Managing the Dynamics of New Product Development Processes

Arie Karniel · Yoram Reich

Managing the Dynamics of New Product Development Processes

A New Product Lifecycle Management Paradigm

Springer

Dr. Arie Karniel
School of Mechanical Engineering
Tel Aviv University
Tel Aviv
Israel
e-mail: ariek@eng.tau.ac.il

Prof. Yoram Reich
School of Mechanical Engineering
Tel Aviv University
Tel Aviv
Israel
e-mail: yoram@eng.tau.ac.il

ISBN 978-0-85729-569-9 e-ISBN 978-0-85729-570-5
DOI 10.1007/978-0-85729-570-5
Springer London Dordrecht Heidelberg New York

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Cover design: eStudio Calamar S.L.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

In contemporary globally competitive markets, the development of new products is considered a critical success factor for organizations. Significant efforts have been invested in the development of methods and tools for improving the management of design processes, being a key element in development processes, especially those related to new products development (NPD). Product lifecycle management (PLM) is a collection of practices, methods, and tools that help organizations cope with the increased complexity of today's engineering challenges. Like most existing methods and tools they have an underlying assumption that enough product-related knowledge is known a-priori, thus planning the development process and its design processes can be done in advance, and then be followed through careful management. However, the large number of failures in introducing new products to markets suggests the opposite. Top-down and bottom-up managerial strategies applied to the planning and management of NPD fail many times, as existing tools and methods are incapable of supporting the process management challenges of NPD processes.

The managerial issues include the planning of unexpectedly changing and iterative processes and their modeling for implementation, execution, monitoring, and simulation. Current commercially available tools (e.g., workflow tools within PLM) typically support predefined processes, whose structure is fixed; existing planning methods typically support predefined process knowledge; and many modeling methods address only non-iterative processes. The result in practice is that planning and monitoring of NPD processes are typically done manually. This creates a gap that is difficult to manage between the roles of product and project managers. While product managers are responsible for the delivery of a product that meet customer and market demands, i.e., the content of the product, project management are responsible for the execution of a product development process so that the product is delivered on time, within budget and risk constraints. However, whereas evolving product information and process information could be managed by their respective established tools, the integration of updated product data into process management is not established. Consequently, project managers base their decisions on past, partially obsolete product information, instead of newly

generated information of the product developed. The obvious result is that project and process plans are inferior to what is possible with updated information and a plausible result is that resources are wasted, leading potentially to project failures.

This book focuses on a framework, entitled Dynamic new-Product Design Process (DnPDP) for the management of NPD processes (iterative processes with process schemes that changes according to the evolving product knowledge). The framework incorporates a model of a closed-loop meta-process management system that controls the dynamically evolving workflow process.

The novel integrated approach of the presented framework merges product-based planning, process modeling, process execution, probabilistic simulations, and simulation based decision-making. The conceptual framework was implemented through the choice of specific methods as building blocks of the meta-process, into a working system. It is important to notice that the specific choice of building block concepts used for implementation is only one among other potential implementations.

The system that implements the framework is capable of process modeling and execution according to plans that change based on evolving product knowledge. Therefore, it can manage iterative processes with dynamic changes of the process structure. Simulations of potential process changes are analyzed using statistical analysis, and analysis results are used for decision-making during the process. As such, the system can be used as a decision-making aid.

The system incorporates two established methods: the Design Structure Matrix (DSM) and Petri nets. The DSM is a method for modeling some aspects of product knowledge, and then using reordering algorithms for planning the design processes based on that knowledge. This method, originally used for pre-defined product knowledge, was extended to deal with evolving product knowledge (typical of NPD). Petri net formalization concepts were used for formally proving process structure properties that result from DSM-based plans.

The interpretation of the DSM-based plan to a process scheme is not unique and its translation to a process workflow may lead to implementation problems. Workflow (WF) nets, being a subclass of Petri nets, provide formal tools for verifying process properties, and establishing the *soundness* correctness criteria. WRI-WF-nets are a subclass of WF-nets that can be hierarchically built and are sound by construction (i.e., keeping correctness criterion). Therefore, they enable an automated process build approach based on the changing product knowledge that is required for NPD simulations. However, WRI-WF-nets that are capable of modeling the typical DSM-based processes do not support the modeling of special logic cases.

Business rules that were defined in this work are used for addressing more general logic cases and define the specific implementation logic. The business rules address the logic layer of the process conversion to DSM nets. DSM nets (developed in this work) are used for process modeling, using (dynamically modifiable) logic activities. The DSM nets are generated according to the DSM reordering by several translation stages. The resulting process scheme plan can be simulated, executed, and formally verified. The formalization methods were

utilized to prove that DSM nets are equivalent to WRI-WF-nets in typical DSM-based process models; therefore, the proposed translation can be automated and is inherently sound. The system was applied to a case study of designing a simplified product. Various settings of product knowledge changes, causing various changes of the design process structure were checked, while analyzing various simulation parameters. The main findings of the case study analysis are that simulations could help in decision-making. Yet, the simulation results need statistical analysis in order to distinguish between significant results and insignificant ones; otherwise, conclusions might be inadequate. Additionally, the complex cases represented by evolving iterative process were highly sensitive to the specific simulation parameters. Therefore, general rules of thumb are inapplicable in such complex cases. The results demonstrated the need for case specific aiding tools in NPD processes. The presented implementation of the conceptual framework demonstrates the feasibility of the suggested framework and its capabilities. The extension and merger of DSM concepts and Petri net concepts presented, enabled an integration of planning, modeling, and implementation that can be automated and support dynamic modifications of the process scheme. The presented integration bridges between DSM and Petri nets, as the reordered and extended DSM are capable of generating WRI-WF-nets.

The main conclusion is that NPD processes can be managed by dynamic workflow tools, though additional knowledge needs to be captured as part of the on-going process. Such knowledge updates can be captured and managed by existing Product Lifecycle Management (PLM) tools. Enhancing PLM capabilities to match the process complexity by the use of the integrated approach (product-based planning, process modeling, execution, and simulation) could help in better management of NPD processes. Once this is accomplished, project managers would be able to carry out their work based on updated product information. It is argued that the integration of dynamic process-scheme management capabilities to existing tools would make the use of computer aided management tools or PLM tools more effective and consequently, prevalent. Altogether, the presented approach creates a consistent base of information for products and processes constituting a new PLM paradigm. This paradigm complements other PLM efforts by focusing on the neglected link between product and process addressed in this research.

This book is geared toward graduate students in, and researchers of engineering design, product development, and development processes. The first three chapters provide solid background to others interested in product development and its management. Bridging the two areas of product development and process management, the book provides each area a window into the other area that could be extended for the benefit of each field. We call PLM tool developers to implement the conceptual enhancements described.

Tel Aviv, May 2011

Arie Karniel
Yoram Reich

Acknowledgments

The work reported in this book was partially funded through the "Reverse Engineering processes management" research project of the CONSIST consortium, under MAGNET program funded by the Industry and Commerce Ministry, and Israel Aerospace Industries.

Partial support was also obtained from The Israel Science Foundation under grant 765/08.

The warmest and greatest gratitude of all goes to my wonderful family, my mother Carmela, my daughters Doron and Dana, and my son Dvir who bring happiness to my life. My wife Einat, my partner and best friend who helped me in more ways than I can possibly recount, I am eternally grateful for her love and care. I could not have done it without her.

In memory of my father David Karniel.

Arie Karniel

I thank my parents Rina and Yoseph Reich, for carefully managing the dynamics of my development; my children Clil and Shaked, for teaching me how I should exercise such management and my partner Nurit, for creating an inspiring atmosphere.

Yoram Reich

Contents

Abbreviations and Symbols

ASA	Adaptive simulated annealing (optimum search method)
Autom	Automation
Appl	Applications
Artif	Artificial
Bus	Business
BPM	Business process management
BR, BRs	Business rule(s) (Implementation options)
C-process	Current process
CAD	Computer aided design
Comput	Computer
Constr	Construction
CDR	Critical design review
CI matrix	Coupling index matrix
CLT	Central limit theorem
CRM	Customer relationship management
DAG	Directed acyclic graph
DMM	Domain mapping matrices
DnPDP	Dynamic new-product design process (the framework)
DPM	Design process matrix
DSM	Design structure matrix
Eur	European
EPC	Event-driven process chain
ERP	Enterprise resource planning
GERT	Graphical evaluation and review technique
Inf	Information
Intell	Intelligence
Int	International
IR	Implementation rules
J	Journal

LLN	Law of large numbers
Manag	Management
MDM	Multiple domain matrix
Manuf	Manufacturing
NPD	New product development
PDM	Product data management
PDP	Product development process
PDR	Preliminary design review
PLM	Product life-cycle management
PN	Petri net
Res	Research
RT-Process	Run time process
SA	Simulated annealing (optimum search method)
SCM	Supply chain management
Softw	Software
Sys	Systems
Technol	Technology
WF (net)	Workflow (nets)
WfMS	Workflow management systems
WRI (WF-net)	Well-handled with regular iterations (WF-net)
WTM	Work transformation matrix
YAWL	Yet another workflow language

A, *B*, *C*, *D*, *E*, *F*, *G*, *H*	Activities
A_i, a_i	Indexed activity
B, Be	Begin activity (*Be* is used when *B* is an activity)
bi	Indexed activity
C	Path (in Petri net); closed loop constant (Eq. 4.1)
CI( )	Mapping function of an activity to its path index in path in *C*
D(*A*), D(*DB*)	Duration of design activity *A*, or design block *DB*
DB, *Dbi*	Indication of a design block (*DB*), indexed *DB*
DBIL, *DBOL*	Design block input logic, and design block output logic
$D_{i,j}$	Distance function (in Eq. 4.1)
DN	DSM net
DnP	Conversion mapping from DSM net to Petri net
E, *En*	End activity, end state (in Petri net). (*En* is used when *E* is an activity)
F	Set of arcs (in Petri net); forward constant (Eq. 4.1)
F_i	Indexed forward link (from a previous activity)
f, f_{ij}	Edge, indexed edge (in conversion to Petri net)
iff	If and only if
i,j,k	Indices

i,o	Starting place (i) and termination place (o) of WF-net
I_{i}	Indexed iteration (feedback) link (from a following activity)
IL, OL	Input logic, output logic
ln( )	Function of mapping an activity to its index
I_{α}	Confidence interval (of confidence level α)
J-A, J-O, J-X	Join-And; Join-Or; Join-Xor;
K	Batch size (statistics)
L, l	*L*-Set of links, *l*-Link
l_k, l_{ij}	Link with single Index, and with two indices (row, column)
LFI, *LFO*	Set of links: forward input, forward output
LII, *LIO*	Set of links: iteration input, iteration output
LR	Learning ratio (of the duration of activity execution $i+1$ to previous execution)
LSI	Set of self iteration links
M	Marking (in Petri net)
m	Number of elements (e.g., in a path)
MF, *MI*	Multi forward link, multi iteration (feedback) links
N, *n*	Number of activities, number of nodes (in Petri Net)
NA	Not applicable
P, *p*	*P*-Set of places, *p*–place (in Petri net)
p, *P*(*A*), *P*(*A*, *B*)	Probability; Self-probability of activity, or design block *A*; Probability of a link from *A* to *B* (in DSM)
$P(\ ), P^{-1}(\ )$	Probability function or Mapping of a DSM link to a triple (two edges and place) in Petri net, and inverse mapping (Definitions 36, 37)
PS, *PE*	Parallel start, parallel end
PN, *PN**	Petri net, extended Petri net
q	Power of *D* (Eq. 4.1)
R	Process resources, number of simulation runs
Ro( )	Permutation function of DSM activities
S	Starting state (in Petri net)
S-A, S-O, S-X	Split-And; Split-Or; Split-Xor;
So( )	Mapping of a link to its source activity
T	Process time (simulation results)
T, *t*	Transition: *T*-Set of transitions, *t*-transition (in Petri net)
T( )	Mapping of DSM activity to Petri net transition
Ta( )	Mapping of a link to its target activity
Ti(*ai*)	Internal time of a design block—in which activity *ai* was added
Tk	Artificail teperature in ASA
W, *X*, *Y*, *Z*	General activities

x, y, x_0, x_i, x_j, x_n	Nodes, indexed nodes (in Petri net)
x_i	Indexed activity
α	Alphabet operator (of a path C); confidence level (statistics)
σ	Firing sequence
$\varnothing$	Empty group (null)
$\bullet p$, $p\bullet$	A node (transition) before / after place
$\bullet t$, $t\bullet$	A node (place) before / after transition
$\forall$, $\exists$	For every, exist
$\cap$, $\cup$	Union, intersection
$\in$, $\notin$	Element of, not element of
$\subseteq$	Sub set
$\{\}$	Set of elements
(.. , .. ,)	List of ordered elements (with or without commas)
$\mid$	Condition (on elements in set)
$\wedge$	And condition
$\otimes$	Composition of nets
$\Rightarrow$	Split logic; then (if-then statement in Petri nets)
$\Leftarrow$	Join logic
$\bullet$, +, $\oplus$	And, OR, XOR logic
$\sum(Ai)$, $\prod(Ai)$, $\otimes(Ai)$	Multiple And, OR, XOR logic
$\xrightarrow{t}$	Transition enabled in marking (of Petri net)

Note Additional symbols that are specific to the statistical annex, in Sect. 14.2, are defined within that annex.

Part I
Current State of the Art and Enhancements of Existing Methods

Chapter 2
Managing Development Processes

Reaching the goal of automating the planning and execution of a DnPDP, as well as simulating the process under diverse conditions and decision options is quite complex. It requires the integration of several approaches. The following chapter describes the methods and approaches used, enhanced, and integrated within the suggested DnPDP framework.

In the following chapter, the literature on process management is surveyed. Then, two major methods are described: the DSM, which is used for planning and simulation; and the Workflow (Petri) nets concepts that are used for process modeling and verification. The use of DSM-based process plan is discussed in relation to the process-logic being applied and the simulation parameters used for scheduling. The implications of the concepts used for process-logic modeling are described followed by a detailed comparative analysis of DSM-based simulation approaches (Karniel and Reich 2009). The use of DSM-based processes for simulation is further discussed in the context of the activity scheduling literature, surveying simulation parameters, and objective targets.

The definitions within and between the integrated fields are inconsistent; therefore, the first section is devoted to explicitly defining the terminology used in this book.

2.1 Development Processes

Product Development Processes (PDPs), and primarily New-Product Development (NPD) processes are highly complex, dynamic, iterative, and unique. They are complex because they involve multiple disciplines (Simon 1981; Subrahmanian et al. 1997; Reich et al. 1999) contributing to the development of complex multi-disciplinary products (e.g., mechatronics), that have limited resources, shortened development time, and increased quality and regulatory demands (Ulrich and Eppinger 2000; Kusiak 2002; Shane and Ulrich 2004). They are dynamic because

A. Karniel and Y. Reich, *Managing the Dynamics of New Product Development Processes*, DOI: 10.1007/978-0-85729-570-5_2,

they evolve continually due to various reasons including market, technology, and organizational changes (Fricke et al. 2000; Otto and Wood 2001). They are inherently iterative due to the interdependencies between the design activities (Yassine and Braha 2003). They are unique, though they may share common features or elements (Smith and Morrow 1999).

A PDP is a collection of related activities targeted to convert a new idea, concept, or market opportunity, into a marketed product. Within PDP, the cyclic Design Process (DP) includes synthesis, analysis, and decision-making stages for migrating from marketing requirements to artifact specification. The DP reflects the specific design requirements, objectives, and constraints applicable to the current product.

The interdependence between design activities in the DP makes that process fundamentally iterative as changes in the design of one component may result in changing the design of other components. While a-priori planned iterations such as testing are an inherent and necessary part of design processes (Lévárdy and Browning 2005), feedbacks due to unplanned changes cause additional iterative rework that cannot be predicted (Westfechtel 1999). Iteration s are considered a major source of increased product development lead-time and cost (Eppinger et al. 1994; Carrascosa et al. 1998; Browning and Eppinger 2002; Whitfield et al. 2005). Therefore, performing project activities in an appropriate sequence is critical in minimizing rework due to information interdependencies (Meier et al. 2007). Planning and predicting the consequences of changes in a product are essential for modifying existing products, especially safety critical products (Eckert et al. 2006).

The need to align the DP with the product's specific characteristics differentiates its planning from other processes, where the relations between activities and the order of activities can be guided by heuristics or just by planner's common sense.

Among all DPs, those related to NPD processes present additional planning complexities as the knowledge required for planning the process is only partially known initially (Reich 2008; Smith and Morrow 1999), and is changing during the development process (Ulrich and Eppinger 2000). Figure 2.1 depicts the product knowledge and design freedom as a function of time (Reich 2008; Ullman 2003). It also shows the quality determined and cost incurred as a function of time. From the quality determined graph, it is clear that the most important time is the beginning of the project where knowledge is scarce. This paradoxical situation (Reich 2008) inevitably leads to changes in the product and the process. Consequently, we cannot hope to define the scope of work needed for a new product, nor we could plan how to manage it a-priori (Westfechtel 1999). Hence, the planning should be repeatedly updated during the process, incorporating the additional product knowledge (new activities, or new relations) that is gained (Karniel and Reich 2007a). Moreover, due to diminishing design degrees of freedom, the design process will involve iterations to redo previously accomplished design activities.

In order to coordinate a complex design process effectively, a workflow model needs to define iterations or coupled dependencies in design processes. However,

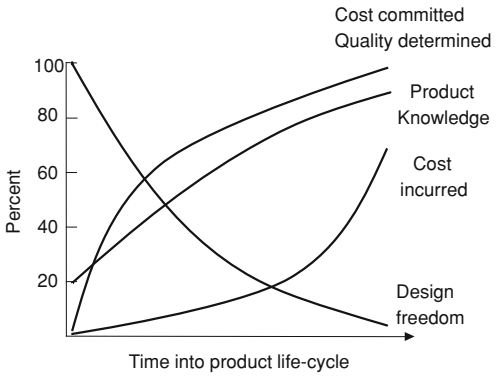


Fig. 2.1 Product knowledge, and design freedom vs. time (Reich 2008, modified from Ullman 2003)

few attempts have been made to develop a systematic framework that considers these complex features based on existing workflow models (Lee and Suh 2006).

If basic needs of NPD processes, such as the above, are not met with proper tools or methods, failures ensue. NPD processes fail in various ways (Bobrow and Shafer 1987; McMath and Forbes 1998; Goldenberg et al. 2001). During the past years, there have been many studies and reviews on the factors that influence the fate of new products (Brown and Eisenhardt 1995; Calantone et al. 1996; Cooper and Kleinschmidt 1995; Di Benedetto 1999; Goldenberg et al. 2001; Griffin and Page 1996; Griffin 1997; Montoya-Weiss and Calantone 1994; Shenhar 1998; Verhaeghe and Kfir 2002; Worren et al. 2002).

Famous failures were analyzed by Bahill and Henderson (2005), categorizing them to requirements, verification, and validation issues. Thomas (2007) analyzed failures of space systems due to inadequate system engineering processes. The engineering process deficiencies were categorized to project planning, project assessment, information management, requirements analysis, architecture design, implementation, verification, and transition process (the capability to provide the service in operational environment). Several cases from both sources related to design processes issues are described in Table 2.1. Since there is an overlap of several case descriptions, they were merged.

The examples demonstrate the importance of proper design process. It is important to state that the implementation of management tools could not resolve all potential issues, e.g., design bugs or decisions not to perform testing. Yet, using the proposed method and management tools could help in better communication (reducing unreported issues as in the GE case, or design misalignment in the Mars orbiter), and could help in mapping required design changes of some system components, due to changes of other components or changes in requirements (Ariane 5, Lewis).

In Fig. 2.2, project management practices are detailed along the dimensions of technological uncertainty and systems' scope. As the system scope or the technological uncertainty increase (or both), the impact of the corresponding practices

Table 2.1 Famous development failures (Bahill and Henderson 2005; Thomas 2007)

System	Year	Failure description
HMS Titanic	1912	Poor quality control of manufacturing the iron rivets caused many of them to fail on hitting the iceberg
Apollo13	1970	The operating voltage of the thermostatic switches for the heaters of the oxygen tanks were changed from 28 to 65 V. However, specifications were not changed, and they were not tested
General electric—rotary compressor for refrigerator	1986	The compressor parts were designed using two powdered-metal. On testing, no compressor failed. However, motor windings were heating, bearing surfaces appeared worn, and the sealed lubricating oil seemed to be breaking down. The latter results were not reported to upper management. The 1988 report indicated that the powdered-metal parts were wearing excessively, increasing friction, burning up the oil, and causing the compressors to fail. During 1989, one million defective compressors were replaced
Hubble space telescope	1990	On orbit it was found that both cameras had distortion (spherical aberration) caused by a flawed shape of the primary mirror. The wrong shape was due to wrong setup of the optical testing equipment used during manufacturing. Tests of the mirror were conducted and found an error, but the error indicators were dismissed. The development of guidance, navigation, and control (GNC) system required additional budget, thus system integration tests were not performed. Such test would have detected the problem. Another shuttle mission was required to fix the problem by adding five pairs of corrective mirrors
Ariane 5 missile	1996	The inertial reference system (and its back up) stopped functioning at 36.7 s after launch. The main failure source was the reuse of software from Ariane 4, specifically a realignment function, but with different implementation and with inadequate testing of the changes consequences by simulation. The Ariane 5's higher horizontal-velocity generated the excessive value, (overflow of the 32-bit register), which caused the inertial system computers to cease operation. The missile blew up on its first launch
Lewis spacecraft—orbit spin	1997	The orbit spin caused off pointing of the solar array. The spacecraft lost control, could not recharge, and was destroyed. The control system was a heritage design (TOMS[a] spacecraft). Yet, there were major changes of configuration and requirement. Lewis had its intermediate/unstable axis toward the sun, while TOMS had its principle/stable axis toward the sun. Thus, minor rotation perturbations caused entering a spin. After thrusters firing in order to control, they were disabled to preserve fuel, leaving the spacecraft uncontrolled. Simulation tests were done using the tools developed for TOMS, without modeling perturbations, initial (small) spin rate of the unstable axis, or larger drag forces applicable to the Lewis mission
Mars climate orbiter	1999	Lockheed Martin used English units for the satellite thrusters while the operator, JPL, used SI units for the ground-based model of the thrusters. The mismatch caused wrong calculated orbit altitude at Mars. Instead of orbiting, it entered the atmosphere and burned up

[a] Total ozone mapping spacecraft

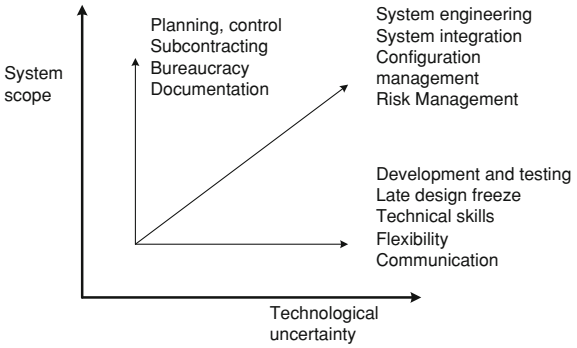


Fig. 2.2 Project-management practices along uncertainty and scope dimensions (adapted from Shenhar 1998)

becomes more important (e.g., testing in a case of technological uncertainity, as in the Hubble case).

Hauser (2001) has identified about eighty factors affecting project success in empirical models. The outputs of the empirical studies were either correlations between independent variables (e.g., marketing proficiency and R&D skill) and dependent variables (e.g., project success), or best practices of successful organizations (e.g., top third of organizations). Unfortunately, these scientific findings are not used by practitioners (Hauser 2001, p. 135): "Many (managers in industry) are aware of the scientific literature that studies the antecedents and consequences of metrics in multi-firm studies. But to fine-tune the culture of their firm, these managers need a method that adjusts priorities based on measures of their firm."

Hauser (2001) takes an adaptive control viewpoint. It treats the executions of actions and decisions as a black box, forcing decision-makers to rely on the mathematical model without understanding the deeper interactions between the project parameters.

A different approach introduces more detailed product information and favors executing simulations and using them to make decision regarding the NPD business and engineering processes. Simulations could provide explanatory power that is lacking from empirical models. Reich and Paz (2008) developed a method that makes use of detailed design information and provides insight about project risk and potential product quality. A similar approach is utilized in this research, using product knowledge for simulating the process as an aid to process management decision-making. Referring back to Fig. 2.1, such simulations that consider evolving product information that are exercised at the initial stages of design, even if they require some resources, would generate valuable knowledge that is so sought for leading to better design decisions.

While many approaches provide modeling methods for improving the design process, they do not provide a systematic guideline for applying the design strategies to the design process operational-level (Lee and Suh 2006). Furthermore, many approaches of process planning assume that the planning is done once. Process execution and monitoring tools are based on a pre-defined process scheme, which was implemented once. Simulation tools used for decision-making are

based on pre-defined static schemes. However, such one-time static process plan assumptions are inadequate for NPD processes due to the partial knowledge at the beginning and knowledge evolution during the process.

2.2 Administrative Approaches for Managing NPD Processes

Organizations use top-down and bottom-up approaches to cope with NPD process management. The top-down approach applies common process management practices by creating high-level, fixed, and structured model of the development process, and using traditional Workflow Management Systems (WfMS). The bottom-up approach manages each activity as it arises, by assigning it to a resource and managing it in a global repository (e.g., WfMS), while ignoring the interdependencies between activities. Both approaches do not support the dynamics of NPD process planning.

NPD processes cannot be reduced to a fixed structure. A-priori breaking the design hierarchically does not address the problem; it simply defers it and the dynamics are concealed inside the model constructs.

Various companies have distinct development processes. The simple linearized process development model depicted in Fig. 2.3 includes typical process stages.

A similar process, based on ISO/IEC 15288 (2002) system life-cycle processes, is described in (Thomas 2007). It includes the stages: concept, development, production, utilization, support, and retirement. Comprehensive comparisons of the development processes in ten industrial companies' case studies are described in (Unger 2003).

Generic models of development processes that are widely acknowledged are Waterfall and Stage-Gate (Cooper 2001; Otto and Wood 2001). Other generic process models that were primarily adopted by the software industry include the Spiral model (Boehm 1986), Evolutionary prototyping (McConnell 1996), and eXtreme Programming (XP) (Beck 2000). The Spiral model includes planned cycles over several development stages; thus, it takes into account the iterative nature of the process. The Evolutionary model identifies the need to refine the prototype until it is acceptable, without predefining specific process activities. In extreme programming, the activities are generally set and specific design activities are planned on a weekly basis.

Focusing on the stages from Concept to Production in Fig. 2.3, and utilizing a more practical modeling approach might yield, for example, the process model depicted in Fig. 2.4. This high-level process includes parallel activities, decision points (e.g., Stage-Gate model; Cooper 2001), and iterations. Being a high-level process, its scheme is typically constant, yet might have *ad-hoc* sub-process changes.

The Specification activity starts in parallel to the Conceptual Design. Decisions regarding iterations of activities can be made in the Preliminary Design Review (PDR) or the Critical Design Review (CDR). The decision taken may be continuing the project, iterating, or project cancellation. The process described

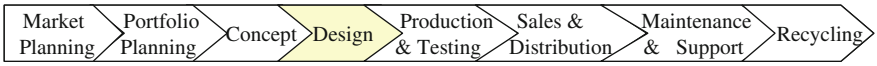
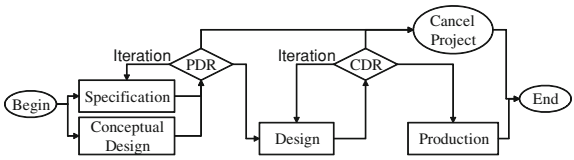


Fig. 2.3 Development process model (linearized)

Fig. 2.4 High-level process model



specifically indicates Begin and End activities that are required for process implementation (Karniel and Reich 2007b).

At the high-level process definition, the process model might be considered relatively invariable and may apply to many products. We shall refer to such pre-defined process model as *Process Skeleton*, as its actual activity content is defined according to the product. Typically, industrial development organizations have different high-level development process skeletons, which represent the organization best practices.

It is typically assumed that lessons learned in one development process could be applied to improving other design processes in similar contexts (Wynn 2007). While such assumption could be appropriate for implementing process plans at a high-level, it is not adequate for the detailed planning of the NPD processes that dynamically evolve based on the particular product requirements. Concluding the implementation of a process modeling tool (Signposting) in several projects, Wynn (2007) indicated the need for synthesizing product-specific information with the process knowledge and simulating the process uncertainties due to iterations.

A distinction should be made between process modeling and the process Capability Maturity Model (CMM). For example, the CMMI (SEI 2002) contains 25 required and expected process areas (such as Requirements Management and Risk Management), and about 185 practices. However, the CMMI does not fully define the interactions, or flow between the practices, and it does not cover the specifics of engineering design for particular products. Thus, CMM may take into account the kinds of activities to include in a process model, but not how to string them together at the level of execution. A CMM is a model of a process's capability and maturity, not a model of the process itself (Browning et al. 2006).

2.3 Process Dynamics Classification

A process case (i.e., the development of a certain product) is dynamically evolving at run-time according to the process scheme. In the current research, we define the following classification of dynamic process change types:

Class I—Constant process scheme dynamics: The dynamics will include the actual decisions being made at run-time and resource allocation. In iterative design processes, the activity content may change in each iteration leading to different types of activity outcome at each iteration (O'Donovan et al. 2003).

Class II—(a) Exception Handling: In a case of an exception (i.e., specific pre-defined situation), a special pre-defined subprocess is added to the main process scheme. It is actually a sub category of class I, since the process is actually pre-defined to a detailed level. Yet, from the user perspective, the process scheme details are elaborated and changed at run-time; thus, each process instance follows a distinct process scheme (Klein and Dellarocas 2000; Russell et al. 2006). A survey of commercial and academic exception handling tools can be found in (Adams et al. 2010). (b) Hierarchy expansion of replacing an activity by a pre-defined subprocess can also be regarded as a sub class of exception handling. A complex behavior can be achieved if the replacement of the activity is done during run-time with one of several optional subprocesses, where the actual subprocess is chosen according to process parameters and according to pre-defined logic. (c) In (Adams et al. 2010) the replacing sub-process can be defined (by an administrator) during the execution of the main process and made available to the main process in case of exception. Such approach could be regarded equivalent to the ad-hoc changes type.

Class III—Ad-hoc changes (excluding product knowledge changes): manually changing the process scheme in unpredicted manner (typically uncontrolled) (Dustdar et al. 2005); the user takes the responsibility for making a correct change. The user is expected to use validations tools for checking the correctness of the changes (van der Aalst and ter Hofstede 2005).

Class IV—Dynamic change of the process scheme due to changes in the product knowledge: (a) adding or deleting activities at each iteration (Rinderle et al. 2004b): changes of the activity order at different iterations based on changes in knowledge (Clarkson et al. 2000; Lévárdy and Browning 2005), where the changes are not pre-defined; (b) changes of whole process blocks in a workflow environment is described in (Reichert et al. 2010) (c) hierarchical expansion of activity, where the activity being expanded was not pre-defined (defined only during run-time), or the sub-process content was not pre-defined, or both (Karniel and Reich 2007a).

Class V—Dynamic change of the Process Scheme due to decision-making changes, e.g., changes in the logic interpretations (defined by BR).

According to the above, the High-level processes, as well as most commercial WfMS tools are classified as having Class I dynamics, i.e., the process scheme is constant. The distinctions between class IV and V are the user perspective and the abstraction level. While product knowledge changes can be considered as design process outcomes (uncontrolled), the changes of the BRs are due to managerial decision-making. Additionally, a BR will typically not add activities, but will change links and activities aggregation to design blocks. A schematic description of the class differences is depicted in Fig. 2.5.

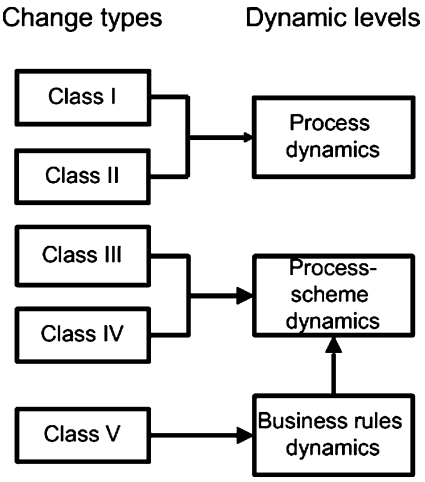


Fig. 2.5 Dynamic process classes

The definitions of change classes IV and V are new (class IV was not fully defined in the literature) and constitute the motivation of the current research. The change classes can be further categorized to *dynamic levels* indicating a suggested ranking of dynamics. Starting from dynamics of the process within a fixed process scheme, next, the scheme itself is changing according to specified rule, and finally, the rules defining the scheme changes are changing dynamically (and change the process scheme). The implications of this classification are further discussed in Chap. 1.

2.3.1 Adaptive Processes

In administrative processes, a case (or process instant) runs through the (predefined) process scheme model. A change of administrative processes scheme should typically apply to all the process cases, which may be in different process stages (Rinderle et al. 2004a). A survey of adaptive workflow changes for administrative processes appeared in (Klein and Dellarocas 2000).

Sadiq et al. (2001) made the following classification of methods for dealing with process scheme change: Flush—current process instances complete according to old process model. Abort—current instances are aborted and restart according to the new model. Flush and Abort are the most common solutions in commercial workflow systems. Migrate—new process segments are integrated into the process in parallel to replaced process segments. Instance actions are flushed through the replaced segment. Adapt—the process is altered individually for each instance according to its state. Build—the whole process is built at run-time corresponding to the particular situation.

A combined approach for managing multiple process cases was described in Reichert et al. (2005). Process *Migration* was performed for most process cases. If the changes could not be implemented, the cases were flushed through the changed process segments; Ad-hoc changes followed process *Adaptation*, i.e., each case had its individual process instance segments that differed from the general process.

However, in design processes, and particularly NPD processes, the product-related knowledge is partial at the beginning, and is evolving during the process. Since process scheme changes (activities and their links) are anticipated, a cycle of process planning, modeling, verification, execution, and simulation should be repeated. Such Process Management Cycle approach is required for every process that is both iterative and changing due to evolving knowledge (Rinderle et al. 2004a; Lévárdy and Browning 2005). Any constant scheme approach, which can be successfully employed under other conditions, fails to support the requirements of NPD processes (and other less challenging design processes). Rinderle et al. (2004a) indicated that "current adaptive WfMS do not allow propagating WF type (process scheme) changes to individually modified WF instances," which is important for the adequate support of long-running workflows, or in NPD case, for unique processes.

Since in design processes, the changes are local to one case only, a *Build* approach is required: building the process at run-time in correspondence to its particular situation and knowledge. This approach is used in this study.

2.4 Process-Models Classification

For the purpose of our study, it is convenient to make the following process-model classification: pre-defined processes (*P-process*), current plan processes (*C-process*), and run-time process (*RT-process*) (Karniel and Reich 2007b). Each process type has its own characteristics. As previously defined in the ontology, the design activities are considered as subprocesses, i.e., there is no notation of an atomic task. Such concept is easily adapted to building hierarchical processes, where each activity may have a more detailed content. A similar concept of three-tier approach including planning layer, scheme layer and instance layer was presented in (Narendra 2004).

2.4.1 Pre-Defined Processes (P-process)

Two types of P-processes are the *high-level*, best-practice processes, and *administrative* processes. A high-level process is the initial skeleton of the design process (c.f., Fig. 2.4). At such high-level, the detailed design stage is completely modeled by one activity (as the actual process details are set according to the product). Using the hierarchical approach, this activity becomes a subprocess, once

there is some additional knowledge about its structure. Using a hierarchical exception handling approach, this activity could be replaced by a pre-defined subprocess (according to a pre-defined set of options). Administrative processes are relevant to many design activities and can occur within any design activity. A common design activity subprocess is the Engineering Change Order (ECO), used for changing the design of a component once the component was released (Loch and Terwiesch 1999). Another example could be the assignment of a part number to a new component; typically, such activity requires cooperation of an engineer with purchasing and logistics employees. Current workflow systems perfectly fit the management of these (sub) processes.

Both process types are typically well-established, with relatively low rate of changes. Changes can be implemented as ad-hoc changes; otherwise, flush or abort methods are used.

2.4.2 Current Processes (C-process)

The current process is the process scheme plan, which should be followed at run-time. For fixed process schemes (Class I), the pre-defined process is the current process. In exception handling, the optional process schemes are not presented to the user; thus, the user observes the current process sections that are applicable to the current case. In (Reichert et al. 2005), local deviations of each case from the pre-defined scheme are allowed; thus, a deviation of the current model is kept for each case running through the process.

According to our approach, since design processes are considered unique, each design process has its own current process scheme. Since the process plan is expected to change, the current process scheme may change in time. In different points in time, there might be different current process schemes.

2.4.3 Run-Time Process (RT-process)

The run-time process follows the current process scheme plan. For a constant DAG process (no iterations), the RT-process is identical to its C-process. Once we have iterations we need to manage the actual path of the process, i.e., manage history and the number of times the process has passed through a planned activity. A simple example is depicted in Fig. 2.6 showing some optional RT-processes (Fig. 2.6b) derived from the fixed C-process scheme (Fig. 2.6a).

Iteration s within a fix process scheme could be modeled and monitored. Colors can be used for visually indicating which activities are currently active, which were already visited, and which were not. A history list can be used for indicating the number of times an activity was executed. However, when trying to manage an iterative process over a changing scheme, a path between two activities could

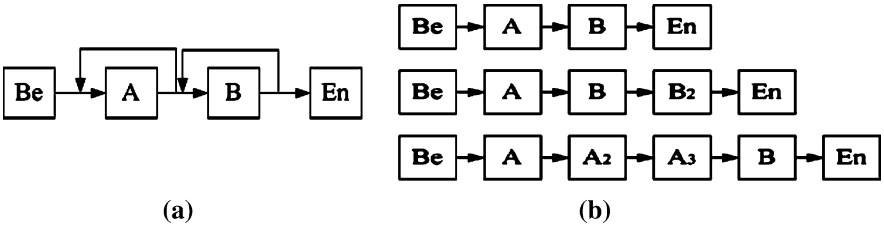


Fig. 2.6 RT-process of iterative C-process. **a** C-Process. **b** RT-Process

change between the iterations; therefore, the current process scheme can no longer be used for recording the run-time process, and an additional process model is required. The RT-process will actually follow partial sections of the changing C-process (examples in Chap. 1).

The different function each process type has, derived different characteristics. For example, C-process (and P-process) should model the optional feedback links. RT-process model has no feedback links; it progresses forward (with time) from one activity to the next. In case of self-iteration, it actually progresses from the first occurrence of an activity to its second occurrence.

2.4.4 WfMS Tools

WfMS tools are nowadays typically embedded in broader context such as Enterpise Resource Planning (ERP), Product Lifecycle Management (PLM), and Business Process Management (BPM) solutions. Nevertheless, in spite of their proliferation they support only a small fraction of the change types from Fig. 2.5. Available WfMS tools support pre-defined processes (Class I), where the process model is fixed before execution. The tools vary in the types of process logic (workflow patterns) that can be implemented by them. A comparison of commercial tools by their expressiveness (ability to define various logic types) is presented in (van der Aalst et al. 2003a), while an academic tools comparison appears in (van der Aalst and ter Hofstede 2005).

Some workflow tools do support ad-hoc changes (Class III) of the process scheme for acyclic (no iterations) processes (Reichert et al. 2005), or more elaborated pre-defined subprocesses being added at run-time, i.e., Exception Handling (Class II). However, they do not support general changes in process scheme (Heller and Westfechtel 2003; Weske et al. 2004); and support neither activity decomposition, nor recursive activity execution (Madhusudan 2005). Furthermore, they do not support the planning of process dynamics, or process verification (Verbeek and van der Aalst 2004). Rinderle et al. (2004a) indicated that commercial WfMS do not actually support process scheme changes during run-time. Either they allow changes in scheme level without taking into account the

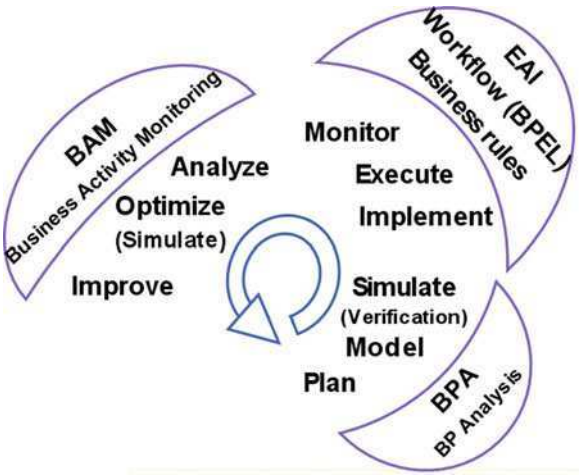


Fig. 2.7 Management of fixed scheme processes

workflow instances (e.g., MQ Series Workflow, Vitria Business Ware) or they propagate changes without consistency checks (Staffware).

Managing the product-specific activities' details, at sub-process level, based on product knowledge (e.g., modeled by the DSM method), is beyond the capabilities of current tools. Due to practices and WfMS limitations, NPD processes dynamics are not managed (as well as other less complex PDP processes).

2.5 A Model of Managing Fixed Scheme Processes

A general view of the overall management of fixed scheme processes is depicted in Fig. 2.7 (Karniel and Reich 2007c). The management process has three main stages: Business Process Analysis (BPA), Workflow, and Business Activity Monitoring (BAM), each having sub stages. This model (a contribution of the current research) aligns the terms and models of commercial tool providers.[1] The BPA includes process planning, process modeling, and business process simulations (BPS) for verification of the plan. The Workflow part is augmented with business rules and Enterprise Application Integration (EAI) to additional application. It includes the process scheme implementation at a workflow engine, and the execution and monitoring of process cases. Finally, the BAM includes performance analysis, optimization, and improvement of the process scheme (Savitha et al. 2005). The activity monitoring is at the overall process scheme level and not the monitoring of a specific process case (included in the workflow stage). Closely related terms are the Business Intelligence (BI), and Business Performance

[1] There is no single model applied by all providers. See Commercial tools web sites.

Management (another BPM). Simulation tools that are used for business correctness verification can be used for analyzing performance check as well.

In a survey of BPM tools (van der Aalst et al. 2003b) the management model has four stages: process design, system configuration, process enactment, and diagnosis. In the design stage manual planning is assumed, then process modeling and verification are done. The system configuration can be mapped to EAI and implementation. Enactment is related to execution and monitoring of the cases; and diagnosis to analysis and optimization using tools such as CPN (Jensen et al. 2007).[2]

Commercial products that are mapped to the first stage are ARIS, Pallas Athena-Protos (Jansen-Vullers and Netjes 2006). The stage regarding process implementation and execution (Workflow) includes many providers (offering modeling and simulation capabilities as well) that typically define themselves as BPM providers [e.g., Oracle, IBM, IBM (Filenet), SAP, Fujitsu, Ultimus, Tibco (was Staffware), Pallas Athena (Flower)], or PLM providers [e.g., Dassault Systems, Siemens (was UGS), PTC, Oracle (was Agile)]. The latter group focuses on management of engineering process integrated with computer-aided tools (CAX, where X = D for design, X = M for Manufacturing, etc.). BAM providers include Tibco, IBM (was Cognos), Oracle (was Hyperion), and SAP (was Business Objects).

PLM solutions are of special interest to our study since they emerged from PDM solutions that were focused on managing product-related knowledge (e.g., CAD files, BOM, and meta data). In recent years, most PLM tools have integrated GANTT like capabilities for process planning (including import/export to MS-project). To date, most PLM solution enhanced the process management capabilities and they typically include ad-hoc changes (Class III). Yet, PLM systems are not exploiting the existing managed product knowledge or the capabilities to manage relations data for planning design processes, i.e., there is no integration between the product knowledge and the project planning tools such as GANTT. Moreover, the current GANTT-based plans are not integrated with the embedded WfMS tools.

Advantages and disadvantages of tools used in industry for process improvement are described in (Wynn 2007), indicating that the reviewed tools do not suit the needs of planning design processes. The case study presented (Rolls-Royce) indicated the business need for planning and re-planning design process activities at the component level for improving the process progress, monitoring, and control.

Despite the availability of tools, their use in the NPD context seems to be poor. Swink (2002) has surveyed 132 NPD projects, querying the impact of 18 tactics and tools, which are typically suggested for accelerating the development phase. The survey results indicated improvement of time-based performance when using

[2] CPN is a Colored Petri Net based process simulator, a free tool by the University of Aarhus, Denmark.

computerized project-scheduling tools (e.g., GANTT based tools like MS-project). The project lateness ((actual time—planned)/planed*100%) was 10% for extensive users of such tools versus 45% for projects that did not utilize computerized tools. However, project-scheduling tools were used only in 40% of the NPD projects.

It can be claimed (and can be tested by a future survey) that the meager capabilities of existing tools (in the context of NPD process requirements) exacerbate the relatively poor usage of computer-aided tools for process management of NPD processes.

References

Adams M, ter Hofstede A, Russell N, Aalst WVD (2010) Dynamic and context-aware process adaptation. In: Wang M, Sun Z (eds) Handbook of research on complex dynamic process management Techniques for adaptability in turbulent environments. IGI Global, Hershey, pp 104–136

Bahill AT, Henderson SJ (2005) Requirements development, verification, and validation exhibited in famous failures. Sys Eng 8(1):1–14

Beck K (2000) Extreme programming explained. Addison-Wesley, Boston

Bobrow EE, Shafer DW (1987) Pioneering new products: a market survival guide. Dow Jones-Irwin, New York

Boehm B (1986) A spiral model of software development and enhancement. ACM SIGSOFT Software Eng Notes 11(4):14–24

Brown SL, Eisenhardt KM (1995) Product development: past research, present findings, and future directions. Acad Manag Rev 20:343–378

Browning TR, Eppinger SD (2002) Modeling impacts of process architecture on cost and schedule risk in product development. IEEE Trans Eng Manag 49(4):428–442

Browning TR, Fricke E, Negele H (2006) Key concepts in modeling product development processes. Sys Eng 9(2):104–128

Calantone RJ, Schmidt JB, Song XM (1996) Controllable factors of new product success: a cross-national comparison. Marketing Sci 15(4):341–358

Carrascosa M, Eppinger SD, Whitney DE (1998) Using the design structure matrix to estimate product development time. Proceedings of DETC'98 ASME design engineering technical conference (Design Automation Conference), Atlanta, GA

Clarkson PJ, Melo AF, Eckert CM (2000) Visualization of routes in design process planning. In: Proceeding of the 4th international conference on information visualisation (IV2000), IEEE Comput Society, London, pp 155–164

Cooper RG (2001) Winning at new products, 3rd edn. Perseus Publishing, Cambridge

Cooper RG, Kleinschmidt EJ (1995) Benchmarking the firm's critical success factors in new product development. J Prod Innovation Manag 12:374–391

Di Benedetto CA (1999) Identifying the key success factors in new product launch. J Product Innovation Manag 16(6):530–544

Dustdar S, Hoffmann T, van der Aalst WMP (2005) Mining of ad-hoc business processes with TeamLog. Data Knowledge Eng 55:129–158

Eckert CM, Keller R, Earl C, Clarkson PJ (2006) Supporting change processes in design: Complexity, prediction and reliability. Reliability Eng Safety Sys 91(12):1521–1534

Eppinger SD, Whitney DE, Smith R, Gebala D (1994) A Model-based method for organizing tasks in product development. Res Eng Design 6(1):1–13

McMath RM, Forbes T (1998) What Were They Thinking? Times Business-Random House, New York
Fricke E, Gebhard B, Negele H, Igenbergs E (2000) Coping with changes: causes, findings, and strategies. Sys Eng 3(4):169–179
Goldenberg J, Lehmann DR, Mazursky D (2001) The idea itself and the circumstances of its emergence as predictors of product success. Manag Sci 47:69–84
Griffin A (1997) The effect of project and process characteristics on product development cycle time. J Marketing Res 34(1):24–35
Griffin A, Page AL (1996) PDMA success measurement project: recommended measures for product development success and failure. J Product Innovation Manag 13:478–496
Hauser JR (2001) Metrics thermostat. J Product Innovation Manag 18:134–153
Heller M, Westfechtel B (2003) Dynamic project and workflow management for design processes in chemical engineering. Proceedings of the 8th international conference on process system engineering (PSE 2003), Kumming, China
Jansen-Vullers M, Netjes M (2006) Business process simulation: a tool survey. In Workshop and tutorial on practical use of coloured petri nets and the CPN tools, Aarhus. http://www.daimi.au.dk/CPnets/workshop06/. Accessed July 2010
Jensen K, Kristensen LM, Wells L (2007) Coloured petri nets and CPN tools for modelling and validation of concurrent systems. Int J Software Tools Technol Transfer 9:213–254
Karniel A, Reich Y (2007a) Managing dynamic new product development processes. Proceedings of the 17th annual international symposium of the international council on system engineering INCOSE'07, San Diego, CA, June 2007
Karniel A, Reich Y (2007b) A coherent interpretation of DSM plan for PDP simulation. In Proceeding of the international conference on engineering design, ICED 07, Paris, August 2007
Karniel A, Reich Y (2007c) From planning to executing NPD processes. In: The 4th annual israeli national conference on system engineering INCOSE_IL'07, Herzliya
Karniel A, Reich Y (2009) From DSM based planning to design process simulation: a review of process scheme logic verification issues. IEEE Trans Eng Manag 56(4):636–649
Klein M, Dellarocas CA (2000) Knowledge-based approach to handling exceptions in workflow systems. J Comput Supported Collaborative Work 9(3/4):399–412
Kusiak A (2002) Integrated product and process design. J Eng Design 13(3):223–231
Lee H, Suh HW (2006) Workflow structuring and reengineering method for design process. Comput Ind Eng 51:698–714
Lévárdy V, Browning TR (2005) Adaptive test process—Designing a project plan that adapts to the state of a project. 15th Annual international symposium of the international council on system engineering (INCOSE), July 2005
Loch CH, Terwiesch C (1999) Accelerating the process of engineering change orders: capacity and congestion effects. J Product Innovation Manag 16(2):145–159
Madhusudan T (2005) An agent-based approach for coordinating product design workflows. Comput Ind 56:235–259
McConnell S (1996) Rapid development: taming wild software schedules. Microsoft Press, Redmond
Meier C, Yassine A, Browning T (2007) Design process sequencing with competent genetic algorithms. J Mech Design 129(6):566–585
Montoya-Weiss M, Calantone R (1994) Determinants of new product performance: a review and meta-analysis. J Product Innovation Manag 11:397–417
Narendra NC (2004) Flexible support and management of adaptive workflow processes. Inf Sys Frontiers 6(3):247–262
O'Donovan BD, Clarkson PJ, Eckert CM (2003) Signposting: modeling uncertainty in design processes. International conference on engineering design (ICED 03), Stockholm, 19–21 August
Otto KN, Wood KL (2001) Product Design. Prentice-Hall, New Jersey

Reich Y (2008) Preventing Breakthroughs from Breakdowns. Proceedings of the 9th biennial ASME conference on engineering system design and analysis, ESDA2008, Haifa, Israel
Reich Y, Paz A (2008) Managing product quality, risk, and resources through quality function deployment. J Eng Design 19(3):249–267
Reich Y, Subrahmanian E, Cunningham D, Dutoit A, Konda S, Patrick R, Westerberg A, the n-dim group (1999) Building agility for developing agile design information systems. Res Eng Design 11(2):67–83
Reichert M, Rinderle S, Kreher U, Dadam P (2005) Adaptive process management with ADEPT2. 21st international conference on data engineering (ICDE'05), pp 1113–1114
Reichert M, Bauer T, Dadam P (2010) Flexibility for distributed workflows. In: Wang M, Sun Z (eds) Handbook of research on complex dynamic process management: techniques for adaptability in turbulent environments. IGI Global, Hershey, pp 137–171
Rinderle S, Reichert M, Dadam P (2004a) Correctness criteria for dynamic changes in workflow systems—a survey. Data Knowl Eng 50(1):9–34
Rinderle S, Reichert M, Dadam P (2004b) Flexible support of team processes by adaptive workflow systems. Distributed Parallel Databases 16(1):91–116
Russell N, van der Aalst WMP, ter Hofstede AH (2006) Exception handling patterns in process-aware information systems. In the 18th conference on advanced information system engineering CAiSE'06
Sadiq S, Sadiq W, Orlowska ME (2001) Pockets of flexibility in workflow specification. Lect Notes in Comput Sci 2224:513–526
Savitha S, Vikas K, Holmes S (2005) Web-log-driven business activity monitoring. Computer 38(3):61–68
SEI (2002) Capability maturity model® integration (CMMISM), Version 1.1. Carnegie Mellon University Software Eng Institute, Pittsburgh
Shane SA, Ulrich KT (2004) Technological Innovation, product development, and entrepreneurship in management science. Manag Sci 50(2):133–144
Shenhar AJ (1998) From theory to practice: toward a typology of project-management styles. IEEE Trans Eng Manag 45(1):33–48
Simon HA (1981) The sciences of the artificial. MIT Press, Cambridge
Smith RP, Morrow JA (1999) Product development process modeling. Des Stud 20(3):237–261
Subrahmanian E, Reich Y, Konda SL, Dutoit A, Cunningham D, Patrick R, Thomas M, Westerberg AW (1997) The *n*-dim approach to building design support systems. In: Proceedings of ASME design theory and methodology DTM '97, New York, ASME
Swink M (2002) Product development: faster, on-time. Res Technology Manag 45(4):50–58
ISO/IEC, Systems engineering: system life cycle processes, ISO/IEC 15288, International organization for standardization, 1 Nov 2002
Thomas LD (2007) Selected systems engineering process deficiencies and their consequences. Acta Astronautica 61(1–6):406–415
Ullman DG (2003) The mechanical design process, 3rd edn. McGraw-Hill, New York
Ulrich K, Eppinger SD (2000) Product design and development, 2nd edn. McGraw-Hill, New York
Unger DW (2003) Product Development process design: improving development response to market, technical, and regulatory risks. Dissertation. Massachusetts Institute of Technology, June
van der Aalst WMP, ter Hofstede AHM (2005) YAWL: Yet another workflow language. Inf Sys 30(4):245–275
van der Aalst WMP, ter Hofstede AHM, Kiepuszewski B, Barros AP (2003a) Workflow patterns. Distributed Parallel Databases 14(1):5–51
van der Aalst WMP, ter Hofstede AHM, Weske M (2003b) Business process management: a survey. International conference on business process management, BPM 2003, Eindhoven, Netherlands. Lecture notes in computer science, vol 2678. Springer, Netherlands, pp 1–12

Verbeek HMW, van Der Aalst WMP (2004) XRL/Woflan: Verification and extensibility of an XML/Petri-net-based language for inter-organizational workflows. Inf Technol Manag 5:65–110

Verhaeghe A, Kfir R (2002) Managing innovation in a knowledge intensive technology organization (KITO). R&D Management 32(5):409–417

Weske M, van der Aalst WMP, Verbeek HMW (2004) Advances in business process management. Data Knowledge Eng 50(1):1–8

Westfechtel B (1999) Models and tools for managing development processes. Lecture notes in computer science, vol 1646. Springer, Berlin

Whitfield RI, Duffy AHB, Gartzia-Etxabe LK (2005) Identifying and evaluating parallel design activities using the design structure matrix. International conference on engineering design, ICED05, Melbourne, August

Worren N, Moore K, Cardona P (2002) Modularity, strategic flexibility, and firm Performance: a study of the home appliance Industry. Strategic Manag J 23:1123–1140

Wynn DC (2007) Model-based approaches to support process improvement in complex product development. Dissertation, University of Cambridge

Yassine A, Braha D (2003) Ccomplex concurrent engineering and the design structure matrix method. Concurr Eng Res Appl 11(3):165–176

Chapter 3
Design Process Planning Using DSM

Complex product development processes can be managed by mapping them through various kinds of project flowcharts and diagrams. The commonly used GANTT and PERT charts are inadequate for planning design processes, as they do not effectively model the design activities interdependencies and process iterations (feedback loops) (Lévárdy et al. 2004; Yassine 2007). A comprehensive survey of methods used for modeling design processes is given in (Browning et al. 2006). The use of multiple views of the different representations for different stakeholders is presented in (Clarkson and Hamilton 2000; Flanagan et al. 2006; Keller et al. 2006; Wynn 2007).

The Design Structure Matrix (DSM) was introduced by Steward (1981a) and extended by Eppinger et al. (1994). It facilitates mapping of interdependencies between elements, and is utilized to capture the required product knowledge. The DSM representation can be used for various applications (Abdelsalam and Bao 2006); it provides the means to identify serial, parallel, and iterative design flows; and model and manipulate iterations and multidirectional information flows (Eppinger et al. 1997); therefore DSM can be used for establishing a plan based on product information (Dong and Whitney 2001).

The (initial) DSM is matrix representation of a system graph, Fig. 3.1a. In such graph, a node represents a system element; an edge (further referred to as link) joining two nodes represents the relationship between two system elements. Arrow direction represents the influence from one element to another. The resultant graph is called a directed or *digraph*.

The equivalent DSM that represents the same directed links is depicted in Fig. 3.1b. For example, the directed edge from element *b* to element *e* is marked by 1 in column *b* row *e*. This DSM is an initial ordering used for DSM-based algorithms discussed in the following sections.

A. Karniel and Y. Reich, *Managing the Dynamics of New Product Development Processes*, DOI: 10.1007/978-0-85729-570-5_3,

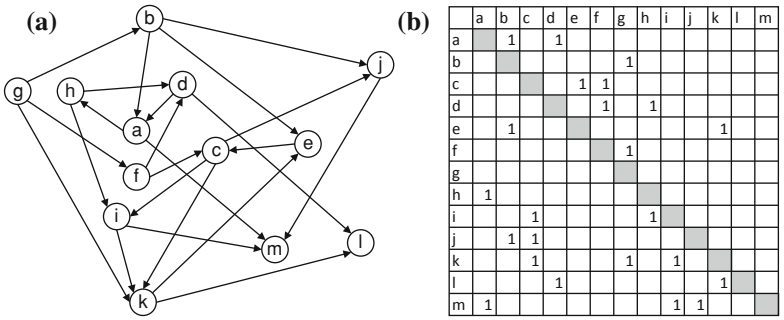


Fig. 3.1 From system graph to DSM **a** System graph **b** DSM

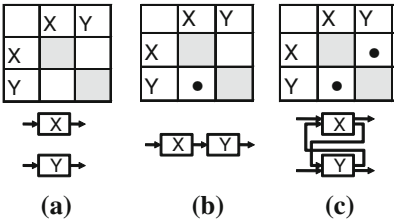


Fig. 3.2 Activity relations types in DSM, and corresponding process schemes (Adapted from Eppinger et al. 1994)

3.1 DSM Definition

DSM is a square matrix, which uses off-diagonal entries to signify the dependency of one element on another. In modeling design processes (Activity-based or Parameter-based), the lower diagonal (subdiagonal) portion represents a precedent activity relationship (downstream activities); i.e., a marking in cell $\{j, k\}$ (row j, column $k, j > k$) indicates that activity k should follow activity j. For each activity, its row shows its inputs and its column shows its outputs. Process representation by the subdiagonal is similar to PERT/CPM techniques. However, an entry in the upper diagonal portion of the DSM matrix denotes an iterative process (a link to an upstream activity). The outcome of a particular activity can directly affect a previously performed activity, which should be performed again, i.e., rework.[1]

Serial activities (synonyms: sequential, dependent, precedence) are linked by a forward link, and indicate a serial process, Fig. 3.2a. Parallel activities (synonyms: concurrent, independent) have no relation links, Fig. 3.2b. Coupled activities (synonyms: interdependent, mutually dependent) have forward and feedback links, Fig. 3.2c. The initial DSM in Fig. 3.1b was reordered to have minimum feedback

[1] Few articles use the opposite convention; upper diagonal expresses forward links (e.g., Browning 2001; Lévárdy et al. 2004). When graphically adding the activity duration, such representation resembles GANTT. The common approach (subdiagonal indicates forward links) is used in this research.

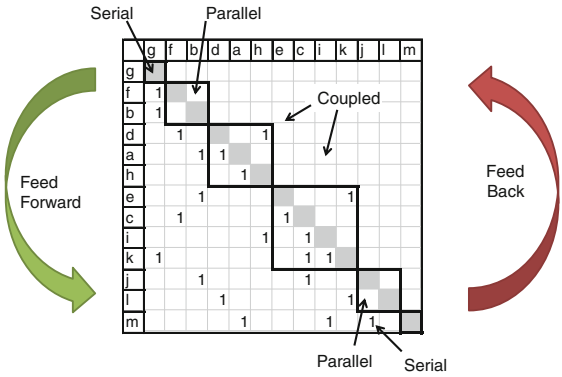


Fig. 3.3 Activity relations types in an ordered DSM

links. The activity relation types are demonstrated in the resultant DSM, Fig. 3.3. A similar example is presented in (Yassine and Braha 2003).

Activities (graph elements) *f* and *b* are parallel, and both serial to *g*. Activities (*d,a,h*) and (*e,c,i,k*) are coupled. Note that *j* and *l* are in parallel, and also *l* and *m* could be in parallel, but *m* is serial to *j*.

Additional relation type, ***overlap*** activities, can be considered as a parallel relation (Yassine and Braha 2003; Whitfield et al. 2005). Cho and Eppinger (2001, 2005) modeled two overlapping activities as parallel activities with additional lead-time, and a constraint preventing the latter activity from completing before the former activity. Nevertheless, it was indicated that this model does not scale to multiple activities having multiple iteration paths. Maheswari and Varghese (2005) considered communication time between two activities and defined overlapping as the time when a former activity can release information to the latter activity prior to its completion, or the latter activity needs the information at a certain point after its beginning.

Coupled activities form an activity loop (cycle), which may have many forms in the DSM representation. Few examples of activity loops containing three activities are depicted in Fig. 3.4. An activity loop may have an internal coupling loop, Fig. 3.4d and e. The activities order ascribes different interpretations of the dependency links. The processes described by Fig. 3.4b and c have the same set of links, but the link from *Z* to *Y* has a different interpretation (feedback in (b), and forward in (c)); therefore it represents different types of process logic. Concurrent process interpretations are depicted in Fig. 3.4f–j, as detailed in the following sections.

3.2 DSM Type Classifications

DSM types can be classified according to their marking type and according to their utilization category. DSM provides the means to identify serial, parallel, and iterative design flows; and models iterations and multidirectional information flows (Eppinger et al. 1994).

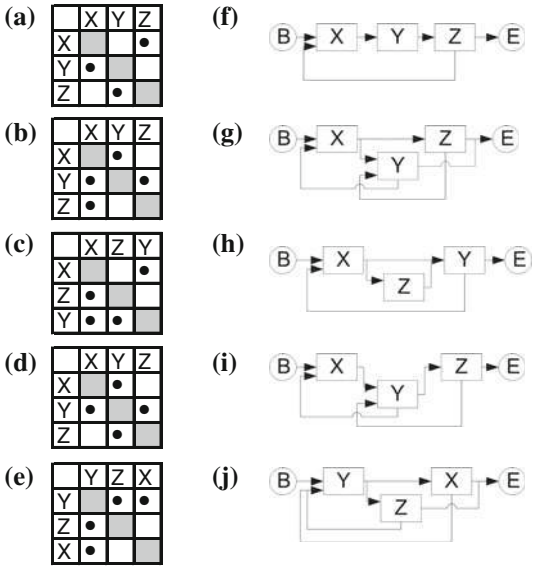


Fig. 3.4 Coupled activities form activity loops

All marking types are used in DSM based algorithms (see next section), and at the various categories. Steward (1981a) used tick-marks ('*x*') to represent precedence relations, later, referred to as *Binary DSM* (Eppinger et al. 1994). These marks concern the existence of a link, as in the above examples.

The DSM marking was extended to *Numeric DSM*, i.e., having numbers (typically integers) instead of ticks. The numbers can represent a measure of the degree of relation or its importance in ranking. Level numbers were proposed for a manual *Tearing* algorithm by (Steward 1981b), and used in a similar manner in (Choo et al. 2004); numeric values were used to indicate the number of iterations in (Abdelsalam and Bao 2006). Dependency strength or *Importance* ranking of dependency degree can be used for sequencing optimization algorithms. Further refinement to *Probability DSM* was presented in (Smith and Eppinger 1997a). In this case, the numbers are normalized (in the range [0,1]). They can represent the probability of performing activities (Browning and Eppinger 2002; Sered and Reich 2006); or risk assessment (product of normalized likelihood and impact factors) (Eckert et al. 2006).

The traditional classification of DSM studies lists four main categories (Browning 2001): Component-based or Architecture DSM; People-based (Team-based) or Organization DSM; Activity-based or Schedule DSM; and Parameter-based DSM. The first two are considered static, representing existing elements. The latter two are defined as time-based, as their ordering implies flow. A different classification (Fig. 3.5) is based on the type of relations between the the DSM items, whether they are non-directional or directional. This classification addresses additional DSM types involving more product or project aspects than the previous four such as functions, resources, and requirements DSMs. More importantly,

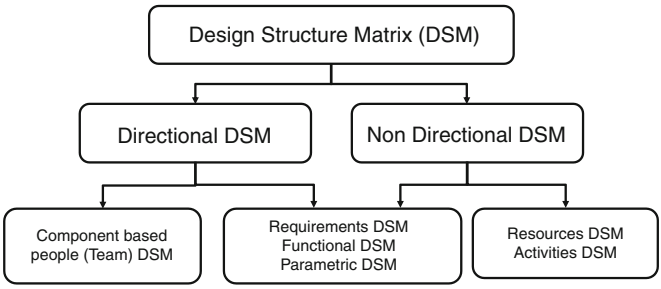


Fig. 3.5 DSM categories

some of the product information types are represented by both directional and non-directional influences (Karniel et al. 2005). This classification is instrumental in defining the kind of algorithms that are suitable for DSM-based planning as presented in the next section.

Eppinger and Salminen (2001) presented the interactions between different aspects (people, activities, and components) using additional (non-square) linkage matrices. For example, when used to model design processes, one additional matrix presents the dependencies between multiple components and multiple process activities.

The Multiple Domain Matrix (MDM) is an extension of the basic DSM structure (Maurer 2007). A MDM includes several DSMs (ordered as block-diagonal matrices) that represent the relations between elements of the same domain; and corresponding Domain Mapping Matrices (DMM) that represent relations between elements of different domains. The four complexity domains addressed are market, product complexity (diversification and modularization), process, and organization.

3.3 DSM-Based Algorithms

There are three main types of DSM-related planning algorithms: *Partitioning*, *Clustering* and *Sequencing* (includes Tearing). Partitioning is done by reordering the DSM rows and columns such that the new DSM arrangement does not contain feedback marks (iterative behavior), or those marks are moved as close as possible to the diagonal. Such reorganization yields decomposition of the process activities into clusters of interdependent activities. Partitioning is the commonly used algorithm for reordering DSM activities and achieving minimum iterations. Minimum iteration marks are expected to yield optimal processes (Steward 1981a; Kusiak et al. 1995; Smith and Eppinger 1997b). Partitioning results are in a lower part of the matrix, or block-diagonal matrix in the case of coupled activities.

Partitioning methods include the Path Searching method (Sargent and Westerberg 1964; Steward 1965; Gebala and Eppinger 1991); the Reachability Matrix method (Warfield 1973); the Triangulation algorithm (Kusiak et al. 1994); and the

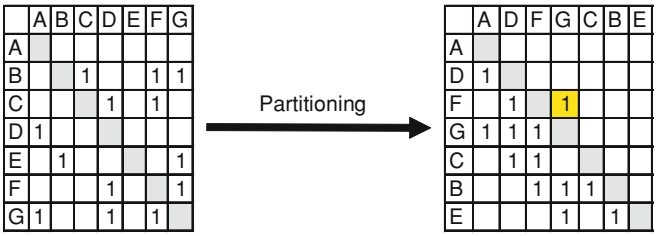
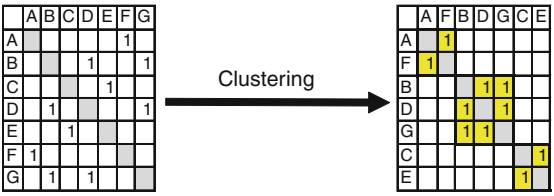


Fig. 3.6 Partitioning

Fig. 3.7 Clustering



powers of the Adjacency Matrix method (Norman 1965; Ledet and Himmelblau 1970, in Abdelsalam and Bao 2006). An example of partitioning result is depicted in Fig. 3.6.

Ideal partitioning with no feedback marks is unlikely to exist (Yassine et al. 2000). Therefore, coupled activities that form an activity loop require additional reordering algorithms. Tearing (Steward 1965; Steward 1981b) is a process of removing feedback marks (or assignment of lower priority). Tearing of a Component-based DSM may imply modularization or standardization of components (Sered and Reich 2006). In a parametric-based DSM, tearing may imply confidence in the initial estimation of the parameter value (Lévárdy et al. 2004). The tearing process is typically done manually based on additional knowledge required to establish the meaning of removing the feedback marks. A process-based tearing is done in (Choo et al. 2004) to the feedback links with low-significant values in order to reduce coupling in the process plan.

Clustering algorithms are typically used for identifying groups of highly connected components (clusters) in the context of Component-based DSM, or Team-based DSM (Browning 2001). In Activity-based DSM context, several clustering algorithms were presented to improve work teams formation (Fernandez 1998; Whitfield et al. 2002; Sharman and Yassine 2007). Clustering was used in (Danilovic and Browning 2007), for analyzing cross-domain relations (e.g., product components and process activities). A simple clustering case is depicted in Fig. 3.7.

Sequencing (by optimization) algorithms can be defined as algorithms that are used for both minimizing feedbacks (Partitioning), and ordering the coupled activities within a loop (in a manner similar to Tearing), trying to have the feedback links close to the diagonal.

In Fig. 3.8, all the activities are linked with forward and feedback links. A partitioning algorithm would not change the order. Using (manual) tearing we can

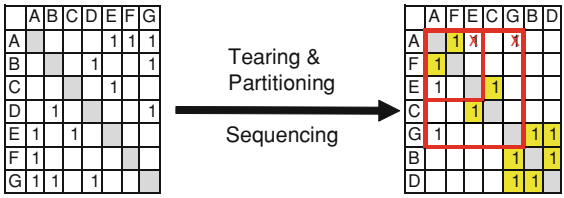


Fig. 3.8 Sequencing

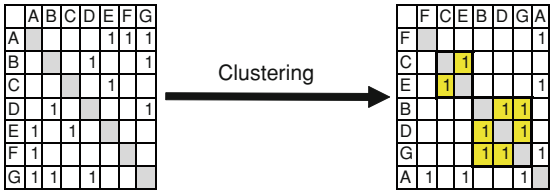


Fig. 3.9 Clustering

remove the links from activity *C* to activity *E*, and the link from *G* to *A*; then use partitioning and get a block-diagonal DSM. A sequencing algorithm may have an equivalent result. Subclusters are defined according to the optimization objective function. Optimization methods are further discussed in Sect. 3.3.2. An optimization algorithm (used in this example) that combines sequencing and clustering (Karniel et al. 2005) is detailed in Sect. 4.2.

For the same initial DSM as in the example above, a clustering algorithm using team-based clustering (e.g., Yassine and Braha 2003), or component-based clustering (e.g., Whitfield et al. 2002) reorders the DSM as in Fig. 3.9. Activity *A* is interpreted as an integration team activity or a binding component.

A comprehensive survey of matrix-based and graph-based algorithms used for analysis of relations between elements is detailed in (Maurer 2007). The survey presents examples, definitions, and the potential use of the algorithms for system structure analysis.

3.3.1 DSM Reordering: Partitioning Procedure

The reordering of the DSM to a block-diagonal matrix (Partitioning/Sequencing) can be done using the following procedure (Gebala and Eppinger 1991). Only off-diagonal links are considered. A binary DSM is assumed; if the DSM has other values, any non-zero value is replaced by '1'. The following procedure is explicitly elaborated since its details are used for the formal definition of DSM properties in Sect. 9.4. Step 3 is an improvement of the procedure proposed in (Steward 1965).

Partitioning procedure:

1. Go through the elements list.

2. If an element without input from other elements, (empty row) is identified, then: Place the element (row, column) in top of the DSM; remove the element from the elements list; and disregard this element in the matrix (row, column). Go to 1.
3. If an element without output to other elements, (empty column) is identified, then: Place it at the bottom of the DSM; remove from the elements list; and disregard this element in the matrix. Go to 1.
4. If no elements remain, go to 6 (the matrix is partitioned; otherwise, the remaining elements contain circuits.)
5. Determine circuit (e.g., using Path search procedure); and collapse the circuit to one element (e.g., using Collapse procedure). Go to 1.
6. Expand the collapsed circuits, if there were such.

Path search procedure:

1. Starting from an element, follow the dependencies until the element is reached again. Stop.
2. The circuit is the list of elements from the first appearance to the second (not included) appearance of the same element.

Note: minimal path is not guaranteed.

Collapse procedure:

1. Sum all columns and all the rows of the elements within the circuit to a representative element (any element in the circuit).
2. Delete all rows and columns of the other circuit elements.
3. Turn to 1 each number that is greater than 1.
4. Replace the number on the diagonal with zero.

3.3.2 Optimization Methods in DSM-Based Algorithms

Meier et al. (2007) presented a detailed comparison of objective functions for optimization algorithms performing Partitioning and Sequencing. The purpose of such objective functions is minimizing feedback marks; seeking feedback marks that are close to the DSM diagonal as possible, yielding short iteration cycles; pushing marks to the lower-left corner of the DSM to increase concurrency; minimizing iteration; or maximizing concurrency. The optimization methods utilized in that study were based on Genetic Algorithms (GA).

Genetic algorithms maintain large population of candidate solutions. The population is evolved by random generation of new candidates and selection of candidates from the old and new candidates. The selection gives priority to those that best fit a fitness criterion.

Generating a new candidate is done using *Crossover* (choosing two parents and taking some characteristic from one and the rest from the other) and *Mutation* (random change of a characteristic). A scheme of encoding, decoding, and evaluating candidates is required (Davis 1991).

An optimization algorithm that combines Sequencing (i.e., Tearing and Partitioning) and Clustering was developed by Karniel et al. (2005). This algorithm was used in the context of reordering computation tasks within a constrained reverse engineering process. The computation tasks required solving geometry constraints between surfaces. Tasks rearrangement was utilized for improving the overall computation. The optimization algorithm was used to find an optimal (or near global optima, in case of multiple solution) reordering of the tasks. The optimization relies on Simulated Annealing (SA) search method. In SA, an artificial "temperature" is introduced for determining the probability of accepting a non-minimum result; thus, allowing the solution to be pulled out from local minima. The "temperature" is gradually lowered and, at the end, the system is hopefully inside the basin of attraction of the global minimum (or in one of the global minima, if more than one exists). Implementation involves the following ingredients (Kirkpatrick et al. 1983):

1. Concise description of the system configuration
2. Random generation of rearrangement "moves"
3. Quantitative objective function
4. Annealing schedule of the temperatures and length of time at each step for the system to evolve

The description requirement is equivalent to the encoding and decoding requirement of GA, the moves generation is equivalent to candidate generation, and objective function to the fitness function. The main difference is that SA tries to avoid local minima by accepting non-minimum results, while GA assumes that the large population and operators would avoid local minima. However, the latter assumption is criticized as the selection process after several generations is biased due to the influence of crossover, and its computation behavior is impossible to predict. Changes in the parameters of mutation rate and crossover rate may yield different behavior of the GA process.

In Classical Simulated Annealing, the global minimum is achieved if and only if the temperature is decreased logarithmically (Lester 1996). However, Lester describes a variant called Adaptive Simulated Annealing (ASA) where the annealing schedules for the temperatures Tk decrease exponentially in annealing-time k, for particular problems. The algorithm developed in this research, used the ASA version (further discussed in Sect. 4.2).

It is important to note that the required calculation for planning of a design process is complex. Braha and Maimon (1998) modeled the design process as an automaton, and proved that computational complexity of the planning problem is NP-Hard. Meier et al. (2007) proved that DSM sequencing is NP-Hard by formulating it as a Quadratic Assignment Problem (QAP), which is a known combinatorial optimization problem. The problem is further entangled when considering resource scheduling.

3.4 Using DSM for Planning: The Data Collection Process

DSM data collection can be done through questionnaires and interviews of the design process participants (Eppinger et al. 1994; Whitfield et al. 2002; Browning and Eppinger 2002). However, it was found that different participants may have different perspectives regarding the design activity interactions and dependencies (also differences whether information is provided or not); therefore, group discussions seem to be better procedure for DSM data collection (Danilovic 1999; Danilovic and Browning 2007).

Yet, data acquisition through questionnaires and interviews is resource intensive. A report of documenting relations of 140 elements in a matrix form indicated duration of several months (Alexander 1964; in Maurer 2007).

Binary dependency data is relatively easy to define (or assumed to be so despite opinion differences between team members). Estimation of numeric values is harder, typically a scale of dependencies (high, medium, low) is used with additional no dependency value (zero). A direct estimation of probabilities was found hard to accomplish for developing useful models (Ford and Sterman 1998; Yassine 2007). Using the dependency scale, the estimations are converted to ordinal influence strength (e.g., {1, 2, 3} in (Pimmler and Eppinger 1994) or {3, 2, 1} in (Danilovic 1999)).

Direct conversion of influence estimations to probability figures was done in (Smith and Eppinger 1997b) using {0.5, 0.25, 0.05}, respectively; the figures {0.15, 0.1, 0.05} were used in (Huberman and Wilkinson 2005); and {0.3, 0.2, 0.1} in (Yassine et al. 2003). In all these cases, there was no indication of the probability estimation source.

Sered and Reich (2006) used a staged process. A ranking scheme was used to estimate the influence of specification changes. The ranking values {9, 6, 3, 1} are representing {high, medium–high, medium–low, and negligible} influence, respectively. Zero indicated no influence. A normalization method (Rogers and Bloebaum 1994) was used to normalize the probabilities to the range [0.05, 0.95]. Using Markov-chain simulation enforces the sum of probabilities in each row to be less than one. Additional linear scaling was performed to ensure that the maximal sum of the rows is less than 1.

Yassine (2007) used the following procedure for assessing rework probabilities. First, interdependencies were marked. Then, for each mark the Information Variability (likelihood to changes, ranking 1 to 3 (low to high)), and the Task Sensitivity (to information change) were defined. Finally, Information Variability and the Task Sensitivity were multiplied, thus getting the ranking values {1, 2, 4, 6, and 9}. A review of design processes, and a calibration process identified that the maximal feedback probability was about 52%, assigned to rank {9}. Additional results indicated that a linear scaling of the ranks was the most appropriate.

PLM tools have the required functionality to capture dependency data. Geometrical dependencies can be automatically extracted from CAD data using the CAD relations. However, this might lead to very large DSM that is too large to

cope with or to understand. Additionally, other relation types would not be captured.

A manual option is to use the relations management capabilities in PLM for adding relation values that can be further used for creating and updating the DSM. Such approach is more appealing since it can address all relation types during the design process. Yet, engineers should be willing to make the additional effort of data entry.

In the current work, we assumed that the required data is available. In most DSM-related articles the data was collected by the researcher. Tools for data entry by users were reported in Wynn (2007).

The conversion of influence or dependency values into probability values (calculated probability figures) has an underlying assumption that greater dependency is reflected in greater chance of design activity repetition; hence, component redesign. The linearized conversion suggested by Yassine (2007) is therefore an appealing concept used in the DnPDP framework (Karniel and Reich 2007) as detailed in Sect. 4.1.

References

Abdelsalam HME, Bao HP (2006) A simulation-based optimization framework for product development cycle time reduction. IEEE Trans Eng Manag 53(1):69–85

Alexander C (1964) Notes on the Synthesis of Form. Harvard University Press, Cambridge

Braha D, Maimon O (1998) A mathematical theory of design: foundations, algorithms and applications. Kluwer, Boston

Browning TR (2001) Applying the design structure matrix system to decomposition and integration problems: A review and new directions. IEEE Trans Eng Manag 48:292–306

Browning TR, Eppinger SD (2002) Modeling impacts of process architecture on cost and schedule risk in product development. IEEE Trans Eng Manag 49(4):428–442

Browning TR, Fricke E, Negele H (2006) Key concepts in modeling product development processes. Syst Eng 9(2):104–128

Cho SH, Eppinger SD (2001) Product Development Process Modeling Using Advanced Simulation. ASME Conf on Design Theory and Methodology (DECT 2001/DTM), Pittsburgh, PA, September

Cho SH, Eppinger SD (2005) A simulation-based process model for managing complex design projects. IEEE Trans Eng Manag 52(3):316–328

Choo HJ, Hammond J, Tommelein ID, Austin SA, Ballard G (2004) DePlan: a tool for integrated design management. Autom Constr 13:313–326

Clarkson PJ, Hamilton JR (2000) Signposting, A parameter-driven task-based model of the design process. Res Eng Design 12(1):18–38

Danilovic M (1999) Loop: leadership and organization of integration in product development. Dissertation. Linköpings Universitet

Danilovic M, Browning TR (2007) Managing complex product development projects with design structure matrices and domain mapping matrices. Int J Project Manag 25(3):300–314

Davis L (1991) Handbook of genetic algorithms. Van Nostrand Reinhold, New York

Dong Q, Whitney DE (2001) Designing a requirement driven product development process. Proc of ASME Design Eng Technical Conf and Comput and Inf in Eng Conf, DTM-21682, 1–10

Eckert CM, Keller R, Earl C, Clarkson PJ (2006) Supporting change processes in design: Complexity, prediction and reliability. Reliability Eng Saf Syst 91(12):1521–1534

Eppinger SD, Salminen V (2001) Patterns of product development interactions. Int Conf on Eng Design, ICED 01, Glasgow, 21–23 August

Eppinger SD, Whitney DE, Smith R, Gebala D (1994) A Model-based method for organizing tasks in product development. Res Eng Design 6(1):1–13

Eppinger SD, Nukala MV, Whitney DE (1997) Generalized models of design iterations using signal flow graph. Res Eng Design 9:112–123

Fernandez C (1998) Integration analysis of product architecture to support effective team co-location. SM Thesis, Massachusetts Institute of Technology

Flanagan TL, Eckert CM, Keller R, Clarkson PJ (2006) Bridging the gaps between project plans and reality: The role of overview. Proc of Tools and Methods of Competitive Eng (TMCE 2006), 1:105-116, Ljubljana, Slovenia

Ford DN, Sterman JD (1998) Expert knowledge elicitation to improve formal and mental models. Syst Dynamics Review 14(4):309–340

Gebala DA, Eppinger SD (1991) Methods for analyzing design procedures. ASME 3rd Int Conf on Design Theory and Methodology, 227–233

Huberman BA, Wilkinson DM (2005) Performance variability and project dynamics. Comput Math Organiz Theory 11:307–332

Karniel A, Reich Y (2007) Managing dynamic new product development processes. Proc of the 17th Annual Int Symposium of The Int Council on Sys Eng INCOSE'07, San Diego, California, June

Karniel A, Belsky Y, Reich Y (2005) Decomposing the problem of constrained surface fitting in reverse engineering. Comput Aided Des 37:399–417

Keller R, Flanagan TL, Eckert CM, Clarkson PJ (2006) Two sides of the story: Visualising products and processes in engineering design. Proc of the 10th Int Conf on Inf Visualistion (IV 2006), IEEE Computer Society, London

Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220(4598):671–680

Kusiak A, Larson N, Wang J (1994) Reengineering of design and manufacturing processes. Comput Ind Eng 26(3):521–536

Kusiak A, Wang JR, He DW, Feng CH (1995) A structured approach for Analysis of Design Processes. IEEE Trans Compon Manuf Technol—Part A 18(3):664–673

Ledet WP, Himmelblau DM (1970) Decomposition procedures for the solving of large scale systems. Adv Chem Eng 8:185–254

Lester I (1996) Adaptive simulated annealing (ASA): lessons learned. J Control Cybern 25(1):35–54

Lévárdy V, Hoppe M, Browning TR (2004) Adaptive test process—An integrated modeling approach for test and design activities. In: The Product Development Process, Proc of DETC'04 ASME 2004 Design Eng Technical Conf and Comput and Inf in Eng Conf, Salt Lake City, Utah, September

Maheswari JU, Varghese K (2005) Project Scheduling using dependency structure matrix. Int J Project Manag 23:223–230

Maurer M (2007) Structural Awareness in complex product design. Dissertation, Technischen Universität München, Germany

Meier C, Yassine A, Browning T (2007) Design process sequencing with competent genetic algorithms. J Mech Des 129(6):566–585

Norman RL (1965) A matrix method for locating of cycles of a directed graph. AIChE J 11: 450–452

Pimmler TU, Eppinger SD (1994) Integration analysis of product decompositions. ASME Conf on Design Theory and Methodology, Minneapolis, MN, 343-351, September

Rogers JL, Bloebaum CL (1994) Ordering design tasks based on coupling strengths. AIAA, paper no. 94-4326

Sargent RWH, Westerberg AW (1964) Speed-up in chemical engineering design. Trans Instn Chem Engrs 42:T190–T197

Sered Y, Reich Y (2006) Standardization and modularization driven by minimizing overall process effort. Comput Aided Des 38(5):405–416

Sharman D, Yassine A (2007) Architectural valuation using the dependency structure matrix and real options. Concurrent Eng 15(2):157–173

Smith RP, Eppinger SD (1997a) Identifying controlling features of engineering design iteration. Manag Sci 43(3):276–293

Smith RP, Eppinger SD (1997b) A predictive model of sequential iteration in engineering design. Manag Sci 43(8):1104–1120

Steward DV (1965) Partitioning and Tearing Systems of Equations, Journal of the Society for Industrial and Applied Mathematics: Series B. Numer Anal 2(2):345–365

Steward DV (1981a) Systems Analysis and Management: Structure Strategy and Design. Petrocelli Books, NJ

Steward DV (1981b) The design structure system: A method for managing the design of complex systems. IEEE Trans Eng Manage 28:71–74

Warfield JN (1973) Binary matrices in system modeling. IEEE Trans Syst Man Cybern 3: 441–449

Whitfield RI, Smith JS, Duffy AHB (2002) Identifying component modules. Artificial Intelligence in Design, Cambridge, UK

Whitfield RI, Duffy AHB, Gartzia-Etxabe LK (2005) Identifying and evaluating parallel design activities using the design structure matrix. Int Conf on Eng Design, ICED05, Melbourne, August

Wynn DC (2007) Model-based approaches to support process improvement in complex product development. Dissertation, University of Cambridge

Yassine A (2007) Investigating product development process reliability and robustness using simulation. J Eng Des 18(6):545–561

Yassine A, Braha D (2003) Complex concurrent engineering and the design structure matrix method. Concurrent Eng Res Appl 11(3):165–176

Yassine A, Joglekar N, Braha D, Eppinger SD, Whitney D (2003) Information hiding in product development: The design churn effect. Res Eng Design 14(3):131–144

Yassine A, Whitney D, Lavine J, Zambito T (2000) Do-it-right-first-time (DRFT) approach to design structure matrix restructuring. In: Proceedings of the 12th international conference on design theory and methodology, Baltimore, Maryland, USA, September

Chapter 4
DSM Enhancements

4.1 DSM Data

The following section presents the proposed process for gathering DSM data (Karniel and Reich 2007b). As in other DSM-based process planning methods, the product data is deriving the process plan. A key issue in the current work is the utilization of data changes throughout the design process. The detailing level is not pre-defined and is changing and adapting to the applicable product knowledge. Collecting the data throughout the design process requires a tool. An existing applicable tool could be a PLM product, where other product relation links (e.g., father-son) are managed. A specific tool for collecting the data may also apply but would be less effective. The process of collecting the DSM data as part of the overall DnPDP is fully described in Sect. 8.5. The following example reflects the initial knowledge at the conceptual design stage.

The following four substages were identified and enhanced in the research.

1. Identify product related design activities (at the required detail level).

A simplified model is the assignment of one design activity to each product component. This simplification is useful for automating the transferring process, and is used in the following examples. However, complex relations may apply between the product structure and the assigned activities (Eppinger and Salminen 2001). Danilovic and Browning (2007) used Domain Mapping Matrix (DMM) for comparing and analyzing the relations in different domains. Maurer (2007) presented Multiple Domain Matrix (MDM) that combines DSMs and DMMs into one multi layered matrix, which has DSM like properties (i.e., same rows and columns). Since the definition of product components by the designer is not limited to physical entities, this simplification is not restrictive. The translation of the more comprehensive relations could be automated as well by using the MDM structure.

A. Karniel and Y. Reich, *Managing the Dynamics of New Product Development Processes*, DOI: 10.1007/978-0-85729-570-5_4,

Table 4.1 Dependency influence ranking

Influence value per parameter	Value
No Influence	0
Limited influence (potentially some changes)	1
Medium influence (some change are required)	3
Major influence (re-design)	9

2. Identify the dependencies and assign their value.

In the context of the current research, dependencies between activities are defined as relations between the activities that influence the ordering of the activities in planning the process. The dependencies might be induced, e.g., dependencies between product components through design parameters (assuming a design activity per component, as in the following example), or direct precedence relations between activities (e.g., the separation of design and testing activities, and assignment of a sequential link between the two). The latter case, assignment of direct logic relation can be done at this stage of product data gathering or at a later stage.

In the first case, for each design parameter, the influence value should be estimated and the influence direction assigned. The optimization method developed in this research allows for assigning non-directed dependency links, i.e., links that keep forward direction in any DSM reordering. Such link indicates influence without a specific direction. Current DSM methods use only directed links. The ranking scheme suggested is using the concepts of (low, medium, and high) influence according to the ordinal scale in Table 4.1. Though the scale is actually ordinal, its values are set as if it was a ratio scale in order to allow summation of the influence values over parametric links between activities. Therefore, the ordinal terms (low, medium, and high) are interpreted as the numbers 1, 3, and 9.[1]

The resulting ranking values are aligned with previous studies (Sered and Reich 2006; Yassine 2007).

Using the above values, the summation of influence values of a change in component *A* which causes a change in *B* can be interpreted as follows: nine design parameters with low influence (of a design change in *B* due to design change in *A*) have an equivalent effect to one design parameter whose influence is complete re-design. The influence values in (Yassine 2007) were calculated as multiplications of the likelihood of change values range {1–3} with sensitivity to change {1–3}. Sered and Reich (2006) used the values {1, 3, 6, 9}. The additional ranking values, e.g., {6} in (Sered and Reich 2006), and {2, 4, 6} in (Yassine 2007), can be interpreted as summation of several influence entities.

An illustration of a system[2] is depicted in Fig. 4.1. It is part of a laser direct imaging plate/image-setter for the prepress market industry. The following

[1] Similar values are used in other product development related tools such as Quality Function Deployment (QFD).

[2] The same system is used for the example in Sect. 12.2, with extended data.

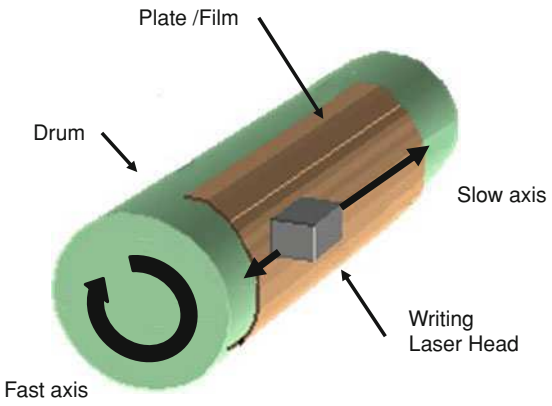


Fig. 4.1 Product illustration

	W	X	Y	Z
W (frame)		Weight 3	Weight 1 Size 1 Location 1	
X (drum)	Grip 3 Pressure 3			
Y (laser)				Processing Time 3
Z (control)		Acceleration 3 Resolution 1		

Fig. 4.2 Product knowledge

Table 4.2 Influence values of dependencies between parameter links

From	To	Parameter description	Influence	Value
X	W	Weight	Med	3
X	Z	Acceleration	Med	3
X	Z	Rotation resolution	Low	1
Y	W	Weight	Low	1
Y	W	Size	Low	1
Y	W	Location	Low	1
W	X	Gripping force	Med	3
W	X	Beam pressure	Med	3
Z	Y	Processing time	Med	3

components were identified: X (drum), Y (laser writing head), W (frame), and Z (control system). The influence values for the example are described in Table 4.2.

The product knowledge is represented in Fig. 4.2. The matrix rows and columns are the product components (sub systems). In each cell, there is a list of parameters and figures. The figures indicate the influence of a change of one component (column) on another component (row) due to the specified parameter, according to Table 2.1. The influence figures are directed, i.e., the influence is asymmetric.

Fig. 4.3 DSM scaling to probabilities **a** Numeric DSM, **b** Probability DSM

	W	X	Y	Z
W		3	3	
X	6			
Y				3
Z		4		

(a)

	W	X	Y	Z
W		0.25	0.25	
X	0.5			
Y				0.25
Z		0.33		

(b)

3. Summing the influence values (in each cell) to create a numeric DSM.

Summing was used in (Sered and Reich 2006), and it suits the linear conversion to probability values in step (4). Summing the figures in each cell for the above example yields a Numeric DSM shown in Fig. 4.3a. Even when starting with the proposed limited set of values, the summation process yields a larger variety of influence values, as the number of parameters considered is unlimited.

4. Assigning scaled probability values for creating a probability DSM.

Assignment of probabilities is required for simulation purposes. Direct assessment of probabilities was indicated to be problematic (Browning et al. 2006). In some DSM-based simulation articles, the probabilities (or probabilities and relative impact) are assumed as known by the users (Cho and Eppinger 2001; Browning and Eppinger 2002; Eckert et al. 2006), or some values are assumed to represent the influence value (Huberman and Wilkinson 2005; Smith and Eppinger 1997b).

The justification of setting probability values that correspond to influence values is logically appealing but is not fully determined in the literature. Some estimation approaches used the influence values to calculate the probability. Sered and Reich (2006) used a scaling process suggested in (Rogers and Bloebaum 1994). Yassine (2007) used a direct scaling based on maximal probability assigned to maximal influence value.

The logic of scaling also varies. Sered and Reich (2006) scaled the values in a way such that the maximal sum of values in a column is 0.95, i.e., the conversion does not address a specific maximal value. Yassine (2007) has found that probability $p = 0.52$ has suited the actual project data for maximal influence value used {9}.

In the current work, we choose to follow the linear scaling, and use the value $p = 0.5$ as corresponding to maximal influence value. The choice of this maximal value is arbitrary, since this figure is the only one that has been validated for some real case. Further research is required to validate actual probability figures and their relation to the assessment of influence (impact or risk). It can be anticipated that different engineering fields and different product types will tend to have different actual iteration probabilities.

The use of linear conversion can be justified. It keeps the DSM values with the same relations; thus, the next step of DSM reordering could either be done using the numeric values or the probability values with no change in ordering results

(when using the algorithm in (Karniel et al. 2005)). Therefore, the consequence of changing the numeric values can be easily tracked (while using the transformation by (Rogers and Bloebaum 1994; Sered and Reich 2006) is more complex to track). Furthermore, the linear scaling of the probability values was demonstrated to appropriately represent measured iteration probability results (Yassine 2007).

Using the previous example, the DSM is linearly converted to Probability DSM; and the maximal value 6 was converted to maximal probability 0.50, Fig. 4.3b.

4.2 DSM Optimization

The optimization problem defined in (Karniel et al. 2005) utilized a modified DSM that included directed links (e.g., design activity B gets information from design activity A) and undirected links (i.e., information passed between activities but it does not matter which activity is done first). This modification implied that after reordering, directed links might become feedback links (if they are part of an activity loop), while undirected links can "change direction" and always be forward links. Recalling the definition in Sect. 3.1, link l_{jk} is a directed forward link if $k > j$. If the link is undirected then the link can become l_{kj}. In an activity loop, if one link can change its direction, then the loop "breaks" and the activities of that loop could be ordered with no feedback links.

The sequencing algorithm has the following steps:

1. Matrix partitioning for initial arrangement and finding activity loops. Partitioning is done using DSM Reachability matrix method adopted from the DSM site at MIT (DSM 2009).[3]
2. For each activity loop, checking if it includes a non-directed link. If it does, change the temporary direction (to break the loop) and go to step 1. The remaining activity loops become initial clusters or *Partition clusters* for the optimization step.
3. Use the optimization algorithm for further decomposing large initial clusters to sub-clusters (tearing); merging small cluster or activities with non-directed links; and reordering the matrix. The outcomes of this step are *Optimal clusters*.

The term cluster indicates a group of activities that are closely related, and therefore are planned to be executed in parallel as a Design block. Activity loop may indicate that the activities in the loop should be performed together, since they depend on each other. Therefore, the loops identified as results of step 2 are candidates for clustering.

[3] This algorithm, presented in Sect. 4.2 is further used as part of the formal definitions in Sect. 6.7.

The algorithm in step 3 searches for better clustering results by minimizing the cost function in (Karniel et al. 2005). The results may differ if another cost function is used.

In describing the cost function in Eq. 4.1, the more general term *matrix element* A_{ij} is used instead of the term link, and this term is hereby replacing the term link. The matrix elements indicate influence values (Numeric DSM), but since the objective cost function has linear expressions of the matrix values, the use of linear conversion from numeric values to probability values keeps the same reordering results. Thus, reordering by using influence values or probability values are equivalent.

$$\left[\sum_{c} N_c \left(F \sum_{ic\ >\ jc} A_{ic,jc} + C \sum_{ic\ <\ jc} A_{ic,jc} \right)\right] + F \sum_{id\ >\ jd} A_{id,jd} D^q_{id,jd} + C \sum_{id\ <\ jd} A_{id,jd} D^q_{id,jd} + \sum_{id} A_{id,id} D^q_{id} \tag{4.1}$$

The four expressions of the cost function are summed over the entire DSM with the following rules:

1. Expression one: If elements i and j are in the same cluster, we indicate them as ic and jc, respectively. The priority value assigned to the link $A_{ic,jc}$ is multiplied by constant factors *F*or *C* and by the cluster size N_C, where $F > 0$ is Forward constant (applicable when $A_{ij} > 0$, $i > j$); and $C > 0$ is the Closed Loop constant (applicable when $A_{ij} > 0$, $i < j$).
2. Expressions two and three: If elements i and j are in distinct clusters, we indicate them as id and jd, respectively. In addition to multiplying by constants F or C, the numeric value $A_{id,jd}$ is multiplied by $D^q_{id,jd}$, which is the penalty of a link for not being part of a cluster. The function $D_{id,jd}$ corresponds to a distance measure between elements i and element j, $D_{id,jd} = \text{abs}(i - j) + 1$. The power q is a parameter derived from the required minimal cluster size (set by the user).
3. Expression four: The partitioning algorithm (step 1) may end with elements that belong to several clusters. If an element *id* has entries in several clusters, its self-value $A_{id,id}$ is multiplied by D^q_{id}, where D_{id} is the sum of all cluster sizes between the first and last appearance of the element. $D_{id} = \sum_k N_k$, where k runs from the first cluster (to which the element id belongs) to the last cluster and N_k are the cluster sizes.

The optimization algorithm searches for a better clustering, by reordering the matrix elements and by generating different clusters (i.e., different grouping). A clustering candidate is generated using the following modification actions (adapted from the hierarchical clustering program ECOBWEB (Reich and Fenves 1992):

1. Move an element from one cluster to another;
2. Remove an element from a cluster and make it a new cluster;
3. Join two clusters;

4. Split a cluster into two clusters;
5. Change the order between 2 elements in a cluster; and
6. Change the order of clusters.

The number of clusters and their size are not predefined. The initial guess is the result of partitioning (i.e., close loop groups of activities). The selection of clusters and elements (on which an action is performed) is random using a uniform distribution.

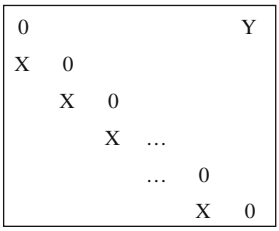
According to simulated annealing, a better result is always accepted. However, with a given probability ("temperature"), a worse result may also be accepted. The probability of accepting a worse result is lowered according to the simulated "temperature". A search step with a given probability ("temperature") is done a predefined number of times. Then, the probability is lowered for the next set of trials. An exponential temperature decrease was used following Lester (1996). Using the Adaptive Simulated Annealing (ASA) search assures convergence to a near global minimum (given enough time). Enough time, according to our experience (Karniel et al. 2005) was set to N^3, where N is the number of design activities in an activity loop.

Implementation notes:

1. The size of the problem search space is $N! \cdot 2^{N-1}$, where N is the number of activities in an activity loop. $N!$ is the number of permutations, since a different order is a different solution, and 2^{N-1} is the number of possible allocations to clusters of a specific order. Obviously, no algorithm exists for finding the global optimum for an arbitrary problem.
2. The last term in Eq. 4.1 is required in order to reduce (or eliminate) multiple entries of the same object in several different clusters (as might be the result of partitioning).
3. The self-iteration probability value does not affect the ordering (i.e., the index of the activity).
4. The power $q > 1$ is derived from a required cluster size. In this work, q was arbitrarily set to $q = \log_2 5 = 2.32$. This value emerged from the analysis of a link chain, with no internal loops (i.e., A is linked to B, which is linked to C and so forth). If $N_C < 2^q$, where N_C is the cluster size, the elements are always clustered. Otherwise, the elements might not be clustered (each being a cluster of its own) according to the other parameters, see following example.
5. The closed loop constant, C, is always set to be bigger than the forward constant, i.e., $C > F$.
6. We can choose F, C, and q values in order to control the optimization behavior.

The following example presents a closed loop with N elements, each element is linked to the next one, and the last one linked to the first. Let us assume that all link values are equal ($= X$) except for one ($0 < Y < X$). Since $F < C$, a minimum feedback links optimization (resulting in one feedback link) will order the objects such that the link with priority Y is in a "feedback position" as shown in Fig. 4.4.

Fig. 4.4 Optimal clustering



A one-cluster solution cost (i.e., a design block with all elements in the same cluster) according to Eq. 4.1 is:

$N[F(N-1)X + CY]$ (Only the first term is applicable).

The cost of a non-clustered solution (each element is a separate cluster) is:

$F(N-1)X2^q + CYN^q$ (Terms two and three, respectively).

A non-clustered solution is preferred to the one-cluster solution when

$$F(N-1)X2^q + CYN^q < N[F(N-1)X + CY]$$

By reordering, we get a non-clustered solution only when the following inequality holds, Eq. 4.2.

$$\frac{Y}{X} < \frac{F}{C} \times \frac{(N-2^q)(N-1)}{(N^q - N)} \quad \text{for } N > 2^q \tag{4.2}$$

For large N, the right hand term is approximately $F/(C \cdot N^{q-2})$. If we set $q > 2$ this function is decreasing as N grows.[4] Hence, if Y is small enough (minimum zero) the optimization will not cluster the activities to a design block but rather keep them separated (and serialized).

A one cluster solution is always preferred (for every Y) if $N \leq 2^q$, (e.g., $q = 2.32$, $N \leq 5$) since the left side of Eq. 4.2 has negative value. Therefore, small cycles (small being defined by selection of q) will always tend to cluster to a design block.

Overall, by assigning F, C, and q, we can control the size of the required cluster for a given ratio Y/X.

The usage of clustering the activities into designs block is intended to find substructure within larger cycles. Partitioning algorithms stop at the stage of identifying loops. Optimization sequencing is required to identify the subprocesses within the loop. The above optimization procedure performs clustering and sequencing (including tearing); thus, can separate the activities further into closely interconnect activity groups, which become design blocks.

Design block can be typically implemented as parallel activities with aligned start and end (Smith and Eppinger (1997a); Huberman and Wilkinson 2005; Yassine et al. 2003); yet, additional process-logic implementation options may

[4] See a detailed analysis of the function $F(q, N) = (N - 2^q) \times (N - 1)/(N^q - N)$ in Sect. 14.3, Annex C.

Table 4.3 Optimal ordering and clustering

#	Configuration	Cost function	Cost
1	w−x−y−z	$F(0.5 \times 2\text{^}q + 0.33 \times 3\text{^}q) + C \times 0.25 \times (2 \times 2\text{^}q + 3\text{^}q)$	384.60
2	[wxyz]	$4(F(0.5 + 0.33) + C(3 \times 0.25))$	201.96
3	[wxzy]	$4(F(0.5 + 0.25 + 0.33) + C(0.25 + 0.25))$	140.96

apply as further discussed in Chap. 10. Some results of applying the optimization criterion to the example in Fig. 4.3 are depicted in Table 4.3. The parameters values are: $F = 3$, $C = 64$, and $q = 2.32$.

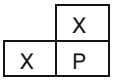
The first line is the initial ordering without clustering. The second line keeps the same order but with clustering (all activities in one DB). The third line is the case of clustering to one DB but with different internal order, which is the optimal result. As long as $q > 2$, and $F < C$, changes of parameters do not change the order of cost results (i.e., cost([wxzy]) < cost([wxyz]) < cost(w−x−y−z)) and the optimal solution in this case is always one DB.

A case of activity cycle with six activities is presented in Chap. 5. There, the optimal result is separation of the cycle into two DBs. If the optimization is used for reordering only (without clustering), the optimal results do not follow the minimum feedback rule, but try to get the feedback marks as close as possible to the diagonal (see Sect. 14.4).

While tearing can reduce planning complexity, the option of using undirected links (Karniel et al. 2005) seems to be even more powerful. For example, the case of a loop is depicted in Fig. 4.4. If any of the links is undirected (i.e., it does not matter which component is designed first), changing the link place (i.e., *Y* changing its place to be a forward link) will "break" the loop, allowing uncoupled ordering. In Karniel and Reich (2011), it was proved that a simple loop can always be reordered to have one feedback link; hence, if there is an undirected link the loop can be "broken" (see Sect. 9.4).

4.3 DSM Self-Iterations

Self-iterations are not discussed in the DSM literature; however, their definition is important for simulation purposes. Self-iteration of a design activity could always be separated to design, check, and decide activities. Such distinction might be justified if different resources perform the distinct activities, but it is a redundant overhead if done by a single resource. Self-iterations represent practical processes, e.g., a design of a part that did not conform to some work standard and should be re-done. Utilizing self-iterations in simulation enhances the simulation options, but complicates the simulation execution as iteration occurrences need to be addresses



(a)

	WX	YZ
WX	0.625	0.25
YZ	0.33	0.25

(b)

Fig. 4.5 Representing self-iterations in DSM **a** Single activity self-iteration, **b** Self-iteration of Merged Design-blocks

(see Sect. 10.7). The representation of self-iteration in DSM is straight forward, using the typically unused diagonal values, as depicted in Fig. 4.5a.

Additional source of using self-iteration is derived from the definition of DBs. The interrelation probabilities assigned to activities within a DB are used for calculating the probability of the whole DB, using the following calculation model[5]:

$$Pd = 1 - \prod(1 - Pi) \tag{4.3}$$

where Pd is the combined merged probability of a DB, and Pi are the probability values of the cells being merged. The probabilities of cells, which are on forward direction (sub diagonal) to design activities within the DB are merged to the probability of forward link to the merged DB. The probabilities within the DB (including diagonal self-iteration probabilities) are all considered as feedback links and are merged to the self-iteration merged probability of the DB. In Fig. 4.5b, the calculations were applied to the probability DSM in Fig. 4.3, where W and X activities are merged to a single DB, and activities Y and Z are merged to a second DB. The probabilities 0.5 and 0.25 are merged to the self-probability 0.625. The other probabilities are merged in the same manner (but in this simple case keep the same value).

The probability calculation model is based on the combined probability of independent events (OR option). For the case of two probabilities Pa and Pb, the resulting probability of Eq. 4.3 reduces to $Pd = Pa + Pb - Pa \times Pb$. The interpretation of links within the DB as feedback links indicates that any of the links (even the forward links) may actually cause iteration (as the activities are parallel).

Equation 4.3 is computationally appealing since it keeps the commutative, associative, and distributive properties. Yet, this calculation needs to be validated by measurements. Alternatively, only feedback probabilities and self-probabilities could have been considered for calculations of merged feedback probability and self-probability, and only forward link for merger of forward links probability. In such case, cummutativity, associativity, and distributivity are not kept for the calculations of the merged forward links.

Additionally, the calculation should refer to the cases where iteration of a design activity is required while the design activity did not complete by applying either (a) increasing the duration of the activity, or (b) initiating a new execution. Such cases of applying self-iterations in conjunction with BRs are demonstrated in Sect. 10.7.1.

[5] It is generalization of $P(A \cup B) = P(A) + P(B) - P(A) \times P(B)$, for independent events.

Applying a validated calculation model based on experimental results is left for future research. Using self- iterations probabilities (Karniel and Reich 2007a) is unique to the current framework.

References

Browning TR, Eppinger SD (2002) Modeling impacts of process architecture on cost and schedule risk in product development. IEEE Trans Eng Manag 49(4):428–442

Browning TR, Fricke E, Negele H (2006) Key concepts in modeling product development processes. Sys Eng 9(2):104–128

Cho SH, Eppinger SD (2001) Product development process modeling using advanced simulation. ASME Conference on design theory and methodology (DECT 2001/DTM), Pittsburgh, PA, September

Danilovic M, Browning TR (2007) Managing complex product development projects with design structure matrices and domain mapping matrices. Int J Project Manag 25(3):300–314

DSM web site (2009) http://www.dsmweb.org. Accessed July 2010

Eckert CM, Keller R, Earl C, Clarkson PJ (2006) Supporting change processes in design: Complexity, prediction and reliability. Reliab Eng Saf Sys 91(12):1521–1534

Eppinger SD, Salminen V (2001) Patterns of product development interactions. International Conference on Engineering Design, ICED 01, Glasgow, Aug. 21–23

Huberman BA, Wilkinson DM (2005) Performance variability and project dynamics. Comput Math Organiz Theor 11:307–332

Karniel A, Belsky Y, Reich Y (2005) Decomposing the problem of constrained surface fitting in reverse engineering. Comput-Aided Des 37:399–417

Karniel A, Reich Y (2007a) Simulating design processes with self-iteration activities based on DSM planning. IEEE Proceedings of the international conference on systems engineering and modeling ICSEM'07 33–41, Haifa, March

Karniel A, Reich Y (2007b) Managing dynamic new product development processes. Proceedings of the 17th annual international symposium of the international council on system engineering INCOSE'07, San Diego, California, June

Karniel A, Reich Y (2011) Formalizing the implementation of DSM-based process planning for NPD. IEEE Tran on Sys Man & Cybernetics, Part A 41(3):476–491

Lester I (1996) Adaptive simulated annealing (ASA): lessons learned. J Control Cybern 25(1):35–54

Maurer M (2007) Structural Awareness in complex product design. Dissertation, Technischen Universität München, Germany

Reich Y, Fenves SJ (1992) Inductive learning of synthesis knowledge. Int J Expert Syst: Res App 5(4):275–297

Rogers JL, Bloebaum CL (1994) Ordering design tasks based on coupling strengths. AIAA, paper no. 94-4326

Sered Y, Reich Y (2006) Standardization and modularization driven by minimizing overall process effort. Comput Aided Des 38(5):405–416

Smith RP, Eppinger SD (1997a) Identifying controlling features of engineering design iteration. Manag Sci 43(3):276–293

Smith RP, Eppinger SD (1997b) A predictive model of sequential iteration in engineering design. Manag Sci 43(8):1104–1120

Yassine A (2007) Investigating product development process reliability and robustness using simulation. J Eng Des 18(6):545–561

Yassine A, Joglekar N, Braha D, Eppinger SD, Whitney D (2003) Information hiding in product development: The design churn effect. Res Eng Des 14(3):131–144

Chapter 5
Simulations

The expectation of minimum iteration marks to yield optimal processes in terms of total time and robustness to activity duration (Steward 1981; Kusiak et al. 1995; Smith and Eppinger 1997a) is an appealing general planning heuristic. However, the counter example simulation results presented in (Browning and Eppinger 2002; Abdelsalam and Bao 2006) contradicted the basic assumption, by demonstrating that shortest process time required more iterations than minimal.

In (Browning and Eppinger 2002) process duration and process cost were estimated using iterations with overlap. Increasing the concurrency of the activities increased the number of iteration marks, and the number of iterations. Thus, cost has increased, but due to overlapping only part of the activity had to be executed again, and the overall duration decreased. The example in (Abdelsalam and Bao 2006) demonstrated a case where more feedback links yielded a shorter process. The marking indicates the number of repetitions; thus, a reordering with more feedback marks, where the span of iterated activities was smaller, has created an overall shorter process time.

A simplified example that demonstrates the approach of (Abdelsalam and Bao 2006) is depicted in Fig. 5.1. In (a), there is one marking that indicates two iterations of the whole process, i.e., each activity is executed three times. Assuming equal activities duration X for all activities we get a total process time of $T = 12*X$. A reordering of the process is presented in (b). The first activity D executes, then C, then due to iteration D and C are executed again, then B executes, iteration of B and C, then A, and finally iteration of B and A. Overall, D and A have executed twice; B and C have executed three times; and the total process time is $T = 10*X$.

It should be noted that the minimum iteration marks concept was developed for a binary DSM, where all marks are equal, while the markings in the above examples have values (iteration probability values in the former case, and number of iterations in the latter case).

Simulating iteration decisions is a simplification of real life situations where such decisions are done according to the status of the design. In the above

A. Karniel and Y. Reich, *Managing the Dynamics of New Product Development Processes*, DOI: 10.1007/978-0-85729-570-5_5,

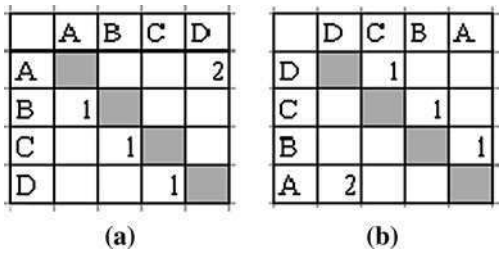


Fig. 5.1 The impact of minimum iterations marking **a** minimum iterations **b** minimum time

examples and in the current research the design itself is not modeled, only the decision to iterate is modeled. The main conclusion drawn from such examples is needed for simulations to guide the planning for process specific conditions, where rules of thumb are inapplicable.

5.1 Using DSM for Simulation

Though DSM-based modeling helps in identifying iterative loops and guides process planning; the reordering algorithms that use only the DSM structure are criticized as inadequate for process optimization (Browning and Eppinger 2002; Abdelsalam and Bao 2006).

The actual use of DSM values for simulation purposes has no broad agreement; different interpretations are used. Furthermore, the DSM information and the DSM structure (after the use of a reordering algorithm) could be interpreted to process logic in several ways. Following are descriptions of various interpretations of DSM for simulation. A detailed comparison (Karniel and Reich 2009a) is presented in Sects. 7.4, 7.5.

Abdelsalam and Bao (2006) used explicit assignment of the required number of iterations. The main assumption was that all activities within an activity loop are affected by the iteration and all should be reworked, i.e., there is an implicit forward link between any two consecutive activities. Forward link values are not used in the simulation. The process is deterministic and serial. Its progress is according to the DSM structure, and the assigned number of iterations of the feedback links. The repeating iterations duration is summed to get the final process duration as criterion for reordering. Since the duration time is stochastic, the process time varies between different run time simulations.

Several studies (Smith and Eppinger 1997a; Huberman and Wilkinson 2005; Yassine et al. 2003) referred to a case of a clique (fully parallel activities). In this case, all activities are coupled; they belong to the same activity loop; and are performed in parallel due to mutual interdependencies, without preceding relations. Since activities iterations are performed in parallel until the whole process completes, the process structure can be defined as deterministic. Yassine et al. (2003) mentioned subprocesses that may have internal order. Huberman and Wilkinson (2005) indicated that the analysis done might apply to each subprocess

with coupled activities. In all those cases, diagonal elements are used, with different interpretations; yet, they do not affect the process progress. Simulations are performed to study the implication of information transfer delays (Yassine et al. 2003) or fluctuations in the interaction values (Huberman and Wilkinson 2005). In the latter case, diagonal elements are used to indicate the autonomous completion rate of the activity.

Choo et al. (2004) presented a combination of DSM-based planning with resource planning, which adds resource constraints. The results of the DSM, calculated by the Analytical Design Planning Technique (ADePT), is a combination of partitioning and tearing. Feedback links with low value are disregarded; thus, large coupled activity blocks are divided into sub-blocks. Sub-blocks are serialized, i.e., activities in the sub-block need to wait for previous (sub) blocks to be completed, and the activities within the sub-block are parallel (they can start in parallel or must end together).

Smith and Eppinger (1997b) described a serialization of coupled activities. A reward Markov chain process is suggested by which the process could progress one activity at a time. The process can either iterate to a previous activity or continue to the next activity. The probability DSM values are used to calculate the Markov chain choice probabilities. Feedback probabilities are taken directly from the probability DSM, but forward probabilities are evaluated in steps according to the process progress.

Sered and Reich (2006) used a parametric-based approach for simulating design modularization and standardization. A sensitivity rate is defined for expressing the sensitivity of a change in the design of one component on the design of another component due to a specific parameter. A cell in the Coupling Index (CI) matrix is the sum of sensitivities. A heuristic procedure (Rogers and Bloebaum 1994) is used to convert the CI matrix values to probability values; and a normalization step is performed to ensure that the maximal sum of out links probabilities is less than one (0.95). The final step is essential due to using serialized Markov chain—random walk simulation. After each activity, the next one might be any linked activity, either forward or feedback, according to direct use of the assigned probabilities (unlike in (Smith and Eppinger 1997b), where an activity can be activated only if its previous one has completed).

Melo and Clarkson (2001) added a Markov chain simulation to the Signposting system. The simulation is used to establish the best task-order in terms of cost and risk to reach the design goal. At each step, any potential task (according to required input status), can be performed. Hence, though the process chooses one-step at a time, parallel activities are being considered. The state of the development process is assessed according to confidence of parameters value.[1]

In (Lévárdy and Browning 2005), activities (including activity iterations) were selected one at a time (as in (Melo and Clarkson 2001)); however, the selection is limited to the most applicable activity according to the target function.

[1] An example of activity states inputs and outputs is given in Sect. 6.6.2, Fig. 6.16.

The presented example progresses one activity at a time, but in general, multiple activities may be performed in parallel. Activity iterations were defined by preconditions on the required value and their confidence, in a similar manner to Signposting. The initial state, the optional activities, and the activity relations are stochastically selected (Monte Carlo). At each process step, all potential activities are checked, and an adaptive selection process is used to choose the best activity to follow, or iterate, according to process state. The process state is evaluated at each decision point after each activity according to cost and duration estimations. The number of iterations is not predefined. The process progress is deterministic once the process data is set. Since the process data used for decisions are probabilistically set at the initialization stage, the process is similar to a Markov chain. Furthermore, the process data could be stochastically changed during the process. Unlike the process in Signposting, making a certain choice may invalidate other potential options; consequently, the process simulation is not equivalent to a parallel process simulation.

Coates et al. (2003) described real time coordination and scheduling system, optimizing resource utilization. The resources perform computational analysis tasks, agent communication, resource management, tasks management, and optimization, in an integrated approach. The analysis tasks have triangular precedence interdependencies (i.e., subdiagonal DSM); however, the optimization task is depended on completing active tasks and it iterates if it is expected to result in better ordering solution.

Browning and Eppinger (2002) defined the feedback probabilities (upper diagonal) logic as first-order iteration, the forward probabilities (subdiagonal) are defined as second-order iterations. After the first execution of the activity its forward links are considered to have a probability value of one; in subsequent iterations, the actual probability values are utilized. No restrictions were made regarding sum of probability values. The same simulation is used by Yassine (2007) for estimating the probability figures assigned to numeric values for validating the transformation between numeric and probability DSM.

Cho and Eppinger (2001, 2005) introduced an integrated project management framework. A DSM-based analysis of the project is used for activities sequencing with minimum number of feedbacks. The process model with additional information regarding activities, overlap (percentage and impact), rework (probability and impact), duration variability (leaning curve, stochastic influences), and resources constraints, are used for process simulation and improving the process plan.

Criticism of DSM-based planning relates to the minimum iteration assertion. Browning and Eppinger (2002) demonstrated a case with coupled activities, which suggested that minimum iterations do not provide the optimal process ordering. (Whitfield et al. 2005) suggested increased parallelism, i.e., minimal number of process stages, as additional trade-off criterion to iterations minimization. In general, the relation between the objective function proposed (i.e., minimum feedback links) and the actual goals of decreasing the process' overall duration, cost, and risk are not well established (Meier et al. 2007). Abdelsalam and Bao (2006) utilized an optimization criterion based on duration and presented a case in

which the optimal sequence with minimum time had more iteration (twelve), than a solution with minimal iterations sequence (eight).

A survey of simulation methods that were used in the context of design processes is presented in (Wynn 2007). The survey demonstrates, by detailed examples, the process modeling methods used for defining the simulations. The modeling methods were classified into Task networks, Queuing models, Multi-agent models, and System dynamics models. Task networks are further classified into Task precedence, Task dependency, and Dynamic task models. Petri nets, which are further analyzed in the current study, were classified as task precedence nets. DSM-based modeling methods were classified as task dependency models, where dependencies between tasks do not directly imply precedence. Signposting (Clarkson and Hamilton 2000), Extended Signposting (O'Donovan et al. 2004; Flanagan et al. 2006), and Adaptive Test Process (Lévárdy et al. 2004) were classified as System dynamics models, where dependencies are not directly represented.

The scheduling literature addresses the need to satisfy desired project quality requirements such as minimum time, resources, or other objective functions (Brucker et al. 1999). The activity relations regarded may include traditional precedence constraints (finish-start) (Brucker et al. 1999), or Generalized Precedence Relations (GPR) (Elmaghraby 1995).

The main problems addressed in the scheduling literature are Resource-constrained Project Scheduling Problem (RCPSP), and uncertainty in activities duration (with or without resource constraints) (Brucker et al. 1999; Elmaghraby 1995; Kolisch and Padman 2001; Herroelen and Leus 2005). Proactive scheduling that accounts for statistical knowledge of uncertainty focuses on the schedule robustness to changes, (e.g., by buffers). Reactive scheduling involves revising the schedule when unexpected events occur, typically resources and time variations, but also adding a new activity (Herroelen and Leus 2004; Vonder et al. 2007). However, the scheduling of iterative activities is typically not considered (Herroelen and Leus 2005).

Consequently, on top of process planning, simulation techniques are required for analyzing process objectives, while handling additional process data such as: time, cost, rework effort (Browning and Eppinger 2002); risk propagation (Eckert et al. 2006); communication time (Maheswari and Varghese 2005); or handling process data variations such as uncertainty and learning (Cho and Eppinger 2005).

The basic DSM model used for process planning typically does not include the activity duration, duration changes due to iteration (learning), and the impact of iterations or rework. Therefore, the information presented in DSM is partial and was criticized as insufficient for process plan evaluation.

Most DSM-based simulations address duration issues using Monte Carlo sampling for the stochastic activities duration. Resources issues were addressed in (O'Donovan et al. 2003). Resource scheduling, based on DSM plan was applied to assignment of computer resources in real time calculations (Coates et al. 2003); and to a weekly assignment of project resources in (Choo et al. 2004),

5.2 DSM-Based Simulation Parameters

The DSM-based simulations were used for several purposes including DSM reordering (not based on minimal number of feedback links), and calculations of processes variables (e.g., duration). The following comparison is focused on DSM reordering; therefore, some of the articles discussed earlier are not part of the following comparison.

The two articles by (Smith and Eppinger 1997a, b) do not describe actual simulations, but process computations. Choo et al. (2004) utilized minimum feedback links for DSM ordering. Yassine et al. (2003) used deterministic DSM with local finished work, system tasks, hidden completed system work, and five types of work transformation matrices to calculate the process open issues.

The typical DSM-based planning algorithms do not utilize the diagonal. However, if there is only one simulation parameter involved per each activity, then the diagonal cells could be used to present it. Such presentation was used in (Sered and Reich 2006) for presenting the design effort of an activity. A summary of simulation parameters used in DSM-based simulations appears in Table 5.1. Simulation parameters might address activities properties, marked by (A); or relations between activities that are presented by additional matrices (M).

The various optimization objectives yield broad and diverse range of simulation parameters. Moreover, the range expands with the type of parameter changes. Some studies have stochastic parameters within a frame of a deterministic process (Abdelsalam and Bao 2006; Huberman and Wilkinson 2005; Coates et al. 2003). Other studies have deterministic parameters within a changing process, where the process route is probabilistically selected (Melo and Clarkson 2001; Sered and Reich 2006); and some studies use both sources of process parameters change.

5.3 Learning Curve

In two of the studies indicated in Table 5.1, a learning curve Learning curve is assumed, i.e., assuming that the activity duration decreases with iterations. Three typical models of the activity duration are depicted in Fig. 5.2 (Andersson 2001).

In (Browning and Eppinger 2002), it was assumed that the first execution takes 100% of the duration and in all the following iterations, the duration is reduce (a step function) to a given percentage of the initial one. Each activity is given a different improvement curve (IC) percentage. In (Cho and Eppinger 2005), the learning curve is defined such that the duration is reduced at each iteration to a percentage of the former execution until it reaches a minimum duration (it is a multi steps function, which could approximate the third option in Fig. 5.2). The former case could be expressed as defining the duration at second execution as minimum duration.

Table 5.1 DSM-based simulations parameters mapping

Source	Optimization objective	Simulation parameters	Parameter type
Abdelsalam and Bao (2006)	Minimum duration	Duration (A)	Stochastic duration
Huberman and Wilkinson (2005)	Minimum instability due to fluctuations	Work transformation fluctuations	Stochastic fluctuations
Sered and Reich (2006)	Minimum design effort	Effort (A) DSM links values (M)	Link values assigned per decision (standardization/ modularization) Probabilistic path choice
Melo and Clarkson (2001)	Minimum cost/risk	Path choice	Probabilistic path choice
Coates et al. (2003)	Maximum Resource utilization	Actual Duration (A); Resource availability	Stochastic duration
Lévárdy and Browning (2005)	Minimum project risk	Duration (A) Cost (A) Technical performance (A)	All parameters are stochastically generated
Browning and Eppinger (2002)	Minimum risk factor of duration and cost	Duration (A) Cost (A) Learning Curve(A) Rework probability (M) Rework impact (M)	Stochastic: Triangular distribution of Duration and Cost (min, expected, max)
Yassine (2007)	Minimum duration/cost variability	Duration (A) Cost (A) Task Volatility (influence) (M)	Stochastic: Triangular distribution of Duration (min, expected, max)
Cho and Eppinger (2001, 2005)	Minimum mean and variance of time	Duration (A) Learning curve (A) Overlap amount (M) Overlap impact (M) Rework probability (M) Rework impact (M)	Stochastic: Triangular distribution of Duration (min, expected, max) Learning curve (initial, % reduction, minimum)

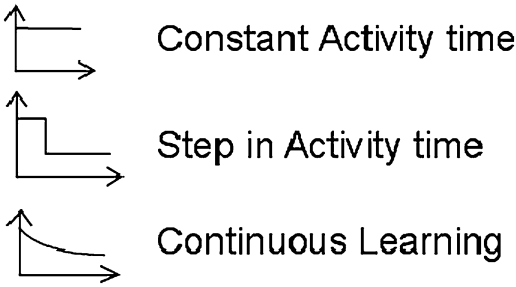


Fig. 5.2 Activity duration changes (reproduced from Andersson 2001)

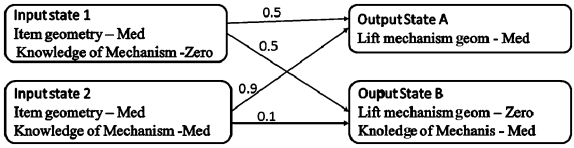


Fig. 5.3 Activity states due to knowledge (adapted from O'Donovan et al. 2003)

An interesting approach was described in (O'Donovan et al. 2003), extending the Signposting system (Melo and Clarkson 2001) to include resources and learning (additional knowledge regarding the problem) in the activity states. Learning has impact on the probability to reach different states of the design activity, rather than duration change; and learning status is changing in different activity states (that describe iterations). An example is depicted in Fig. 5.3, where the input state 1 with zero knowledge (along with other parameters of duration and cost) has 50% probability to yield a success of the design activity (i.e., output state A, create lift mechanism geometry); while a state where such knowledge exists (input state 2) has 90%. However, if on the first iteration the design activity did not succeed (output state B), then knowledge it gained toward the next iteration.

It should be noted that adding details to the states requires coherent definition of more activity states (that indicate iterations). If the number of potential activity states does not increase according to the additional parameters (e.g., still being kept as four states) the additional parameters do not really provide more process options.

In the current work, two models were checked: a constant duration (i.e., no learning), and decreasing the activity duration (resembling Cho and Eppinger 2005). However, no minimum duration was set. Thus, when activity duration reaches a threshold any further iterations have zero time and are not executed (a way of limiting the number of iterations). The Learning Ratio (LR) is assumed to be known (this assumption is used in the current study), and its estimation is not discussed in the DSM literature.

5.4 Objective Function

The objective function of the simulations varies. Two of the studies in Table 5.1 are using a DSM-based plan with minimum iteration criterion. Once the plan is set, it is assumed that there are no feedback links, and a scheduling problem is solved.

Coates et al. (2003) maximized resource utilization (and minimized time) by resource scheduling according to the actual duration of tasks, given the predefined DSM-based plan. In (Choo et al. 2004), the duration and resource availability parameters are used for implementation of activity weekly schedule according to the plan (considering the information needs, and resources availability). In both cases, simulation parameters are not used for further process plan optimization.

Browning and Eppinger (2002) suggested calculating the risk factors for duration and cost, which are the integral of the impact of unwanted result, multiplied by the probability of such result (based on simulation). The impact is calculated as a square of the difference between unwanted outcome and required outcome when overrunning the scheduled due date, or the budget, respectively.

Lévárdy and Browning (2005) used a weighted project risk that includes duration, cost, and technical performance risks. The risk is evaluated in each process step and guides choosing the best task to be performed next.

The simulated parameters in (Sered and Reich 2006) are the DSM links according to decision regarding modularization or standardization of a product component. Simulation results are used to calculate the best process plans, and the overall results are then used for choosing the components that should be standardized or modularized.

Minimum process duration is used in (Cho and Eppinger 2005; Abdelsalam and Bao 2006). The variance of the process duration is additionally considered in (Cho and Eppinger 2005) and is the objective function in (Yassine 2007).

5.5 Statistical Analysis for Decision-Making

Using Monte Carlo simulation (Metropolis and Ulam 1949), the simulated processes proceed by generating random numbers from probability density functions $f(x)$. Parameters of the resulting process may not have a formal distribution, but such distribution can be generated by multiple repetitions of the simulation. Asmussen and Glynn (2007)[2] indicated that according to the Law of Large Numbers (LLN), and the Central Limit Theorem (CLT), the distribution generated by the simulation converges to the actual distribution (that might be unknown, or cannot be expressed by a formula).

Discrete probability mass distribution of the process time can be generated for evaluating mean and standard deviation. In (Cho and Eppinger 2005), probability mass distribution was used for comparing various model assumptions; in (Abdelsalam and Bao 2006) for comparing various DSM rearrangement cases; and in (Huberman and Wilkinson 2005) for comparison of fluctuations strength. Browning and Eppinger (2002) generated the distributions of cost and duration, and then used them to calculate the risk of overrun.

[2] Statistical definitions and equations are described in Annex B, Sect. 14.2.

One of the issues that needs to be addressed is the number of simulations required for the result to converge. In many cases, a large number (assumed to be large enough) is taken: 800 in (Abdelsalam and Bao 2006), 1000 in (Cho and Eppinger 2005), and 10,000 in (Huberman and Wilkinson 2005). Asmussen and Glynn (2007) describe a typical approach of estimating the required number by a small size batch of simulations. Browning and Eppinger (2002) defined criteria for estimating convergence of the distribution by running batches until the relative mean and deviation of the additional batch is below a threshold. However, such relative threshold has no statistical meaning.

Given the simulation results, a decision-making procedure should take place, typically by choosing the best results according to the criterion. Since the results are distributed, statistical measures should be taken to support the decisions, i.e., to check if the results are statistically significant. However, the DSM-based studies reviewed do not address that requirement, and typically just compare mean and standard deviations without checking their significance. The importance of evaluating the validity of advice derived from process simulation was emphasized by Wynn (2007).

The expected value of a large number of simulation results have properties of a normal distribution . If we split the simulation runs to K sections of size M, the expected values taken from these sections have a t-test distribution. Furthermore, a function over the expected value of simulation parameters has a normal distribution; and function estimations over expected values taken from several sections have a t-test distribution. These properties are used for estimation of variance and confidence interval of the function (Asmussen and Glynn 2007); see the statistical Annex B, Sect. 14.2.

In (Karniel and Reich 2009b), the properties of multiple simulation repetitions are used for supporting the decision-making by statistically evaluating hypothesis regarding differences between the results of applying different business rules. More details appear in Sect. 11.7.1, and examples are found in Sect. 12.7.

References

Abdelsalam HME, Bao HP (2006) A simulation-based optimization framework for product development cycle time reduction. IEEE Trans Eng Manag 53(1):69–85

Andersson J (2001) Multi objective optimization in engineering design: applications to fluid power systems. Dissertation. Institute of Technology, Linköpings universitet, Sweden

Asmussen S, Glynn PW (2007) Stochastic simulation algorithms and analysis. Springer, Berlin

Browning TR, Eppinger SD (2002) Modeling impacts of process architecture on cost and schedule risk in product development. IEEE Trans Eng Manag 49(4):428–442

Brucker P, Drexl A, Möhring R, Neumann K, Pesch E (1999) Resource-constrained project scheduling: Notation, classification, models and methods. Eur J Oper Res 112:3–41

Cho SH, Eppinger SD (2001) Product development process modeling using advanced simulation. ASME Conf on design theory and methodology (DECT 2001/DTM), Pittsburgh, PA, September

Cho SH, Eppinger SD (2005) A simulation-based process model for managing complex design projects. IEEE Trans Eng Manag 52(3):316–328
Choo HJ, Hammond J, Tommelein ID, Austin SA, Ballard G (2004) DePlan: A tool for integrated design management. Autom Constr 13:313–326
Clarkson PJ, Hamilton JR (2000) Signposting, a parameter-driven task-based model of the design process. Res Eng Des 12(1):18–38
Coates G, Duffy AHB, Whitfield I, Hills W (2003) An integrated agent-oriented approach to real-time operational design coordination. Artif Intell Eng Des Anal Manuf 17:287–311
Eckert CM, Keller R, Earl C, Clarkson PJ (2006) Supporting change processes in design: Complexity, prediction and reliability. Reliab Eng Safety Sys 91(12):1521–1534
Elmaghraby SE (1995) Activity nets: A guided tour through some recent developments. Eur J Oper Res 82:383–408
Flanagan TL, Eckert CM, Keller R, Clarkson PJ (2006) Bridging the gaps between project plans and reality: The role of overview. Proc of tools and methods of competitive eng (TMCE 2006), 1:105-116, Ljubljana, Slovenia
Herroelen W, Leus R (2004) Robust and reactive project scheduling: A review and classification of procedures. Int J Prod Res 42(8):1599–1620
Herroelen W, Leus R (2005) Project scheduling under uncertainty—Survey and research potentials. Eur J Oper Res 165:289–306
Huberman BA, Wilkinson DM (2005) Performance variability and project dynamics. Comput Math Organiz Theory 11:307–332
Karniel A, Reich Y (2009a) From DSM based planning to design process simulation: A review of process scheme logic verification issues. IEEE Trans on Eng Manag 56(4):636–649
Karniel A, Reich Y (2009b) Statistical analysis of process simulations, in Int Conf on Eng Design (ICED' 09), Stanford, CA
Kolisch R, Padman R (2001) An integrated survey of deterministic project scheduling. Omega 29(3):249–272
Kusiak A, Wang JR, He DW, Feng CH (1995) A structured approach for analysis of design processes. IEEE Trans Compon Packag Manuf Technol - Part A 18(3):664–673
Lévárdy V, Browning TR (2005) Adaptive test process—Designing a project plan that adapts to the state of a project. 15th Annual int symposium of the int council on sys eng (INCOSE), July
Lévárdy V, Hoppe M, Browning TR (2004) Adaptive test process—An integrated modeling approach for test and design activities. In: The product development process, proc of DETC'04 ASME 2004 Design eng technical conf and comput and inf in eng conf, Salt Lake City, Utah, September
Maheswari JU, Varghese K (2005) Project Scheduling using dependency structure matrix. Int J Project Manag 23:223–230
Meier C, Yassine A, Browning T (2007) Design process sequencing with competent genetic algorithms. J Mech Des 129(6):566–585
Melo AF, Clarkson PJ (2001) Design process planning using a state-action model. 13th Int conf eng design (ICED 01), Glasgow
Metropolis N, Ulam S (1949) The monte carlo method. J Am Stat Assoc 44:335–341
O'Donovan BD, Clarkson PJ, Eckert CM (2003) Signposting: Modeling uncertainty in design processes. Int conf on eng design (ICED 03), Stockholm, August 19-21
O'Donovan BD, Eckert CM, Clarkson PJ (2004) Simulating design processes to assist design process planning. Proceedings of ASME Design Eng Technical Conf, Salt Lake City, Utah, USA
Rogers JL, Bloebaum CL (1994) Ordering design tasks based on coupling strengths. AIAA, paper no. 94-4326
Sered Y, Reich Y (2006) Standardization and modularization driven by minimizing overall process effort. Comput-Aided Des 38(5):405–416
Smith RP, Eppinger SD (1997a) Identifying controlling features of engineering design iteration. Manag Sci 43(3):276–293

Smith RP, Eppinger SD (1997b) A predictive model of sequential iteration in engineering design. Manag Sci 43(8):1104–1120

Steward DV (1981) The design structure system: A method for managing the design of complex systems. IEEE Trans Eng Manag 28:71–74

van de Vonder S, Demeulemeester E, Herroelen W (2007) A classification of predictive-reactive project scheduling procedures. J Scheduling 10(3):195–207

Whitfield RI, Duffy AHB, Gartzia-Etxabe LK (2005) Identifying and evaluating parallel design activities using the design structure matrix. Int conf on eng design, ICED05, Melbourne, August

Wynn DC (2007) Model-based approaches to support process improvement in complex product development. Dissertation, University of Cambridge

Yassine A, Joglekar N, Braha D, Eppinger SD, Whitney D (2003) Information hiding in product development: The design churn effect. Res Eng Des 14(3):131–144

Yassine A (2007) Investigating product development process reliability and robustness using simulation. J Eng Design, 18(6):545–561

Chapter 6
Process Modeling Using Workflow-Nets

6.1 Process Modeling

Once a project plan is set, various methods can be used for its implementation. The previously reviewed DSM based simulations do not use a formal method for implementing the process model. Such methods include proprietary workflow models, Graphical Evaluation and Review Technique (GERT), Task nets, pi-calculus, and Petri nets (discussed in the next section).

For its implementation, the process is defined by the Process scheme model representing the precedence relations in addition to process logic. Generalized precedence relations (GPR) (Elmaghraby 1995) include the typical end–start relations, start–start, end–end, and iterations (cycles). The overlap relation was defined in Cho and Eppinger (2001), as a start–start relation with delay. The relations are implemented by defining the Input Logic (IL) of an activity (preconditions) and the Output Logic (OL) (post-conditions).

The GERT method (Taylor and Moore 1980; Neumann 1990; Barjis and Dietz 2000; Zanddizari 2006) addresses probabilistic routing and feedback loops; and can be used for process duration calculations or simulations. The GERT technique includes the modeling of systems in a network form, and analysis through simulation of stochastic networks having logical nodes and directed branches. GERT uses activity on arc (AOA); nodes are considered as systems states, and arcs represent transitions leading from one state to the other. Graphical symbols are used to define activity relations and the process logic. An extended set of symbols was presented in Barjis and Dietz (2000), implementing probabilistic and deterministic exits from a node, and multiple initial and final nodes. Like in other modeling methods, the sequence of tasks is assumed fixed (Cho and Eppinger 2005).

Task nets are directed graphs that represent the input and output relationships among the various tasks in the process. Tasks (activities) are represented by nodes (i.e., activity on node, AON), and arcs are linking between the tasks. Output of a

A. Karniel and Y. Reich, *Managing the Dynamics of New Product Development Processes*, DOI: 10.1007/978-0-85729-570-5_6,

source task is the input of the target task through the link. An arc may represent a parameter generated by the source task and needed by the target task (Braha 2002), control flow, or iteration flow (Heller and Westfechtel 2003). Task nets were used to represent dynamic evolution of design processes in Heimann et al. (1996), and were enhanced in Westfechtel (1999) by including control flow links, data flow links, and feedback links. Formally, a task net is based on a graph rewriting system that enables adding and hierarchically modifying the net during process execution. The work of Heimann et al. (1996) was criticized by Reichert and Dadam (1998), indicating that tasks to be added needed to be pre-defined in the scheme; deleting and changing the tasks sequence were not available; and correctness issues were not addressed. The representation of off-diagonal links in DSM and the arcs in task net are very similar (Braha 2002). This similarity is used in defining the DSM net in Sect. 9.4.

The pi-calculus (Milner 1999) process algebra is a textual annotation of a process model that provided the formalism for modeling of changing, interacting processes. The pi-calculus process expressions define the process state, and the relations between the process components [which are defined as (sub) processes or agents]. The process state can evolve to other states through state transitions. Pi-calculus describes communication (input and output prefixing, and the data); concurrency construct (parallel execution); new name assignment (e.g., new link creation); and process replications. The pi-calculus extends the Calculus of Communication Systems (CCS) (Milner 1980), by adding mobility [e.g., the ability to define a new link (channel) between processes through an existing link; and agent ability to divide or "die"]. Processes can be described in a compact manner, utilizing the recursive structure of the process expression; however, the interpretation of the system is complex and requires the use of transition rules that describe all the potential transitions to next process states in order to calculate the process transitions. The pi-calculus syntax and semantics are described in Annex A, Sect. 14.1.

Petri net (Reising and Rozenberg 1998) are established graphical formal language for process model specification. Petri nets can be used as diagrammatic tool for modeling and analyzing distributed system behavior including sequential, conditional, parallel, and iterative routing (van der Aalst and van Hee 2002). Petri nets formal proofs (detailed in the following sections) were utilized in the current research for establishing formal proofs of DSM nets properties. Choosing Petri nets for formalization was based on their relative user-friendliness, being more close to the simple Task nets used for implementation. The use of pi-calculus as a formal vehicle should be further investigated. Comparison examples of pi-calculus and Petri nets were developed in this work and are described in Annex A, Sect. 14.1.

6.2 Process Correctness

Process correctness can be defined as ensuring certain process behavioral characteristics (e.g., reaching a steady state or reaching a termination state). Process behavior may be verified using numerous simulations, yet verification of

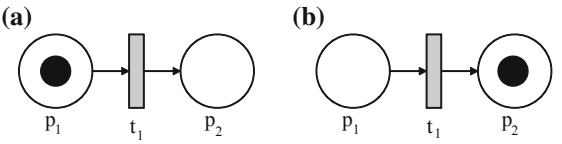


Fig. 6.1 Petri net basics

process characteristics according to the process structure is more fundamental and illuminating. Process correctness verification is profoundly discussed in the workflow literature (van der Aalst 2001; Rinderle et al. 2004; Sadiq et al. 2004, Farrell et al. 2007), but seldom mentioned in the DSM literature. This omission, coupled with the lack of clear translation from DSM representation to a process scheme, might lead to generating processes that may fail (Karniel and Reich 2009). Consequently, any proposed process scheme should be verified. Moreover, in NDP processes, the product knowledge changes, and the DSM-based plans should be updated (Karniel and Reich 2007a). Consequently, automation of the DSM translation is required. Thus, it becomes essential to guarantee that the resulting process models are correct and executable. Such automation might also be very useful for large-scale processes.

Correctness criteria are therefore defined to ensure the required behavioral characteristics. For directed acyclic graphs (DAG) the correctness criteria are described in Sadiq and Orlowska (1999). van der Aalst (1998) defined the *Soundness* criteria for WF-nets, later evolved to compositional soundness criteria (van Hee et al. 2003; Ping et al. 2004). Karniel and Reich (2007b) defined correctness criteria for dynamic scheme processes in the context of design processes.

If a process is *sound*, then it has the required characteristics. Checking soundness can be done using diagnostic tools (e.g., "Woflan" system, Verbeek et al. 2001). Ping et al. (2004) defined WRI-WF-nets (Well-handled with Regular Iterations) that are inherently sound, and proposed a verification approach, where processes can be hierarchically built as sound processes. This approach was used in the current research.

The following section describes Petri nets, then the more specialized workflow WF-nets, and WRI-WF-nets; and finally their formal definitions that are later used for defining the DSM conversion to a process scheme (Sect. 9.4).

6.3 General: Petri Nets

A Petri Net in is a bipartite directed graph with two node types called *places* and *transitions* that are connected by directed *edges* (arcs). Place can contain *tokens*. A transition may 'fire', i.e., consume tokens from its input places and add tokens to its output places according to 'firing rules'. Process activities are modeled by transitions; while places correspond to conditions. *Marking,* representing the process state, is the distribution of tokens over the places. In Fig. 6.1, the transition t_1 has one input place p_1 and one output place p_2. The transition is ready to fire

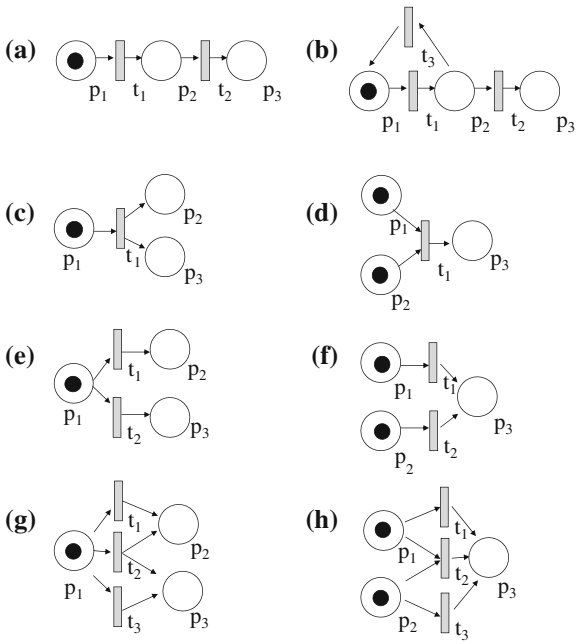


Fig. 6.2 Logic representation in Petri net **a** Sequence **b** Iteration **c** Split-And (Concurrency, Parallel split) **d** Join-And (Synchronization) **e** Split-Xor (Exclusive choice) **f** Join-Xor (Simple Merge) **g** Split-Or (Multiple Choice) **h** Join-Or (Multiple merge)

since p_1 has a token. The process state, i.e., marking has changed from $M_0 = (1, 0)$ at (a) to $M_1 = (0,1)$ after firing (b).

Process logic can be modeled by combinations of transitions and places as depicted in Fig. 6.2. The yet another workflow language (YAWL) provides a more convenient graphical representation of the distinct logic patterns (van der Aalst and Hofstede 2005).

The edge capacity defines the number of tokens consumed by the transition from each input place, or the number of tokens produced to each output place. In this work, we assume that all edges have capacity = 1. A transition can fire when all its input places have enough tokens to be consumed (according to our assumption, each input place should have at least one token). In Fig. 6.2d, the transition may fire only when its two input places have a token. There is no conservation of the number of tokens. In Fig. 6.2c, transition t_1 has consumed one token in its input place and produced two tokens, one in each output place. In (d), two tokens are consumed and one is produced.

Being enabled (i.e., having all required token in input places) does not indicate immediate activation (i.e., firing). It is assumed that firings of the transitions are not simultaneous. The consumption of input tokens is assumed immediate on firing, and all required tokens are consumed simultaneously. When produced, all tokens are produced simultaneously.

There are several options for the timing of producing the tokens. The option typically used in Petri net literature is an immediate production of the output

tokens, i.e., the system spend time only in states, transition between states in instantaneous.

Another option is that the transition (activity) has duration. Actually, the latter option can be presented by the former one if we define an initial transition (the activity started) and a final transition (the activity completed). Thus, an activity is defined by a set of transition-place-transition nodes. We find it more intuitive (with less graphic symbols) to use the second representation option, though it has a full equivalence by the first one.

Furthermore, there is no predefined order of firing. Thus, in a Split-Xor (e), both transitions are enabled; at a certain time, one of them may fire (not both simultaneously), and at that time, the other transition will (immediately) be disabled (as there are no input tokens).

Split-Or (g) and Join-Or (h) are frequently represented the same way as Split-Xor (e) and Join-Xor (f), respectively; creating inconsistency (Dehnert and van der Aalst 2004). However, the complexity of the explicit correct representation is combinatorial, i.e., representing the possibilities of three options would require six transitions. This may explain the general implicit use of (e) and (f) to represent the Split-Or and Join-Or functions. Additionally, in many cases the terms Split-OR and Join-Or are used to indicate Figs. (e) and (f), respectively, which are actually Split-Xor and Join-Xor logic. Such misuse happens in some of the following examples as well.

A proper representation of iteration Fig. 6.2b is using two transitions, one representing the design activity, and one representing the decision to iterate (send feedback), or otherwise continue; each transition having its own properties (activity duration, and probability to send feedback). Such representation is described in Andersson (2001), and is using a Split-Xor logic (e).

Other iteration-logic options, described in following section, require definitions that are more complex. Implementation of additional constructs such as overlapping and alternative routing was suggested in Jun et al. (2006), using a DSM net and an extended probability DSM. A comprehensive study of construct definitions (workflow patterns) was elaborated in (van der Aalst et al. 2003a, b).

6.4 WF-Nets

WF-nets (workflow nets) (van der Aalst 1998) are a class of Petri nets used for specifying the control aspect of workflow processes. The *soundness* criteria for WF-nets were defined in van der Aalst (1998), later defined as *1*-soundness, and enhanced in van Hee et al. (2003) for net compositions that keep the *generalized soundness* properties. van Hee et al. (2004) proved that for any WF-net, it could be decided if the generalized soundness criteria are met or not.

Ping et al. (2004) defined WRI-WF-nets (Well-handled with Regular Iterations) that are constructively sound; they are restricted to regular iterations; therefore, activities within the iterated part do not activate some other objects outside that

process section during iteration. The WRI-WF-nets recursive definition supports hierarchical modeling. The following section formally describes the WF-net properties.

6.5 Correctness Criteria

Correctness criteria are used to describe the proper behavioral characteristics of the process. Their definitions are context oriented depending on the required process, e.g., a project oriented process, such as the design process, the process should have defined start and end, and a typical requirement would be to reach the process end.

Typically, the goal is to define the existence of required characteristics according to the structure of the process, the *process scheme*. While the analysis of a fixed process scheme can assure the required properties of a given process, a dynamic process scheme, which changes during the process require an inherent build approach that can assure the required properties while changing the *process scheme* Ping et al. (2004).

For DAG processes, the following criteria were described in Sadiq and Orlowska (2000). The processes should be free of Deadlocks and free of Lack of Synchronization. The first criterion reflects a case of joining exclusive Split-Or (choice) paths with a Join-And (synchronize), which results in a deadlock conflict. The second requirement reflects a case of joining Split-And (fork) concurrent paths with a Join-Or (multi merge), which introduces lack of synchronization conflict. Lack of synchronization manifests as unintentional multiple activation of an activity.

Gruhn and Laue (2006) defined the nesting properness criterion that describes the process structure nesting in an event-driven process chain (EPC), which has activities and logic controls. Each subprocess between a pair of logic controls (e.g., Split-And and Join-And) is additional level in the process nesting. A properly nested process has no "jumps" (i.e., no links that are targeted "outside" of the subprocess). This property is the process equivalent to using structured loops versus "goto" spaghetti code. This criterion applies to DAG as well as iterative processes.

The process examples in Fig. 6.3a and b were presented in Sadiq and Orlowska (1999). In the net diagram, a circle indicates Join-Or (input), or Split-Or[1] (output) logic; a rectangle indicates an activity with Join-And (input), or Split-And (output) logic.

In Fig. 6.3a, a Split-Or logic is used thus only one of the following activities can perform. The marked activity waits for inputs from both activities, but since only one of them could have been activated, we have a deadlock. Same description applies to Fig. 6.3c using Petri net symbolic (transitions and places). Only one of the transitions can consume the initial token, and produce a token in its output;

[1] This is an example of typical terms misuse: the actual logic indicated is Split-Xor as in Fig. 6.2 e.

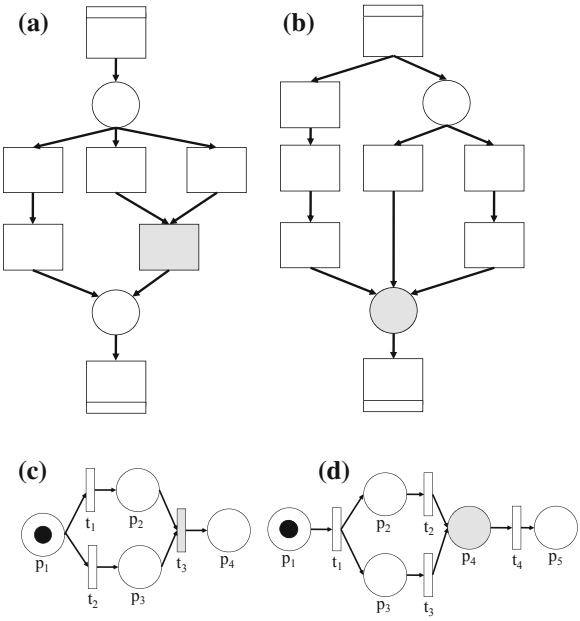


Fig. 6.3 Deadlock and Lack of Synchronization **a** Dead lock—net diagram **b** Lack of synchronization—net diagram **c** Dead lock—Petri net **d** Lack of synchronization—Petri net

thus, the marked transition will never fire, as it requires having at least one token in each input place.

In Fig. 6.3b, the initial activity has a Split-And logic (i.e., two routes are activated in parallel). The marked Join-Or will get a signal from the left path and additional signal from one of the other paths (that have a Split-Or logic), causing lack of synchronization. A Petri net case of lack of synchronization is depicted in Fig. 6.3d. The marked place will get two tokens and the following transition will be activated twice.

In both cases, we have two parallel paths starting (by split) with one type of logic and completing (join) with another type. In the deadlock case, we split with Split-Or and join with Join-And. In the lack of synchronization case, we split with Split-And and join with Join-Or. van der Aalst (2000) presented the 'well-handled'condition, which is the interpretation of both requirements, in terms of a Petri net. A well-handled Petri net has no fully separated parallel paths between different types of nodes (a formal definition is presented in Sect. 6.7).

As indicated by Gruhn and Laue (2006), the proper nesting criterion for EPC can also be described in terms of the well-handled condition by converting an EPC to a Petri net as described in van der Aalst (1999).

A WF-net proper process, defined according to the soundness (*1*-soundness) criteria (van der Aalst 1998) has the following properties (formally defined in Sect. 6.7):

1) From every process state, which is reachable from the initial state, a firing sequence leads to the termination state. The process should terminate eventually.
2) Once the terminal state was reached, there are no open issues; formally: there are no tokens in places other than the termination state.

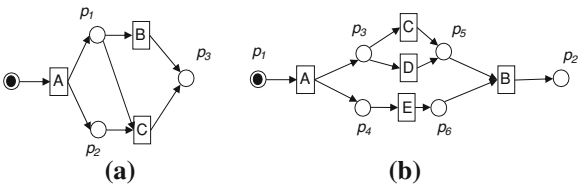


Fig. 6.4 Soundness and well-handled criteria of DAG Petri nets **a** Not well-handled and not sound **b** Sound and well-handled

3) There are no "dead" activities (i.e., no activities that could not execute due to unmet pre-conditions).

The Petri net in Fig. 6.4a represents a process that is not well-handled. The process in this case is not sound since if transition (activity) *B* fires, then *C* could never fire and the process completes with a token left in p_2.

The process in Fig. 6.4b is sound. After the execution (firing) of *A*, the process splits to parallel paths, in one path either *C* or *D* execute, and in the other *E*. Activity *B* waits for both paths to complete. There are two paths from *A* to p_5 (through *C* and through *D*, respectively), but since both paths include p_3 they are not fully separated and the process is well- handled.

The Soundness criteria correspond to the absence of basic errors in a workflow model (Farrell et al. 2007). The concepts of correctness criteria, and specifically soundness, are utilized in the current work to interpret the DSM planning (reordering results) into a process scheme. Such implementation bridges a gap in the literature, between the DSM planning phase and process definition and implementation phase.

It should be noted that process characteristics that are considered unacceptable in the context of administrative processes (typically addressed in Workflow literature), are very common in the Design Process context (Smith and Morrow 1999). Examples are iterations and lack of synchronization. Iterations solve the 'Dynamic change bug' (van der Aalst 2001), i.e., changes in previous stages of the process, and as indicated are typical to design processes.

Lack of synchronization in terms of administrative process tasks means that a task gets a starting signal and then additional unsynchronized input that require restarting. Yet, in the context of design activities, such occurrences are expected and accepted. Correctness criteria for processes with dynamic process scheme in the context of design activities were defined in Karniel and Reich (2007b), and are detailed in Sect. 9.19.2.

6.5.1 *Well-Handled Processes and Soundness of Iterative Processes*

Defining soundness for iterative processes requires the assumption of a "fair process" i.e., there is an equal probability of firing and no specific order of firing (van der Aalst and van Hee 2002). For example, in Fig. 6.2b, if transition t_3 always fires before t_2, then the cycle can repeat forever. The fairness assumption is

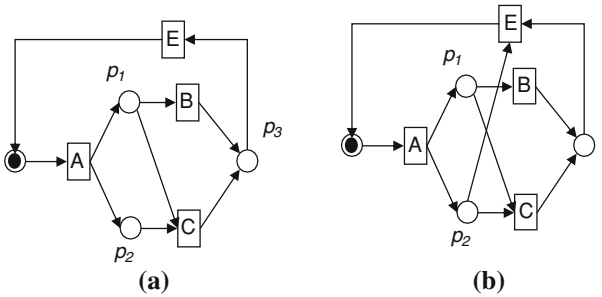


Fig. 6.5 Cyclic not well-handled Petri nets **a** Not well-handled and not sound **b** Not well-handled and sound

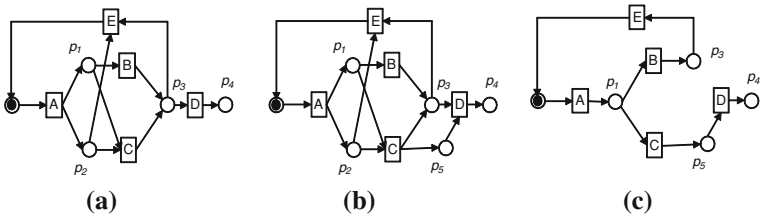


Fig. 6.6 Continuing a not well-handled Petri net **a** Not well-handled and not sound **b** Not well-handled and sound **c** Sound and well-handled

applicable in other cases as well, and a sound "fair" process will eventually complete.

The well-handled condition is very powerful in the case of acyclic (DAG) processes. An acyclic process that is not well-handled will not be sound. A process with a deadlock, Fig. 6.3c, does not meet requirement (1) of soundness. Lack of synchronization, Fig. 6.3d, as well as the example in Fig. 6.4a may yield remnant tokens; thus, violating requirement (2) of soundness.

However, the well-handled criterion is less powerful in the case of cyclic processes. A cyclic process can be sound without being well-handled as presented in the following examples (a contribution of the current research). The cyclic process presented in Fig. 6.5a was created by adding an iteration path to the process in Fig. 6.4a. The problems of the previous process are now aggravated as multiple tokens can accumulate in p_2 (each time B fires). The new iterative process is not sound.

In Fig. 6.5b, a link was added between p_2 and E. The process now represents the option either to execute activity C (E is disabled) or if B executed (fired) then E (iteration) would follow. Eventually C will execute completing the process without excess tokens. This iterative process is sound; however, it is not well-handled. In addition to the distinct paths from A to p_3, there are also distinct paths from p_2 to E, i.e., the additional link did not change the process into a well-handled process. The latter example demonstrates the capability to utilize iterations for overcoming problems of acyclic processes.

The soundness problem was solved in Fig. 6.5b, only for the case of p_3 being the final place. However, if the process can continue, as demonstrated in Fig. 6.6a,

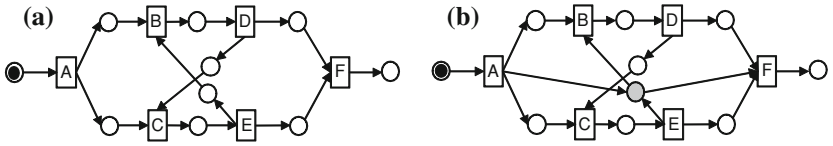


Fig. 6.7 Cyclic Petri nets **a** Well-handled with a deadlock **b** Sound and not well-handled

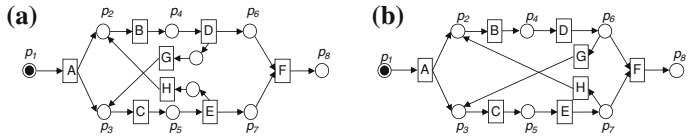


Fig. 6.8 Iterations of parallel paths **a** Live lock process **b** Sound process

then activity D could follow B, and a token would be left in p_2, i.e., in this case the process is not sound.

In this case, adding the place p_5 in Fig. 6.6b makes the system sound. After the execution of A, if B executes, then E can follow (D is not enabled); and once C executes (E is disabled), only D can follow. Yet, this process logic can be equivalently described by the sound and well-handled process scheme in Fig. 6.6c.

Another type of process deadlock in a cyclic net is depicted in Fig. 6.7a. The cycle is *(BDCEB)*. Transitions (activities) B and C are waiting to tokens at the output of E and D, respectively, while D waits to B, and E waits to C. This process is well-handled but is not sound.

In Fig. 6.7b, a link was added from transition A to the marked place, and then a link to F. In this process, there is no deadlock, and the activities are actually serialized in the order *(ABDCEF)*. The process is sound. The process is not well-handled having two separated paths from A to the marked place.

In Fig. 6.8a, the activities (BD) start in parallel with activities (CE). By the completion of D, activities (CE) start again through G, and by the completion of E, activities (BD) start again through H. The cycle is endless (a "live lock"). The token accumulated in p_6 and p_7 cause activity F to execute repeatedly. Since the process never completes, it is not sound. The process is not well-handled, e.g., there are two separated paths from A to p_2 (and from A to p_3).

The process in Fig. 6.8b is not well-handled (as the previous process), but it is sound. Like any cyclic process it may perform endlessly under certain circumstances, for example, assuming that activity G always executes before F (even if F was enabled). Yet, if the process was "fair", it would complete eventually.

This process demonstrates a proper process scheme implementation of an activity cycle. Transitions G and H are considered as logic activities. Transitions A and F could either represent design activities or logic activities.

Unlike previous examples that did not describe parallel processes (Figs. 6.6b, 6.7b), the process in Fig. 6.8b does not have a well-handled equivalent.

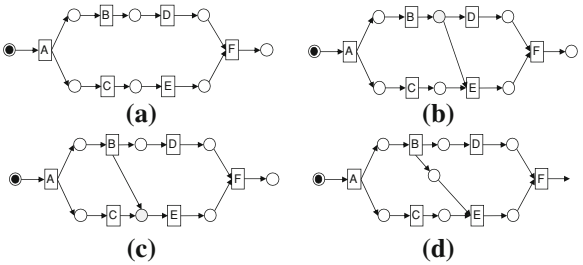


Fig. 6.9 Changes in a DAG process **a** Parallel path process **b** Change leading to deadlock **c** Lack of synchronization **d** Implementing the change properly

6.6 Process Scheme Modifications

One of the major problems addressed in the workflow related literature is the modification of the process scheme. Process modifications of a fixed scheme process might be a managerial requirement because of the analysis and improvement in the BAM stage (Fig. 2.7). Changing the process scheme is an intricate issue since the process planner intentions may lead to improper process. The following examples present some potential problems in modeling process changes.

A proper process with two parallel paths of activities (*BD)* and (*CE*) is depicted in Fig. 6.9a.[2] We want to make a change such that activity *E* waits for the completion of *B*. An implementation as in Fig. 6.9b will lead to a deadlock (two paths from the marked place to activity *F*. Either *D* can execute or *E*, same as in Fig. 6.3). While such case is easy to detect in this simple example, in a more complicated one such mistake may happen (Sadiq et al. 2004).

The implementation in (c) yields a lack of synchronization (two paths between activity *A*, and the marked place, i.e. Activity *E* will be initiated twice), and does not fulfill the requirement of *E* waiting for the completion of *B*. The proper change is depicted in (d).

On next iteration of the process planning, we want to add the option of repeating activities (*BD*). A "straight forward" implementation of a cycle (by adding activity *G* and a place) may lead to a deadlock, Fig. 6.10a. Transition *B* will endlessly wait for a token from the output place of *G*. Such cyclic deadlocks can be solved by a *proper siphon*. A siphon[3] is defined as a set of places in the Petri net such that their input transitions (activities) are subset of their output transitions. A proper siphon is a non-empty siphon (van Hee et al. 2004). The siphon *S* is marked in Fig. 6.10b, where $S = \{p_2, p_3, p_4\}$, the input transitions are $\{B, D, G\}$, and the output transitions are $\{B, D, G, F\}$. There is an initial marking of a token at place p_4.

[2] Using instantaneous transitions (with immediate production of output tokens), this process represented two parallel activities, each with start and end transactions, while other transitions are used for representing logic.

[3] Formally (see additional definitions in Sect. 6.7), *S* is a siphon if $\bullet S \subseteq S\bullet$, where $\bullet S$ are the set of input transitions of the places that belong to *S*, and $S\bullet$ are the output transitions of these places.

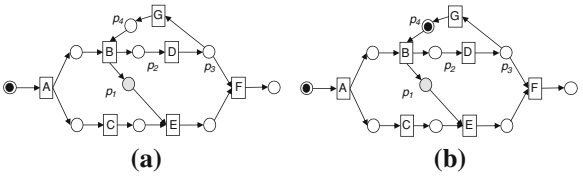


Fig. 6.10 Cyclic process **a** Cycle deadlock **b** A siphon marking–not sound

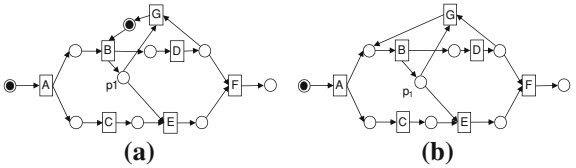


Fig. 6.11 Sound cyclic processes **a** Sound process with siphon **b** Sound process

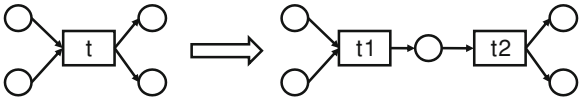


Fig. 6.12 Transition refinement

This solution in turn will cause a non-sound improper process since repetitions of (*BDG*) may cause accumulation of tokens in the place p_1 (marked).

In Fig. 6.11a, we get a sound solution by adding a link from place p_1 to activity *G*. In this case, (*BDG*) can repeat while *E* did not execute. Another sound solution without using a siphon is presented in Fig. 6.11b. In both cases, the processes are not well-handled.[4]

The above examples (Figs. 6.10, 6.11) demonstrate the options of using siphon or not using siphon for handling cyclic (iterative) processes. In the following sections, siphons are not used.

6.6.1 Process Expansion

In some cases, the required change is expansion of the process scheme, i.e., combining processes, or detailing sub processes. An example of transition refinement of a (van Hee et al. 2003) is depicted in Fig. 6.12.

In the general case, such expansion may cause process correctness issues. For a DAG process it was proposed (Mangan and Sadiq 2003) to restrict the process structure types to sequential and pairs of Split-And (fork) and Join-And (synchronize) that can be used during run-time for building the process.

In van der Aalst and van Hee (2002), expansions of a DAG process are defined such that the subprocesses are well-handled; e.g., a place can be extended by a

[4] There is a path from p_1 to *F* through *E*, and another path through (*G, B,* and *D*).

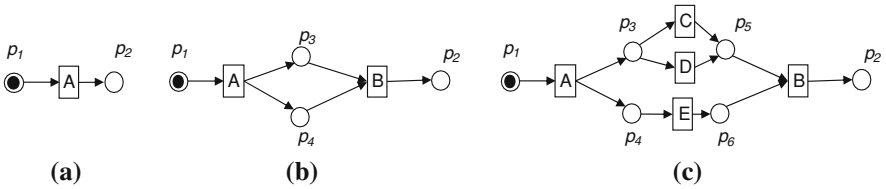
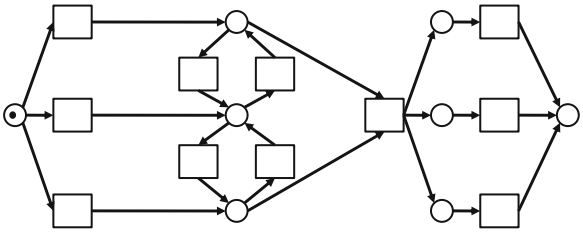


Fig. 6.13 Process expansion

Fig. 6.14 *3-sound* process



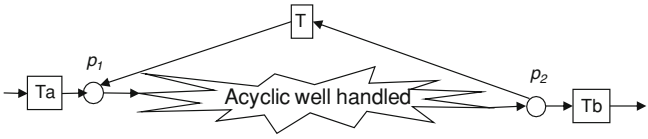


Fig. 6.15 Well-handled with regular iteration process

Split-or and Join-Or; a transition can be extended by Split-And and Join-And; and both can be extended by a sequential path. For example, the transition *A* in Fig. 6.13a is expanded by Split-And and Join-And pair to the process in (b). The place p_3 is then expanded by Split-Or and Join-And pair; and the place p_4 is expanded by a sequential path to the process in (c).

The same idea could be used in the other direction, i.e., reduction rules. Reduction rules were used in Sadiq and Orlowska (1999), to check process correctness. If a given acyclic scheme could be reduced to a sequence then the original process scheme has no deadlocks or lack of synchronization.

A more general approach applicable for processes with cycles was presented in van Hee et al. (2003). For ensuring a sound process any subprocess should have generalized soundness process, i.e., it should be *K-sound* for any natural number *K*. *K-soundness* indicates that if the process starts with *K* tokens at the initial place it will complete with *K* tokens at the final place. A sound process (van der Aalst 1998) is therefore *1-sound*.

An example of a *3-sound* process that is not *1-sound* and not *2-sound* was presented in Ping et al.(2004), Fig. 6.14. If the subprocess presented in Fig. 6.14 is the expansion of another process, and that other process would not provide three tokens to the input place, then the whole process would not be sound. The process in Fig. 6.14, as can be easily noticed, is not well-handled.

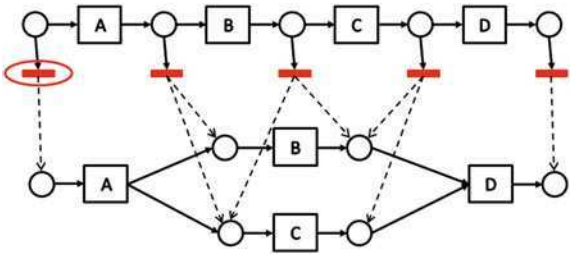


Fig. 6.16 From serial to parallel process (reproduced from Rinderle et al. 2004)

Ping et al. (2004) defined acyclic well-handled subprocesses, and well-handled subprocesses with regular iteration. The latter is a WF-net with an acyclic well-handled process between the first and the final placed plus a transition from the final place to the initial place as illustrated in Fig. 6.15.

The simplest subprocess with regular iteration is a simple iteration, Fig. 6.2b. WRI-WF-nets were defined as processes that are built by using an expansion method. Starting with an acyclic well-handled process it can be expanded by acyclic well-handled subprocesses or by well-handled with regular iteration subprocesses. The parallel process in Fig. 6.9a could easily be obtained using that method. However, the example in Fig. 6.9d could not. Though the scope of WRI-WF-nets processes is limited, it is useful for presenting certain types of processes resulting from DSM based plans, as presented in Sect. 9.5.

6.6.2 Dynamic Process Changes

A well-known change addressed in the workflow literature is the dynamic conversion from a serial to parallel process and vice versa. A solution of the first problem was defined in Ellis et al. (1995) using "jumpers" (a jumper is marked by a red cycle, Fig. 6.16) from a state (place) in the serial process to the parallel process. However, there is no DAG mapping from the parallel process to the serial one, which covers all the potential cases (specifically, a token before *B* and a token after *C*).

Trying to make the latter change (from parallel to serial) would create the "dynamic change bug". When no transition is available, the solution proposed is either not implementing the change, or wait until the process reaches a section where a transition is applicable (Rinderle et al. 2004; Reichert et al. 2005).

The dynamic process change examples using net diagram were addressed in Reichert et al. (2005). Petri net implementations are depicted in Fig. 6.17. The initial process has a parallel execution (using Split-And and Join-And logic). The required process change, adding activity *G* is depicted in (b). Fig. 6.17c represents a case where the change requirement was assigned after the process has passed the zone of change (i.e., could not be implemented). Such change type was referred in Rinderle et al. (2004) as changing the history. In an acyclic process without iterations such change cannot be implemented, and therefore in Reichert et al. (2005) it

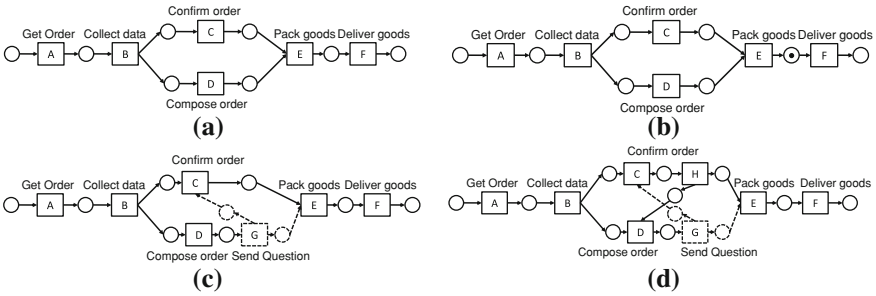


Fig. 6.17 Change in a parallel process **a** Initial Process **b** Required process modification **c** Change of history **d** Structural conflict

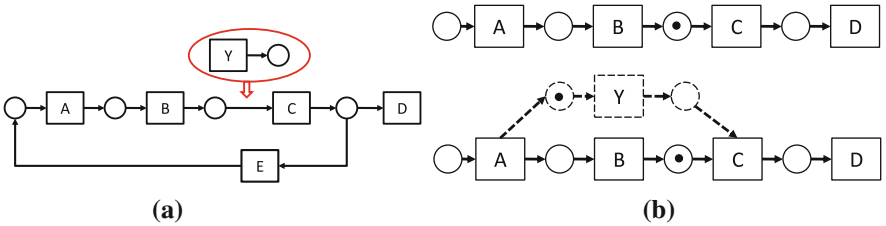


Fig. 6.18 Dynamic process changes **a** Loop tolerance **b** Marking issues

is reported as not compliant to the required change. In (d), the process has an ad-hoc change (*H*), such that implementing the required process modification would create a deadlock cycle, and therefore the case is reported as non compliant due to structural conflict. Note: the non-compliance conclusion of the article actually depends on the implementation of the process scheme; a sound implementation of such cycle (contribution of the current research) was presented in Fig. 6.8b.

Other dynamic process change types identified in (Rinderle et al. 2004) were loop tolerance (adding a new activity during a loop), Fig. 6.18a; and marking issues, Fig. 6.18b. In a Petri net, adding a parallel path after the process has passed the split required adding tokens to the process marking (before *Y*).

Therefore, the properties of a valid transformation are required. An algorithm for finding valid transformation (if such exist) was proposed with order of complexity being $O(n^4(n!)^2)$, where n is the number of workflow nodes (van der Aalst 2001).

6.7 Formal Process Definitions

Most of the following formal definitions of the WF-nets, if not otherwise stated, were adapted with minor modifications from (van der Aalst and van Hee 2002).

Readers who are less interested in the formal definition can skip this section.

Definition 1 (*Petri net*) A Petri net is a triple $PN = (P, T, F)$.

(1) P and T are finite disjoin sets of places and transitions, respectively ($P \cap T = \phi$).
(2) F is a set of arcs (flow relations), $F \subseteq (P \times T) \cup (T \times P)$.
(3) A place p is called an input place of a transition $t \in T$, if and only if (iff) a directed arc from p to t exists. The set of input places to t is marked $\bullet t = \{p \in \mathrm{P} \mid (p, t) \in F\}$. The output places set of transition t, $t\bullet = \{p \in P \mid (t, p) \in F\}$. The notations $\bullet p$ and $p\bullet$ are correspondingly the pre-set and post-set of transitions.
(4) A pair (PN, M) of a Petri net PN, and a start marking M is called a net system. M is the state, or marking of the Petri net, i.e., the distribution of tokens over places. $M(p)$ is the number of tokens in place p.
(5) For two markings: M_1 and M_2, $M_1 \leq M_2$ iff $\forall p \in P$, $M_1(p) \leq M_2(p)$.
(6) A marking M changes by firing a transition t, which may fire only if it is enabled.
(7) A transition t is enabled in marking M, (written $M \xrightarrow{t}$), iff every input place of t contains the required number of tokens.
(8) If a transition t is enabled in marking M, it may fire. On firing, tokens are consumed (removed) from every input place and tokens are produced (added) to every output place.[5] The number of tokens consumed or produced is defined by the edge capacity (arc weight).
In the current research we address the case of edges having capacity $= 1$ for all edges, so each input place should have at least one token. $\forall p \in \{\bullet t\}$, $M(p) \geq 1$. According to our assumption, one token is removed from each input place and one token added to each output place. Given a Petri net and a marking M, we have the following notations:
(9) $M_1 \xrightarrow{t} M_2$: transition t is enabled in M_1; its firing in M_1 results in M_2.
(10) $M_1 \xrightarrow{\sigma} M_n$: the firing sequence $\sigma = (t_1\, t_2\, t_3 \ldots t_{n-1})$ leads from marking M_1 to marking M_n, through markings $M_2, M_3, \ldots, M_{n-1}$.
(11) A marking M_n is reachable from M_1 iff a firing sequence σ exists.
(12) The empty sequence ε is enabled by any marking M.

Definition 2 (*Path in PN, Elementary path*)

(1) In the Petri net PN, a path C from a node (place or transition) x_0 to a node x_m is a non-empty sequence $(x_1, \ldots, x_m)$ such that $(x_i, x_{i+1}) \in F$ for $1 \leq i \leq m - 1$.
(2) C is an elementary path iff $\forall x_i, x_j\, i \neq j \Rightarrow x_i \neq x_j$, (i.e., nodes do not repeat).
(3) Alphabet operator α is defined by $\alpha(C) = \{x_1, \ldots, x_m\}$, where $C = (x_1, \ldots, x_m)$ (Ping et al. 2004).

[5] The use of weighted arcs (i.e., arcs which transfer more then one token), is not addressed in this research, and is not required for the following proofs.

Definition 3 (*Strongly connected*) A Petri net *PN* is strongly connected iff for every pair of nodes x and y, there is a path C leading from x to y.

Definition 4 (*State machine*) A Petri net *PN* is a state machine iff each transition has exactly one input place and one output place. $\forall t \in T\ |t\bullet| = 1$ and $|\bullet t| = 1$.

There should be only one token at the initial state (and therefore at all times); thus, the process state is equivalent to the marking by that token.

Definition 5 (*Well-handled*) A Petri net *PN* is well-handled, iff for any pair of nodes x, y such that one node is a place and the other is a transition, and for any pair of elementary paths C_1 and C_2 leading from x to y. $\alpha(C_1) \cap \alpha(C_2) = \{x, y\} \Rightarrow C_1 = C_2$.

Rephrasing the definition, if there are two paths between distinct type nodes, there should be an additional node (other than x or y) that belong to both paths, thus the process is well-handled.

Definition 6 (*WF-net*) A Petri net *PN* is a WF-net, iff:

i. PN has two special places i (the starting place, or source), and o (the termination place, or sink). Place $i \in P$ satisfies $\bullet i = \varnothing$; and place $o \in$ P satisfies $o\bullet = \varnothing$; and
ii. every node $x \in P \cup T$ is on a path from start place i to termination place o.

This definition implies a project-like structure with defined start and completion. The requirement that transitions (activities) should be on a path from start to end implies process completeness, no loose ends.

Definition 7 (*extended WF-net*) Given a WF-net *PN*, the extended Petri net of PN, $PN^* = (P^*, T^*, F^*)$, is defined by adding a transition $t^* \notin T$ that connects place o with place i as follows: $PN^* = (P, T \cup \{t^*\}, F \cup \{(o, t^*), (t^*, i)\})$.

Proposition 1 (WF-net properties) *If PN is a WF-net with starting place i, and termination place o, then:*

(1) *i is the only starting place. For any place* $\forall p \in P$: $\bullet p \neq \varnothing$ *or* $p = i$;
(2) *o is the only termination place. For any place* $\forall p \in P$: $p\bullet \neq \varnothing$ *or* $p = o$;
(3) *the extended Petri net PN* is strongly connected; and*
(4) *if Petri net PN has starting place i and termination place o, and its extended Petri net PN* is strongly connected, then every node* $x \in P \cup T$ *is on a path from start place i to termination place o in PN; and PN is a WF-net. Proof: in van der Aalst* (1998).

Definition 8 (*Special states*) Two special states (markings) are defined for a Petri net *PN*: starting state S (with tokens only at place i) is a special initial state. The End state E (with tokens only at place o), is a special termination state.

(1) S_K is the starting state of *PN* with $k > 0$ tokens iff $\forall p \in P,\ p \neq i,\ S_K(p) = 0 \wedge S_K(i) = k$.
(2) E_K is the end state of *PN* with $k > 0$ tokens iff $\forall p \in P,\ p \neq o,\ E_K(p) = 0 \wedge E_K(o) = k$.

For convenience, the default marking used is $S = S_1$ and $E = E_1$ (indicating one token) in van Hee et al. (2004).

Definition 9 (*1-Soundness*) A procedure modeled by a WF-net *PN* is sound iff:

i. For every state M reachable from starting state S, by a firing sequence σ_{SM}, there is a firing sequence σ_{ME} leading from state M to state E.

$$\forall_M (S \overset{\sigma_{SM}}{\rightarrow} M) \Rightarrow (M \overset{\sigma_{ME}}{\rightarrow} E)$$

ii. State E (termination) is the only state reachable from S with a token in place o

$$\forall_M (S \overset{\sigma_{SM}}{\rightarrow} M \wedge M \geq E) \Rightarrow (M = E)$$

iii. There are no dead transitions in (*PN; S*). For every transition t there is at least one state M, reachable from the starting state, by sequence σ_{SM}, which enables this transition; and firing transition t in M results in M^*:

$$\forall_{t \in T} \exists_{M\,,M^*} (S \overset{\sigma_{SM}}{\rightarrow} M \overset{t}{\rightarrow} M^*)$$

Identifying design activities with transitions in the context of design processes, and based on the soundness criteria, the extended process properties requirements are:

(1) The first requirement indicates that the process can reach its completion. An assumption is made (van der Aalst 1998) that the process should terminate eventually.
(2) Once the terminal state was reached, there are no activities with unmet pre-conditions (there are no tokens in places other than the termination place).
(3) There are no 'dead' activities i.e., activities that could not be performed. Additionally, there are no activities that did not perform.

Definition 10 (*well-structured*) A WF-net is well-structured iff *PN** (the short-circuited net) is well-handled.

Hierarchical Composition of WF-nets

The following definitions were adapted from (Ping et al. 2004), based on the work in (van Hee et al. 2003).

Definition 11 (*K-sound*) A WF-net PN is K-sound for a natural number k iff:

i. $\forall M (S_K \overset{\sigma_{SM}}{\longrightarrow} M) \Rightarrow (M \overset{\sigma_{ME}}{\longrightarrow} E_K)$
ii. $\forall M (S_K \overset{\sigma_{SM}}{\longrightarrow} M \,\Lambda\, M \geq E_K) \Rightarrow (M = E_K)$
iii. $\forall t \in T \; \exists_{M,\,M^*} \; S_K \overset{\sigma_{SM}}{\longrightarrow} M \overset{t}{\longrightarrow} M^*$

This is an expansion of the soundness criteria in Definition 9 (*1*-sound). It also aligns the definition proposed in van Hee et al. (2003), with Definition 9.

Definition 12 (*G-sound: General soundness*) A WF-net *PN* is *G*-sound iff it is *K*-sound for every natural number $k > 0$. The term *G*-sound in Definition 12 replaces the term *Sound* used in (van Hee et al. 2003); and the term *1*-sound replaces the term *Sound* used in van der Aalst and van Hee (2002), for Definition 9.

Definition 13 (*WF-net composition*) Activity refinement (transition refinement in van Hee et al. 2003) is the hierarchical concept of replacing an activity by a sub-process. Let $PN_1 = (P_1, T_1, F_1)$, $PN_2 = (P_2, T_2, F_2)$ be two WF-nets such that the start and termination places in PN_2 are i_2 and o_2 respectively; $T_1 \cap T_2 = \varnothing$, $P_1 \cap P_2 = \varnothing$, $t_1 \in T_1$ and $t_a, t_b \notin (T_1 \cup T_2)$. $PN_3 = (P_3, T_3, F_3)$ is the WF-net composition obtained by replacing t_1 in PN_1 by PN_2. $PN_3 = PN_1 \otimes_{t_1} PN_2$ is defined as follows:

i. $P_3 = P_1 \cup P_2$
ii. $T_3 = (T_1 \backslash \{t_1\}) \cup T_2 \cup \{t_a, t_b\}$
iii. $F_3 = \{(x,y) | (x,y) \in F_1 \wedge x \neq t_1 \wedge y \neq t_1\} \cup \{(x,y) | (x,y) \in F_2\}$
$\cup \{(x,t_a) | (x,t_1) \in F_1\} \cup \{(t_b,y) | (t_1,y) \in F_1\} \cup \{(t_a,i_2)\} \cup \{(o_2,t_b)\}$

Proposition 2 (*G*-soundness of WF-net composition) *Let both PN_1 and PN_2 be G-sound WF-nets and $t_1 \in T_1$ be a transition of PN_1, WF-net $PN_3 = PN_1 \otimes_{t_1} PN_2$ is also G-sound. Proof: in (van Hee et al. 2003).*

Proposition 3 (*K*-soundness of WF-net composition) *Let PN_1 be K-sound and PN_2 be G-sound WF-nets and $t_1 \in T_1$ be a transition of PN_1, WF-net $PN_3 = PN_1 \otimes_{t_1} PN_2$ is also K-sound. Proof: in Ping et al. (2004).*

Definition 14 (*WA-WF-net: Well-handled and Acyclic*) A WF-net is a WA-WF-net iff it is well-handled and acyclic.

Definition 15 (*WRI-WF-net: Well-handled with Regular iterations*)
(1) Any WA-WF-net is a WRI-WF-net.
(2) Let $PN_1 = (P_1, T_1, F_1)$, $PN_2 = (P_2, T_2, F_2)$ be two WRI-WF-nets and $t_1 \in T_1$, then $PN_3 = PN_1 \otimes_{t_1} PN_2$ is a WRI-WF-net.
(3) Let $PN_1 = (P_1, T_1, F_1)$, $PN_2 = (P_2, T_2, F_2)$ be two WRI-WF-nets and $t_1 \in T_1$, then $PN_3 = PN_1 \otimes_{t_1} PN_2^*$ is a WRI-WF-net (where PN_2^* is the extended Petri net).
(4) WRI-WF-nets can only be obtained either by (1), (2), or (3).

Theorem 1 *WRI-WF-nets are 1-sound. Proof: in Ping et al. (2004)*

Corollary 1 *A WRI-WF-net is G-sound. Proof: in Ping et al. (2004)*

References

Andersson J (2001) Multi objective optimization in engineering design: applications to fluid power systems. Dissertation. Institute of Technology, Linköpings Universitet, Sweden

Barjis J, Dietz JL (2000) Business process modeling and analysis using GERT networks. In: Filipe J (ed) Enterprise Inf Sys, pp 71–80

Braha D (2002) Partitioning tasks to product development teams. Proceedings of ASME Int Design Eng Technical Conf DETC'02, Montreal, Canada, September 29–October 2

Cho SH, Eppinger SD (2001) Product Development Process Modeling Using Advanced Simulation. ASME Conf on Design Theory and Methodology (DECT 2001/DTM), Pittsburgh, PA, September

Cho SH, Eppinger SD (2005) A simulation-based process model formanaging complex design projects. IEEE Trans on Eng Manag, 52(3):316–328

Dehnert J, van der Aalst WMP (2004) Bridging the gap between business models and workflow specifications. Int J Coop Inf Sys 13(3):289–332

Ellis CA, Keddara K, Rozenberg G (1995) Dynamic change within workflow systems. In: Proceedings of International ACM Conference COOCS 95, Milpitas, CA, pp 10–21

Elmaghraby SE (1995) Activity nets: A guided tour through some recent developments. Eur J Oper Res 82:383–408

Farrell ADH, Sergot MJ, Bartolini C (2007) Formalising workflow: a CCS-inspired characterisation of the YAWL workflow patterns. Group Decision and Negotiation, vol 16. Springer, Berlin, pp 213–254

Gruhn V, Laue R (2006) Complexity Metrics for Business Process Models. In: 9th International Conference on business Inf Sys (BIS 2006). Lecture Notes in Informatics, vol 85. Klagenfurt, Austria, pp 1–12

Heimann P, Joeris G, Krapp C, Westfechtel B (1996) DYNAMITE: Dynamic task nets for software process management. Proceedings 18th International Conference on Software Eng, Berlin, Germany, pp 331–341

Heller M, Westfechtel B (2003) Dynamic project and workflow management for design processes in chemical engineering. Proceedings 8th International Conference on Process Sys Eng (PSE 2003), Kumming, China

Jun HB, Park JY, Suh WH (2006) Lead time estimation method for complex product development process. Concurr Eng Res Appl 14(4):313–328

Karniel A, Reich Y (2007a) Managing dynamic new product development processes. Proceedings of the 17th Annual International Symposium of The Int Council on Sys Eng INCOSE'07, San Diego, CA, June

Karniel A, Reich Y (2007b) A coherent interpretation of DSM plan for PDP simulation. In: Proceedings of the International Conference on Eng Design, ICED 07, Paris, August

Karniel A, Reich Y (2009) From DSM based planning to design process simulation: a review of process scheme logic verification issues. IEEE Trans Eng Manag 56(4):636–649

Mangan P, Sadiq S (2003) A constraint specification approach to building flexible workflows. J Res Pract Inf Technol 35(1):21–39

Milner R (1980) A Calculus of Communicating Systems. Lecture Notes in Computer Science, vol 92. Springer, Berlin

Milner R (1999) Communicating and Mobil Systems The π–calculus. Cambridge University Press, Cambridge

Neumann K (1990) Stochastic project networks: Temporal analysis scheduling and cost minimization. Springer, Berlin

Ping L, Hao H, Jian L (2004) On 1-soundness and soundness of workflow nets. In: Daniel Moldt (ed) Proc of the 3rd Workshop on Modelling of Objects, Components, and Agents, DAIMI PB, vol 571. Aarhus, Denmark, pp 21–36

Reichert M, Dadam P (1998) ADEPTflex—Supporting dynamic changes of workflows without losing control. J Intell Inf Sys 10:93–129

Reichert M, Rinderle S, Kreher U, Dadam P (2005) Adaptive process management with ADEPT2. 21st International Conference on Data Eng (ICDE'05), pp 1113–1114

Reising W, Rozenberg G (1998) Lectures on Petri nets I: Basic models. Lect Notes Comput Sci 1491:1998

Rinderle S, Reichert M, Dadam P (2004) Correctness criteria for dynamic changes in workflow systems—A survey. Data Knowl Eng 50(1):9–34

Sadiq W, Orlowska ME (1999) Applying graph reduction techniques for identifying structural conflicts in process models. Lect Notes Comp Sci 1626:195–209 Jan

Sadiq W, Orlowska ME (2000) Analyzing process models using graph reduction techniques. Inf Sys 25(2):117–134

Sadiq S, Orlowska ME, Sadiq W, Foulger C (2004) Data flow and validation in workflow modeling. Proceedings of the International Conference in Res and Practice in Inf Technology, ADC'2004, Dunedin, New Zealand

Smith RP, Morrow JA (1999) Product development process modeling. Des Stud 20(3):237–261

Taylor BW, Moore LJ (1980) R&D project planning with Q-GERT network modeling and simulation. Manag Sci 26(1):44–59

van der Aalst WMP (1998) The application of Petri nets to workflow management. J Circuits Sys Comput 8(1):21–66

van der Aalst WMP (1999) Formalization and verification of event-driven process chains. Inf Softw Technol 41:639–650

van der Aalst WMP (2000) Finding control-flow errors using Petri net based techniques. In: vd Aalst W, Desel J, Oberweis A (eds) Bus Process Manag, Lecture Notes in Computer Science, vol 1806. Springer-Verlag, Berlin, pp 161–183

van der Aalst WMP (2001) Exterminating the dynamic change bug: a concrete approach to support workflow change. Inf Sys Frontiers 3(3):297–317

van der Aalst WMP, ter Hofstede AHM, Kiepuszewski B, Barros AP (2003a) Workflow patterns. Distrib Parallel Dat 14(1):5–51

van der Aalst WMP, ter Hofstede AHM, Weske M (2003b) Business process management: A survey. Int conf on Bus Process Manag, BPM 2003. Lecture Notes in Computer Science, vol 2678. Springer, Eindhoven, pp 1–12

van der Aalst WMP, ter Hofstede AHM (2005) YAWL: Yet another workflow language. Inf Sys 30(4):245–275

van der Aalst WMP, van Hee K (2002) Workflow Management Models Methods and Systems. MIT Press, Cambridge

van Hee K, Sidorova N, Voorhoeve M (2003) Soundness and separability of workflow nets in the stepwise refinement approach. In: van der Aalst WMP, Best E (eds) Application and Theory of Petri Nets 2003, Lecture Notes in Computer Science, vol 2678. Springer, Berlin, pp 337–356

van Hee K, Sidorova N, Voorhoeve M (2004) Generalised Soundness of workflow nets is decidable. In: Cortadella J, Reisig W (eds) Application and Theory of Petri Nets 2004, Lecture Notes in Computer Science, vol 3099. pp 197–216

Verbeek HMW, Basten T, van der Aalst WMP (2001) Diagnosing workflow processes using Woflan. The Computer J 44(4):246–279

Westfechtel B (1999) Models and tools for managing development processes. Lecture Notes in Computer Science, vol 1646. Springer, Berlin

Zanddizari M (2006) Set confidence interval for customer order cycle time in supply chain using GERT method. In: Proceedings of the 17th IASTED international Conference on Modelling and Simulation, Canada, pp 213–218

Chapter 7
Logic Issues of DSM-Based Processes

In this section, we present a comparative review of DSM-based process schemes in DSM literature (Karniel and Reich 2009). The survey proposed a classification model that segregated the different DSM-based approaches, and is further used to analyze the verification consequences of the logic implementation practiced in each case. Two aspects of process logic verification are discussed: (a) Presentation and modeling of process logic, and (b) Verification of process logic.

No previous DSM-based related work has comprehensively discussed logic verification issues; thus, the formulation and examples developed for this comparison are contribution of the current research to the analysis of the DSM-based literature. The definitions and analysis tools were further used in the development of the DnPDP framework (Karniel and Reich 2007) presented in Chap. 8.

7.1 Presenting Process Logic in DSM

Presentation of process logic within the DSM is described in (Clarkson and Hamilton 2000; Browning 2001). In both cases, different symbols are used to indicate AND/OR logic. The presented DSM models had no distinction between output logic and input logic. The following example (Karniel and Reich 2009) demonstrates the inability of a single DSM to represent both input logic and output logic at the same time. Furthermore, Clarkson and Hamilton (2000) indicated that such marking created confusion when trying to reorder the matrix. In both cases, this line of research was not continued.

Assuming four activities, *W*, *X*, *Y*, and *Z*, with DSM markings labeled 'a' to 'd', as depicted in Fig. 7.1a. The Output Logic of activities *W* and *X* is defined as Split-And (to both *Y* and *Z)*. The Input logic for *Y* is Join-And (from *W* and *X*) and the input logic of Z is Join-Or (from *W* and *X*).

A. Karniel and Y. Reich, *Managing the Dynamics of New Product Development Processes,* DOI: 10.1007/978-0-85729-570-5_7,

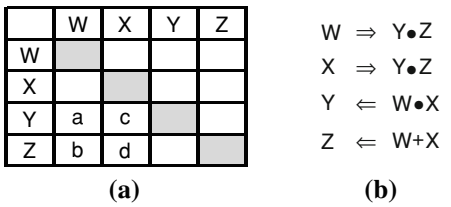


Fig. 7.1 Marking of input logic and output logic

When defining the symbols: Split (⇒); Join (⇐); And (•); and Or (+); the above logic descriptions are formulated by symbolic representation shown in Fig. 7.1b.

Actually, there is no marking scheme that can convey such logic in one matrix. Since *W* and *X* have the same output logic, the markings in their columns should be the same, $a = c$ and $b = d$. The input markings (in the row) of *Y* should be equivalent to the column of *W* (i.e., $a = a$, $b = c$) and the column of *X* (similarly, $a = d$), thus we get $a = b = c = d$. On the other hand, they should differ from those in *Z*, i.e., $a \neq b$ or $c \neq d$. Overall, the requirements result in a contradiction.

As further detailed in Sect. 7.3, Input Logic and Output Logic are typically different; therefore trying to express the output logic of A and the input logic of B on the link from A to B is bound to lead to definition problems. Separating the logic presentation to two matrices, Input Logic and Output Logic will overcome the above problem. The different logic matrices proposed in (Lee and Suh 2006), can present the logic, however their integration into one matrix can lead to a problem as in the above example.

Furthermore, if the logic is defined prior to rearrangement of the matrix, improper process may result after arrangement. For example, assigning a Join-And input logic of forward links to *Y* (links from *X* and *Z* in Fig. 3.4c). A DSM rearrangement might cause this logic to become a Join-And of a forward link and a feedback link (Fig. 3.4b). In such case, the resulting process is not sound, as activity (*Y*) may become a dead activity (*Z* may send a signal to End without sending a feedback signal), see Fig. 3.4g.

7.2 DSM Limitations

Current DSM algorithms do not address some requirements regarding process logic, which are applicable to design process planning. The logic requirement described in (Abdelsalam and Bao 2006), is to avoid reordering of activities within activity loop if the internal ordering has a specific meaning (e.g., testing should not precede design).

Coates et al. (2003) described complex logic conditions. The process described had design tasks and optimization scheduling activity. Assume that design task *B*, has a precedent design task *A*, Fig. 7.2a. The decision whether to perform the scheduling activity *S* depends on process data, e.g., would the anticipated time reduction due to optimization worth performing that activity. The following logic

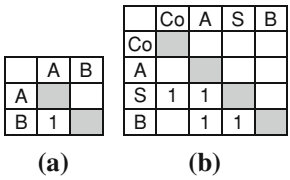


Fig. 7.2 Changing the design process plan at Run Time. **a** S does not activate. **b** S does activate

applies: If scheduling optimization activity S is performed after the completion of task *A*, according to condition Co, then task *B* should wait for completion of S; otherwise, task *B* may start once task *A* has completed. Definitely, such logic cannot be expressed in a DSM. To do so there is a need to switch between two DSM configurations depicted in Fig. 7.2a and b respectively. Actually, the scheduling activity S might be inserted between any two successive design tasks.

7.3 Process Verification Issues

There are several issues related to assigning logic to a process: first, the definition of the process logic; then, the presentation and modeling of process logic; and finally, the verification of the process logic to ensure process correctness. The importance of verifying the process logic manifests in examples where the DSM planning is converted to a logically correct process for specific DSM data; but changes in the DSM data might yield undetermined process scheme. Notwithstanding the importance of process logic verification, it is disregarded in the DSM-based simulation literature.

As illustrated in the following examples (Karniel and Reich 2009a), iterative processes are inherently subject to logic problems. Thus, the process logic should be strictly defined. The examples present logic definition problems that manifest in the process scheme interpretation of the DSM data.

Coupled activities in a Binary DSM , using Join-And logic, represent a deadlock (cf., Fig. 3.2c). Activity *X* waits for an input and a completion signal from activity *Y*. Activity *Y* waits for an input and a completion signal from *X*.

Some sort of separation between forward links logic from feedback links logic is therefore required. Typically, this case is solved by using Join-Or logic, or Join-Xor logic; i.e., wait for forward link signal or feedback link signal. Such input logic definition is sufficient if the process is serialized, i.e., there are no other options. In a serialized process, the result of using Join-Xor is equivalent to using Join-Or.

If the process is progressing in parallel paths, there are multiple input forward links and multiple feedback links, and the required logic is more complicated. Besides the separation between forward and feedback links, the logic of the multiple forward links and feedback links should be defined. For example, Join-And logic may apply for multiple forward links, which implies waiting for all previous activities to complete. In Fig. 3.4c (process in Fig. 3.4h), activity *Y* should wait for *X* and for *Z* to complete. Join-Or logic may apply to multiple feedback

links. In Fig. 3.4e, Join-Or logic indicated that *Y* should iterate if *X* or *Z* have sent feedback signal. Using Join-And logic in the latter case would mean that *Y* iterates only if both have sent feedback signals. Such process scheme definition may violate the second condition of *Soundness*; e.g., *Z* sends feedback signal to *Y,* and *X* sends forward signal to End (Fig. 3.4j).

The Output logic, which defines a decision algorithm, may have many variants: Split-Xor (Exclusive-or) when using a serialized simulation; Split-And of the forward links when using parallel simulation; and Split-Or on the second iteration of a parallel simulation allowing to chose one or more of the linked activities as defined in (Browning and Eppinger 2002).

As discussed, distinction is typically made in the Output logic between forward links and feedback links (e.g., using Split-Xor). Using Split-Or or Split-And logic for that purpose will yield additional interesting requirement options. For example, in the case of three coupled activities in Fig. 3.4d (process in Fig. 3.4i), using Split-Or or Split-And as the Output logic, once *Y* has completed, may result in additional iteration from both *X* and *Z*. Since *Y* Input logic should be Join-Or (being a parallel process), and the duration of *X* and *Z* are different in the general case, *Y* might be requested to iterate again (e.g., signal from *X)* before completing prior iteration (e.g., due to a signal from *Z*).

Such case was previously defined as Lack of Synchronization problem, but is acceptable or considered more effective (Terwiesch and Loch 1999) in the context of design processes; therefore, this case should be addressed. One option is to increase the duration of the current *Y* activity iteration. For example, Cho and Eppinger (2001, 2005) described an algorithm for recalculating activity duration in the case of overlapping activities. Another option might be waiting for the current iteration to complete and performing an additional iteration of *Y*. While the latter option increases the overall process duration in this case, waiting might be beneficial in more complex cases.

The requirement of having Starting state and End state, is implicitly supported by assuming they are the states before the first activity and after the final activity, respectively. An explicit definition of such logic requirement was described in (Abdelsalam and Bao 2006; Brucker et al. 1999). While assuming Begin and End logic activities is obvious for serial processes, their definition is critical in parallel processes for indicating the initiation and termination of parallel activities. The problem of initiating parallel activities is addressed once all potential activities that can start are checked. Yet, termination of parallel activities is not explicitly defined. For specific DSM data, the problem may not occur. However, having different DSM data, or changing the DSM data may result in an undetermined process.

The requirement to perform each activity at least once is necessary for design processes. This property is assured for serial simulations in which, if feedback iteration did not occur, the next activity according to DSM order becomes available once its previous activity has completed at least once (Smith and Eppinger 1997b), or the next activity according to DSM is enforced (Abdelsalam and Bao 2006).

A potential logic problem of undetermined process scheme arises in the Signposting system (Clarkson et al. 2000). A parameter value is the outcome of

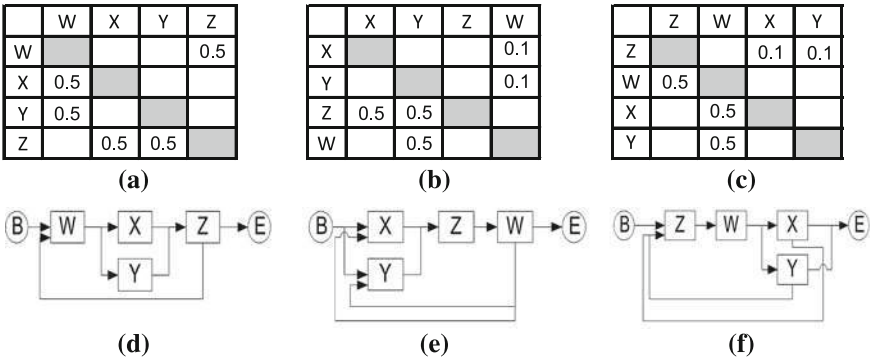


Fig. 7.3 Activity loops defined by DSM coupled activities

design activity. Activity iterations are defined according to the level of confidence in the value assigned to the parameter. Given a required confidence level of a parameter, a confidence map defines the confidence level of its input parameters. Each activity iteration (increasing the confidence level of the outcome parameter) is modeled by a task. The precedence links between required confidence levels define the precedence of tasks. The process serialization in that system has two underlying assumptions: 1) the links between tasks (activity iterations) create at least one continuous route from the first to the last activity; and 2) Each activity has at least one task, which is on some route from begin to end. Since Confidence maps definitions do not ensure these properties, an analysis is required to check the above assumptions before any simulation.

An example of the above considerations is depicted in Fig. 7.3. Four coupled activities (activity loop) are presented; two of them, *X* and *Y,* are parallel. In the three cases presented, the activities have similar links, but with different probability values. Figure 7.3a presents the case of equal probabilities. Recalling that minimum iterations expected to yield optimal processes; the arrangement in (a) represents the minimal feedback ordering (same order as for a Binary DSM). Equivalent result applies if *X* and *Y* are replaced. Setting different probability values can yield the other two combinations as minimum, e.g., using the optimization procedure by Karniel et al. (2005) (Detailed in Sect. 4.2). In (b), the values of links from *W* to both *X* and *Y* were set to $p = 0.1$.

Setting both links from *X* and *Y* to *Z* as $p = 0.1$ would result in the DSM structure in (c).[1] In the first case (a), the activities are connected to the first activity and the last activity; the other two cases introduce parallel initiation (b) and parallel termination (c).

Multi parallel initiation and termination of multiple paths could be handled by the following algorithm: The first step adds Begin and End logic activities,

[1] Other results may occur by setting different values for the feedback and forward coefficients in (Karniel et al. 2005).

which are first and last respectively. The second step connects the Begin activity to all parallel activities that do not have forward link input; and connects all activities with no forward link output to the End activity. Additionally the Output logic of the Begin activity should be Split-And (otherwise parallel processes would not start); and the Input logic of the End activity should be Join-And (otherwise the process may reach a termination state while not all activities have completed, see Fig. 3.4g and j). This algorithm implements requirements that are equivalent to the first part of the *Soundness* criteria. The results of applying the procedure to the three cases are depicted in Fig. 7.3d–f, respectively; and used for interpreting the process schemes in Fig. 3.4f–j. This procedure is further implemented in constructing the DSM net.

An important distinction, not being emphasized in the literature, relates to the actual use of DSM. Two modes of work are established in the copious DSM literature: (a) modeling an existing process by DSM; then using DSM algorithms to improve the process; finally translating the DSM back to a process; and (b) using DSM to model activity interaction (e.g., based on parametric relations), manipulating the DSM, and then translating the results to a process. There is a major distinction between the two cases. In the first case, there will typically be a path from Begin activity to the End activity through all activities, though it is not systematically assured. In the latter case, typically there will not be such path; thus, using verification means is essential (Karniel and Reich 2007).

7.4 Classification of DSM-Based Simulations

Different interpretations of the DSM values for simulation were used in the literature. The current section classifies the approaches, and the next section compares the actual logic interpretation of the approached described in Sect. 5.1. Three main differentiating parameters can be addressed: The information used, i.e., DSM type; the simulation approach: parallel or serialized; and the procedure of choosing the next step: Simulation progress type.

All DSM types were used: Binary DSM[2]; Probability DSM, where probability figures are used for simulation; and direct use of Numeric DSM with explicit number of iterations. The Work Transformation Matrix (WTM), first proposed by Smith and Eppinger (1997a), is used to calculate the remaining work to be done. Combination of Numeric and Probability DSM were used in (Cho and Eppinger 2001, 2005).

Using Probability DSM for estimating iteration probabilities has two main challenges: (a) Estimating the probability values; and (b) Interpreting the figures at simulation time. The use of Probability DSM varies. Sered and Reich (2006) used

[2] The Binary DSM itself cannot be directly used as a probability DSM since feedback iterative links with probability $p = 1$, would yield infinite loops; in general, feedback probabilities should always be less than one.

the values directly, indicating the probability to proceed from activity to another (either forward or feedback). Smith and Eppinger (1997b) used the feedback probabilities directly and have a more complex derivation of forward probabilities due to process serialization. Browning and Eppinger (2002) used the feedback probabilities directly and the forward probabilities were used directly only at subsequent iteration steps.

Three main simulation progress methods (i.e., choice of next activity or activities) were utilized: Deterministic, Markov chain, and Stochastic (Monte-Carlo) simulations. The deterministic simulation approach indicates that the process progress is fully defined by its DSM structure. Markov-chain processes are based on probabilistic progress from one activity to the other, where the selection of the next state depends only on the current state. The choice might be done within parallel (multi path) or serial activities. Monte-Carlo process is stochastically choosing one or many activities (in parallel) of all the applicable activities. Unlike the Markov-chain, the choice might depend on previous history.

A distinction should be made between the choice of the process-progress simulation method and stochastic (Monte Carlo) choice of other simulation parameters. Additional parameters that contribute to the process variability include: activity duration (fixed, changing according to learning curve, or stochastic), and changes in interdependency values (binary, adding or removing interdependencies, or changes in magnitude). The impact of such parameters is added to the effect of the changes in the process structure. The inclusion of such changes further complicates the system behavior because their effect is highly process scheme dependent (Huberman and Wilkinson 2005).

Table 7.1 summarizes the classification of surveyed DSM-based simulation tools according to the above parameters. The selected articles fully describe the implementation of the process logic for the given DSM structure. Additional publications describe simulation results but do not describe the process logic implementation.

Two articles (Smith and Eppinger 1997a, b) do not describe simulations but do describe the process logic to be implemented. In addition, the logic defined in (Smith and Eppinger 1997b), is repeated in (Huberman and Wilkinson 2005; Yassine et al. 2003), and used in (Cronemyr et al. 2001), for simulating process improvements.

Since the following simulation-type classification refers only to the process progress parameter, it may differ from the classification of the simulation processes as described by the authors.

7.5 Logic Comparison of DSM-Based Simulations

In many of the articles, the underlying simulation rules are not explicit. The simulation processes, which were described in Table 7.1, utilize different type of logic constructs. The differences are summarized in Table 7.2. The indications include references to potential simulation obstacles.

Table 7.1 DSM based simulations mapping

Source	Simulation progress method	Process type	DSM type
Abdelsalam and Bao (2006)	Deterministic	Serialized	Numeric
Smith and Eppinger (1997b)	Deterministic	Fully parallel (Coupled activities)	Numeric (to WTM)
Huberman and Wilkinson (2005)	Deterministic	Fully parallel (Coupled activities)	Numeric (to WTM)
Yassine et al. (2003)	Deterministic	Fully parallel (Coupled activities)	Numeric (to WTM)
Choo et al. (2004)	Deterministic	Serialized blocks; Parallel activities within block	Numeric
Smith and Eppinger (1997a)	Markov chain	Serialized	Probability
Sered and Reich (2006)	Markov chain random walk	Serialized (Coupled activities)	Numeric to probability
Melo and Clarkson (2001)	Markov chain	Serial choice (Parallel)	Binary, Triangular
Coates et al. (2003)	Monte Carlo equivalence	Parallel	Binary, Triangular +Logic
Lévárdy and Browning (2005)	Similar to markov chain	Serial choice (Parallel)	Numeric (Symbols)
Browning and Eppinger (2002)	Monte Carlo	Parallel	Probability
Yassine (2007)	Monte Carlo	Parallel	Numeric to probability
Cho and Eppinger (2001, 2005)	Monte Carlo equivalence	Parallel	Probability (+ Numeric overlap)

Since process progress logic is the only issue being compared, the comparison is conducted subject to assuming deterministic activity duration. It is also assumed that the DSM was already reordered according to the applicable algorithm.

The simulation logic of the presented approaches is compared according to the following logic issues presented in Sect. 7.3. Parallel Start (**PS**) indicating the initiation of parallel activities; Parallel End (**PE**) indicating the logic required for completing parallel activities (e.g., Fig. 7.3c); Input Logic in case of Multiple Forward links (***IL* MF**); Input Logic in case of Multiple Iteration[3] (feedback) links (***IL* MI**); The separation of Forward links from Iteration (feedback) links at Input Logic (***IL* F/I**) required for preventing deadlocks (cf. Sect. 6.5); Output Logic in case of Multiple Forward links (***OL* MF**); Output Logic in case of Multiple Iteration (feedback) links (***OL* MI**); and Output Logic separation of Forward and Iteration (Feedback) links (***OL* F/I**).

[3] The term Iteration link temporarily replaces the term Feedback link, for distinguishing the initials from Forward link.

Table 7.2 Mapping the DSM-based Logic

Source	PS	PE	IL MF	IL MI	IL F/I	IL formulation IL ⇐	OL MF	OL MI	OL F/I	OL formulation OL ⇒
Abdelsalam and Bao (2006)	N/A	N/A	J-X	J-X	J-X	$(\otimes(F_i)) \oplus (\otimes(I_i))$	N/A (d)	S-X (f)	S-X	$(\otimes(I_i)) \oplus F_{next}$ (g)
Smith and Eppinger (1997b); Huberman and Wilkinson (2005); Yassine et al (2003)	S-A	J-A	J-O	J-O	J-O	$(\sum(F_i)) + (\sum(I_i))$	S-A	S-A	S-A	$(\prod(I_i)) \bullet (\prod(F_i))$
Choo et al (2004)	S-A	J-A	J-A	N/A (c)	N/A (c)	$\prod(F_i)$	S-A	N/A (c)	N/A (c)	$\prod(F_i)$
Smith and Eppinger (1997a)	N/A	N/A	J-X	J-X	J-X	$(\otimes(F_i)) \oplus (\otimes(I_i))$	S-X (e)	S-X	S-X	$(\otimes(I_i)) \oplus (\otimes(F_i)) \oplus F_{ready}$
Sered and Reich (2006)	N/A	N/A	J-X	J-X	J-X	$(\otimes(F_i)) \oplus (\otimes(I_i))$	S-X	S-X	S-X	$(\otimes(I_i)) \oplus (\otimes(F_i))$
Melo and Clarkson (2001)	Imp (a)	Imp (b)	J-A	N/A (c)	N/A (c)	$\prod(F_i)$	S-A	N/A (c)	N/A (c)	$\prod(F_i)$
Coates et al (2003)	S-A	J-A	J-A	N/A (c)	N/A (c)	$\prod(F_i)$	S-A	N/A (c)	N/A (c)	$\prod(F_i)$
Lévárdy and Browning (2005)	Imp (a)	Imp (b)	J-A	J-O	J-X	$(\prod(F_i)) \oplus (\sum(I_i))$	S-A	S-X	S-X	$(\otimes(I_i)) \oplus (\prod(F_i))$
Browning and Eppinger (2002); Yassine (2007)	S-A	Imp	J-A	J-O	J-X	$(\prod(F_i)) \oplus (\sum(I_i))$	S-A	S-O	S-X	$(\sum(I_i)) \oplus (\prod(F_i))$
Cho and Eppinger (2001, 2005)	S-A	J-A	J-A	J-O	J-O	$(\prod(F_i)) + (\sum(I_i))$	S-A	S-O	S-X	$(\sum(I_i)) \oplus (\prod(F_i))$

Once the logic parameters are defined, a symbolic formulation of the logic can be generated, and is expressed by the following symbols:

Logic indication: Split ($\Rightarrow$) for Output logic; Join ($\Leftarrow$) for Input logic.
Logic operations: + (OR); • (AND); $\oplus$ (XOR, Exclusive-Or);
Short form of multi-variable logic operations, where A_i represents a link signal, which can be a forward link F_i or Iteration (feedback) link I_i:
Multiple Or: $\sum(A_i) = A_1 + A_2 + \cdots + A_n$;
Multiple And: $\prod(A_i) = A_1 \bullet A_2 \bullet \cdots \bullet A_n$; and
Multiple Xor: $\otimes(A_i) = A_1 \oplus A_2 \oplus \cdots \oplus A_n$.

The Input logic formulation expresses the pre-conditions of the activity, and has three distinct parts: logic of forward links to the activity from upstream activities, logic of iteration (feedback) links from downstream activities, and the logic, which might differentiate between these types.

Notations (for Table 4.2):
Imp: Implicit; J-A: Join-And; J-O: Join-Or; J-X: Join-Xor; S-A: Split-And; S-O: Split-Or; S-X: Split-Xor; N/A: Not Applicable;

Logic formulation:
F_i: forward links; I_i: iteration (feedback) links; Split ($\Rightarrow$); Join ($\Leftarrow$);
+ (OR); (AND); $\oplus$ (XOR);
$\sum(A_i) = A_1 + A_2 + \cdots + A_n$; $\prod(A_i) = A_1 \bullet A_2 \bullet \cdots \bullet A_n \otimes(A_i) = A_1 \oplus A_2 \oplus \cdots \oplus A_n$

Notes

a Parallel activities are implicitly started in parallel using a serialized check.

b Parallel completion is undefined; but could be implicitly assumed. Problem could be avoided by linking parallel activities to the last activity.

c No iteration links. In Choo et al. (2004), either tearing or making the activities parallel, but without run time iterations.

d Next activity only (other potential activities with forward links are disregarded).

e Forward links are available at stages (e.g., Fig. 1.2e); *X* can advance from *Y* only after *Z* is complete.

f The case of multiple iterations is not strictly defined; consequently, the simulation might be undetermined in such cases.

g The process may proceed only to the next activity (F_{next}).

h The process may proceed to the ready activity (F_{ready}) (see e); or any activity that has been completed once, or it may iterate.

For example, $IL \Leftarrow (\prod(F_i)) \oplus (\sum(I_i))$(Browning and Eppinger 2002; Lévárdy and Browning 2005) indicates that either the activity waits for all previous activities to complete (Join-And), or (exclusive) starts once any of the downstream activities has completed and sent iteration signal; however, it does not accept both. In (Browning and Eppinger 2002), it is indicated that if a downstream activity *B* is active while and upstream activity *A* (which is linked to *B*) has started to iterate, the downstream activity *B* stops; i.e., the upstream activity *A* will not get an additional iteration (feedback) signal from *B* while it operates.

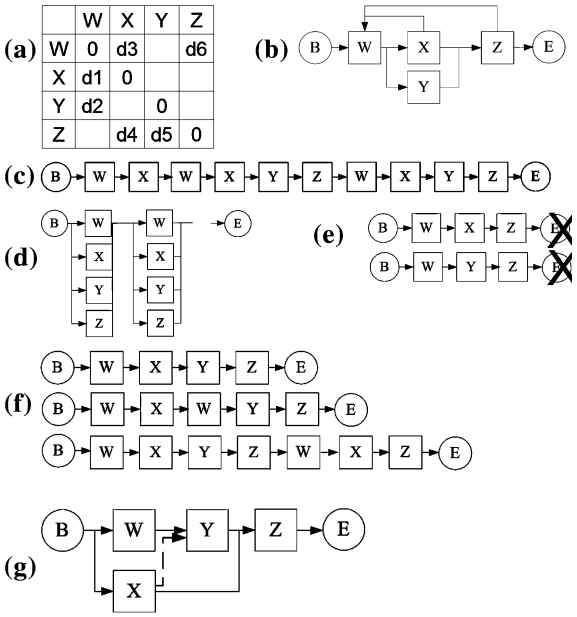


Fig. 7.4 Run time examples of logic implementation

In Cho and Eppinger (2005), an activity can get signals both from upstream or downstream activities. A second forward link signal to B may follow an iteration of A. $IL \Leftarrow (\prod (F_i)) + (\sum (I_i))$

The output logic indicates a decision process, i.e., sending a signal that the current activity has completed. For example, $OL \Rightarrow (\sum (I_i)) \oplus (\prod (F_i))$ in (Browning and Eppinger 2002; Cho and Eppinger 2005), implies that a signal is either sent through an iteration link (or links) or (exclusive) signals are sent to all forward links. Being a decision procedure, the order of checks might be significant. The order is significant for parallel activities, and is not significant for probabilistic choice of serial processes (where the sum of probabilities must be one).

The examples in Table 7.2 are roughly ordered in an increasing potential of run time (RT) process variety. The process logic circumscribes the diversity of process scheme under the assumption of stochastic activity duration, i.e., how many different RT processes could be created for a given DSM by using the defined logic. The following sections demonstrate the implementation of the different logic options.

Figure 7.4 demonstrates some implementation differences. A DSM structure depicted in Fig. 7.4a is represented by the process scheme in Fig. 7.4b.

(1) The deterministic definition in (Abdelsalam and Bao 2006) would always yield the same RT process scheme (same order of activities) for a given DSM. Using integer feedback values $d3 = d6 = 1$ (indicating one iteration), we get the RT process in Fig. 7.4c.

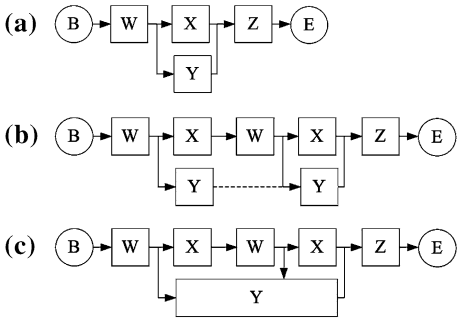


Fig. 7.5 Run time implementation of parallel process logic

(2) Implementing parallel execution of all coupled activities according to the logic in (Smith and Eppinger 1997a; Huberman and Wilkinson 2005; Yassine et al. 2003), referred to as design block (DB) in (Karniel and Reich 2007), yields a RT process with potential multiple iterations of all the activities. Execution of all activities and the first iteration are presented in Fig. 7.4d.

(3) The serialized logic of (Smith and Eppinger 1997b) may result in many potential RT processes; three examples are depicted in Fig. 7.4f. However, any of the process examples in Fig. 7.4e could not be a result of the process logic due to the progress limitation of forwarding only one activity at a time (see Table 7.2, note e).

(4) If we set $d1 = d3 = 3$ and all others values $di = 1$ in Fig. 7.4a, then according to the logic in (Choo et al. 2004), the feedback link $d6$ would be torn; and the resulting DSM would include a sub-block of the coupled activities W and X. In the RT process (Fig. 7.4g), these parallel activities do not iterate. The dashed link indicates the option of parallel completion (before Y could begin). However, if all $di = 1$ as in example (2), all activities are coupled, and the resulting RT would be the same as in Fig. 7.4d.

(5) Using the DSM entries in (4), but with parallel process implementation (e.g., Lévárdy and Browning 2005; Coates et al. 2003; Melo and Clarkson 2001), can yield the RT process in Fig. 7.5a, and iterations of the process following the feedback link from Z activity.

A more interesting case is execution of the feedback link from activity X in Fig. 7.4b. The RT process (Fig. 7.5b) is the result of such iteration by the logic implemented by (Browning and Eppinger 2002; Cho and Eppinger 2005; Yassine 2007). The logic defined in (Cho and Eppinger 2005) would yield many different RT process schemes depending on the actual duration of the activities, due to the concurrent execution paths and the impact of logic input, which may accept signals from both feedback and forward activities. For example, it could result in increasing the time of activity Y due to additional input from W in Fig. 7.5c.

However, the order of presentation in the table does not indicate any preference of one logic definition over another. The different logic definitions may represent different business cases. The case of applying parallel-logic for coupled activities (Smith and Eppinger 1997a; Huberman and Wilkinson 2005; Yassine et al. 2003)

implies that all activities start together and the process completes once all have completed. Such logic is unacceptable once there is defined serialization of the coupled activities, e.g., in the case of testing activity (Lévárdy and Browning 2005), or specific serialization requirement (Abdelsalam and Bao 2006). Another example is the use of Join-And, which is the typical input logic used for forward links (from upstream activities) in case of multiple process paths. However, if coupled activities are to be processed in parallel, such logic is inapplicable; and Join-Or logic must be used.

References

Abdelsalam HME, Bao HP (2006) A simulation-based optimization framework for product development cycle time reduction. IEEE Trans Eng Manag 53(1):69–85

Browning TR (2001) Applying the design structure matrix system to decomposition and integration problems: A review and new directions. IEEE Trans Eng Manag 48:292–306

Browning TR, Eppinger SD (2002) Modeling impacts of process architecture on cost and schedule risk in product development. IEEE Trans Eng Manag 49(4):428–442

Brucker P, Drexl A, Möhring R, Neumann K, Pesch E (1999) Resource-constrained project scheduling: Notation, classification, models and methods. Eur J Oper Res 112:3–41

Cho SH, Eppinger SD (2001) Product Development Process Modeling Using Advanced Simulation. ASME Conf on Design Theory and Methodology (DECT 2001/DTM), Pittsburgh, PA, September

Cho SH, Eppinger SD (2005) A simulation-based process model for managing complex design projects. IEEE Trans Eng Manag 52(3):316–328

Choo HJ, Hammond J, Tommelein ID, Austin SA, Ballard G (2004) DePlan: A tool for integrated design management. Autom in Constr (13):313–326

Clarkson PJ, Hamilton JR (2000) Signposting, A parameter-driven task-based model of the design process. Res Eng Des 12(1):18–38

Clarkson PJ, Melo AF, Eckert CM (2000) Visualization of routes in design process planning. In: Proceedings of the 4th International Conference on Inf Visualisation (IV2000), IEEE Comput Society, 155–164, London

Coates G, Duffy AHB, Whitfield I, Hills W (2003) An integrated agent-oriented approach to real-time operational design coordination. Artif Intell Eng Des, Anal Manuf 17:287–311

Cronemyr P, Rönnbäck AO, Eppinger SD (2001) A decision support tool for predicting the impact of development process improvements. J Eng Des 12(3):177–199

Huberman BA, Wilkinson DM (2005) Performance variability and project dynamics. Comput & Math Organ Theory 11:307–332

Karniel A, Belsky Y, Reich Y (2005) Decomposing the problem of constrained surface fitting in reverse engineering. Comput-Aided Des 37:399–417

Karniel A, Reich Y (2007) Managing dynamic new product development processes. Proc of the 17th Annual Int Symposium of The Int Council on Sys Eng INCOSE'07, San Diego, California, June

Karniel A, Reich Y (2009) From DSM based planning to design process simulation: A review of process scheme logic verification issues. IEEE Trans Eng Manag 56(4):636–649

Lee H, Suh HW (2006) Workflow structuring and reengineering method for design process. Comput Ind Eng 51:698–714

Lévárdy V, Browning TR (2005) Adaptive test process—Designing a project plan that adapts to the state of a project. 15th Annual Int Symposium of the Int Council on Sys Eng (INCOSE), July 2005

Melo AF, Clarkson PJ (2001) Design process planning using a state-action model. 13th International Conference Eng Design (ICED 01), Glasgow

Sered Y, Reich Y (2006) Standardization and modularization driven by minimizing overall process effort. Comput-Aided Des 38(5):405–416

Smith RP, Eppinger SD (1997a) Identifying controlling features of engineering design iteration. Manag Sci 43(3):276–293

Smith RP, Eppinger SD (1997b) A predictive model of sequential iteration in engineering design. Manag Sci 43(8):1104–1120

Terwiesch C, Loch CH (1999) Measuring the effectiveness of overlapping development activities. Manag Sci 45(4):455–465

Yassine A (2007) Investigating product development process reliability and robustness using simulation. J Eng Des 18(6):545–561

Yassine A, Joglekar N, Braha D, Eppinger SD, Whitney D (2003) Information hiding in product development: The design churn effect. Res Eng Des 14(3):131–144

Part II
The Integrated Model Dynamic New-Product Developmment Process

Chapter 8
Dynamic New-Product Design Process

This chapter describes the concepts, methods, and enhancements that compose this research. Due to the variety of subjects, simple examples are added to each issue, as well as simple integrated examples. The implementation of a fully integrated example is presented in Chap. 1.

Section 8.1 presents the main DnPDP framework concepts required for managing dynamic process changes of NPD processes. The proposed integrated framework (Karniel and Reich 2007b) is presented as a closed-loop process (Sect. 8.2), extended to a decision-making process by additional control loop (Sect 8.3). The computational view of the framework separates it into process steps forming the controller Meta-process (Sect. 8.4). The controller accepts product data as feedback of the design process. The (changing) product data is converted to a process scheme. The main conversion steps go from product data to Probability DSM, DSM then to Design Process Matrix (DPM), and then to process scheme. These steps, defined as the planning cycle, are overviewed in Sect. 8.5. The evolving process models are discussed in Sect. 8.6.

8.1 Model Description

Clearly, the product structure is not completely defined at the early design stage of NPD processes; it is evolving as the development process progresses. Consequently, the DSM is not static. As the DSM changes according to the product knowledge, the process scheme based on the DSM should also change. Hence, a procedure for DSM translation, reflecting one cycle from planning to implementation, should be performed each time a change in product knowledge occurs.

The following sections describe the integrated approach for iterative planning, modeling, and executing processes with dynamic process scheme (Karniel and Reich 2007b). When integrating process planning (via DSM) and execution

A. Karniel and Y. Reich, *Managing the Dynamics of New Product Development Processes*, DOI: 10.1007/978-0-85729-570-5_8,

(through workflow engine tools), the limitations of both are exacerbated. In the first step of process planning, DSM representations of product data give rise to many process interpretations (Karniel and Reich 2009, see Sect. 7.5). The difference between business model descriptions (e.g., DSM) and the unambiguous workflow specification required for their implementation and execution was emphasized by (Dehnert and van der Aalst 2004). When these interpretations, defined as *Business Rules*, are formulated and organized, they drive the interpretation of a DSM-based plan to a concurrent process plan model, the *Process Scheme*. Formalizing the BRs, including the case of self-iterations, allows constructive verification of the process scheme based on Work-Flow (WF) nets (van der Aalst 1998), and additional requirements specific to design processes. The DnPDP approach presented in this section consolidates process planning and re-planning at run time due to evolving product knowledge; dynamic process implementation; and process execution and simulation.

8.2 Closed-Loop Process Framework

The process in Fig. 8.1 describes the framework's closed-loop controlling process. The control process controls the dynamically changing design process together with its dynamic process scheme changes. This process has two types of entries: *external* entries, which are not aligned with process propagation and might change at any time; and *internal* entries, which are dependent on the actual progress of the process. This description is defined as the framework *operational view*.

The framework includes three main blocks: the *integrated -process generator* (1)[1] that generates the dynamic process schemes, a *process engine* (9) that executes the run time process according to the (updated) process scheme, and a *product-based process scheme generator* (11) that generates the required process scheme according to the gained product knowledge.

The integrated-process generator (1) merges external entries of required changes, with internal feedback entries, and generates the process scheme for the next time step accordingly. The external entries are skeleton processes (predefined best practices) and their changes (2); any *ad-hoc* change (3); product requirements (4); and resource constraints (5). The internal feedback entries are the process status (assumed to be defined within the system) (7), and the DSM-based process plan (6) that is generated according to current product knowledge. That plan (6) is generated by the product-based process scheme generator (11), using the DSM-based procedure and the planning algorithm. The specific algorithm used could be replaced (e.g., for checking differences between planning algorithms). The input to the latter generator (11) is the product knowledge (10), which is entered by

[1] The numbers in parentheses indicate index reference in Fig. 8.1.

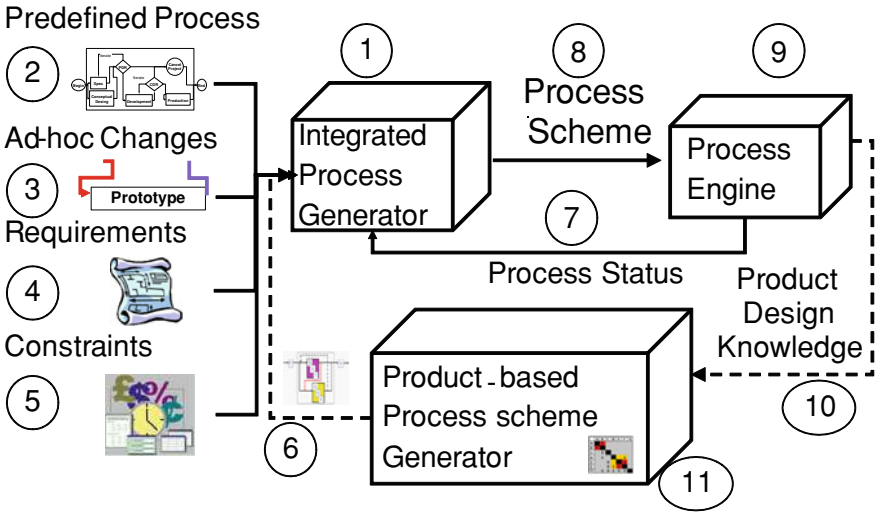


Fig. 8.1 DnPDP framework—closed-loop process

developers during the design process, and is the sources of DSM-based calculations.

The process engine (9) is assumed to have equivalent capabilities to current workflow engines, with resources optimization capabilities (not addressed in this research); yet, it is following a changing process scheme and therefore (unlike current engines) has the capability to record the actual Run Time (RT-process). The importance of this capability for modeling a dynamic process scheme is demonstrated in Sect. 11.4.

8.3 Decision-Making View

The same system is also used for simulation, where simulation results can aid the managers to make decisions regarding the process. One type of decisions may be the choice of applicable business rules for interpreting the DSM-based plan. This additional aspect is depicted in Fig. 8.2.

The system appears twice, once used for the actual monitoring of the process, and additionally used as a simulation tool. The actual process inputs and actual process output (process status and product knowledge) are the input to the simulator that is stochastically generating multiple scenarios and can supply results, which are then used for decision-making. Decisions might be manual (as graphically represented) or can be the output of a decision algorithm. This description of the process is regarded the *decision-making view*.

An additional view describing the repeated calculations of the framework is given by the following meta-process.

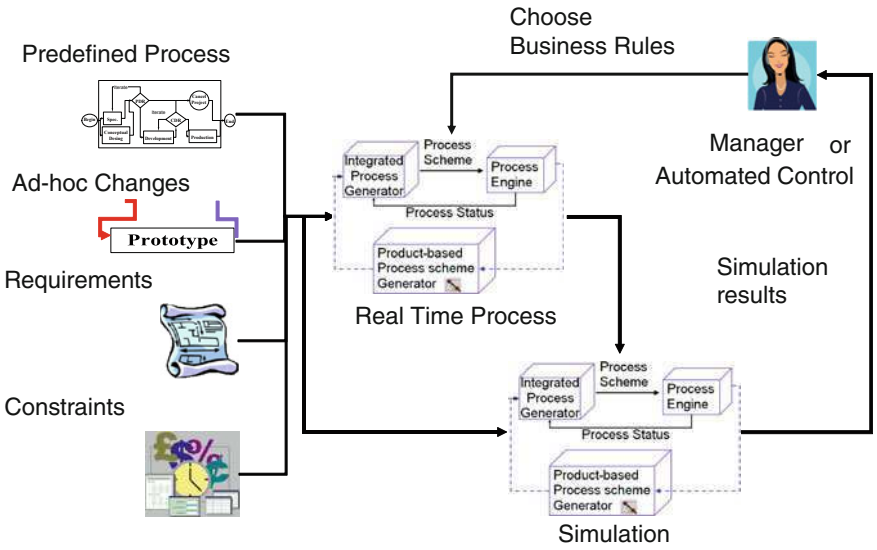


Fig. 8.2 DnPDP framework—decision-making view

8.4 Meta-Process Description

The closed-loop iterative meta-process is depicted in Fig. 8.3. The diagram presents the meta-process activities (procedures or processes) followed by results (information or a model), and entries of information (e.g., product knowledge and business rules). This description is defined as the framework *computational view*.

Current input information at time t is utilized for planning the process for the next time step $t + 1$. The information created at the next time step $t + 1$ becomes the information used for the next meta-process calculation. Steps (I) to (III) are done within the "product-based process scheme generator", Steps (IV) and (V) are done as part of the "integrated process generator", and step (VI) is executed by the "process engine".

In the first step (I), the product knowledge is converted into a DSM. Product knowledge includes the product structure translated to design activities at the required detail level (typically, the work of one resource, e.g., designer, applied to a product component, e.g., subassembly is considered as one activity); and the influence of changes in the design created by one activity on design work performed by other activities. Then, the DSM is reordered to create a process plan assuming the process is starting now. The one-time plan assumption is typically used in DSM-based simulations; however, in the current framework, the plan is updated in every time step according to current project knowledge.

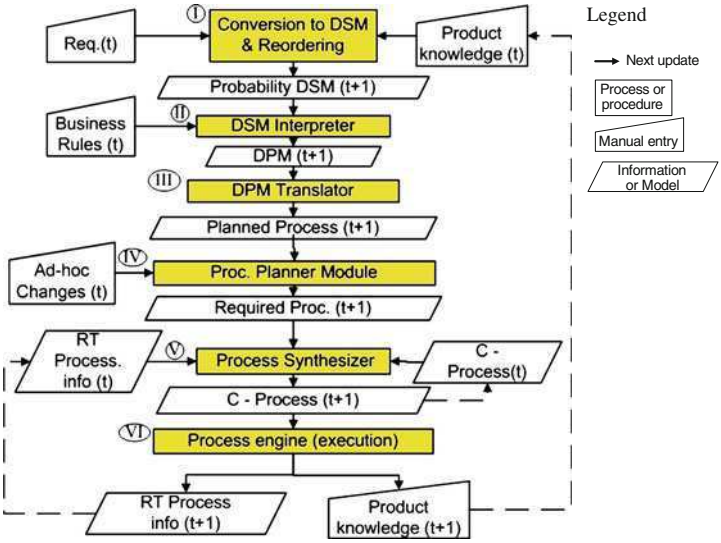


Fig. 8.3 Meta process model

Methods of acquiring the DSM information are described in (Sered and Reich 2006; Yassine 2007). It is assumed that such information can be captured and managed by Product Data Management (PDM)/Product Life cycle Management (PLM) systems. This step is further detailed in the next section.

The second step (II) is interpreting the ordered DSM into a DPM, making use of BRs that define the process logic (not described by the DSM). These BRs might be set manually or changed during the process; thus, the rules defined at time t apply to the planning of process scheme for time $t + 1$.

In step three (III), the DPM is translated into the *planned process scheme*. The translation is considered technical, founded on WF-nets (Petri nets) proofs.

In step four (IV), updating the process scheme requires integration of the *planned process scheme* based on the current product knowledge (the process plan as if the process starts from scratch); and any *ad-hoc* changes to the process (e.g., changes to the skeleton process, or adding a new activity such as prototyping to reduce development risk). This integration yields the *required process scheme*, which is still an abstraction detached from the actual process status.

In step five (V), the process synthesizer (within the integrated-process generator) utilizes the *required process* (planned for $t + 1$), the *current process scheme* C-Process(t) and the actual status of the run time process RT-process(t); and calculates the *current process scheme* for the next time-step C-Process($t + 1$).

Step six (VI) is the actual execution of the design process. It is not part of the calculations, but it provides the data for the next step. This step is simulated in the simulation mode.

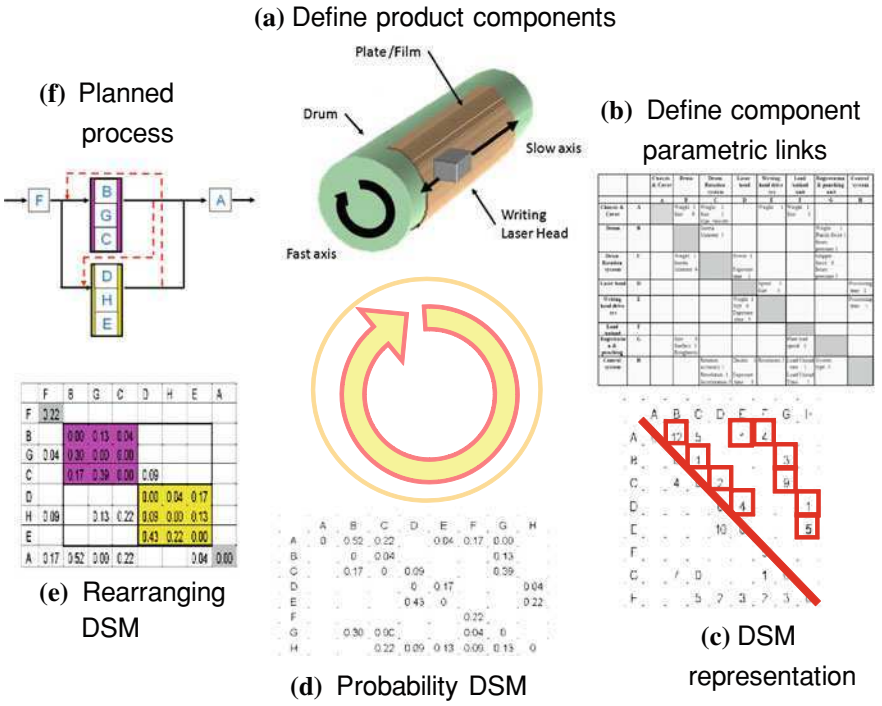


Fig. 8.4 Process-planning cycle

8.5 From Product Knowledge to Planned Process Scheme

The procedure of translating from DSM to planned Process Scheme, steps (I) to (III) in the meta-model (Fig. 8.3), is defined as the *process planning* cycle. The process-planning cycle is mainly focused on step (I), and has the following sub-stages, depicted in Fig. 8.4 (Karniel and Reich 2007d).

In the following sections, the planning cycle stages in Fig. 8.4 are elaborated, and are mapped to the meta-process steps of Fig. 8.3, in order to align both views. Stages (A) to (E) are aligned with previous work in the literature, and are quite similar to several other publications; yet, some enhancements are unique to the current work and are distinguished as sub-stages. Stage (F) is relating to steps (II) and (III); it is significantly different from any previous work, and is a main contribution of the current work.

Step (I) stages**:**

(A) Defining design activities according to the identification of product components and assigning design activities to these components.
(B) For each design activity, defining the parameters that influence other design activities based on parametric links between components.

(B1) For each parameter, assigning its influence value (the influence it may have on design changes) and setting the influence direction.

(C) Summing the influence values for each DSM cell (in order to get the Impact DSM representation).

(D) Assigning probability values (preliminary probability DSM) and scaling them; thus, creating a Probability DSM.

(D1) Adding self-iteration probabilities (Karniel and Reich 2007a).

(E) Rearranging the Probability DSM. The DSM can be reordered using an optimal Sequencing and Clustering algorithm (Karniel et al. 2005); or by using other reordering algorithms (see the DSM review in Sect. 3.3.1).

(E1) Merging activities into Design-blocks (grouped activities), then calculating the merged probabilities. This step is the main source for the requirement to add the definition of self-probability activities in the DSM (Karniel and Reich 2007a).

Step II stages:

(F) Translating the DSM into a process scheme.

(F1) Interpreting the DSM to a Design Process Matrix (DPM) by explicitly adding logic activities (e.g., process Begin, and End activities; Input logic and Output logic per each design activity; or Design-block Begin and End activities). Adding logic is done according to BRs that are set according to business requirements.

(F2) Assigning deterministic links ($p = 1$). This assignment can be done only after the rearranging the DSM, as otherwise rearrangement may yield a feedback link with $p = 1$ that will cause an infinite loop. Deterministic links describe a "must follow" logic that is used for connecting logic activities to design activities (Karniel and Reich 2007c). Such links may also be used for indicating specific order, i.e., testing should always follow a design activity (Lévárdy and Browning 2005).

Step III stages:

(F3) Converting the DPM into an equivalent Current process scheme (C-Process).

Transition step from product knowledge to reordered DSM, using similar approach to sub-steps (A) to (E) (excluding sub-steps B1, D1, and E1) were previously described and detailed in many articles, (Rogers and Bloebaum 1994; Eppinger et al. 1994; Eppinger et al. 1997; Rogers et al. 1999; Browning and Eppinger 2002; Sered and Reich 2006). Sub-step (B1), (D1) and (E1) are unique to the current research (Karniel 2009). Sub-steps (B1) and (D1) were demonstrated in Sect. 4.1. In step (E), almost any reordering procedure could be used as long as it

keeps the property that a feedback link is necessarily representing an activity cycle. This property is guaranteed for Partitioning (Karniel and Reich 2011), and actually for any minimum feedback procedure (otherwise a better DSM reordering could be found, i.e., which minimizes the number of feedback links).

Adding self-iteration probabilities do not influence the rearrangement procedure (as diagonal values are typically ignored).

The transformation to process scheme logic for simulating the results is not unique and is typically done in an informal way, as previously discussed in Sect. 5.1. The formal approach for interpreting steps (II) and (III) (stages F1, F2, and F3 respectively) is detailed in Chap. 1.

8.6 Building and Modeling an Evolving Process

In the current framework, design activities are considered as subprocesses, i.e., there is no notation of an atomic task. Such concept is easily adapted in building hierarchical processes, where each activity may have a more detailed content. Applying the concept requires a definition of the required granularity. Yet, the level of granularity is not predefined, and the approach used enables detailing or grouping (e.g., merging to DBs). Hence, another flexible dynamic dimension is added to the process model.

Process modeling according to the presented approach is iterative (it repeats every time there is a change of process knowledge), and has many stages, involving the integration of several process models. The different process model types are predefined processes (P-process), current processes scheme plan (C-process), and run time process (RT-process) (see Sect. 2.4).

A high-level, best-practice predefined process is the initial skeleton of the development process (c.f. Fig. 2.4). At that level, the complete design stage is modeled by one activity. Using the hierarchical approach, this activity becomes a subprocess (with actual process details set according to the product) once there is some additional knowledge about the product structure. Using a hierarchical Exception Handling approach, this activity could be replaced by a predefined subprocess (according to a predefined set of options) (Klein and Dellarocas 2000).

The C-process is calculated each time when there are new product knowledge inputs that may change the plan. There is only one current process at a given time, which is followed by the run time process. However, there are many C-processes over the process progress; there are many interim stages of creating the current process; and furthermore, there might be several optional C-processes to choose from at a certain time. All these process options are managed in parallel.

The RT-process follows the C-process that applies for the next period. Using the hierarchical updating approach, the design activity is elaborated according to the currently available and desired subprocess level. In turn, each of the design activities could be further detailed to a subprocess. These subprocesses may be more detailed design activities or predefined administrative processes.

Administrative processes that are relevant to many design activities can occur within any design activity. Some examples are an Engineering Change Order (ECO), used for changing the design of a component once the component was released. Another example could be the assignment of a part number to a new component; typically, such task requires cooperation of an engineer with purchasing and logistics employees. All current workflow systems can manage these administrative (sub) processes, but not the changing design activities. A useful approach that should be further investigated is the use of distributed workflows for the administrative subprocesses that are controlled by the main process, as described in (Reichert et al. 2010).

References

Browning TR, Eppinger SD (2002) Modeling impacts of process architecture on cost and schedule risk in product development. IEEE Trans Eng Manag 49(4):428–442

Dehnert J, van der Aalst WMP (2004) Bridging the gap between business models and workflow specifications. Int J Cooperative Inf Sys 13(3):289–332

Eppinger SD, Whitney DE, Smith R, Gebala D (1994) A model based method for organizing tasks in product development. Res Eng Des 6(1):1–13

Eppinger SD, Nukala MV, Whitney DE (1997) Generalized models of design iterations using signal flow graph. Res Eng Des 9:112–123

Karniel A (2009) Managing the dynamics of product development processes for new product development. Dissertation, Tel Aviv University, Israel

Karniel A, Reich Y (2007a) Simulating design processes with self-iteration activities based on DSM planning. IEEE proceedings of the international conference on symposium Engineering and modeling - ICSEM'07, 33–41, Haifa, March

Karniel A, Reich Y (2007b) Managing dynamic new product development processes. Proceedings of the 17th annual international symposium of the international council on system engineering INCOSE'07, San Diego, California, June

Karniel A, Reich Y (2007c) A coherent interpretation of DSM plan for PDP simulation. In proceedings of the international conference on engineering design, ICED 07, Paris, August

Karniel A, Reich Y (2007d) From planning to executing NPD processes. In the 4th annual israeli national conference on system engineering−INCOSE_IL'07, Herzliya

Karniel A, Reich Y (2009) From DSM based planning to design process simulation: A review of process scheme logic verification issues. IEEE Trans Eng Manag 56(4):636–649

Karniel A, Reich Y (2011) Formalizing the implementation of DSM-based process planning for NPD. IEEE Tran on Sys Man & Cybernetics, Part A 41(3):476–491

Karniel A, Belsky Y, Reich Y (2005) Decomposing the problem of constrained surface fitting in reverse engineering. Comput Aided Des 37:399–417

Klein M, Dellarocas CA (2000) Knowledge based approach to handling exceptions in workflow systems. J Comput Supported Collaborative Work 9(3/4):399–412

Lévárdy V, Browning TR (2005) Adaptive test process—designing a project plan that adapts to the state of a project. 15th Annual international symposium of the international council on system engineering (INCOSE), July 2007

Reichert M, Bauer T, Dadam P (2010) Flexibility for distributed workflows. In: Wang M, Sun Z (eds) Handbook of research on complex dynamic process management: Techniques for adaptability in turbulent environments. IGI Global, Hershey, PA, pp 137–171

Rogers JL, Bloebaum CL (1994) Ordering design tasks based on coupling strengths. AIAA, paper no. 94-4326

Rogers JL, McCulley CM, Bloebaum CL (1999) Optimizing the process flow for complex design projects, design optiization. Int J Prod Process Improvement 1:281–292

Sered Y, Reich Y (2006) Standardization and modularization driven by minimizing overall process effort. Comput Aided Des 38(5):405–416

van der Aalst WMP (1998) The application of petri nets to workflow management. J Circ Sys Comput 8(1):21–66

Yassine A (2007) Investigating product development process reliability and robustness using simulation. J Eng Des 18(6):545–561

Chapter 9
From DSM to DSM Net

9.1 Introduction: DSM Translation Concepts

The following sections describe the integration of the DSM planning model with process modeling approaches of Petri nets. First, the process correctness criteria for the Dynamic new-Product Design Process (DnPDP) are presented in Sect. 9.2. Most requirements are adapted from WF-net correctness criteria.

Additional requirements, which are more specific to the iterative nature of the PDP processes, are added. Next, Sect. 9.3 presents the translation stages with a simple example. In Sect. 9.4, the DSM representation and translation steps to DSM net are formally defined. Petri net based results (Chap. 1) are used to prove that the process model resulting from the translation of DSM to process scheme is correct. Readers who are less interested in the formal definition can skip Sects. 9.5, 9.6, and 9.7.

9.2 DnPDP Correctness Criteria

Karniel and Reich (2007c) defined the following correctness criteria for a dynamically changing process scheme (DnPDP):

1. The PDP has project characteristics; it should have a defined start and a defined end (termination state).
2. From the currently given state, the process should be able to reach the termination state of the current process scheme.
3. Reaching the termination state (outcome of executing the End activity) should imply:
   (a) All the design activities (and all their iterations) have completed and
   (b) Each design activity has been performed at least once.
4. The process should be traceable.

A. Karniel and Y. Reich, *Managing the Dynamics of New Product Development Processes*, DOI: 10.1007/978-0-85729-570-5_9,

5. Despite the iterative nature of the process, which enables infinite loops, the process should be enforced to complete in a finite time.

Requirements 1, 2, and 3(a), echo the WF-net requirements. Requirement 1 was demonstrated and discussed in Sect. 7.3. Requirement 2 in the WF-net literature applies to the static process scheme; while in the current work, it applies for dynamic changes of the process scheme. This requirement indicates that a change of the process scheme at a current process state should be such that it allows the process to terminate. The implication of the requirement is: (a) the process analysis should apply to the current state (not for every possible state) and (b) the analysis can be done for each of the process schemes that are applicable to the current state using momentarily static process scheme.

Requirement 3(b) implies the completeness of the design process, assuming that in the DSM are required. This assumption implies a relation of *what* should be done (assign design activity to product component), rather than specific design operations (i.e., *how* should the design be done). If the process planning was accounting for specific operational tasks with multiple parallel options to choose between (i.e., many ways of performing the design), then requirement 3(b) should be modified. Using the concept of required activity without specifying definite operations and applying 3(b) has a major benefit, which is the ability to set a strict simulation termination criterion that is easily checked. Otherwise, checking the process correctness would have been more complex; i.e., requirement 3(b) should be modified to checking that at least one of the potential options of performing the activity was executed. Requirement 4 establishes the need of process records and may have regulatory implications (e.g., Sarbanes and Oxley 2002). The operational aspect of this requirement is keeping the records due to following a changing scheme and for allowing rollback. The last requirement is practical: it is required for simulation, and it follows practical behavior. In practice, no project has unlimited duration (unlimited resources); thus, at some point the process will be enforced to complete or be cancelled.

9.3 Translating an Ordered DSM into a Process Scheme

Translating an ordered DSM into a process scheme has the following main steps (cf. Fig. 9.1):

1. Translating the DSM into a Design Process Matrix (DPM), adding logic activities and defining the logic assigned to the logic activities according to BRs.
2. Converting the DPM into an equivalent DSM net. The DSM net becomes the *current* process scheme (C-Process).
3. Using the C-Process for simulating the *run time* process scheme (RT-process).

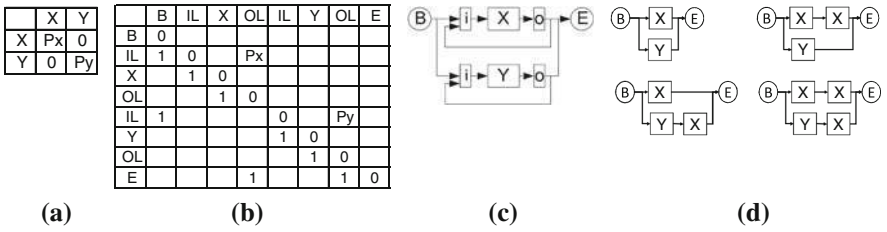


Fig. 9.1 DSM Translation to process scheme **a** DSM **b** DPM **c** C-Process **d** RT-Process

The following example relates to conversion of two parallel independent activities from DSM to DPM, then to C-process scheme, and finally some potential RT-process examples are presented.

The Logic activities added in Step 1 consist of Process Begin and End activities, and Input and Output Logic activities according to Implementation Rules, Fig. 9.1b.

The interpretation from DPM to the C-Process is structurally straightforward, Fig. 9.1c, since the process logic is neither detailed in the DPM nor in the resulting process scheme (DSM net). Unlike Petri nets, the logic is an additional layer. The benefit of such separation is the ability to change the logic during run time according to the process status (e.g., number of iterations performed).

Iteration of a design activity is presented as a new activity in the RT-process (i.e., may have different properties than a previous execution). Due to the iterative nature of the activities, many RT-processes could be derived from a single C-process scheme, Fig. 9.1d.

9.4 Formal Definition of DSM Conversion to a Process Scheme

The DSM represents serial activities (sequential) by forward links; parallel activities (concurrent, independent) have no relation links; and coupled activities (mutually dependent) form a close loop of forward and feedback links.

By establishing relations of the DSM-based process scheme to the specialized WRI-WF-nets (subtype of Petri nets), we can use proven Petri net results for a constructive build approach of the process scheme. We first define the DSM and DSM reordering by partitioning; then, conversion to design process matrix (DPM) including a step of adding logic activities; and conversion to a DSM net graph. We prove the conversion of the DSM net to a Petri net, and finally the conversion to WF-net. We prove that certain implementations of the DSM-based plan to a DSM net can always be constructed as WRI-WF-net, i.e., in a hierarchical way. A resembling approach is used in Robidoux et al. (2010) for converting reliability block diagrams to colored Petri nets.

The DSM reordering by a partitioning algorithm (or an algorithm where feedback link implies an activity cycle in the DSM e.g., Karniel et al. 2005)

is important since it provides the basic mechanism to establish a process plan that can be converted to a WRI-WF-net (Karniel and Reich 2011). The main outcome is the conversion of block-diagonal DSM to WRI-WF-nets, which includes the conversion of the fully parallel and the serialized processes as special cases. This conversion establishes the further use of design blocks. Additional proof relates to the implementation of DSM-based plan resulting from optimal sequencing algorithm (where activity cycles are further decomposed to subcycles). It is proved that the implementation is WRI-WF-net.

9.5 DSM Definitions and Properties

The definitions, corollaries, lemmas, theorems, and propositions are numbered in sequence to the numbering in Sect. 6.7 (Karniel and Reich 2011). They are referred by their initial letter and number. The links between the items are diagrammatically sketched in Sect. 14.5.

Definition 16 (*DSM index, DSM reordering*)

(1) $DSM = (As,\ L)$ *is a square matrix.* Activities $a_i \in As,\ 1 \leq i \leq N$ are the ordered labels of rows and columns, where As is the activity set. L is the set of links, and $l \in L$ indicates direct link between activities (a matrix marking).
(2) $In(a_i) = i$ is the mapping of an activity to its order index (having same row and column index) in the DSM.
(3) $Ro(DSM_a) = DSM_b$ is a permutation function (reordering) of DSM activities. Ro is reversible. $Ro = I$ (the identity reordering) iff $\forall x_i,\ In(x_i \in DSM_a) = In(x_i \in DSM_b)$.

Definition 17 (*DSM Directed link, Source, and Target mapping*) DSM directed link is a marking, such that the column is the link source and the row is the link target.

1. $So(l) = a_j$ is a mapping of a link to its source activity a_j (column).
2. $Ta(l) = a_i$ is a mapping of a link to its target activity a_i (row).
3. A link is the *Output* link of activity a_j when its source is a_j (i.e., the column of a_j).
4. A link is the *Input* link of activity a_i when its target is a_i (i.e., the row of a_i).

Two indexing options are used: Single index $l_k = (a_j,\ a_i)$ or two indices $l_{ij} = (a_j, a_i)$, where i is the DSM row (target) and j is the link column (source). For any specific DSM ordering, there is a one-to-one mapping between these options. The index k is counted along the row from top to bottom (rows from left to right). The single indexing is also used for indicating unspecified links $l_k = (x_j, x_i)$ (i.e., no specific location).

Definition 18 (*Forward link, Feedback link, Self-iteration*)
(1) A forward link $In(So(l)) < In(Ta(l))$ (column < row, below diagonal).

(2) Feedback link $In(So(l)) > In(Ta(l))$ (above diagonal).
(3) Self-iteration link is a link where $In(So(l)) = In(Ta(l))$ (on the diagonal).

Choosing the subdiagonal to represent forward links is a common practice in DSM literature. Self-iterations are a subtype of feedback links. They are distinctly referred since they are unique to this research; typically, this link type is not used in the DSM literature (Karniel and Reich 2007a).

Definition 19 (*Link Sets*) The following sets of links can be defined for activity a_i:

$LFI(a_i) = \{l = (a_k, a_i) | \ k < i\}$ forward input links;
$LII(a_i) = \{l = (a_k, a_i) | \ k > i\}$ iteration (feedback) input links;
$LFO(a_i) = \{l = (a_i, a_k) | \ i < k\}$ forward output links;
$LIO(a_i) = \{l = (a_i, a_k) | \ i > k\}$ iteration (feedback) output links; and
$LSI(a_i) = \{l = (a_i, a_i)\}$ self-iteration link.

Definition 20 (*Regular DSM*) A DSM is regular iff $\forall a_i$, $LSI(a_i) = \varnothing$ (only off diagonal links, no self-iterations).

Definition 21 (*Acyclic DSM*) A regular DSM (D5) is an acyclic DSM iff $\forall a_i$, $LIO(a_i) \cup LII(a_i) = \varnothing$ (i.e., all the links are forward links).

Such DSM models a DAG process (as GANTT or PERT; see Fig. 9.9).

Note Acyclic DSM is not defined by not having cycles (despite the name), but as having forward links (input and output) only; not having cycles is a property proved later.

Definition 22 (*DSM Path, Elementary*)

(1) A DSM path $C = (x_1, \ldots, x_m)$ is a non-empty sequence from an activity x_1 to another activity x_m, through directed links $l_i \in L$ $l_i = (x_i, x_{i+1})$, $1 \le i \le m-1$. The path length m is the number of activities in the sequence.
(2) C is an elementary path iff $\forall x_i, x_j \ i \ne j \rightleftarrows \Rightarrow x_i \ne x_j$ (i.e., nodes do not repeat).
(3) An alphabet operator α is defined by $\alpha(C) = \{x_1,\ldots,x_m\}$.
(4) The operator β is defined by $\beta(C) = \{l_1,\ldots,l_{m-1}\}$.
(5) $C \subset DSM$ iff $(1)\forall x_i \in \alpha(C) \wedge \forall l_i \in_\beta (\mathrm{C}) \Rightarrow x_i \in As \wedge l_i \in L$.
(6) CI is a non-unique path index mapping of activity a_i indicating its position in path C, $CI(a_i) = \{j_k\}, 1 \le j_k \le m$, where k is the number of applicable indices.

Elementary path and Alphabet operator definitions are equivalent to Definition 2.

Definition 23 (*Forward Path*) A path $C = (x_1, \ldots, x_m)$ is a forward path iff $\forall x_i, x_{i+1}, 1 \le i \le m-1$ the link $l_i = (x_i, x_{i+1})$ is forward link (D18).

Corollary 2 *A path C is a forward path iff* $\forall x_i, x_j \ i < j \Rightarrow In(x_i) < In(x_j)$.

Corollary 3

(1) *If C is a forward path, then C is an elementary path* (D22) (*i.e., activities do not repeat*).
(2) *In an acyclic DSM* (D21)*, any path C is a forward path.*

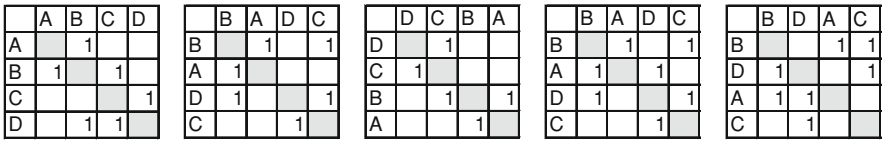


Fig. 9.2 DSM types **a** DSM with cycles **b** Cyclic DSM **c** Block-diagonal DSM **d** Cyclic DSM Modified **e** Nominal Cyclic DSM

Definition 24 (*Activity cycle, Simple activity cycle, Self-cycle*)

(1) Path $C = (x_1,\ldots,x_m)$ is an activity cycle iff $In(x_1) = In(x_m)$ (i.e., first and last activities in the path are the same).
(2) An activity cycle C is a simple activity cycle iff $\forall i, j \notin \{1, m\} \Rightarrow In(x_i) \neq In(x_j)$ (or equivalently $\forall i \notin \{1, m\}\, CI(x_i)$ is unique); otherwise, if $\exists i, j\, i < j$, $i, j \notin \{1, m\} \wedge In(x_i) = In(x_j)$ there are subcycles.
(3) Self-cycle $C = (x_1, x_2)$, $In(x_1) = In(x_2)$ (shortest activity cycle and a simple cycle).

Corollary 4 (Cycle simplification) If C is an activity cycle with subcycles, a simple activity cycle C^* can be obtained from C by replacing any simple subcycle in C, with its first activity, or replacing self-subcycle by its first activity.

Note Any cycle can be simplified, but the resulting simplified cycle might not include all the activities that were included in the original cycle. Example in Fig. 9.2b, from the cycle (BABDCDCB) we can obtain (BAB) by replacing (BDCDCB) with B; or obtain (BDCB) by first replacing (BAB) with B and then replacing (CDC) with C (equivalently replacing (DCD) with D).

Corollary 5 (Reordering a cycle) *If a path* $C = (x_1, \ldots, x_m) \subset DSM_a$ *is an activity cycle, then,* $\forall Ro$, $DSM_b = Ro(DSM_a)$,

$C = (x_1, \ldots, x_m) \subset DSM_b$ *is an activity cycle* (*only the order indexes* (D16) *of the activities will change*).

Proposition 4 (No cycles in an acyclic DSM) *If DSM is acyclic* (D21) *then it has no activity cycles* (D24).[1]

Proof In an acyclic DSM, every path is a forward path since there are only forward links, $\forall x_i, x_j, i \neq j$, $In(x_i) < In(x_j)$; therefore, there is no link that can connect to the initial activity in the path (to form a cycle).

Note Though it seems obvious by name, acyclic DSM (D21) was defined by not having feedback links. Not having cycles is a property. A general DSM with no cycles that was not reordered by partitioning may have feedback links (i.e., is not acyclic). Such DSM can be reordered by a partitioning algorithm to an acyclic DSM as further proved. □

[1] Despite the name acyclic DSM was not defined as not having cycles, therefore the property should be proved.

Lemma 1 (Partitioning procedure for activities with forward links) *In a reordered DSM, if an activity is placed in different order according to steps* (2) *and* (3) *of the Partitioning procedure* (Sect. 3.3.1)*, then* $LII(a_i) = \varnothing$ *and* $LIO(a_i) = \varnothing$ (D4)*. It has neither feedback input nor feedback output links.*

Proof We keep two indices, *Current-first* and *Current-last*, initiated to 1 and N (number of activities), respectively. According to step 2, each activity x_i that does not have input from other activities (being currently checked) is moved such that $In(x_i) =$ *Current-first*; the index is increased and the activity is no longer checked. Respectively (step 3), each activity x_i without output link to other activities (being currently checked) is moved such that $In(x_i) =$ *Current-last,* the index is decreased and the activity is no longer checked.

An activity x_i ordered by step (2) has no input link from other activities x_j being checked (*current-first* $\leq j \leq$ *current-last*), and not from activities, which were ordered by step (3) (that does not have output links). Therefore, there are no input links from any other activity x_j where $In(x_i) < In(x_j)$, (i.e., no feedback input links) $LII(x_i) = \varnothing$. It may have input links from other activities already ordered by step (2), but these will be only forward input links. Furthermore, since the activities x_j previously ordered by step (2), $In(x_i) > In(x_j)$, have no feedback input links, the current activity x_i cannot have feedback output link to these activities, i.e., $LIO(x_i) = \varnothing$. Overall, it cannot have feedback links.

Similar reasoning is applied to activity x_k being ordered by step (3). We conclude that activity x_k cannot have output links to any activity x_j where $In(x_k) > In(x_j)$, i.e., no feedback output links) $LIO(x_k) = \varnothing$ and cannot have input links from activities x_j already ordered by step (3) where $In\ (x_k) < In(x_j)$ (i.e., no feedback input links) $LIO(x_k) = \varnothing$.

Note A formal proof of these properties was not found in the DSM" literature. □

Proposition 5 (Partitioning a DSM without cycles) *When applying the Partitioning procedure to a DSM without cycles, the procedure will terminate by applying step* (2) *only.*

Proof We find an activity with no input links and make it first. Since there are no cycles, such activity must exist. Applying step (2) again, the second activity might have only links from the first, but since the first is no longer considered, it does not have input links from any other activity, and so forth. □

Proposition 6 (Indices of an acyclic DSM) *If two different reordering* $DSM_a = Ro_a(DSM)$ *and* $DSM_b = Ro_b(DSM)$ *exist* (D16)*, and both are acyclic DSM* (D21)*, then,* $\forall x_1,\ x_2\ _\wedge\ l = (x_1,\ x_2) \in L$*, if* $In(x_1) < In(x_2)$ *at* DSM_a*, then* $In(x_1) < In(x_2)$ *at* DSM_b*.*

Proof $l = (x_1, x_2)$ is a forward link in DSM_a, and since DSM_b is also acyclic the link must be a forward link; thus, the index order of the linked activities is kept. □

Proposition 7 (Reordering does not eliminate cycles) *If $DSM_b = Ro(DSM_a)$, and DSM_b is acyclic* (D21)*, then DSM_a has no cycles.*

Proof (by contradiction): If DSM_a had a cycle, reordering of a cycle still keeps it as a cycle (C5); therefore, DSM_b cannot be acyclic (P4). □

Theorem 2 (Partitioning a regular DSM without cycles) *The Partitioning procedure applied to a regular* (D20) *DSM_a will terminate after utilizing only steps* (2) *and* (3) *iff $\exists Ro$, $DSM_b = Ro(DSM_a)$ and DSM_b is acyclic* (*i.e., the DSM can be reordered to an acyclic DSM*).

Proof If the procedure has completed after performing only steps (2) and (3) and there are no additional activities left to be reordered, then by Lemma 1, all the activities that were handled have no feedback links. Consequently, the resulting DSM is acyclic (D21); and the final places of the activities represent the permutation function, i.e., such function exists.

Conversely, if DSM_b is acyclic, then, there are no cycles in DSM_a (P7). Therefore, DSM_a can be partitioned by applying only partitioning step (2) (P5). □

Corollary 6 *If $\exists Ro$, $DSM_b = Ro(DSM_a)$ and DSM_b is acyclic, then applying the Partitioning procedure to any reordering of DSM_a will yield an acyclic DSM as a result.*

Theorem 2 (T2) and corollary 6 (C6) indicate that a DSM whose structure can be represented as a DAG could be reordered to have only forward links.

Theorem 3 (Ordered DSM properties) *A DSM reordered by a partitioning algorithm: $\exists C$* (*activity cycle exists*)*, iff $\exists l$ such that l is a feedback link or a self-iteration link.*

Proof (1) If an activity self-cycle exists, then by definition the link $l = (x_1, x_1)$ is self-iteration link; otherwise,

(2) Considering a simple activity cycle (D24) $C = (x_1, \ldots, x_{m-1}, x_m)$. If any of the links prior to the last is a feedback link, then, we are done. Otherwise, we need to prove that the last link must be a feedback link. If all previous links are forward links $In(So(l)) < In(Ta(l))$, then $In(x_1) < In(x_2) < \cdots < In(x_{m-1})$, therefore, the last link $l = (x_{m-1}, x_m)$ is a feedback link, as $In(x_{m-1}) > In(x_1)$ and $In(x_1) = In(x_m)$. An activity cycle that is not simple has a simple subcycle.

Conversely, (3) if there is a self-iteration link, we have an activity cycle by definition.

(4) If there is a feedback link l, we want to prove that an activity path containing l must be an activity cycle. On the contrary, assuming there is a path containing l that is not a cycle.

According to (P5) the partitioning procedure of DSM with no cycles will terminate by applying step (2). From Lemma 1, activities ordered by step (2) or step (3) of the Partitioning procedure have no feedback links, so we get a contradiction.

Theorem 3 (T3) indicates that if a DSM was reordered by the partitioning algorithm and as a result there is a feedback link, then, the DSM must contain a cycle, i.e., it cannot be represented by a DAG. □

Proposition 8 (Feedback link in a simple cycle) *If l is a feedback link* (D18) *in a DSM reordered by partitioning, then, there exists a simple cycle C* (D24) *in which l is the last link.*

Proof If a feedback link $l = (x_1, x_2)$ exists, then it is a part of an activity cycle (T3). We define the cycle $C = (x_2, \ldots x_1, x_2)$ and turn it to a simple cycle (C3). □

Proposition 9 (Simple cycle reordering) *If* $\exists\ C = (x_1, \ldots, x_{m-1}\ x_1)$ *is a simple cycle* (D24), *and* $l = (x_{i-1}, x_i) \in \beta(C)$, *then* $\exists Ro$, $DSM_b = Ro(DSM_a)$, *and* $\exists\ C^* \vee \subset DSM_b$ *is simple cycle, such that* $l \in \beta(C^*)$ *is the last link and the only feedback link in* C^*.

Proof C^* exists by (P8), and the DSM is reordered as follows. If $min(In(x_j)) = k$ where $x_j \in \alpha(C)$ (the minimal order-index of activities in C), then place x_i such that $In(x_i) = k$ in DSM_b. Then, for $u = i + 1$ *to* $m-1$ in C, place the activities in DSM_b with $In(x_u) = k + (u-i)$, next place x_1. Then, for $v = 2$ *to* $i-1$ in C, place the activities in DSM_b such that $In(x_v) = k + (m-1-i) + v$. Therefore we get the cycle $C^* = (x_i, \ldots, x_{m-1}, x_1, x_2 \ldots, x_{i-1}, x_i)$.

In this simple cycle, all links (except the last) are forward links and the final one is l, a feedback link by construction. We will refer to such reordering as reordering the DSM by C^*, i.e., such that the cycle has one feedback link.

Note The partitioning algorithm completes once all cycles were identified. The additional DSM reordering defined by P9 is required for changing a simple cycle (D24) to have only one feedback link, which is the last link. □

Definition 25 (*Nominal Activity Cycle*) A simple activity cycle C (D24) with length $m > 2$ is a Nominal Activity Cycle iff the last link is feedback link $l_{m-1} = (x_{m-1}, x_m), \mathrm{In}(x_{m-1}) > \mathrm{In}(x_m)$ and all others are forward links.

Note Self-activity cycle has length $m = 2$ and has a self-iteration link. It resembles a nominal activity cycle, but is a distinct cycle type (e.g., there are no other links).

Definition 26 (*Nominal DSM Cycle*) A Nominal activity cycle C (D25) is a Nominal DSM Cycle iff $In(x_m) = In(x_1) = 1$ (i.e., it starts from the first activity in the DSM).

Definition 27 (*Cyclic DSM*) A DSM with N activities is a cyclic DSM iff $\forall i, 1 \le i \le N, a_i \in \alpha(C_k)$, where C_k is some Nominal DSM Cycle with index k.

Examples of cyclic DSM are depicted in Fig. 9.2b and d.

Proposition 10 (Cyclic DSM) *If* $\exists C \subset DSM$ *is a simple cycle, and* $\forall i, 1 \le i \le N, a_i \in \alpha(C)$ (*i.e., all the DSM activities belong to the same simple cycle*), *then, the DSM can be reordered to a cyclic DSM* (D27).

Proof If all activities belong to the same simple cycle we can define another cycle C^*, and set its starting to activity a_1, then reorder the DSM according to C^* (by P9) and get a Nominal DSM Cycle (D26); thus, the resulting DSM is cyclic. □

Definition 28 (*Nominal Cyclic DSM*) A Cyclic DSM (D27) with N activities is a Nominal Cyclic DSM iff $\forall C = (x_1, \ldots, x_{m-1}, x_m)$ that is a simple cycle starting at the first activity $In(x_m) = In(x_1) = 1$, C is a Nominal DSM Cycle (D26) (i.e., all simple cycles from the first activity have only one feedback link).

Definition 29 (*Block-diagonal DSM*) A DSM is a block-diagonal DSM iff there is at least one activity cycle, and there is no single cycle C that include all activities, $\forall C, \exists x_1, x_2, (x_1 \in \alpha(C)) \Rightarrow (x_2 \notin \alpha(C))$.

Corollary 7 (Block-diagonal DSM) *If a DSM is a block-diagonal DSM, then, it is neither acyclic DSM nor cyclic DSM.*

The various definitions are demonstrated in Fig.9.2a–e. Example of a regular (D20) cyclic (D27) DSM is depicted in Fig. 9.2b. The cycles (BAB) and (BDCB) are simple cycles with one feedback link, hence, nominal activity cycles. Both start from (B), thus are both Nominal DSM cycles. Since all activities belong to these cycles, then, this DSM is a Cyclic DSM. Since no other simple cycles start at (B), then the DSM is also Nominal Cyclic DSM.

The DSM in (b) is a reordering of the DSM in (a). The DSM in (a) has cycles (and may become cyclic), but it is not a cyclic DSM. The cycle (ABA) is a Nominal DSM Cycle. However, (ABDCBA) is an activity cycle, but not a Nominal Activity Cycle (has more than one feedback link), therefore it is not a Nominal DSM Cycle. Thus, activities {C, D} do not belong to any Nominal DSM Cycle.

If either the link (B, D) or (C, B) were missing in (a), the DSM could not become a cyclic DSM by any reordering. The result of the partitioning-procedure after removing the link (B, D) in (b) is a block-diagonal DSM demonstrated in (c).

If we add the feedback link (D, A) to the DSM in (b), the result is the DSM in (d). This DSM is cyclic, but no longer Nominal Cyclic DSM, since (BDAB) cycle is not a Nominal Cycle (two feedback links). Reordering the DSM in (d) yields the DSM in (e), which is Nominal Cyclic DSM. This reordering was done according to (P9) by choosing the last link (A, B) in the cycle to be the only feedback link in this cycle.

Lemma 2 (DSM simple activity cycles to Nominal Cyclic DSM) *If* $\forall a_i\ 1 \le i \le N, \exists C_k, a_i \in (C_k)$,*where* C_k*is simple cycle* (D24) *with index k starting at* a_1 (*first activity*), then, $\exists Ro$, $DSM_b = Ro(DSM)$ *such that* DSM_b *is a Nominal Cyclic DSM* (D28).

Proof We need to prove that there is always a reordering such that all the simple cycles C_k starting at the first activity a_1 are Nominal Activity Cycles (D25) (and therefore Nominal DSM Cycles (D26)). □

By (P9), for any simple activity cycle C_k we can reorder the DSM such that only the last link is a feedback link (i.e., Nominal Activity Cycle). We will reorder the DSM according to all the C_k cycles.

Let *CS* be the minimal set of the longest simple cycles such that each of the activities is included in at least one of the cycles in *CS*. Such set exists according to the assumption, but may not be unique. Then, we generate an ordered activity list and reorder the DSM according to that list. The following three cases define a reordering algorithm: (a) If all activities belong to the same cycle, then the ordered list includes the activities in their order in that cycle (removing the last activity a_1). (b) Case of no activity order conflicts between the simple cycles. No conflicts are defined by $\forall\ a_i, a_j, i, j > 1$ if $CI(a_i) < CI(a_j)$ in C_u, then $CI(a_i) < CI(a_j)$ in C_v where *u, v* are cycle indices. In this case, we generate a merged list of activities, such that all ordering constraints are kept. (c) The case of ordering conflicts, i.e., for an activity pair a_i, a_j (other than a_1), $CI(a_i) < CI(a_j)$ in one cycle C_u, and in another cycle C_v $CI(a_i) > CI(a_j)$. In such case, ordering the DSM according to C_u yields a forward path between a_i to a_j (the only feedback link is back to a_1), and respectively there would be a forward path from a_j to a_i in ordering according to $C_{v.}$. Therefore, a subcycle that includes both activities exists.

An "input path"is a path from a_1 to an activity within that subcycle such that the path does not include any other activity, which belongs to the subcycle. Respectively, an "output path" is a path from an activity within the subcycle to a_1, which does not include other activities of the subcycle. Furthermore, this subcycle must have at least two distinct forward "input paths" and two distinct "output paths"; otherwise, there could not be two contradicting simple cycles (the two subcycle sections included in C_u and C_v should have at least two intersection points).

In case (c), the following algorithm steps apply:

(1) Identify the minimal set of longest subcycles that include the activities with conflicts, where each subcycle has two "input paths" and two "output paths".
(2) For each identified subcycle: Identify the activities that are in "input paths" and add them to the input path list (IPL). Identify the activities that are in "output paths" and add them to the output path list (OPL). Choose a path of the activities in the subcycle such that it begins with an activity in the IPL, ends with an activity in the OPL, and includes all the activities in the cycle. There might be multiple options for such choice. If an activity belongs to both the IPL and OPL, then, choose the path option in which the activity is the latest in that path.
(3) Alignment of multiple subcycles and multiple options: choose a set of options that have no conflicts. If there are unresolved conflicts between the subcycles, then there are subsubcycles, and the procedure is applied recursively.
(4) Now, there are the aligned forward paths of the subcycles, input paths of each subcycle, and output paths of each subcycle. These paths have no conflicts so they can be merged. Continue as in case (b). Wrapping up, we have a reordering by which any simple cycle starting from a_1 is a Nominal Activity Cycle (D25).

An implementation example of case (b) refers to Fig. 9.2d. The list of cycles is {BAB, BDAB, BDCB}. CS is {BDAB, BDCB}, there are no conflicts and the merged list is (BDAC) (or it could be BDCA). The resulting reordering is depicted in (e).

	A	B	C	D	E	F	G	H	I	J	K	L
A	A								1			1
B		B		1								
C	1		C				1					
D			1	D		1						
E					E	1						
F			1			F				1		
G				1			G				1	
H		1						H				
I								1	I			
J	1									J		
K					1						K	
L							1					L

(a)

	A	J	F	E	K	G	C	D	B	H	I	L
A	A										1	1
J	1	J										
F		1	F				1					
E			1	E								
K				1	K							
G					1	G		1				
C	1					1	C					
D			1				1	D				
B								1	B			
H									1	H		
I										1	I	
L						1						L

(b)

Fig. 9.3 Reordering a cyclic DSM to a Nominal cyclic DSM **a** A cyclic DSM **b** Nominal Cyclic DSM

An example of case (c) in *L2* is depicted in Fig. 9.3. In Fig. 9.3a, we can identify many cycles such that activities belong to some simple cycles that start from activity *A*. A choice of minimal set of longest cycles can be {(AJFEKGCDBHIA), (ACFEKGLA)}. There are other options.

The conflicts in this set are between a subpath (FEKG) and C within the cycles, the paths ((FEKG)C) in the first cycle, and (C(FEKG)) in the latter. We can identify the following subcycles (CDGC) and (CFEKGC).

For the subcycle (CDGC), the input paths are (AC), (AJFD), and (AJFEKG); thus, the input list is IPL = {C,D,G}. The output paths are (DBHIA), (GLA) (DA), and the output list is OPL = {D,G}. The optional paths of this cycle are (CDG) and (GCD).

For the subcycle (CFEKGC), the input paths are (AC) and (AJF); thus, IPL = {C,F}. The output paths are (CDBHIA), (FDBHIA), and (GLA); thus, OPL = {C,F,G}. The optional paths are (CFEKG) and (FEKGC).

There are two options choosing aligned subcycle paths (CDG and CFEKG) or (GCD and FEKGC). The merger to an ordered activity list is done by merging the cycles, choosing an input path, and choosing an output path. In the first case, the merger should include C in the beginning, G at the end, and D in any position before, within, or after FEK. The optional mergers are: (CDFEKG), (CFDEKG), (CFEDKG), and (CFEKDG). A general way to write these options might be (C[EFK||D]G), where '||' indicates parallel paths. The output paths have no conflicts and no links; therefore, can be assigned in parallel [BHI||L], i.e., (LBHI), (BLHI), (BHLI), and (BHIL). Summing up, for the first option the merger can be described by (AC[EFK||D]G[BHI||L]), multiple parallel ordering options. All the ordering options described by the compressed parallel form are describing the same DSM net.

For the second choice, the path (GC) is a constraint, and we get (FEKGCD) as the only merger option. The multiple ordering options can be described by

Fig. 9.4 Not a nominal cyclic DSM

	A	B	C	D
A		1		
B	1		1	
C		1		1
D			1	

(AJFEKG[CDBHI||L]), where L is parallel to (CDBHI). Ordering according to the second merger choice, and choosing the output paths merger option (AJFEKGCDBHIL), yields the reordering in Fig. 9.3b.

Notes

1. In the case of all activities belonging to the same cycle, the resulting reordered DSM is Nominal Cyclic DSM and a cyclic DSM by (P10).
2. The condition part of *L2* is equivalent to checking if every activity belongs to a simple cycle containing a_1, such that the cycle could be set to have a_1 as the first activity in the cycle.
3. An example where the conditions do not hold is presented in Fig. 9.4. No activity is linked to all other activities by a simple cycle and therefore the DSM cannot be reordered to Nominal Cyclic DSM.

Theorem 4 (DSM partitioning) *Using the Partitioning procedure in* Sect. 3.3.1. *Any DSM will be reordered to one of the following potential results, using the following procedure type:*

(1) *An acyclic DSM* (D21)*, using only procedure steps* (2) *and* (3)
(2) *A cyclic DSM* (D27) *having all its activities belonging to the same activity cycle* (*the procedure will just identify the cycle and will not do any ordering. Ordering of this case is done in L2*).
(3) *A block-diagonal DSM* (D29)*, including the following types:*
   (a) *Parallel distinct cycles,* $\forall a_i \exists C_k$ (C_k*is a simple cycle*)*, for which* $j \neq k \Rightarrow a_i \in \alpha(C_k) \wedge a_i \notin \alpha(C_j)$*; and if* $a_i \in \alpha(C_k)\ a_j \in \alpha(C_g)\ i \neq j\ k \neq g \Rightarrow (a_i, a_j) \in \emptyset$*. Every activity is part of only one simple cycle; if activity belongs to one cycle it has no links to activity in other cycles.*
   (b) *Sequenced cycles,* $\forall a_i$*if* $a_i \in (C_k) \Rightarrow a_i \notin \alpha(C_j)\, j \neq k$*. In this case, there might be activities that are not in any cycle, but they link between cycles; or there might be links between activities in different cycles, e.g.,* Fig. 9.2c *link* (*C, B*).
   (c) *Shared cycles, activities may belong to several cycles.*

Proof Case (1), 'no feedback links after reordering,' is proved by (T2).

In case (2), since all activities belong to a cycle, each has an input and an output links; thus, steps (2) and (3) of the procedure are not performed. All activities are collapsed and then expanded back with no change of order. We get a cyclic DSM according to (*L2*).

In case (3a), all cycles are collapsed (step (5) of the procedure); since no collapsed cycle has input links, they are just ordered one by one and expanded back. As a result, we get blocks along the diagonal.

In case (3b), cycles are collapsed and the remaining activities and the collapsed cycles are reordered. The reordering of remaining activities and collapsed cycles is to an acyclic DSM by steps (2) and (3) (otherwise, we would have additional cycles). Once the cycles are expanded, we get block along the diagonal linked by forward links between them or forward links to activities.

Case (3c): All the cycles that share activities will collapse into one. If an activity belongs to two cycles, the first cycle will collapse then the collapsed cycle will still be part of a cycle (the second one) and would collapse. After all collapsing steps, the temporal DSM might be either like case (2), the whole DSM is one block; like (3a) distinct blocks or (3b). □

Proposition 11 (Creating Nominal Cyclic DSM) *If a DSM D with N activities has the following properties*

(1) $\forall i,\ 2 \le i \le N-1,\ LFI(a_i) \neq \Phi \wedge LFO(a_i) \neq \Phi$ *(forward input and output links);*
(2) $i = 1, \exists LFO(a_1) \neq \Phi$;
(3) $i = N, \exists LFI(a_i) \neq \Phi$; *and*
(4) *there is a feedback link* $l = (a_N, a_1)$*, then D is a Nominal Cyclic DSM* (D28).

Proof Since each activity has input forward links it must be linked to a former activity (i.e., smaller index), and therefore must be linked to activity a_1 through a forward path. Using similar logic each activity must be linked to activity a_N through a forward path. Hence, every activity is on a path from a_1 to a_N. Every activity is part of a Nominal DSM Cycle (D26), which includes the forward path from a_1 to the activity, the forward part from the activity to a_N, and the feedback link (a_N, a_1); and D is a Nominal Cyclic DSM. □

9.6 Converting DSM to Design Process Matrix

The planned activities may have all types of links. For converting the results of a DSM planning to a process, some additional basic structural definitions are required to assure certain process features. The following requirements were defined in Sect. 9.2 (Karniel and Reich 2007c):

(1) The design process should have Start and End activities being defined;
(2) From any given process state, the process should be able to reach the process End;
(3) When the process terminates (End activity executes), this should imply:
    a. all the design activities (and all their iterations) have completed and
    b. each design activity has been performed at least once.

The following algorithm was suggested in Karniel and Reich (2007b), for converting a DSM plan to a Design Process Matrix, the DPM:

(1) Add Begin and End activities, before and after the design activities, respectively.
(2) Connect Begin activity to all parallel activities that do not have forward link input.
(3) Connect all activities with no forward link output to the End activity.

Many implementations of DSM-based plans (either the initial DSM or the reordered DSM) do not fulfill these requirements; for example, implementing a straightforward direct conversion of a DSM to a process scheme (WF-net). In such conversion, an activity is translated to a transition; and each link is translated to a triad (edge, place, edge): an edge from the source transition (activity); a place; and an edge to the target transition (activity). In order to create a valid WF-net we also need to add starting and termination places (Definition 6).

Definition 30 (*Logic activity*) An activity is a Logic activity if it is connected to design activities or other logic activities, does not have self-iteration links, and its duration is zero.

Note in a Petri net, such activity must "fire" once its preconditions are met.

Definition 31 (*DPM*) The DSM is a DPM iff

(1) There is a Begin logic activity that is linked to and does not have input forward links (from other design activities).
(2) There is an End logic activity linked to, which have no output forward link to other design activities.

Corollary 8 *The DPM is a DSM* (*i.e., all definitions apply*).

Corollary 9

(1) *In DPM, every activity belongs to a forward path from Begin to End.*
(2) *In DPM, every forward link belongs to a forward path from Begin to End.*

Proposition 12 (DPM Links) *Every link in a DPM belongs to a path from Begin to End.*

Proof

(1) Self-iteration link $l_{ii} = (a_i, a_i)$. Since every activity a_i is on a forward path (C9), then adding a subcycle (a_i, a_i) to that path will make the self-iteration link l_{ii} a part of the path from Begin to End.
(2) Feedback link $l_{ij} = (a_j, a_i)$ is part of a cycle $C = (a_i, \ldots, a_j, a_i)$. All other links in this cycle are forward links; thus, are part of a forward path from being to end. Therefore, adding the cycle to the forward path will make l_{ij} part of a required path. □

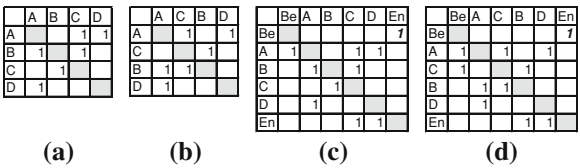


Fig. 9.5 From DSM activities to design process matrix (DPM) **a** Nominal Cyclic DSM **b** DSM with Cycles **c** DPM from DSM (a) **d** DPM from DSM (b)

Proposition 13 (DPM to a Nominal Cyclic DSM) *After a DPM was created* (by D31)*, if we add a feedback link from the End activity to the Begin activity, then, the resulting DPM is a Nominal Cyclic DSM* (D28).

Proof (by P11):

(1) Every design activity has either forward links from other design activities or a forward link from the Begin activity. Respectively, it is either linked to other activities by forward link or linked to the End activity.
(2) In DSM, the first activity has no prior forward link, so it was linked from Begin activity.
(3) In DSM, the last activity has no output forward link, so it was linked to the End activity.
(4) A feedback link from End to Begin was added to the DPM.

Fig. 9.5a depicts a Nominal Cyclic DSM (the two Nominal DSM Cycles are (ABCA) and (ADA); the cycle (BCB) is a subcycle. In this example, activity C has no forward out link; thus, it demonstrates that P11 is a sufficient but not a necessary condition. When reordering the DSM to Fig. 9.5b, the resulting DSM is not a Cyclic DSM since (ABCA) is not a nominal activity cycle (two feedback links).

Applying the proposed algorithm according to P13, we get the DPM in Fig. 9.5c and d, respectively. In Fig. 9.5c, forward link from Begin (Be) was added to activity A only defiand forward links were added from C and D activities to End (En). In Fig. 9.5d, forward links were added from Begin (Be) to A and C activities (meaning they start in parallel), and forward links from B and D activities to End (En) were added. Finally, the (End, Begin) feedback link (bold italic font) was added. In both cases, the DPM is a Nominal Cyclic DSM. □

9.7 The DSM Net

The DSM net is an activity net task net in Westfechtel (1999), that can include design activities and logic activities (e.g., Begin and End). The logic activities implement the process logic. Input Logic (*IL*) defines the preconditions of a design activity. Output Logic (*OL*) defines the decision procedure of sending process control signals (e.g., continue to next activity or make iteration to previous activity).

The motivation for defining logic activities is inherent to the implementation of iterations, as demonstrated in the following simple example.

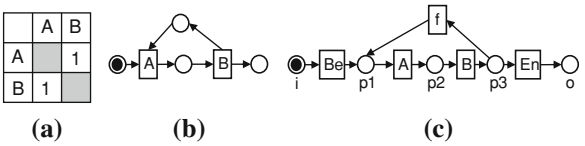


Fig. 9.6 DSM and potential implementations **a** DSM **b** Direct implementation;Petri net (deadlock) **c** A sound implementation; WRI-WF-net

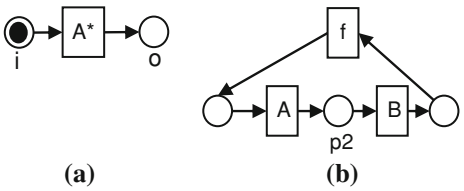


Fig. 9.7 Building the DSM net **a** Initial serial net PN_1 **b** A well-handled net with regular iteration (WRI-WF-net) PN_2

A DSM with activity cycle is depicted in Fig. 9.6a. Its direct conversion to a Petri net is depicted in Fig. 9.6b, resulting in a deadlock. In Fig. 9.6c, a sound process implementation of the cycle is presented, which is a WRI-WF-net.

The process in Fig. 9.6c can be constructed as follows. A serial net PN_1 is depicted in Fig. 9.7a, described by (*i, A*, o*), where places *i* and *o* are the start and termination places and *A** is an activity. Another serial net PN_2 is described by (p_1, *A*, p_2, *B*, p_3), its extended net PN_2* is depicted in Fig. 9.7b. The start and termination places of PN_2 are p_1 and p_3, respectively, and the extended Petri net PN_2* is defined by adding transition *f* between p_3 and p_1. PN_2 is serial and well-handled net; thus PN_2* is well-handled with regular iteration (WRI-WF-net). Refining *A** in PN_1 by PN_2*, according to Definition 13, and renaming *ta* and *tb* to *Be* and *En*, respectively, we get the net in Fig. 9.6c, which is a WRI-WF-net by (D15).

By definition (D15), any transition can be refined by a serial net; thus, the net in Fig. 9.8a can be generated from the net in Fig. 9.6c, as follows. Activity *Be* in Fig. 9.6c is refined to (*Be*, place, *a*1); Activity *A* is refined to (*a*3, place, *A*, place, *a*4); Activity *B* is refined to (*b*1, place, *B*, place, *b*2); Activity *En* is refined to (*b*3, place, *En*); and finally *f* is refined to (*b*4, place, *a*2).

Then, subnets are allocated to logic activities. The subnet consisting of {*a*1, *a*2, *a*3, and *p*1} is defined as *ILa*, the input logic of *A*; *a*4 is defined as *OLa,* the output logic. Using similar allocations for *B, b*1 is defined as *ILb*, and the subnet {*b*2, *b*3, *b*4, and *p*3} is allocated to *OLb.* The result is the Petri net depided in Fig. 9.8b.

In Fig. 9.8c, each sequence of (edge, place, edge) was replaced by a DSM net edge; and the start and termination places were removed to obtain the DSM net. This DSM net can be represented by the extended DPM in Fig. 9.8d. Alternatively, the DSM net is the direct conversion of the extended DPM to a net.

A DSM net can be directly converted to a Petri net (D36). An inverse conversion exists in certain cases (D38 and P14), e.g., from Fig. 9.8b, c. Yet, these

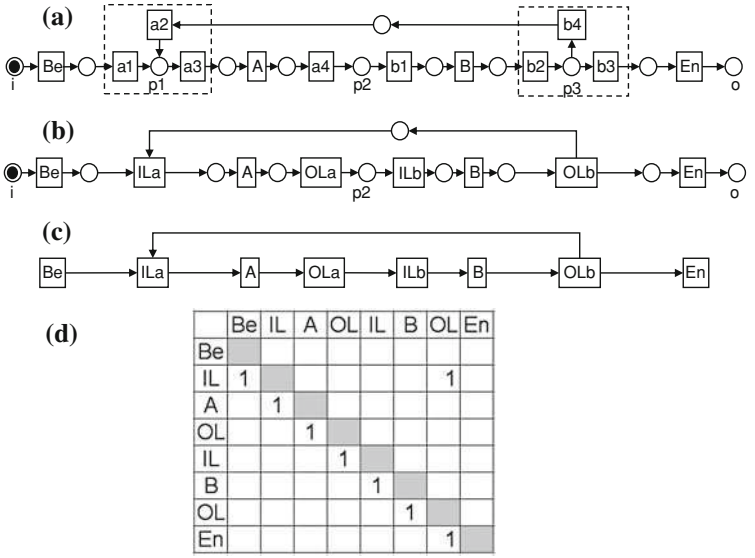


Fig. 9.8 From a WRI-WF-net to DSM net to extended DPM **a** WRI-WF-net expansion **b** Logic activities (Petri net) **c** DSM net (Serial implementation) **d** Extended DPM

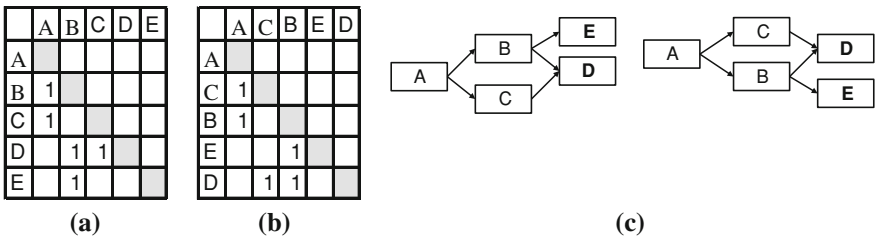
representations are not the same since the logic activities may represent a complex subnet. The complex logic that addresses the requirement of iterations is 'hidden' in the pairs of *IL* and *OL* logic activities.

The overhead of logic activities (e.g., comparing Fig. 9.8c to Fig. 9.6c) is justified when self-iterations are considered, and it simplifies the construction of a process with multiple links.

It is proved that the DSM nets of an acyclic DSM (P16), DSM with self-iterations (P18), and strict block-diagonal DSM (P20), with simple logic (Join-And, Split-And) are equivalent to WRI-WF-nets. Therefore, soundness properties are kept in hierarchical building of the process, i.e., an automated process build approach is possible.

Definition 32 (*DSM net*) DSM net is a tuple $DN = (As, L, In, So, Ta)$, where $a \in As$ are the net nodes, representing design activities and logic activities; $l \in L$ are the net directed edges representing the links between activities. The mappings *In, So, Ta* are the node (activity) index mapping (to DPM), edge (link) source, and edge (link) target, respectively.

The DSM net *DN* and the DPM matrix are equivalent representations. The DSM net can be directly derived from the DPM matrix, and vice versa. Being equivalent representations one can choose either representation form. However, the transformation from DSM via DPM to DSM net is not unique. The DSM matrix may have many orderings that are equivalent to the same DSM net *DN*. As previously described, it is possible to find a permutation function that will reorder the DSM such that each feedback link is the only feedback link in a simple cycle.



(a) (b) (c)

Fig. 9.9 DSM and its corresponding DSM net **a** DSM **b** Reordered DSM **c** Equivalent DSM net

From a process perspective the choice between equivalent DSM orderings is unimportant; yet, differences between reordering that yield different DSM nets are important. We can further assume that the DSM was reordered to an acyclic DSM, a block-diagonal DSM, or a Nominal Cyclic DSM, prior to its conversion to DPM, and to DSM net.

DSM of five activities and five links is presented in Fig. 9.9a. Matrix reordering may yield the DSM in Fig. 9.9b. Both are equivalent to the DSM net in Fig. 9.9c. From a user perspective, the graph positioning may alter; while being the same graph and the DSM net is presented by two equivalent graphical options. Using the descriptive notation used in *L2*, we can describe the different options by (A[B||C][D||E]).

Note The indices of the activities, and therefore the link location indices, are different between the DSM matrices, e.g., link from the activity labeled 'B' to the activity labeled 'E' has the location (2, 5) in Fig. 9.9a and (3, 4) in Fig. 9.9b.

Optimal reordering of the DSM according to Karniel et al. (2005) could result in any of those solutions as minimum (and few others) with the same cost function value. Since the algorithm steps are probabilistic, the resulting solution should not be dependent on the initial guess, unless stuck in local minima.

Corollary 10 *A DSM net is equivalent to a directed graph with nodes* (*transitions, i.e., activities*) *and directed edges* (*links*). *In this graph, there is at most one directed edge* (*at each direction*) *between any two nodes.* $\forall \{a_i, a_j\} \in As$, $(a_i, a_j) = \varnothing$ *or* $(a_i, a_j) = l_{ji} \in L$.

A DSM net can be generated from any DSM; yet, in the current research, we focused on DSM nets that represent DPM, and are based on partitioning ordered DSM, where feedback links indicate activity cycle.

Definition 33 (*Path in DSM net, Elementary*)

(1) Path $C = (x_1,\ldots,x_m)$, $x_i \in As$ is a non-empty sequence from a node (activity) x_1 to a node x_m, such that the directed edge (link) $(x_i, x_{i+1}) = l_i \in L$ for $1 \leq i \leq m - 1$.
(2) C is elementary path iff $\forall x_i, x_j\ i \neq j \Rightarrow x_i \neq x_j$ (i.e., nodes do not repeat).
(3) The operator α is defined by $\alpha(C) = \{x_1,\ldots,x_m\}$.
(4) Operator β is $\beta(C) = \{l_1,\ldots,l_{m-1}\}$.

Elementary path and the operator α definitions are equivalent to (D2).

Definition 34 (*Regular DSM net*) DSM net *DN* is regular iff $\forall l \in L$, $So(l) \neq Ta(l)$.

Definition 35 (*Acyclic DSM net*) Acyclic DSM net is a DSM net *DN* of an ordered DSM iff $\forall l \in L$, $In(So(l)) < In(Ta(l))$.

Conversion of DSM (DSM net) to a Petri net can be easily done in the case of acyclic DSM (DSM net). Converting to a WF-net requires additional links and nodes as subsequently discussed.

Definition 36 (*Converting DSM net to Petri net*) Conversion mapping $DnP = (T, P)$ is defined from DSM net to Petri net, iff

$T(A) \rightarrow T$ is a mapping of DSM activity a_i to Petri Net transition t_i;

$p(l_k) \rightarrow (f_{ik}, p_k, f_{kj})$is a mapping of DSM linkage (with some index k) to a triple: Petri net place node p_k, an edge $f_{ik} = (t_i, p_k)$, and an edge $f_{kj} = (p_k, t_j)$; where $t_j = T(So(l_k))$is the linkage source and $t_j = T(Ta(l_k))$is the linkage target.

Definition 37 (*Converting DSM to WF-net*) A DSM is converted to WF-net, iff

(1) The DSM is reordered by the partitioning procedure and converted to a DPM
(2) The DPM is converted to a DSM net
(3) The DSM net is converted to a Petri net (D36)
(4) A starting place and a termination place are added before Begin and after End activities, respectively.

Definition 38 (*Converting Petri net to DSM net, Inverse conversion*) A conversion from Petri net to DSM net exists, iff after the removal of the special places i (starting place) and its corresponding arc, and the place o (termination place) and its corresponding arc the following mappings exist:

(1) $T^{-1}(T) \rightarrow A$ is the mapping of Petri net transitions to DSM activities.
(2) $\forall p_k \in P$, $p^{-1}(f_{ik}, p_k, f_{kj}) \rightarrow (l_k)$ is the mapping of tuple (edge, place, edge) to a link in a DSM net.

Proposition 14 (Inverse conversion) *A conversion from Petri net to DSM net* (D38) *exists iff* $\forall p \in P$, $|\bullet p| = |p\bullet| = 1$.

Proof Conversion of transition to activity and v.v. is one to one; therefore, we need only to consider places. If each p has only one previous transition and one post transition then the mapping P^{-1} exists, since each place is part of a unique triple that can be converted to a DSM link. Conversely, if the mapping exists, each place is linked only to one previous transition and one post transition. □

Corollary 11 *If a Petri net was defined by conversion of a DSM net, then, it has an inverse conversion to DSM net.*

Proposition 15 (Acyclic DSM to WF-net conversion) *If an acyclic DSM is converted according to* (D37)*, then*

1. *The result is a WF-net* (D6)*;*
2. *The WF-net has no cycles; and*
3. *The WF-net has only Split-And and Join-And logic*.

Proof

1. Following (D6), there are start and termination places by construction. According to (C9) and (P12) all activities and links of the DPM are on a path from Begin to End. Activities are converted to transitions and links to places (and edges). Therefore, every node is on a path from the starting place to the termination place.
2. Since the DSM is acyclic then by (P7), the reordered DSM is also acyclic, and the DPM is acyclic (C8). By (D35) the DSM net is acyclic, and therefore DSM net cannot have cycles (otherwise, the DPM would have feedback links or self-iteration links). The conversion of a link to two edges and a place cannot add cycles, and the resulting WF-net has no cycles.
3. According to the conversion process (D36), $\forall p \in P$, $|\bullet p| = |p\bullet| = 1$; therefore, by the 'firing rules'of the Petri net, a transition waits for all its input places to have a token (Join-And), and once it fires all the output places get a token (Split-And). □

Proposition 16 (Acyclic DSM to WRI-WF-net) *If an acyclic DSM* (D6) *is converted to a WF-net* (D6)*, then the result is a WRI-WF-net* (D15).

Proof the WF-net is acyclic (*P*15). Since $\forall p \in P$, $|\bullet p| = |p\bullet| = 1$, any path from a transition to a place p must be through $\bullet p$, and a path from a place to a transition must be through $p\bullet$; thus, the net is well handled. By (D14), the net is WA-AF net, and by (D15(1)), it is a WRI-WF-net. □

Definition 39 (*Extended DPM*) A DPM is an Extended DPM iff

(1) $\forall a_i$ design activity, $\exists$*ILi OLi,* (input logic and output logic activities, respectively).
(2) The DSM links to a_i are linked to *ILi* (from *OLi, OL* of other activities or from the Begin activity);
(3) DSM links from a_i are linked from *OLi* (to *ILi*, other activities *IL*, or to the End activity);
(4) There is a forward link from *OLi* to $a_{i;}$ and
(5) There is a forward link from a_i to *OLi*.

Proposition 17 (Extended DPM is regular) *Extended DPM* (*D*39) *is a regular DSM* (D20).

Proof Any self-iteration link of a DSM activity becomes a feedback link from the *OLi* to *ILi* (D39), and logic activities have no self-iterations (D30). □

Corollary 12 *The DSM net defined by the extended DPM* (D39) *is a regular net* (D34).

Proposition 18 (DSM with self-iterations and no feedback links to WRI-WF-net) *A DSM with self-iterations and no feedback link can be converted to a WRI-WF-net* (D15).

Proof Since DPM of an acyclic DSM is WRI-WF-net (P16), then, by (D15(3)), an activity (transition) can be replaced by a short-circuited WRI-WF-net, and the result is WRI-WF-net. Therefore, we identify the design activity a_i as t_1; *ILi* and *OLi* as *ta* and *tb*, respectively, in (D13); and the self-iteration link becomes the link which makes *ILi,* a_i and *OLi* a shortcircuited net. □

Definition 40 (*Strict block-diagonal DSM*) A partitioned block-diagonal DSM (D29) is a strict block-diagonal DSM iff $\forall C$, where C is a cycle, then, C is a simple cycle (D9).

Definition 41 (*Enforced cycle simplification*) In a DSM that was reordered by a partitioning procedure, C^* is an enforced simple activity cycle obtained from activity cycle $C = (a_1,\ldots,a_m)$, iff

1. $\forall a_i \in \alpha(C) \Rightarrow a_i \in \alpha(C^*)$ (all the activities in C are included in C^*);
2. $\forall k\, k \notin \{1, m\}, j \neq k \Rightarrow In(a_j) \neq In(a_k)$ (except $In(a_1) = In(a_m)$, the first activity which is also the last activity in C^*, all other activities are included only once in C^*); and
3. Links are added, $\forall k\ k \neq m-1$, $l_k = (a_k,a_{k-1})$ is a forward link, and l_{m-1} the last link is a feedback link.

The enforced simplified cycle emerges from the partitioning procedure. The internal structure of the cycle (the original links) is ignored. This simplification is different from the simplification according to (C4), where subcycles are replaced by the first activity, and the resulting simplified cycle may not include all the activities, which belong to subcycles. For example, in Fig. 9.4, the cycle (*AB-CDCBA*) will be simplified by (C4) to (*ABA*), and by (D41) to (*ABCDA*). Additionally, the last link (*D, A*) does not exist in the original cycle.

Proposition 19 (Forming a strict block-diagonal DSM) *Any partitioned block-diagonal DSM* (D29) *can be converted to a strict block-diagonal DSM* (D40)*, with one feedback link per cycle.*

Proof By (T4), partitioning may result in a block-diagonal DSM. After all cycles have collapsed, any collapsed cycle in this DSM can be replaced by an enforced simple cycle (D41). By (P9), the activities in the simple cycle can be ordered such that there is only one feedback link. □

Proposition 20 (A strict block-diagonal DSM to WRI-WF-net) *If a DSM is strict block-diagonal* (D40)*, then, it can be converted to a WRI-WF-net* (D15).

Proof After the collapse of all cycles according to partitioning procedure, the resulting reduced DSM is an acyclic DSM. By (P16), this reduced DSM can be converted to a WRI-WF-net. Replacing each collapsed node by an enforced simple cycle according to (P19) results in a WRI-WF-net by (D15 (3)). □

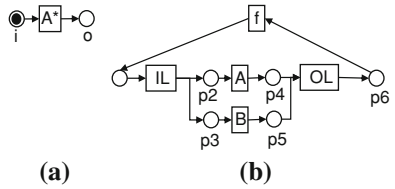


Fig. 9.10 Parallel implementation of activity cycle **a** Initial serial net PN_1 **b** A well-handled parallel net with regular iteration (WRI-WF-net) PN_2

9.8 Conversion Process and Logic Activities

The types of formally proved process logic can describe many practical process plans being discussed in the DSM research literature. The parallel implementation of the formal proofs for a process with two coupled design activities is presented in the following section. The implementation of a simple activity cycle may be done in various ways. Two standard ways are a linearized cycle and a fully parallel cycle.

In the linearized cycle, one activity follows the other, and the last one has an iteration link to the first, as presented in the previous section. In the fully parallel implementation of a cycle all activities start in parallel, and once all are complete, an iteration link may cause all activities to repeat. The latter option was defined in the current work as design block (DB).

The construction of the serialized DB process as a WRI-WF-net for a design activities cycle with two activities is presented in Figs. 9.7, 9.8. The fully parallel implementation of the same DB was built in a similar manner. The starting process is depicted in Fig. 9.10a, which is the same as Fig. 9.7a.

The serial process in Fig. 9.7b is replaced by a parallel version in Fig. 9.10b, where the DB has *IL* and *OL*, and the activities within the DB are linked between the *IL* and the *OL*. The process (starting at p_1 and terminating at p_6) is well handled.

The process is extended by adding the logic activity *f* (regular iteration), which keeps the process as a WRI-WF-net.

Refining A^* in PN_1 by PN_2 according to $D13$ generates the process in Fig. 9.11a. Transitions *ta* and *tb* in Definition 13 are replaced by *Be* and *En*, respectively. The WRI-WF-net represents the parallel implementation of an activity cycle (i.e., DB). If we refine each design activity by a subnet of parallel activities, we can get a structure of parallel implementation of DBs. Such case represents DBs that are in a cycle (actually an activity cycle separated to subcycles).

By expanding the transitions with serial processes (which are WRI-WF-nets), the process in Fig. 9.11b is generated.

The logic activities of the DB (marked by dash lines in Fig. 9.11b) are identified as *ILdb* and *OLdb*, respectively, and replaced yielding the Petri net in Fig. 9.11c. This Petri net has Split-And and Join-And logic of the transitions.

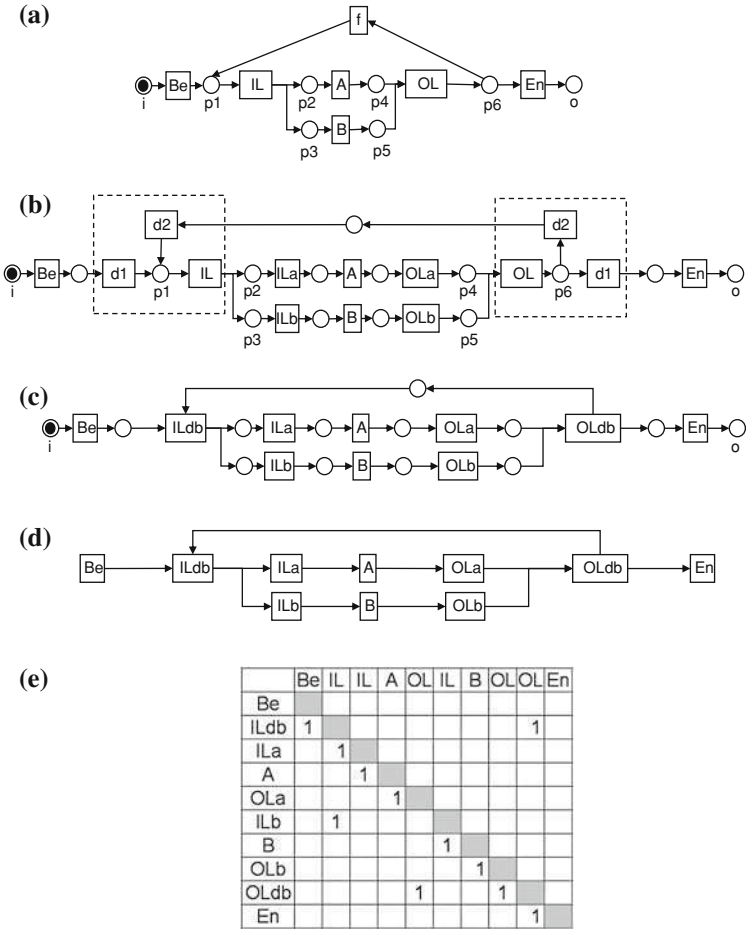


Fig. 9.11 From a parallel WRI-WF-net to DSM net to extended DPM **a** Sound parallel implementation of an activity cycle **b** WRI-WF-net expansion **c** Logic activities (Petri net) **d** DSM net (Serial implementation) **e** Extended DPM

If the logic activities were only transitions, then this process would have represented a deadlock of (*ILdb*). However, the logic activities contain the logic required to avoid deadlocks, as is depicted in Fig. 9.11b.

The logic of *ILdb* activity of the DB describes the case of a single forward and a single feedback input links (like the logic of *ILa* in Fig. 9.8a). Replacing each (edge, place, and edge) in the Petri net with a DSM net edge, we get the DSM net in Fig. 9.11d, which is the equivalent of the DPM in Fig. 9.11e.

The more general case of IL activity logic can be described by $(\prod(F_i)) + (\sum(l_i))$, when using the symbolic formulation Symbolic formulation of Table 7.2 in Sect. 7.5. This logic is schematically depicted in Fig. 9.12a, where FIr_{i} (index

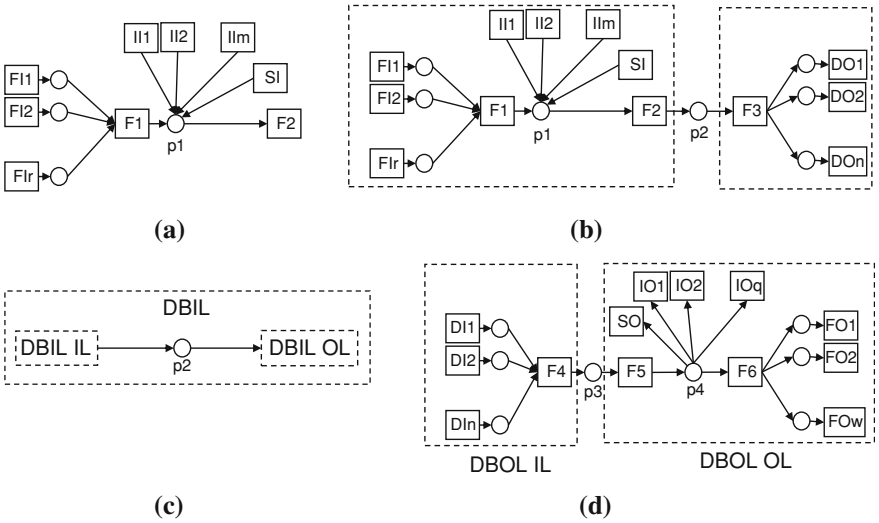


Fig. 9.12 Design block input logic and output logic **a** Input Logic (*IL*) subnet **b** Design block input logic (*DBIL*) explicit output **c** The *IL* and *OL* of *DBIL* **d** Design Block output logic (*DBOL*)

$r_i = \{1 \ldots r\}$) are the transitions accepting forward input links and $F1$ is the implementation of Join-And logic. The transitions IIm_i (with index $m_i = \{1 \ldots m\}$) are the transitions accepting iteration input links from other *OL* activities. Transition *SI* is the self-iteration input side of the self-iteration link. Any of the iteration input transitions, or the self-iteration transition, or the join of forward input transitions ($F1$) can activate $F2$ through the place $p1$. From transition $F2$ there is a split to the parallel activities within the design block. An explicit implementation of Split-And logic is depicted in Fig. 9.12b, where a link to $F3$ is added and the split is done from $F3$ to DOn_i (index $n_i = \{1 \ldots n\}$) transitions, which are linked to the design activities. By adding $F3$ and the explicit split, the logic can be encapsulated into two parts: the *IL* and the *OL* of the DB input logic (*DBIL*) as illustrated in Fig. 9.12c.

The design block output logic (*DBOL*) has a similar structure (which is a mirror of the *DBIL*), depicted in Fig. 9.12c. It can be represented by *DBOL-IL* the Join-And of the parallel design activities within the block and the *DBOL-OL*. Again, the separation to $F4$ and $F5$ transition is not necessary but is convenient for separating the logic segments.

The output logic part (*DBOL-OL*) implementation is described by $(\sum(I_i)) \oplus (\prod(F_i))$, either the self-iteration transition, or any of the iteration output transitions is activated, or all forward output transitions are enabled through $F6$. As further described, this logic is implemented through a probability-based decision procedure.

By assigning the complex logic to IL and OL, as done in the Extended DPM, we make a separation layer between the DSM structure and the applicable logic (undefined in the DSM structure). Such separation has two main benefits:

(a) As indicated in Sect. 7.1. The DSM structure is limited in presenting complex logic.
(b) The separation allows a dynamic assignment of the logic to the *IL* and *OL* logic activities during the process. The importance of this extended dynamic capability of logic changes is further discussed in Sect. 11.11

The partitioning procedure yields serial, fully parallel, and block-diagonal process cases, which are addressed by the DSM literature. The formal proofs of previous sections can be easily applied to the above case where coupled activities within a cycle (fully parallel or blocks in a block-diagonal process) are implemented using serialization or parallelization. In addition, the case of parallel design blocks that is the outcome of an optimal sequencing algorithm (e.g., Sect. 4.2) is also addressed. The implementation of the latter additional case to a process scheme is not addressed in the DSM literature and is a contribution of the current work.

The formal proofs provide the means for implementing all the DSM-based plans into *sound* process schemes. Yet, as previously discussed, the implementation of logic is not unique. Therefore, these logic options do not fully express all the complex process-plan options that are described in the following sections.

9.9 WRI-WF-Nets Competencies and Limitations

The constructive recursive definition of WRI-WF-nets supports hierarchical modeling of the process. Thus, it allows the dynamics expansion of the process scheme as demonstrated in Sect. 12.3.1. Using WRI-WF-net process construction (that includes regular iterations only) is limited, but it addresses the results of the partitioning algorithm and the sequencing algorithm. Therefore, it is suitable for DSM-based process plans.

An implementation of a WRI-WF-net of a parallel process with iteration is depicted in Fig. 9.13a. Each subprocess (*BDG, CEH*) is a closed loop, being a regular iteration of well-handled acyclic process (e.g., p_2-*B*-p_4-*D*-p_6). The process represents the option of each parallel path to iterate. If both paths complete the process may continue to *F* or any of the paths can iterate again. This process is actually an implementation of the reordered DSM in Fig. 9.13b, and is a WRI-WF-net.

A comparable process is defined by the DSM in Fig. 9.14a, where the iterations definitions are different, and instead of two simple cycles, there is a more complex cycle including {*B, D, C, E*}.

The DSM in Fig. 9.14a was converted such that each link in the DSM is replaced by an edge from the transition to a place, and then an edge from the place

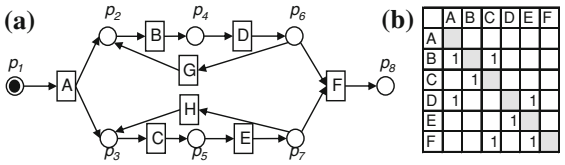


Fig. 9.13 Parallel cyclic activities **a** Sound process **b** Reordered DSM

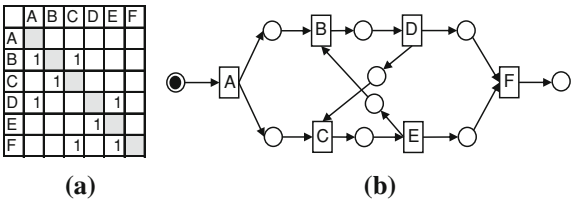


Fig. 9.14 DSM of parallel cyclic activities **a** DSM **b** Conversion to process

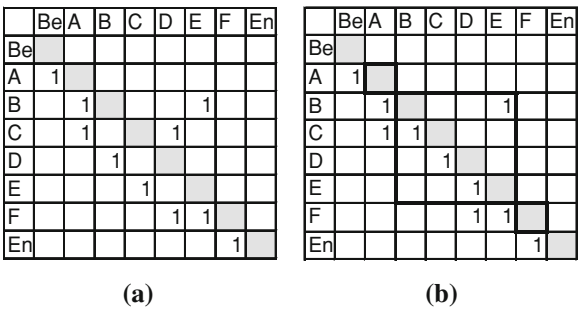


Fig. 9.15 DPM of parallel cyclic activities **a** DPM **b** Strict block diagonal

to the next transition. An initial place connected to activity *A*, and final places connected from activity *F* were added. In this case, *A* and *F* could be considered as design activities or as the Begin and End activities, respectively, being added to basic activity cycle. The resulting process in Fig. 9.14b is a copy of the process in Fig. 34a, i.e., the DSM represents a deadlock.

We assume that *A* and *F* are activities. Using the partitioning algorithm would not change the DSM, which is block-diagonal. With a straightforward implementation, the DSM was transformed to a DPM, Fig. 9.15a. Applying the strict block-diagonal option created the block-diagonal DSM in Fig. 9.15b.

In the strict block-diagonal DSM, the activity cycle of activities (*BCDE*) was converted to a simple cycle (*BCDEB*); a feedback link (*E, B*) was added. The internal links are ignored. Multiple external links that refer to the inner structure of the block are also disregarded.

The resulting DSM-based process can be built as a WRI-WF-net. First, the diagonal blocks were modeled (where DB_1 represented the cycle) in Fig. 9.16a. The net is sequential and therefore a well-handled WRI-WF-net.

Then, the DB1 transition was expanded into a subnet. The subnet is an expansion of iteration net with multiple parallel activities (as in Fig. 9.11a). The resulting process is depicted in Fig. 9.16b. This parallel implementation of the

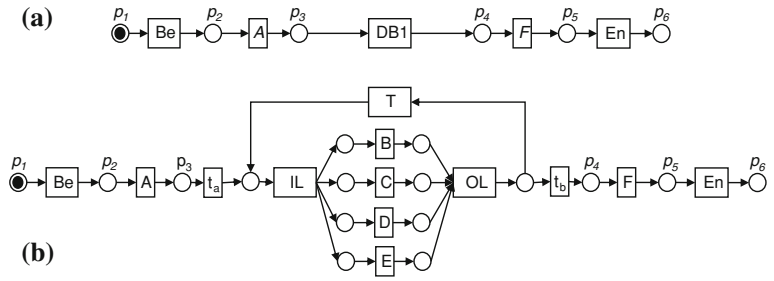
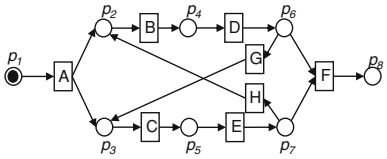


Fig. 9.16 WRI-WF-net implementation

Fig. 9.17 A sound net of iterative parallel paths



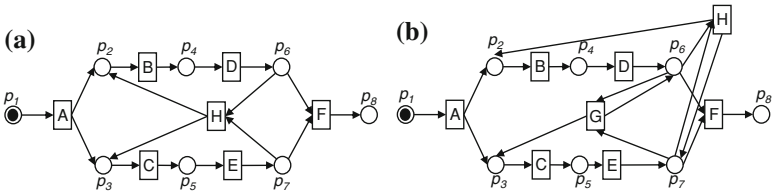


Fig. 9.18 Repetitions of parallel paths **a** Iteration on completion of all **b** Complex iteration logic

simple cycle could be replaced by an iterative serial implementation, which is also a WRI-WF-net.

By using additional logic, the process in Fig. 9.17 represents the internal links of the DSM in Fig. 9.15a. This process (a copy of Fig. 6.8b) is sound. However, it is not a WRI-WF-net, and it could not be created by an expansion of a WRI-WF-net since the iteration edges link different subprocesses.

Additional logic options are presented in Fig. 9.18. The process in Fig. 9.18a is sound: the parallel paths are synchronized and may repeat once both have completed, or the process can continue to activity *F*. In Fig. 9.18b, a more complex logic is demonstrated. Once both paths (*B, D*) and (*C, E*) have completed, the following options apply: (1) the path (*B, D*) may iterate (through *H*); (2) the path (*C, E*) may iterate (through *G*); or (3) the process may continue to *F*.

The above examples demonstrate the usefulness of WRI-WF-nets, as well as their limitations for modeling DSM-based processes. The process examples in Figs. 9.17, 9.18 specify logic options that are not modeled by WRI-WF-nets, but do represent applicable sound process schemes. These processes are *G-Sound* (D12), therefore by P2, when using such construction as subprocesses the resulting

net is *G-sound*. Such logic options are represented in the following sections by implementation rules and business rules. While being *G-sound* can be easily checked for the above examples, a more comprehensive definition of such processes and a way to generate them is required. Further research is required in order to extend the scope of the proofs in the current research to include such logic options as presented in this section.

References

Karniel A, Reich Y (2007a) Simulating design processes with self-iteration activities based on DSM planning. In: IEEE proceedings of the international conference on system engineering and modeling - ICSEM'07, 33–41, Haifa, March

Karniel A, Reich Y (2007b) Managing dynamic new product development processes. In: Proceedings of the 17th annual international symposium of the international council on system engineering INCOSE'07, San Diego, CA, June

Karniel A, Reich Y (2007c) A coherent interpretation of DSM plan for PDP simulation. In: Proceedings of the international conference on engineering design, ICED '07, Paris, August

Karniel A, Reich Y (2011) Formalizing the implementation of DSM-based process planning for NPD. IEEE Trans Syst Man Cybern, Part A 41(3):476–491

Karniel A, Belsky Y, Reich Y (2005) Decomposing the problem of constrained surface fitting in reverse engineering. Comput Aided Des 37:399–417

Robidoux R, Xu H, Xing L, Zhou M (2010) Automated modeling of dynamic reliability block diagrams using colored petri nets. IEEE Trans Syst Man Cybern, Part A 40(2):337–351

Sarbanes P, Oxley MG (2002) Public company accounting reform and investor protection act of 2002, ("Sarbanes (S. 2673) Oxley (H.R. 3763) Act"), Pub. L. No. 107-204, 116 Stat. 745, July 30, 2002. http://frwebgate.access.gpo.gov/cgi-bin/getdoc.cgi?dbname=107_cong_bills&docid=f:h3763enr.tst.pdf. Accessed July 2010

Westfechtel B (1999) Models and tools for managing development processes. Lecture notes in computer science, vol 1646. Springer, Berlin

Chapter 10
Interpretation Using Implementation Rules and Business Rules

10.1 Single Design Activity

A design activity X has a duration property and it may iterate. The activity duration, *duration* (X), is not described by the DSM matrix. The self-iteration probability is $p = Px$, on the DSM diagonal. Translating a single activity DSM to a DPM implies adding the logic activities: Begin, End, Input logic (*IL*), and Output logic (*OL*).

The following *Implementation Rules* (IR) apply:

(IR 1) Logic activity duration is zero and by definition, it does not have self-iterations.

(IR 2) Design activity has one forward input link (from *IL*) and one forward output link (to *OL*).

(IR 3) Logic activities may have multiple input links or multiple output links, but at least one forward input link and one forward output link; with two exceptions: Begin has no input links, and End has no output links.

The case of one design activity is depicted in Fig. 10.1. The DSM (a) is translated to a DPM (b). Logic activities are added: Begin (*B*), End (*E*), Input Logic (*IL*), and Output Logic (*OL*). A marking 1 in the DPM represents either a link with probability $p = 1$ or a logic link (for link from/to design activity to/from logic activity). The self-iteration feedback probability $p = P$ indicates the iteration of X and is set between the Output logic activity and the Input logic activity. The logic applied in the Input logic activity (*IL*) is Join-Or, i.e., signal from Begin or a feedback signal from the Output logic activity. Respectively, the logic applied to the Output logic activity (decision procedure) is Split-Xor; either sending a feedback signal or a signal to the End activity.

The resulting Current process, in Fig. 10.1c, is directly inferred from the DPM. Each off-diagonal element becomes a link. Input logic (i) and Output logic (o) are

A. Karniel and Y. Reich, *Managing the Dynamics of New Product Development Processes*, DOI: 10.1007/978-0-85729-570-5_10,

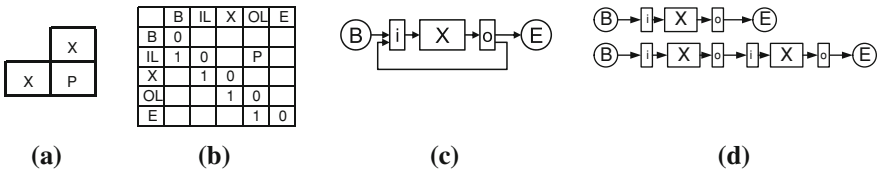


Fig. 10.1 Single design activity. **a** DSM. **b** DPM. **c** C-Process. **d** RT-process

explicitly indicated (they will be implicitly assumed in the following figures). Due to the possible iterations, the Run Time (RT) process in Fig. 10.1d has potentially infinite number of configurations, according to the number of repetitions. Having at least one input forward link and one output forward link (IR3) satisfies requirement 2 (of the correctness criteria, Sect. 9.2), i.e., the process can always complete.

Rule (IR3) implies that all activities are on a route from Begin to End; consequently (after iterations), the process will terminate. The number of iterations is practically limited by a threshold on the feedback probability. Setting $P_{\min}$ as threshold derives the maximal number of iterations, $N_{\max} = \text{ceil}(\log(P_{\min})/\log(P))$ where *ceil* is the nearest greater integer. These limitations satisfy requirement 5.

10.2 Parallel Independent Activities

Two activities may be parallel independent, serial, or coupled. Parallel independent activities process (Fig. 10.2a) has many similarities to one activity process. The X and Y activities can iterate repeatedly according to the Current process, Fig. 10.2c. Results of having one iteration (i.e., two activity executions) at most are presented in Fig. 10.2d.

The parallel independent case requires clear definition of the logic assigned to Begin and End activities. The following Implementation Rules were introduced as a procedure that handles multiple parallel initiation and parallel termination of multiple paths simulation (Karniel and Reich 2009).

(IR 4) The logic of Begin activity is Split-And.
(IR 5) The logic of End activity is Join-And.
(IR 6) If an activity has no previous source (in link), it should be linked from the Begin activity.
(IR 7) If an activity has no target (out link), it should be linked to the End activity.

Both activities (*X* and *Y*) accept links from the Begin activity (to *IL*); with Split-And logic, they start in parallel. Both activities are linked to the End activity (from *OL*); with Join-And logic, the End activity will be enabled once both activities have completed their iterations.

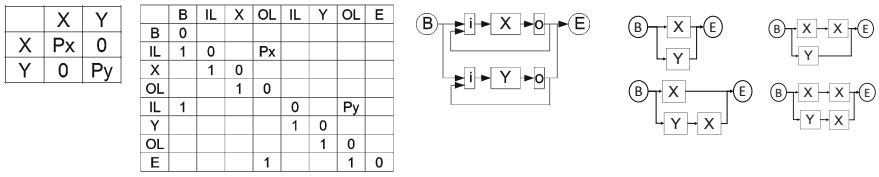


Fig. 10.2 Parallel independent activities (copy of Fig. 9.1 for convenience). **a** DSM. **b** DPM. **c** C-Process. **d** RT-process

For formulating the logic activities, we use the symbolic formulation defined in Chap. 1:

Logic indication: Join ($\Leftarrow$) for Input logic; Split ($\Rightarrow$) for Output logic.
Logic operations: $+$ (OR); $\bullet$ (AND); $\oplus$ (XOR, Exclusive-Or).

The Join operation implies waiting for signals through the input links and is equivalent to Boolean logic equation. Split operation implies a decision procedure, indicating the activities to which signals should be sent. Split-Xor is an ordered choice procedure, e.g., $A \Rightarrow B \oplus C$, means first check sending signal to B, if signal was not sent to B, then send signal to C. Join-Xor is not used in the current work, since the required behaviour is always Join-Or.

The short form of multi-variable logic operations: Multiple-Or $\sum(A_i) = A_1 + A_2 + \cdots + A_n$; Multiple-And $\Pi(A_i) = A_1 \cdot A_2 \cdot \cdots \cdot A_n$; and Multiple-Xor $\otimes(A_i) = A_1 \otimes A_2 \otimes \cdots \otimes A_n$, where A_i represents a link signal, which can be a forward link F_i, or Iteration (feedback) link I_i. Using the above symbols, the following formulation is presented for the implementation of Begin and End:

$$\text{Begin} \Rightarrow \prod(F_\mathrm{i}) \tag{10.1}$$

$$\text{End} \Leftarrow \prod(F_\mathrm{i}) \tag{10.2}$$

Forward links and feedback (iteration) links may have distinct logic operands; the following basic Implementation Rules are defined for both *IL* and *OL*.

(IR 8) Forward links to design activities (lower part of the matrix) have AND logic, on first iteration.
(IR 9) Forward link to the End activity, has XOR (Exclusive-OR) logic with the other links.
(IR 10) Feedback (iteration) links have OR logic.

In simple cases, the Input logic defines accepting signal from all forward links (Join-And), i.e., the activity starts once all its precedent activities have completed; or a signal from any of the feedback links is available; or both (cf. Eq. 10.3). The Out-logic procedure may sends signals to any feedback link, or (exclusively) send signals to all forward links, but not both (cf. Eq. 10.4).

$$IL \Leftarrow \left(\prod (F_{\mathrm{i}})\right) + \left(\sum (I_{\mathrm{i}})\right) \tag{10.3}$$

$$OL \Rightarrow \left(\sum (I_{\mathrm{i}})\right) \oplus \left(\prod (F_{\mathrm{i}})\right) \tag{10.4}$$

10.3 Serial Activities

Serial activities in a design process might be the result of standardization (Sered and Reich 2006), e.g., while the design of a standard part A may influence the design of B, changes in the design of B will not affect the design of A (otherwise A is not standard). Figure 10.3 describes an example of translation stages of a serial process from DSM to RT-process. The link from activity *X* to activity *Y* has probability *Pxy* in Fig. 10.3a. It is translated to a link from Output Logic activity (*OL*) of *X* to Input Logic activity (*IL*) of *Y* in Fig. 10.3b.

The DnPDP correctness requirement 3b (Sect. 9.2), indicating at least one activity execution, imposes different logic requirements on the first execution of the design activity versus further iterations. For example, on its first execution, the activity cannot send forward signal to End activity; sending a forward signal to the next serial activity is required, otherwise the next activity will not be performed. Sending signals to both (next activity and End activity) hinders IR 9 and can cause incorrect processes (see example in Sect. 10.7.2, Fig. 10.10e).

The interpretation of the forward link with probability *Pxy* in case of iterations is not unique. The issue to be considered is: If *X* iterates, can *Y* start in parallel, or should it wait until *X* completes all its iterations. The interpretation may be one of the following Business Rule options

(IR 11): Output logic options: sending completion signal(s) to following serial design activities may follow one of the BRs:

(BR 11.1) Sending only once the activity has completed all its iterations.

(BR 11.2) Sending once the activity has completed its first execution (i.e., early start of next activity); yet, sending a signal to End activity can be done only once all iterations have completed (i.e., IR 9).

The first case (business rule BR 11.1) is a serial process with self-iterations, depicted in Fig. 10.3c. Since activity *Y* may start only after the completion of all iterations of activity *X*, and iterations are serial, the whole process is serial, Fig. 10.3d. In this case, the probability $p = Pxy$ has no simulative meaning. It only indicates a forward process link; thus, it could be translated to probability $p = 1$. This rule is the default rule and it complies with the basic Out-logic rule presented in Eq. 10.4. Typically, the logic of Eq. 10.4 is the default implementation of serial activities with self-iteration in a WF-net.

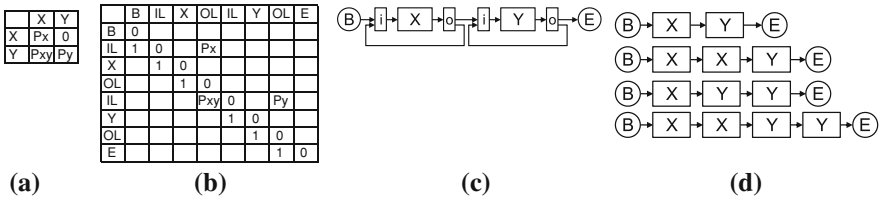


Fig. 10.3 Serial activities process: from DSM to RT-process. **a** DSM. **b** DPM. **c** C-Process. **d** RT-process

10.4 Serial Activities with Parallel Execution

In the latter case (BR 11.2), there might be iterations of the previous activity *X* that are executed in parallel to activity *Y*. The implementation of this case requires additional rules, which are useful for other parallel cases as well. Such possibility is not studied in existing literature.

(IR 12): Output logic: signal to End Activity. Second (or subsequent) execution of an activity (e.g., *X*) may send signal to End Activity, while the following serial activities (e.g., *Y*) have started execution (or have completed):

(BR 12.1) On the second (or subsequent) execution, the activity must be followed by its next serial activities.

(BR 12.2) On the second (or subsequent) execution of the activity, the next activities may follow or the End activity may follow (not both, subject to IR 9).

Four BR combinations change the Output logic and are manifested in the DPM structure. The first was the case of BR 11.1 and BR 12.1, already described by Eq. 10.4. The combination BR 11.1 and BR 12.2 is formulated in Eq. 10.5; BR 11.2 and BR 12.1 in Eq. 10.6; and BR 11.2 with BR 12.1 in Eq. 10.7 (see BR Map in Sect. 10.6).

$$OL \Rightarrow \left(\sum (I_{\mathrm{i}}) \oplus \sum (F_{\mathrm{i}})\right) \oplus \mathrm{End} \tag{10.5}$$

$$OL \Rightarrow \left(\sum (I_{\mathrm{i}}) + \prod (F_{\mathrm{i}})\right) \oplus \mathrm{End} \tag{10.6}$$

$$OL \Rightarrow \left(\sum (I_{\mathrm{i}}) + \sum (F_{\mathrm{i}})\right) \oplus \mathrm{End} \tag{10.7}$$

Allowing early start option (BR 11.2) is deriving Run Time options (IR 13), which are not described by the DPM model. These options can be addressed only in the RT-process scheme.

(IR 13): Iterations of the same activity cannot occur in parallel:

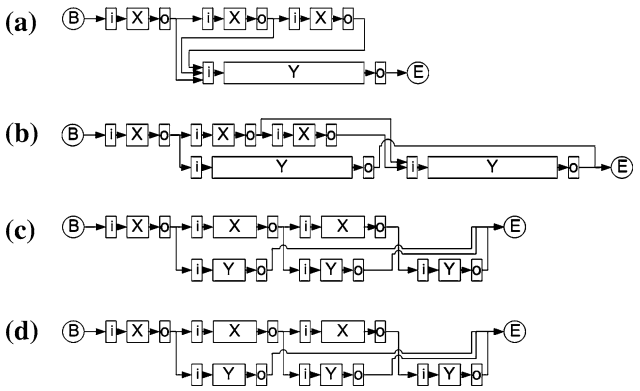


Fig. 10.4 Serial activities RT-process BR 11.2 + BR 12.1. **a** RT: BR 13.1 $d(x) < 0.5 \cdot d(y)$. **b** RT: BR 13.2, $d(x) < 0.5 \cdot d(y)$. **c** RT: BR 13.1, $d(x) > d(y)$. **d** RT: BR 13.2, $d(x) > d(y)$

(BR 13.1): While the activity is executing, input link signals are directed to this activity (i.e., to its input Logic activity).

(BR 13.2): While the activity is executing, input link signals are directed to the next iteration of the activity (next iteration cannot start until current iteration has completed).

Business Rule (IR 13) has correctness implications, avoiding activities proliferation in simulation. In general, it is assumed that the same resource is doing additional iterations of the same design activity (this assumption can be used to decrease the duration of iterative activities, due to learning). If two resources perform the design of the same component (e.g., developing several concepts in parallel), then these activities should be defined distinctly (i.e., not iterations of the same activity). In the context of administrative processes, the options described above were defined as Lack of Synchronization conflict (Sadiq and Orlowska 1999); i.e., multiple requests for the same activity. The first option (BR 13.1) entails defining the consequences of the additional iteration rework. Typically, the activity duration will just increase.

Recalculating the activity duration is similar to overlapping (Cho and Eppinger 2005) and can be addressed by the same approaches: defining how much of the work has been completed, what is the impact of the change, and how much work remains to be done.

The implications of applying the rules are dependent on the activities relative duration. The case where *duration* $(X) >$ *duration* (Y) yields similar Run Time processes for both BRs, Fig. 10.4c and d, respectively. The more interesting case where *duration* $(X) < 0.5^*$ *duration* (Y) is depicted in Fig. 10.4a and b. The relative length of the activity box graphically presents the relative duration. This presentation is imprecise since Logic activities duration is zero. Yet, the Run Time process is presented as if it were sketched over the time axis. The exhibited difference in Run Time process, for self-iterations, due to differences in activities

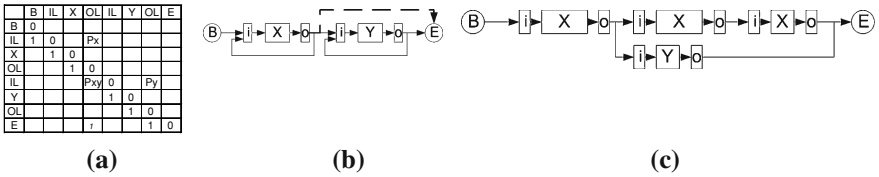


Fig. 10.5 Serial activities BR 11.2 + BR 12.2. **a** DSM. **b** DPM BR 14.1. **c** C-Process BR 12.2

duration, is a new result, not mentioned in the literature. Self-iterations are common in NDP processes, thus modeling, simulating, and understanding their implications is important.

The DPM structure and the Current process scheme, after adding the link to the End activity (at second execution or later), are depicted in Fig. 10.5a and b, respectively. The main structural difference compared to Fig. 10.3 is the additional link from *OL* (Out-Logic) activity following *X* to the End activity (applicable only to the second iteration of *X* activity). The dashed line in the C-process indicates, respectively, that the logic is applicable in the second execution (or subsequent). The process cannot end after the first execution of *X* without performing activity *Y* (correctness criterion 3(b)). This option is always economic in terms of time; yet, quality may degrade since the implications of the change (in *X* design activity on *Y* design activity) are not re-checked. This option is more general than the previous one as all the RT-processes generated in the previous case (BR 11.2 and BR 12.1) could be generated by this case, but not vice versa. An example of a different Run Time process is depicted in Fig. 10.5c.

10.5 Coupled Activities

The coupled activities case may have serialization requirements (e.g., testing activity should serially follow design activities).

(IR 14): Coupled activity execution starts:

(BR 14.1) (serialization) Coupled activity may start after its previous activity (according to DSM) has completed at least once.

(BR 14.2) (parallel) coupled activity may start in parallel to all the other activities in the same activity loop.

Serialization (BR 14.1) logic is described by the four logic-combination cases previously discussed in Sect. 10.4 (see BR Map at Sect. 10.6). The link *Pyx* is assigned from the *OL* of *Y* activity to the *IL* of *X* activity in the DPM, Fig. 10.6b. Serialization is defined by setting $Pxy = 1$, i.e., *Y* must follow *X*. The case of applying BR 12.2 (ability to send signal to End after second execution or to next activities) is depicted in Fig. 10.6b and c. Starting coupled activities concurrently

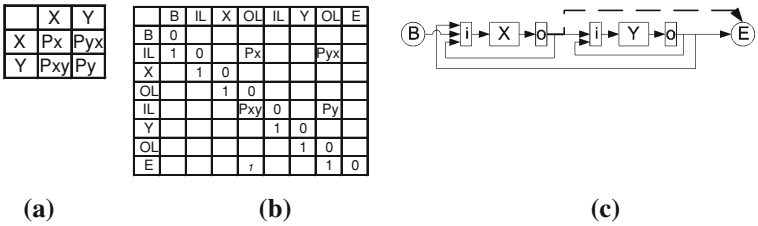


Fig. 10.6 Serialized Coupled activities, BR 14.1. **a** DSM. **b** DPM BR 14.1. **c** C-Process BR 12.2

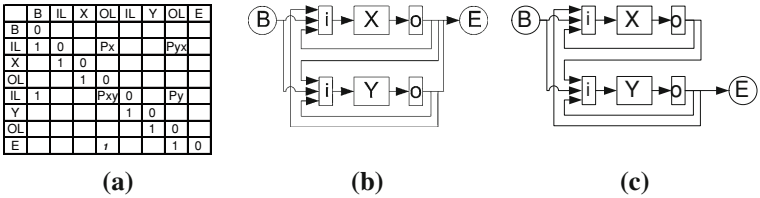


Fig. 10.7 Parallel Coupled activities, BR 14.2. **a** DPM BR 14.2 + BR 15.2. **b** C-Process BR 15.2. **c** C-Process BR 15.1

(BR 14.2) results in processes whose Output logic is similar to the case of BR 11.2 (parallelization of the iterations of serial activities), but with different Input logic. The Input logic for BR 14.2 is formulated in Eq. 10.8 (replacing Eq. 10.3).

$$IL \Leftarrow \text{Begin} + \sum (I_i) + \sum (F_i) \text{ in loop } + \prod (\text{other forward links}) \quad (10.8)$$

The difference is implemented by setting a link with probability $p = 1$ from Begin, Fig. 10.7a. The options described in IR 12 are respectively replaced by the options of IR 15.

IR 15: Sending signal on second (or later) completion of a coupled design activity is done according to one of the following BRs:

(BR 15.1) A coupled design activity should link to the next activity on its completion.

(BR 15.2) A coupled design activity may link to the End activity on its completion.

The DPM in Fig. 10.7a was assigned with rule BR 15.2, i.e., there is a link from *OL* of *X* to End (after second execution), and the resulting C-process is depicted in b. Using BR 15.1, the latter link (to End activity) would not be included and the resulting C-process would be as in c.

The parallel implementation according to IR 14 assumed either serialized of a fully parallel execution of the coupled activities, i.e., DB. Both options were formally proved to fulfill the soundness criterion, by conversion to WRI-WF-net (Proposition 18 and Proposition 20, respectively).

Additional implementation of coupled activities (which is more straightforward, but requires a proof) allows for serial and parallel execution, where feedback

links are applied directly as indicated (using probability matrix). In such case, IR 11 should be modified, to reflect late start (BR 11.1) or early start (BR 11.2). The *OL* of the completed activities and the *IL* of proceeding activities should be aligned; such alignment is inherent if the activities are serial.

IR 16: *OL* and *IL* options for early or late start of proceeding activity may follow one of the BRs:

(BR 16.1) (late start) *OL*—Sending forward signal only once the activity has completed all its iterations. *IL*—Starting the activity execution once all iterations of preceding activities have completed.

(BR 16.2) (early start) *OL*—Sending forward signal once the activity has completed its first execution; yet, sending a signal to End activity can be done only once all iterations have completed (i.e., IR 9). *IL*—Starting execution once all the preceding activities have completed at least one execution (each).

The implementation of BR 16.2 is simple when using a decision algorithm, and complex when using a WF-net due to the *OL* Split-Or logic and the need to identify iterations of activities that are not directly linked to the End activity.

Implementing BR 16.1 is easy to implement by using the default logic of a WF-net, but is complex to implement as a decision algorithm for the *IL* due to the need to evaluate iterations. Self-iterations are easy to follow, but iteration of an activity due to iteration of its preceding activities is more complex. Starting according to an accepted forward link requires checking that all the activities (before its direct preceding activity) did not iterate; otherwise, the direct preceding activity will iterate. Yet, it is still possible that some previous activity will iterate after the decision to start has been made and the activity had started its execution.

The implementation of the process aware logic (i.e., that takes into account the process status) is more complex than the logic expressions, as the decision is no longer local, but requires knowledge of the whole model (identifying previous activities and their relations) and process status. The process performance is derived from the combinations of the business rules (e.g., early or late start with serialized or parallel activities). Simple examples are demonstrated in Sect. 11.7 and integrated examples in Chap. 1.

10.6 Business Rules Map

The relations between Implementation Rules, Business Rules, Input Logic, and Output Logic equations are shown in Fig. 10.8. Input logic is described by Eq. 10.3 (for serial or serialized activities) and Eq. 10.8 (for coupled activities), respectively. The Output logic of the various cases is formulated in Eqs. 10.4–10.7.

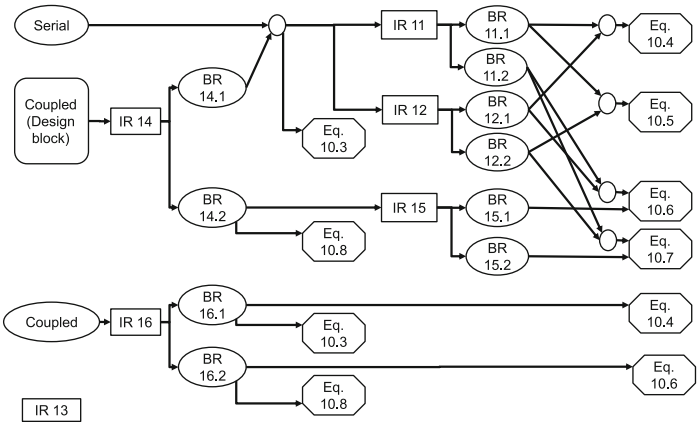


Fig. 10.8 Relations between implementation and business rules

Round nodes indicate "Join-And", i.e., in the case that both BR11.1 and BR12.1 are applied then the logic is according to Eq. 10.4.

10.7 Business Rules Logic Examples

A formal validation of the presented Implementation Rules and Business Rules is left for future research; yet, the following examples demonstrate the implications of the rules by counter examples. Additional implications of rules implementation while utilizing self-iterations are presented in the next section, extending the justification of the above rules.

10.7.1 Self-Iterations

The implication of utilizing the above rules in the case of self-iterations (Karniel and Reich 2007) was fully investigated for the case study of two design activities. The example provides additional insight into iterative process modeling. This analysis elucidates the behavioral richness and the complexity of implementing self-iterations.

First, an enumeration approach was used to generate potential processes, and then the cases were filtered according to the DnPDP correctness criteria (Sect. 9.2). Then, examples of process logic problem are depicted, justifying the definition of some of the criteria.

Applying one iteration at most (maximum two executions) resulted in 74 distinct logically-correct Run time processes, which exhibited some non-trivial

Link Option	Process scheme	Description	Applicable rules/ Eq.
BX	*B-X*	Default link to first activity from B	IR 6 + BR 14.1 / Eq. 10.3
Bs	B- *X*, B- *Y*	Split: from Begin to multiple activities in parallel	IR 6 + BR 14.2 / Eq. 10.8
XY	*X–Y*	Final iteration of an activity links to next activity	BR 12.1 or BR 15.1 / Eqs. 10.4, 10.6
XE	*X-E*	Final iteration of an activity can link directly to E	BR 12.2 or BR 15.2 / Eqs. 10.5, 10.7
YX	*Y–X*	Final iteration of an activity which can feedback to other activity in loop	BR 14.2 / Eqs. 10.6, 10.7, 10.8
YE	*Y-E*	Final iteration of an activity without target should link to E	IR 7 / Eqs. 10.4, 10.5, 10.6, 10.7

Fig. 10.9 Optional run time links

behavioral aspects. The logic verification analysis was done using the implementation rules applied to all the 128 potential process combinations.

10.7.2 Run Time Cases Enumeration

For maximal number of iterations $N_{\max} = 0$, we have exactly one execution of each design activity (as enforced by correctness criterion 3(b)). The number of activities in this case is exactly four (Begin, *X*, *Y*, End). Process generation options for building a Run Time process with two design activities are summarized in Fig. 10.9.

There are two options for starting the process, link from *B* to *X* (BX) or split to both design activities (Bs); two options to link *X* after its completion: link to *E* (XE) or link to *Y* (XY); and similar options for *Y* (YE, YX). The options presented for *X* and *Y* are the only applicable options of the final iteration of an activity (i.e., it cannot iterate to itself in this case as $N_{\max} = 0$). Some options are applicable only for specific BR and specific equations, as indicated.

There are eight potential RT-processes, utilizing the three choices of "process locations": Begin (2 options), *X* (2 options), *Y* (2 options). However, correctness requirements filter out some of them. The examples are depicted in Fig. 10.10. For easier display, we assume the Input and Output logic are embedded in the activity (short RT-process description).

The option (BX + XE), Fig. 10.10a, is wrong since *Y* activity is not executed. Using '*' as 'do not care' symbol, we can define the choices as {BX, XE, '*'}, i.e., the process is equivalent for both choices of the *Y* option (YX or YE), and in both cases is wrong. In general, any serial process combination, which includes only iterations of activity *X*, is incorrect (violating correctness criterion 3b).

The serial case {BX, XY, YE} is depicted in Fig. 10.10b. The combination of choices {BX, XY, YX} and {Bs, XY, YX} depicted in Fig. 10.10c always yields a wrong process, since the process does not reach the End activity, violating

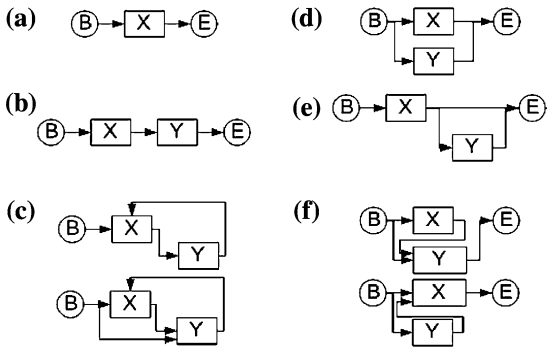


Fig. 10.10 Run Time process without iterations: Correct and wrong options. **a** Wrong: No *Y* activity. **b** Correct: Serial. **c** Wrong: $(X - Y) + (Y - X)$, no End. **d** Correct: Coupled. **e** Wrong: $(X - Y) + (X - E)$. **f** Correct: Coupled, BR 13.1

correctness criterion (1). This combination occurs for the two options of process start {'*', XY, YX} (where '*' indicates either BX or Bs) and is further referred to as (XY + YX).

Correct parallel options are depicted in Fig. 10.10d ({Bs,XE,YE}) and (f) ({Bs,XY,YE} or {Bs,XE,YX}). In the latter case, there should be a difference in the activities duration (e.g., duration (X) < duration (*Y*)), for applying rule BR 13.1 (rule BR 13.2 cannot apply since there are no additional iterations). If both activities have exactly the same duration, the options in Fig. 10.10f are reduced to the case in Fig. 10.10d. The eight RT-process cases described above: four correct cases (in Fig. 10.10 b, d, and f) and four wrong cases (two in Fig. 10.10a and c) are all the potential cases that can be derived from the options in Fig. 10.9.

An additional RT-process is depicted in Fig. 10.10e. Such process cannot be generated from the options presented in Fig. 10.9; *X* cannot link to both *Y* and End. Using BR interpretation, proceeding to End activity cannot be done in parallel to proceeding or feedback to other activities (Xor logic). Proceeding to End should mean completion, while feedback or continuing to other activities has a contradicting intent. Note: Such option is allowed only after second execution of the activity (IR 12), which is not applicable in the current case.

10.7.3 Run Time (RT) with Self-Iterations

Allowing one iteration of each activity, $N_{max} = 1$, either self-iteration or iteration due to the other activity, yields maximum six activities: Begin, first *X* iteration, first *Y* iteration, last iterations of *X* and *Y*, and End. In addition to the link options described in Fig. 10.9, the interim iterations (i.e., first activity execution) have four options detailed in Fig. 10.11.

There are five choices to make based on the linkage options detailed above (five "process locations"): Begin activity (two options), first activity iteration *X*, and first activity iteration *Y* (four options each, combining Figs. 10.9 and 10.11), and final activity iteration *X* and *Y* (two options each). The combinations yield 128

Link Option	Process scheme	Description	Applicable rules/equations 10.x
XX	*X-X*	Self-iteration	Eqs. 10.4, 10.5, 10.6, 10.7
YY	*Y–Y*	Self-iteration	Eqs. 10.4, 10.5, 10.6, 10.7
Xs	$X \begin{cases} X \\ Y \end{cases}$	Split: Iteration and forward link, using the applicable BR	BR 11.2 or BR 14.2 Eqs. 10.4, 10.6
Ys	$Y \begin{cases} X \\ Y \end{cases}$	Split: Choosing multiple iterations (feedback links have OR logic)	Eqs. 10.4, 10.5, 10.6, 10.7

Fig. 10.11 Additional linkage options for iteration

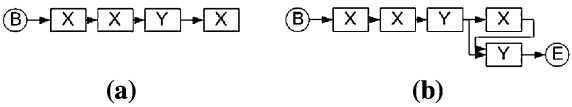


Fig. 10.12 Inadequate: 3rd *X* iteration. **a** Inadequate YX choice. **b** Inadequate Ys choice

potential basic process structures, which include many incorrect processes. The cases where both iterations of an activity are applicable have additional linking options that increase the number of RT-process schemes.

Many potential RT-processes are wrong due to the combination (XY + YX). Such choices, indicated by {'*', '*', '*' XY, YX}, appear in quarter of the cases. However, the process may end (as a correct process), before reaching these choices (e.g. {Bs, XE, YE, '*', '*'}), or might become wrong even before reaching these choices. Therefore, the actual number of incorrect processes due to such choice is 16. Further 16 wrong cases are described by the process path *B-X-E*, using the selection {BX, XE, '*', '*', '*'}. The latter 16 cases include four cases of the type (XY + YX), which are counted as part of the *B-X-E* combination, since the process does not reach the choices (XY + YX). Eight inadequate options, with three executions of *X* (only two allowed), are included in the process path *B-X-X-E*: {BX, XX, '*', XE, '*'}.

Additional four cases are depicted in Fig. 10.12. Two of the cases refer to the selection {BX, XX, XY, YX, '*'}, in Fig. 10.12a. These two cases are equivalent and yield an inadequate process due to three *X* activity occurrences. The other two cases refer to the selection {BX, XX, XY, Ys, '*'}, Fig. 10.12b. In both, after *X* has already iterated, links from *Y* are connected to both *X* and *Y*; thus, a third iteration of *X* is required. The case {BX, XX, XY, Ys, YE} is inadequate since only two executions of *X* are allowed (and Ys includes a third one); the case {BX, XX, XY, Ys, YX} is a wrong process case (XY + YX) as discussed above.

Besides wrong or inadequate options, there are distinct options, which result in the same RT-process; hence, further reducing the count of distinct correct RT-processes. For example, the selection {BX, XY, YE, '*', '*'}, representing four options, is actually equivalent to only one correct process depicted in Fig. 10.10b. Selecting the four options of {Bs, XE, YE, '*', '*'} produces the basic parallel case of the process in Fig. 10.10d. Reminder the latter two "process locations" have two options each.

Applying BR 13.1 or BR 13.2 may add distinct Run Time results. For example, if both iterations of *Y* exist, then the link *XY* may result in $X - Y(1)$ or $X - Y(2)$, where $Y(i)$ indicates the iteration number.

Summary, we have started with 128 *potential* RT-process structures, minus $(16 + 16 + 8 + 4)$ wrong or inadequate options, yielding 84 correct structure cases. Of these, 28 cases are repeated (i.e., equivalent to others). The application of business BR 13.1 or BR 13.2 adds 18 distinct RT cases (assuming different durations of *X* and *Y* activities). In total, by applying the different combinations of BRs to a DSM with two design activities and a maximum of two executions for each activity, we got 56 distinct RT-process structures (i.e., combinations of activities), and total 74 different RT-processes, due to additional combinations of links.

No way was found to enumerateEnumeration the correct cases automatically, due to the complexity of the combinations. In order to do so, there is a need to develop a testing tool that would do so. The approach taken in this work was to embed the rules in the creation process such that only correct processes can be generated.

10.8 Logic Verification Issues

The DSM (and DPM) structure does not fully define the process logic and cannot represent both input logic and output logic in the general case (Karniel and Reich 2009; see Sect. 7.1). Setting logic as additional information, by using additional matrices, using rules, or using logic formulation, is non-trivial. Not every set of rules may apply. The latter assertion is demonstrated by the following examples that provide more insight into iterative process simulation logic and implementation.

A short version of the DPM is used: Begin logic activity (B) and End activity (E) are explicitly represented; *IL* and *OL* are implicitly assumed before and after the design activities, respectively.

Parallel activities example is depicted in Fig. 10.13a. The option of using Split-Or logic for Begin and Join-Or logic for End ($\boldsymbol{B} \Rightarrow \boldsymbol{X} + \boldsymbol{Y}$ and $\boldsymbol{E} \Leftarrow \boldsymbol{X} + \boldsymbol{Y}$) respectively**,** is legitimate in a general process. However, in the context of design processes it is incorrect since we assume that every activity must be performed at least once. The logic $\boldsymbol{B} \Rightarrow \boldsymbol{X} \oplus \boldsymbol{Y}$ will always cause one of the activities not to start. In general, the output logic of forward links, being performed first time, should always be Split-And (to ensure that next activities execute at least once).

When activities may have self-iterations, we must separate the logic applied to links to other activities from the logic applied to the link to the End activity. The logic of the separation must be Split-Xor (e.g., $(\sum(I_i)) \oplus (\prod(F_i))$ in Cho and Eppinger 2005, Table 7.2). Otherwise, the process might terminate before all design activity iterations have completed.

A serial case without self-iterations is depicted in Fig. 10.13b. The link from Begin to *Y* is not required according to the DPM generation rules and is added to

case	DPM	Correct Logic	Incorrect Logic examples
(a)	B: 0; X: 1, 0.1; Y: 1, 0.1; E: 1, 1, 0	**B ⇒X•Y ;** **E ⇐X•Y ;** **X ⇒X⊕E ;** **Y ⇒Y⊕E ;**	**B ⇒X+Y ; B ⇒X⊕Y;** **E ⇐X+Y ;** **X ⇒X+E ; X ⇒X•E ;** **Y ⇒Y+E ; Y ⇒Y•E ;**
(b)	B: 0; X: 1, 0.0; Y: 1, 1, 0.0; E: 1, 0	**B ⇒X•Y;** **Y ⇐B•X;**	**Y ⇐B+X;**
(c)	B: 0; X: 1, 0.1; Y: 1, 1, 0.1; E: 1, 0	**X ⇐X+B;** **X ⇒X⊕Y;** **X ⇒X+Y;** **Y ⇒Y⊕E;** **(E ⇐X•Y)**	**Y ⇒Y+E;** **Y ⇒Y•E;**
(d)	B: 0; X: 1, 0.1, 0.1; Y: 1, 1, 0.1; E: *1*, 1, 0	**X ⇐B+Y ;** **X ⇒X⊕Y ;** **X_2⇒X⊕Y⊕E ;** **X_2⇒(X+Y) ⊕E;** **Y ⇐B+X+Y ;** **Y ⇒(X+Y)⊕E ;**	**X ⇐B•Y ;** **X2 ⇒(X⊕Y)+E ;** **X2 ⇒X+Y+E ;** **X ⇒Y•E ;** **Y ⇒X+Y+E ;** **Y ⇒X•Y•E ; andmore**

Fig. 10.13 Process logic examples

demonstrate the input logic of *Y*. The input logic must be Join-And of all forward links (i.e., *Y* should wait to all previous activities to complete), as it is serial.

The case of adding self-iterations is depicted in (c); *X* and *Y* activities have self-iteration links. According to Eq. 4.3, the Input logic of *X* is $IL \Leftarrow B + X$, i.e., signal from Begin or iteration (from Out-logic activity). In a similar manner, the input logic of *Y* is $IL \Leftarrow X + Y$. The Output logic of *Y* is Split-Xor between iteration and the link to End activity (Implementation Rules IR 9); other options are incorrect. The output logic of *X* has two options: either all *X* iterations should complete before continuing to *Y* (BR 11.1) $(X \Rightarrow X \oplus Y)$ or *Y* might start in parallel to an iteration of *X* (BR 11.2).[1]

Since all iterations should terminate, the End activity should wait for all *X* iterations to complete, though there is no indication of a link from *X* to End. Without such logic the process may end once *Y* completes, while *X* is still iterating (and may cause additional iterations of *Y*). The analysis is quite similar to the examples given in van der Aalst and van Hee (2002). Yet, waiting for activity iteration with no indication of link to End activity is regarded a flawed situation in a simple Petri net.[2]

The case of coupled activities with parallel start and early termination is depicted in (d). The input logic of *X* is (typically) defined as Join-Or of forward

[1] Note: the case $X \Rightarrow X \bullet Y$ is a suboption, i.e., *Y* must start with every iteration of *X*.

[2] This can be solved in high-level Petri net by adding a data link.

and feedback links. In this case, Xor and And logic options are also applicable since *Y* starts in parallel (otherwise, Join-And would cause a deadlock). Similarly, the input logic of *Y* is Join-Or (having multiple activities, the forward links would have Join-And logic, Eq. 10.8). The output logic of *X* does not allow link to End at first execution, but only after iterations (i.e., from second execution X_2).

References

van der Aalst WMP, van Hee KM (2002) Workflow management: models methods and systems. MIT Press, Cambridge, MA

Cho SH, Eppinger SD (2005) A simulation-based process model for managing complex design projects. IEEE Trans Eng Manag 52(3):316–328

Karniel A, Reich Y (2007) Simulating design processes with self-iteration activities based on DSM planning. IEEE Proceedings of the international conference on system engineering and modeling ICSEM'07, Haifa, March, pp 33–41

Karniel A, Reich Y (2009) From DSM based planning to design process simulation: a review of process scheme logic verification issues. IEEE Trans Eng Manag 56(4):636–649

Sadiq W, Orlowska ME (1999) Applying graph reduction techniques for identifying structural conflicts in process models. Lect Notes Comput Sci 1626:195–209

Sered Y, Reich Y (2006) Standardization and modularization driven by minimizing overall process effort. Comput Aided Des 38(5):405–416

Chapter 11
Dynamic Changes

11.1 Dynamic Change Levels

The presented framework engaged three levels of dynamics: the process level (process dynamics within a predefined scheme), the process scheme level (run time changes of the process scheme), and the BR level (decision-making model, i.e., changing the way that process scheme changes are implemented). Using the dynamic process changes classification definitions, in Sect. 2.3, the implementation of the following dynamic levels (Fig. 2.5) is further elaborated (Fig. 11.1):

1. The *process level* dynamics indicates the dynamics at run time (Class I), i.e., following the process scheme and making choices between process branches (e.g., continue or iterate). Exception handling (Class II) implementation is done once the pre-defined branch is required, and is not presented to the user as part of the process scheme unless needed. Pre-defined Hierarchical expansion (replacing an activity in the process by a pre-defined subprocess according to process parameters) can also be regarded as process level dynamics since the scheme might not have been presented to the user but was pre-defined.
2. The *process scheme level* dynamics includes Ad-hoc changes (Class III), hierarchy expansion (not pre-defined) (Class IV), and process content changes. Ad-hoc changes may be used to implement "best practice" scheme changes resulting from changes in the business process, which are relatively rare (typically, such change is postponed to next project) or utilized for unforeseen process options. A dynamic hierarchical process expansion is an expansion of an emerging activity (not predefined) by a pre-defined subprocess, or an expansion of a pre-defined activity by an emerging subprocess, or both not predefined.
3. *Business Rules* dynamics refers to changes in the interpretation rules of a DSM to process scheme, i.e., changes in the way the process scheme changes are made. For example, initially, interpreting a DB as a set of serial, iterative

A. Karniel and Y. Reich, *Managing the Dynamics of New Product Development Processes,* DOI: 10.1007/978-0-85729-570-5_11,

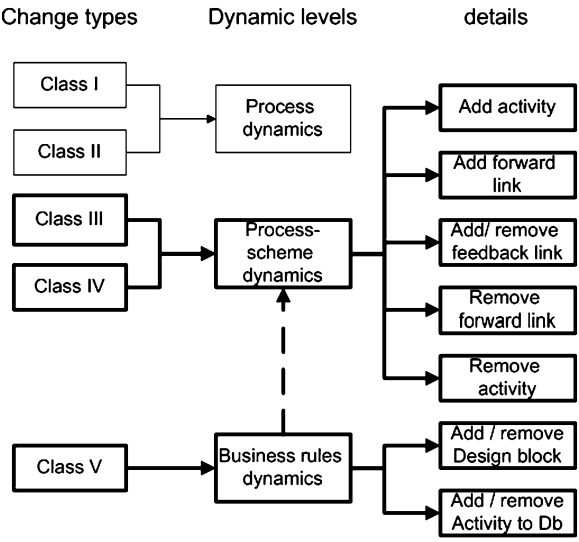


Fig. 11.1 Dynamic scheme change types

activities (due to insufficient resource) in part of the simulation; and subsequently during run time, change the amount of resources and interpret an iteration of the DB as parallel activities.

The first dynamic level is well understood, though Class II changes are complex to implement (Russell et al. 2006). Changes of BR (level 3) would cause changes of the interpretation of the planned process (even if the DSM did not change); thus, adding more process variations. The BRs are controlled by the planner; providing control over process parameters, and are considered part of the decision-making process (Fig. 8.2).

A complex issue is the integration of planned process scheme changes (level 2 dynamics) into the current process scheme (see Fig. 8.3), where the planned processes are either the result of DSM interpretation, or are the required processes (if ad-hoc changes were applied). BRs dynamics directly affect the DB content, and indirectly change the process scheme dynamics. The dynamic types are depicted in Fig. 11.1, which is an extension of Fig. 2.5.

Implementation of a process scheme change is dependent on the required change type and the process status.

11.2 Scheme Change Types

In order to keep the process scheme correctness (DnPDP correctness criteria, Sect. 9.2), the implementation of changes should be done carefully. Ad-hoc changes do not necessarily follow any pattern. Their implementation is left to the user. Various tools (e.g., "Woflan" system, Verbeek et al. 2001) may help to check

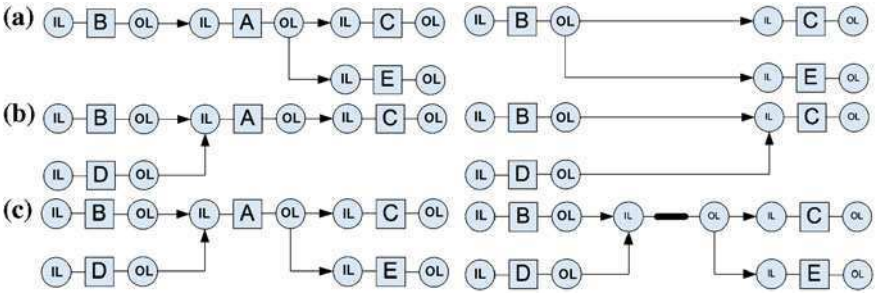


Fig. 11.2 Design task removal

the proposed change. A change that cannot be implemented (violating correctness requirements) should be notified to the user. We would further assume that ad-hoc changes are the type of adding activity or link and not their removal, and have no special issues.

This work focuses on the more structured changes resulting from product knowledge changes via DSM interpretation. The discussion that follows focuses on the DSM structure after reordering (i.e., design activities and links). It should be noted that in the extension of the DSM to DPM, additional logic activities are added with associated links to the design activities, and the DSM links between design activities become links between input and output logic activities (*IL* and *OL*, respectively).

Scheme change types can be classified as follows (Fig. 11.1):

1. *Add activity* Any added activity should have input forward link and output forward link to other activities. This satisfies requirement (2), by Corollary 9.8.
2. *Adding forward links* may reduce parallelism (if activities were previously not linked). Such addition can immediately be implemented based on a BR, as there are no special process logic issues involved.
3. Adding/removing feedback links may have implications to the definition of DBs. If the addition is within a defined DB, there will be no out of block process impact. There might be issues related to the internal planning of the block when it is considered as a process.
4. Removing forward link (from *OL* of the design activity to *IL* of another design activity) requires special treatment. If both the source activity and the target activity have additional outgoing/incoming forward link, respectively, the link could be removed immediately. If it is the only outgoing forward link, it should be replaced with forward link to the End activity. If it is the only incoming forward link, it should be replaced by a link from Begin activity.
5. Removing activity. Removal of activity is a complex task, as the removal should not impair the process correctness. Process status check is required according to the implemented process logic. Removal of an activity would mean in general removal of its links. The removal of activity *A*, in Fig. 11.2 can be done by replacing it with a dummy activity (zero duration,

i.e., short-circuiting the *IL* and *OL* of *A*). In the case of one incoming forward link to the *IL* of *A* (e.g., from *OL* of activity *B*), the replacement by dummy operation is equivalent to replacing all the links from *OL* of *A* by links from the *OL* of *B*, see Fig. 11.2a. The same logic may apply if there is only one outgoing forward link from *OL* of *A* to *IL* of activity *C*, Fig. 11.2b. However, the major problem is when there are multiple incoming forward links and outgoing forward links. In this case, replacement by dummy is the only possibility, Fig. 11.2c. Therefore, replacement by dummy is the option being always implemented.[1]

11.3 Process Status Considerations

The implementation of changes is dependent on the process status. If the required change has no immediate effect at run time on the process, it could be implemented immediately; otherwise, it might be postponed. Typically, adding process activities and forward links have no restrictions.

Considering the scheme changes types:

1. An activity that can be activated (having its information prerequisites fulfilled) can immediately start (and might cause iterations of other activities). Application of rules BR 11.1 or BR 11.2 would have different impact.
   According to BR 11.2, the additional design activity may start if any of its previous activities had completed at least one execution (though it might be executing now), while BR 11.2 will postpone the activation.
2. When a forward link to an already active activity is added, the change impact depends on the applicable business rule (e.g., BR 13.1 will merge to current execution, and BR 13.2 will start next iteration). If the target activity accepts such input, it would affect the activity duration; otherwise, the information (through the link) would contribute to the initiation of the next iteration.
3. Addition or removal of feedback links (after reordering) may define changes in DBs. The implementation of such changes can be done immediately if the block is inactive (i.e., no activity in the DB is active). If the DB is active, the implementation depends on additional BRs (following).
4. Removal of a forward link depends on the status of its source activity. If the source activity (or an iteration of the activity) is active, the link might be removed only after the completion of the activity, and activation of next activity according to that link. It should be noted that the operation time of a link is zero; therefore, the order of performing process updates is important. First, the source activity should complete; then the target activity should get the signal; finally, the link could be removed.

[1] If there is a self-iteration link (*OL* to *IL* of *A*), it should be removed (avoiding infinite loops of process with zero duration that consume computing resources).

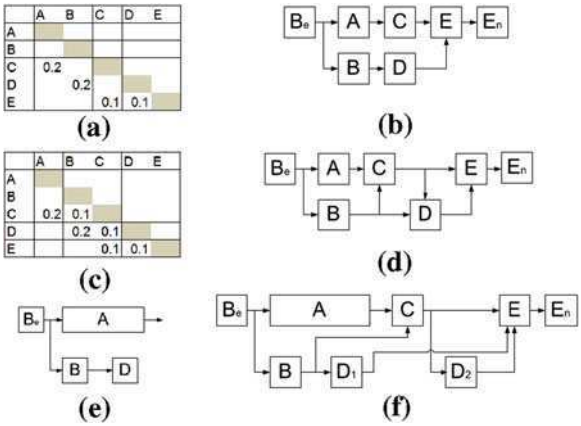


Fig. 11.3 Scheme changes lead to iterations. **a** Initial knowledge. **b** Initial plan (C-Process). **c** Updated knowledge. **d** Updated plan (C-Process). **e** RT-process. **f** RT-process

5. Removal of a link when the source is inactive requires confirming with IR 2 and IR 3, i.e., a single link from the design activity to the OL logic activity or from the IL logic activity to the design activity cannot be removed. A link from the OL logic activity to another IL logic activity can be removed if it is not a single forward link. If it is a single forward link, it must be replaced. If it is a single forward input link to the IL activity, it should be replaced by a link from the Begin activity (IR 6). If it is a single out link from the OL activity, it should be replaced by link to the End activity (IR 7).
6. Removal of an activity that has never performed could be done immediately. Removal of a design activity that is currently performing is postponed until the activity is completed. However, as completion depends on activity duration, the duration may change. In a similar manner to duration increase, BRs should define whether duration should decrease; thus, the activity terminates immediately (on $t + 1$), or the activity should complete to get some meaningful results, and then removed.

The following implementation rule indicated the options of defining the duration of a removed design activity.

(IR 17): Duration of removed design activity:

(BR 17.1) (As soon as possible): Activity duration is reduced to be its current duration

(BR 17.2) (Minimal duration value): Activity duration is set to be the maximum of either minimal duration or current duration

11.4 Dynamic Scheme Changes May Cause Iterations

The implementation of scheme changes may cause activity iterations even if the planning does not indicate so: for example, the DSM in Fig. 11.3a, whose process

plan is in b. At a certain time it was realized that activity *C* needs input from *B* and activity *D* needs input from *C*, see DSM and process plan in Fig. 11.3c and d, respectively. If this additional knowledge is obtained before executing *C* and D, then there is a direct shift from the plan in (b), to the plan in (d). However, if activity *A* has a long duration (e.g., $D(A) > D(B) + D(D)$), depicted in Fig. 11.3e such that the rectangle length indicates relative time, then activity *D* should start early after B and iterate as shown in Fig. 11.3f.

It should be noted that activity *E* needs to wait until the second execution of activity *D* (indicated as D_2). However according to the process plan it should just wait for the completion of *C* and *D*. Actually, the presented (parallel) run time process could be described as if the order of activities was (*ABDCE*); hence, the additional link was equivalent to a feedback link. In a similar manner, if $D(B) > D(A) + D(C)$, then an iteration of *C* would be required, since *C* would have completed before *B*.

The above simple example demonstrates the importance of the distinction made between the C-process, and the RT-process. In the case of a fixed process scheme, the separation of current process plan (C-process) from run time (RT-process) is merely a visual representation of the process progress. However, this separation is essential for modeling the process dynamics. The RT-process model keeps the interim process information that is required for the implementation-logic of dynamic changes.

11.5 Design Block Changes

Link changes either addition/deletion or change in value in the DSM may cause different ordering and different allocation to DBs. Implementation of changes in DB content is subject to BRs. A business rule should indicate if the change should be done as soon as possible (immediately), or the DB has to complete at least one design cycle (resembles the minimum duration option), or wait until the DB completes the current cycle. The latter option is less agile and less responsive to changes, but more robust and may decrease the overall number of iterations.

(IR 18): Changes in design block content:

(BR 18.1) (As soon as possible): Change parallel completion constraints immediately

(BR 18.2) (One cycle completion): All design activities in design block should have completed at least once

(BR 18.3) (Current cycle completion): All parallel activities in design block should complete the current cycle

Example of a change in the DB content, and its implementation is presented in the next Chapter (Sect. 12.7). The implementation of ‘as soon as possible’ rule

(BR 18.1) depends on the execution status of the activity, the DB to which it is added, and the DB from which it is removed (see examples in Sect. 12.7.2). Since as default every design activity is part of a DB (even if it is the only activity in that block), then every assignment to a DB or removal from a DB, changes the DB content. DBs are defined by their content; hence, a change that empties a DB from any activity causes the DB to be removed. The DB ID cannot be reused in order to avoid mistakes.

11.6 Change of Process Scheme by Business Rules

Generating business cases can follow distinct rules. The following rule options were considered:

- Implementation of learning curve with learning ratio of $LR = 0.5$ (the ratio of Duration of activity execution $i + 1$ to the duration of execution i) versus no learning ($LR = 1$);
- Early (BR 16.2) or Late (BR 16.1) start of activity or DB.
- Merger to executing activity (BR 13.1), or starting a new iteration (BR 13.2)
- Exit option (jumping to the end activity) on second iteration (BR 12.2 or BR 15.2), or not (BR 12.1 or BR 15.1, respectively).[2]
- DB change implementation: ASAP (BR 18.1), at least one execution (BR 18.2), or by the end of current DB iteration (BR 18.3).

Out of the 48 potential combinations of the above rules, (see following analysis in Sect. 11.8), two extreme cases can be defined:

- The "short" option—applying learning curve $LR = 0.5$; Early start; Merger in case of executing activity; Exit option to End activity; and immediate implementation of DB change.
- The "long" option—no learning $LR = 1$; Late start; Start new iteration in case of executing activity; no Exit option; and implementation of DB change by the end of the current iteration.

The "short" option implies maximal RT-process parallelism, minimal impact of multiple iterations, and more options to keep the process time minimal (due to exit and merger) options, and quick response to changes in terms of reorganizing the process.

The "long" option indicated higher impact of multiple iterations, i.e., indicates that more iterations might lead to much longer time, having more serial quality checks, and slower communication and reorganization.

[2] Reminder, the business rules of IR 15 apply to coupled activities, while IR 12 applies to serial activities.

11.7 Simulating a Dynamic Scheme Process

11.7.1 Simulation-Based Statistics for Decision-Making

Statistical analysis of simulation results is used for Decision-making. Based on the property of large number R of simulation results, the LLN and CLT determine that a function over the expected values of a simulation is normally distributed (Asmussen and Glynn 2007).[3] A simulation run was defined as running the process twice using two sets of process parameters (e.g., different business rules). The function used is the difference between the estimated parameters (e.g., the total process times) of the runs.

We can use the default assumption (null hypothesis) of no difference between the mean values of the estimated difference parameter between the cases. Under such assumption, the expected value of the difference function is zero. Due to the distribution of simulation results, even when the same set of process parameters is executed twice, the difference between the total process time of the two cases is not zero.

We calculate the Confidence interval I_α with confidence level 1-α using Eq. 14.6, Sect. 14.2. The decision-making hypothesis (no difference) is accepted if the result is within the interval, and rejected otherwise. Since the check is two tailed, if the null assumption is rejected, we accept one of the sides accordingly, with a statistically significant difference.

In order to make such analysis, it is required to have a large enough number of simulations R. The error decreases with square root of R, and is linear to the standard deviation (Eq. 14.6). We can calculate the required number of simulations $R > 1.96^2 \times s^2/0.05$, for $\alpha = 5\%$, where s^2 is the calculated variance estimator (Eq. 14.5). R is estimated according to a small batch of runs (e.g., 50).

Other potential procedures are using the t-test comparison of two simulation types (sets of parameters), or an ANOVA test for comparing multiple simulation sets. In either case, the normal distribution of the expected value is used. For a different variance t-test, one can perform $R = MK$ simulations for each simulation set, where M is number of sections (or groups) of simulation, each with K runs, and K should be large enough. For each of the M sections we calculate the mean (expected value, being normally distributed) and the standard deviation.

11.7.2 Probabilities Interpretation at Run Time

Probabilities are used for DSM reordering. On implementation, we convert the DSM to DPM (adding Begin, End, and the applicable links). The probability value of a forward link that defines serial progress is ignored. The probability values are

[3] See Sect. 5.5

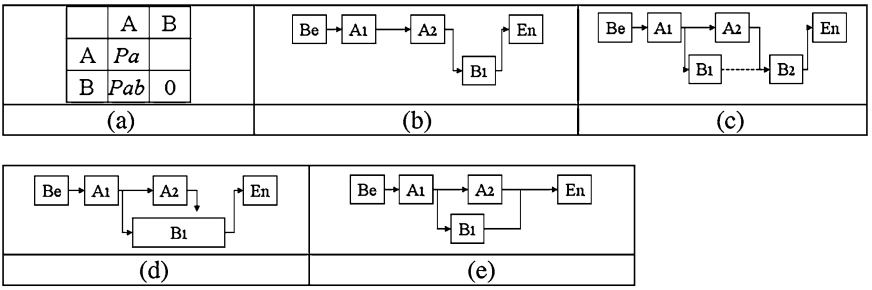


Fig. 11.4 RT-processes of various Business Rules. **a** DSM. **b** Late start. **c** Early start. **d** Merge. **e** Exit

considered for progress decisions only for: feedback links, self-iteration links, and in the special case of forward links within serialized coupled activities (on second execution of the activity).

When DB logic is applied (parallel coupled activities), all "internal" forward and feedback probabilities are replaced by the self-iteration merged probability of the DB, and "external" links (to other DBs or other activities) are replaced by the integrated probability, according to Eq. 4.3.

Progress decisions are made in output logic implemented in *OL* activities. According to Eqs. 10.4–10.7, feedback links are considered first (including self-iterations), then forward links, and finally links to End (if applicable). The latter links have no probability assignment as the links always indicate serial progress to End (same logic applies to links from Begin).

In the decision procedure, a number Q is randomly selected. If $Q < p$, where p is the applicable probability, the link is activated. Feedback links were defined to be in a Split-OR mode, i.e., any combination of feedback links may apply.

The probability of a forward link within serialized coupled activities is considered on second (or later) execution of the activity, using the same Split-Or logic iterations (according to Eq. 10.8). Other probability figures are not considered, i.e., treated as having a binary value.

11.8 Business Rules Combinations at Run Time

The implication of a BR on the process performance (duration and required resources) is dependent on the actual duration of the activities. The following simple examples demonstrate the various possibilities, having two activities with a serial link from A to B with self-iterations of A. Activity A executes twice, activity B executes twice at most. The DSM of the process is depicted in Fig. 11.4a.

The run time processes are illustrated in Fig. 11.4. The process total duration T, the required resources R, and reference to process figure are indicated in

Table 11.1 Duration and Business Rules at run time

Case	D($A1$)	D($B1$)	D($A2$)	D($B2$)	LR	Early/ late	Merge/ new	Exit/ cont.	T	R	Ref RT
1	30	10	30		1	Late	N/A	N/A	70	70	(b)
2	30	10	30	10	1	Early	N/A	Cont.	70	80	(c)
3	30	10	30		1	Early	N/A	Exit	60	70	(e)
4	30	10	15		0.5	Late	N/A	N/A	55	55	(b)
5	30	10	15	5	0.5	Early	N/A	Cont.	50	60	(c)
6	30	10	15		0.5	Early	N/A	Exit	45	55	(e)
7	10	30	10		1	Late	N/A	N/A	50	50	(b)
8	10	30	10	30	1	Early	New	Cont.	70	80	(c)
9	10	30	10		1	Early	Merge	N/A	40	50	(d)
10	10	30	10		1	Early	N/A	Exit	40	50	(e)
11	10	30	5		0.5	Late	N/A	N/A	45	45	(b)
12	10	30	5	15	0.5	Early	New	Cont.	55	60	(c)
13	10	30	5		0.5	Early	Merge	N/A	40	45	(d)
14	10	30	5		0.5	Early	N/A	Exit	40	45	(e)

Table 11.1. The cases represent shortest path options, subject to iteration of A. The durations are represented in some time units, and the assumed resources consumption is one resource per activity per time unit. The shortest path (A–B) without iterations has duration and resources of 40 units.

The combinations of the following rules are demonstrated for two activity duration settings: first D($A1$) = 30, D($B1$) = 10; and second D($A1$) = 10, D($B1$) = 30:

- Implementation of learning curve with learning ratio of $LR = 0.5$ (the ratio of Duration of activity execution $i + 1$ to the duration of execution i) versus no learning ($LR = 1$);
- Early start (BR 16.1) versus Late start (BR 16.2);
- Merger to executing activity (BR 13.1) versus initiating a New iteration (BR 13.2); and
- Exit option of second iteration (12.2) versus continuing by enforcing iteration of the next serial activity (12.1).

Notes The options Merger/New iteration are inapplicable (N/A) for the case D(A) = 3·D(B), and for Late start (i.e., such decision does not occur). The Exit/ Continue options are inapplicable in the case of Late start. Furthermore, there might be either Merger or Exit, but not both. Therefore, out of the 32 potential combinations there are actually only 14 cases, described in Table 11.1.

Including a learning process decreases the impact of iterations on the overall process performance. Yet, the performance is more affected by the other rules and the relative duration of the activities.

Late start serializes the execution of iterations (for this simple case); thus, makes total process duration equivalent to the resource, i.e., there is one active resource at a time. Cases that are more complex are presented in Chap. 1.

	A	B	C	D
A				
B	0.3		0.2	
C		0.2		
D			0.1	

(a)

	A	BC	D
A			
BC	0.3	0.36	
D		0.1	

(b)

Fig. 11.5 Serialized and Parallel process execution. **a** Serialized process. **b** Design block (Parallel)

The implications of using Early start are dependent on the other rules used. It can contribute to shortening the process when early termination options such as Exit or Merge are applicable. Enforcement of continuing with iteration of next serial activity always result with additional resources (in comparison to late start), and might result with larger process duration (cases 8 and 12 in Table 11.1). Yet, enforcing serial activities might be required for quality (e.g., if the B was a testing activity).

The Merge and the Exit options were both applicable only with Early start since there were no feedback links. These options can contribute to shortening the process time. In this simple example, they yielded the same results.

The implications of utilizing these BRs in the case of iterations due to loop of activities are demonstrated in Chap. 1.

11.9 Applying Design Blocks at Run Time

Business Rules indicate different business cases, and may indicate different strategies. Choosing the appropriate strategy might be done by using rules of thumb; e.g., if there are enough resources make the activities as much as parallel to reduce overall time. However, in the case of iterative processes, the "best" strategy may not be very clear. The following simple example of the implication of applying different BRs was presented in (Karniel and Reich 2007).

Two business cases are studied, depicted in Fig. 11.5. The first case is a serial performance of coupled activities (i.e., one activity at a time), applying BR 14.1, Fig. 11.5a; the second example is performing the activities as a DB (in parallel), applying BR 14.2. The DB has a self-iteration merged probability $p = 0.36$, according to Eq. 4.3 (Sect. 4.2.3).

The duration of the DB is defined as the maximal duration of its activities. The duration assigned to activity X is defined as $\mathrm{D}(X)$. The activity durations are: $\mathrm{D}(A) = 1$; $\mathrm{D}(B) = 2$; $\mathrm{D}(C) = 3$; and $\mathrm{D}(D) = 4$. The duration of the DB $\mathrm{D}(BC) = \max(\mathrm{D}(B), \mathrm{D}(C)) = 3$.

In this example, there are no parallel initial or parallel completion issues, Begin links to activity A, and End links from activity D. BR 12.1 and BR 15.1 (no direct

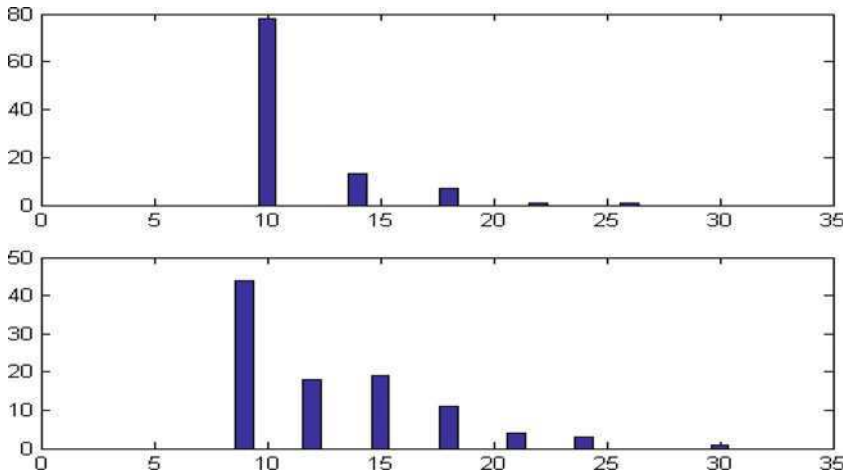


Fig. 11.6 Processes duration distribution

"jump" to End) are implemented, respectively in the serial and parallel examples. BR 13.2 was used, i.e., initiating a new iteration (and not extending activity duration).

The distribution of the overall process duration (for 100 runs) is depicted in Fig. 11.6 indicating duration results of the serialized process ($P1$) and process with DB ($P2$). The X axis is the overall process duration; the Y axis is the number (and %) of runs with the indicated duration. The distribution has discrete values and a decreasing shape (due to decreasing probability of iterations). It is apparent that the distribution is right (positive) skewed, and not a normal distribution.

In order to analyze the results, several decision statistic parameters were compared: average duration (Cho and Eppinger 2005), median (not mentioned in literature), and pairwise comparison (Reich and Paz 2008).

The results for average and standard deviation of averages, and average and standard deviation of medians are depicted in Table 11.2. Results are presented for the following parameters: probability of iteration of the coupled activities (both getting the values: 0.2, 0.4, 0.6), and differences of duration between the activities D(B) and D(C). The parameters of the process with DB are derived using Eq. 4.3 for the combined DB self-iteration merged probability P(BC); and the max duration for the DB duration D(BC).

The results are averaged from 10,000 run time simulations. The average, median, and their respective standard deviations were calculated from 100 averages (medians) of 100 runs, for each set of parameters.

For the range of parameters checked, the decisions based on simple comparison of the different statistical parameters (average, median, differences) resembled, but were not similar. The differences of average and median were not statistically significant. The decisions, according to each of the statistical measures are marked by colors: green (gray) indicates that process $P1$ was preferred, and pink (light gray)

Table 11.2 Business Rules comparison

	Serial				Design Block		Serial (avg)		Db (avg)		Serial (med)		Db(med)	
#	P (B-C)	P (C-B)	D(B)	D(C)	Self P (BC)	D (BC)	Avg-avg (P1)	Std – avg (P1)	Avg – avg (P2)	Std-avg (P2)	Avg-med (P1)	Std – med (P1)	Avg-med (P2)	Std – med (P2)
1	0.6	0.6	1	2	0.84	2	12.39	0.55	17.40	0.97	10.91	0.56	14.10	1.09
2	0.6	0.6	1	5	0.84	5	20.10	1.14	35.36	2.83	16.82	1.11	27.70	3.21
3	0.6	0.6	4	5	0.84	5	27.49	1.82	35.91	2.81	22.68	1.95	27.60	3.31
4	0.4	0.4	1	2	0.64	2	9.99	0.31	10.56	0.44	8.10	0.53	9.07	0.32
5	0.4	0.4	1	5	0.64	5	14.96	0.61	18.93	1.21	11.12	0.84	15.32	1.16
6	0.4	0.4	4	5	0.64	5	20.05	0.89	19.06	1.05	14.05	0.45	15.12	0.74
7	0.2	0.2	1	2	0.36	2	8.74	0.16	8.12	0.18	8.0	0.0	7.0	0.0
8	0.2	0.2	1	5	0.36	5	12.52	0.31	12.79	0.45	11.0	0.0	10.0	0.0
9	0.2	0.2	4	5	0.19	5	16.15	0.51	12.71	0.45	14.0	0.0	10.0	0.0

Table 11.3 Decision-making based on difference

#	Paired	Difference *R*=1000 Interval (5%)				Difference *R*=10000 Interval (5%)			
	% T(P1) >T(P2)	Avg	Std	Confidence Interval	Decision	Avg	Std	Confidence Interval	Decision
1	35.08	-5.04	12.52	0.76	T(P1 < T(P2)	-5.02	12.48	0.24	T(P1)< T(P2)
2	32.63	-15.79	30.03	1.87	T(P1)< T(P2)	-15.86	29.99	0.59	T(P1)< T(P2)
3	41.62	-8.47	32.76	2.02	T(P1)< T(P2)	-8.42	32.72	0.64	T(P1)< T(P2)
4	49.43	-0.55	5.43	0.36	T(P1)< T(P2)	-0.56	5.41	0.11	T(P1)< T(P2)
5	48,95	-3.91	12.91	0.80	T(P1)< T(P2)	-3.96	12.85	0.25	T(P1)< T(P2)
6	54.13	0.98	14.97	1.01	T(P1)~ T(P2)	0.98	14.89	0.32	T(P1)> T(P2)
7	68.79	0.62	2.54	0.15	T(P1)> T(P2)	0.61	2.54	0.05	T(P1)> T(P2)
8	69.33	-0.26	5.79	0.36	T(P1)~ T(P2)	-0.26	5.76	0.11	T(P1)< T(P2)
9	70.72	3.43	6.72	0.41	T(P1)> T(P2)	3.44	6.68	0.13	T(P1)> T(P2)

indicates preference of the DB *P*2 process. Having enough samples should finally create a normal curve of the averages, allowing comparison of averages with statistical significance.

Pairwise comparison (Reich and Paz 2008) results are depicted in Table 11.3. Pairwise comparison is a count of how many times the duration of process *P*1 (serialized) was longer than the duration of process *P*2 (with DB), D(*P*1) > D(*P*2). Such result is a compound simulation result that includes running the process twice, once with serialized parameters and once with DB parameters and comparing the duration results.

The direct counting of pairwise comparison of simulation results is presented in the "paired" column of Table 11.3. The first process is the preferred when its duration is longer in less than 50% of the cases. This measure had self-similarity in various ranges, i.e., it had relatively the same results for 1,000 cases and 10,000 cases. A decision-making criterion based on these results indicates that when the percentage of D(*P*1) > D(*P*2) is less than 50%, then *P*1 (serial) is preferred,

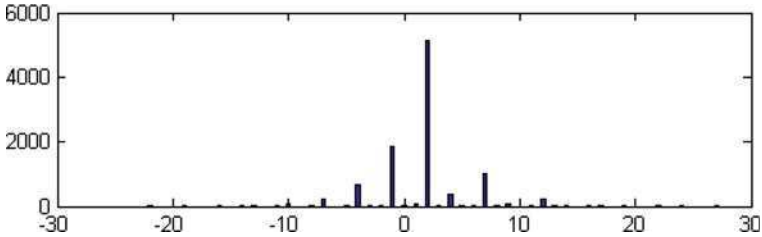


Fig. 11.7 Difference function distribution

otherwise $P2$ is preferred. Using the simulation results (of pairwise comparison for decision-making) give similar results to the other two previous methods.

Using the decision-making method described in Sect. 11.7.1, we compute the difference function distribution. In order to estimate the required number of simulation runs we made 100 runs for each set of parameters. For the serial process (case A), the estimates variance is $s_A^2 = 9.17$, thus for $\alpha = 5\%$, $K_A > s_A^2 \times 1.96/0.05 = 704.7$. In the same manner we get for the DB (case B), $s_B^2 = 20.09$, $K_B > 1{,}543.6$. Using the results of 10,000 simulations is more than enough.

The difference function distribution for the example in Fig. 11.6 is depicted in Fig. 11.7. As expected, the difference of the basic process (i.e., no iterations) is the most common outcome ($10 - 8 = 2$).

The results of performing difference calculation statistics ($Dif = T(P1) - T(P2)$) for 1,000 runs and 10,000 runs are presented in Table 11.3. The calculations were made for $\alpha = 5\%$. Under the assumption of no difference, the interval is $\left[-Z_{1-\alpha/2} \cdot s/\sqrt{R} \quad Z_{1-\alpha/2} \cdot s/\sqrt{R}\right]$, where s is the standard deviation calculated using Eq. 14.5, and $Z_{1-\alpha/2}$ is the normal distribution percentile ($Z = 1.96$ for $\alpha = 5\%$). If the average result is out of the interval, we can decide that the total time of the serial process ($P1$) is significantly larger (smaller) than the parallel process ($P2$). Since the interval is symmetric, only the upper bound of the confidence interval is presented.

Comparing the results in Table 11.3 with the results of Table 11.2, it was found that the decisions based on large sample ($R = 10{,}000$) were the same as the decisions based on averages. The difference function distribution converges to the distribution of the averages. Therefore, in the case of very large sample, such alignment of the results is expected.

A smaller sample ($R = 1{,}000$) was less decisive and the decision-making result indicated insignificant difference (within the confidence interval) between different process parameters, i.e., a "don't care" decision. These "don't care" cases (#6 and #8) were aligned with the cases where using average, median, and paired results yielded different decisions. Having an indifferent zone is an important feature of using confidence interval, and more than enough (i.e., too many simulation) seems to be too much (Karniel and Reich 2009).

Adding simulation runs neither change much of the average, nor the deviation of results; but only the calculated confidence interval. This interesting result should be further investigated.

11.10 Logic Implementation

The implementation of logic is complex. Two main issues should be addressed:

(1) ensuring the completion of the process according to the correctness criteria requirements (Sect. 9.2), and
(2) avoiding multiple occurrences of the same activity in parallel (same activity is not done multiple times at the same time).

The first issue includes five requirements. The first two (project characteristic, and reaching a termination state) are assured by using the conversion to DPM. Item (4)—traceability is fulfilled by using the *RT*-process model. Item (5)—finite time, is typically applicable by the use of probability; it can be enforced by limiting the total number of activities, or could be implemented by limiting the number of iterations of a link.

To ensure items (3a) and (3b), the logic of the End activity is set not to allow completion until all activities have completed at least once, and all iterations have completed.

The second issue is not inherent to a process with parallel activities while enabling iterations (Cho and Eppinger 2005) and self-iterations (as depicted in Sect. 4.3); therefore, it should be enforced. The condition represents a common practice, in which one activity is done by a resource (or several resources) at a time, i.e., it can execute multiple times but not at the same time.

Due to dynamic changes (next section) each activity is part of a DB (a DB may have one activity). To ensure conformance with "one execution at a time" property, the following logic is defined for a DB:

- Condition A: for each activity in the DB, all previous iterations of the activity have completed (or not started).
- Condition B: the DB may start if all its pre-conditions have been completed (i.e., all activities that precede the activities in the DB have completed at least once, including the Begin activity).
- Condition C: The DB has completed once and got a feedback link signal (either from another DB or by self-iteration).

The DB may start if (A) and (B or C).

Additional conditions reflect the BR applied, e.g., early start of activity (once the preceding activities have completed once), or late start (after completion of all iterations). Typically, in a case of early start, re-activation is expected that may occur during the activity execution. If the activity (or DB) is already active, a re-activation may do the following:

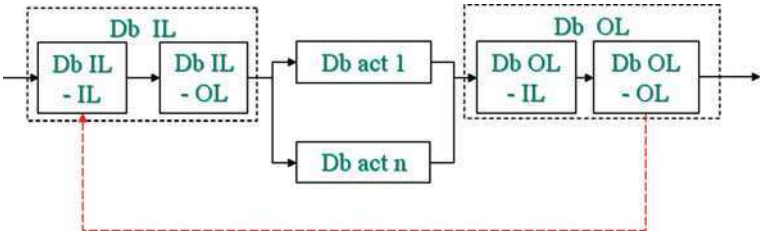


Fig. 11.8 Design block structure

(1) the activity time is expanded, or
(2) a new iteration is added (waiting for the current execution to complete).

11.11 Implementation of Design Block Changes

Since the DB content can change during run time, the implementation of all processes is done using DBs. Accordingly; a DB might contain only one activity. While there is an overhead of managing the DB data and logic for a single activity, it is easier to manage the overall process logic by not using special logic for a single activity and different logic for activities coupled in a DB. Moreover, the DB structure provides the required logic for self-iteration.

DB implementation includes Input Logic activity and Output logic activity, where each has (internal) IL and OL, see Fig. 11.8.

The DBIL-IL activity of the DBIL may receive links from other DB's, either forward or feedback; self-iteration link (dashed red line), and link from Begin activity. It is linked to an DBIL-OL logic activity which has Split-And logic to all activities in the DB (parallel start). The activities are linked to an DBOL-IL logic activity with Join-And (parallel completion). The latter is linked to the DB logic activity DBOL-OL that may have self-iteration link, feedback link, and forward link to other DBs, or a link to End activity.

The DB identity is defined according to its design activities content; once the content is altered, the DB gets a new ID. Examples of the relations of the DB input logic and output logic to Petri nets and WRI-WF-nets were presented for the case coupled activities in 9.7 (serialization) and 9.8 (parallel), respectively.

References

Asmussen S, Glynn PW (2007) Stochastic simulation algorithms and analysis. Springer, Berlin
Cho SH, Eppinger SD (2005) A simulation-based process model for managing complex design projects. IEEE Trans on Eng Manag 52(3):316–328

Karniel A, Reich Y (2007) From planning to executing NPD processes. In the 4th Annual Israeli National Conference on System Engineering—INCOSE_IL'07, Herzliya

Karniel A, Reich Y (2009) Statistical analysis of process simulations. In: International conference on engineering design (ICED' 09), Stanford, CA

Reich Y, Paz A (2008) Managing product quality, risk, and resources through quality function deployment. J Eng Design 19(3):249–267

Russell N, van der Aalst WMP, Hofstede AH ter (2006) Exception handling patterns in process-aware information systems. In: The 18th Conference Advanced Inf. Systems Engineering CAiSE'06

Verbeek HMW, Basten T, van der Aalst WMP (2001) Diagnosing workflow processes using Woflan. Comput J 44(4):246–279

Chapter 12
Implementation Example

The main applications presented in the following example are the monitoring of a dynamically changing process scheme; the implication of the simulation of dynamic planning capabilities; and the potential use of the simulation results for process-related decision-making.

The example represents an NPD environment, where process activities are derived from the product structure and the relations between product components, which are subject to changes during the design as new knowledge becomes available.

First, the contribution and the value of the example are discussed in Sect. 12.1. The product used in the example is described in Sect. 12.2. The initialization and progress of the dynamic design process through the predefined high-level process scheme is presented in Sect. 12.3. Dynamic planning and implementation of a relatively simple case, the planning of the conceptual design phase, is presented in Sect. 12.4. The section includes detailed examples of the impact of various process parameters. Section 12.5 is devoted to planning of the design phase, presenting the full implementation of the DnPDP framework and the implications of dynamic process changes. It includes detailed examples of all the process-planning stages from DSM data collection through the conversions, the use of DB, and business rules. The process continues in Sect. 12.6, and in Sect. 12.7 modifications are made and implemented through design block changes. The value of (early) changes in product knowledge (as a source of process plan changes) is examined. The example is summarized in Sect. 12.8.

12.1 The Value of Simulations for Model Verification

Smith and Morrow (1999) surveyed process-modeling techniques used for PDP. The goal of a modeling was defined to be the creation of predictive model that

A. Karniel and Y. Reich, *Managing the Dynamics of New Product Development Processes,* DOI: 10.1007/978-0-85729-570-5_12,

improved managerial decision-making. The following judging criteria were defined for such model to have useful predictive value: The model addresses an important managerial issue; the decision-making is based on information that is available and accurate; the assumptions and simplifications of the model are reasonable.

Four model validation levels were defined: face validity; applying the model to realistic data sets; model utilization to guide decision-making in an experimental environment; and guiding decision-making in the 'real' world.

1. Face validity implies that the modeling topic, data, assumptions, and tractability seem reasonable to those who are familiar to the field of product development management.
2. Applying the model to retrospective data sets gathered from industry. The models are used to show that they could have guided decision-making.
3. Model utilization to guide decision-making in a well-controlled and repeatable environment; thus, demonstrating decision-making improvements. Yet, the types of problem likely to be encountered in experimental settings are often very simple.
4. Guidance of decision-making for actual cases was defined as the ultimate validation goal. However, there is no objective comparison between the changed and the unchanged situation.

It was indicated that the models reviewed were validated to level two at most.

The usefulness of the DSM method, the need for process implementation and execution, and the benefits of simulation were discussed and proved in the vast literature of any of these issues. However, the transformation from a DSM-based plan to a process scheme, and the interpretation of the logic based on the BRs that are the main issue of the current research do need verification and validation.

12.1.1 Study Cases

A good way to validate the overall concept and its details would be to test it in industry in multiple study cases. An applicable test case for performing such check is often a large project (as done in most DSM-related test cases). However, several issues made such testing difficult to perform in the current research:

- Knowledge evolution requires the implementation of a tool to collect the changing design information (required for building the DSM). It is assumed that the information could be collected using existing PLM tools (or even ERP tools). Technically, such implementation requires only the addition of few data objects, and integration to the current tool. However, implementing such changes and integration requires an organization that utilizes PLM, and that is willing to invest the resources for performing both the integration and the experiment.

- Current workflow tools do not have the required separation between process scheme and actual scheme, which is necessary for the implementation of iterative processes. Therefore, in order to allow testing by users, there is a need to develop an acceptable Graphic User Interface (GUI).

Consequently, implementing such tool and performing a real case study, or even a controlled experiment was beyond the scope (and resources) of the current work.

The next good option might be a comparison to standard example cases, or using data gathered from industry. However, no cases of implementing changed product data into a changing process scheme were found in the literature. The current management practices do not support documentation of the required data; therefore, such data was not available.

Therefore, the last option remaining was face validity, using simulation results of an example, as described in the current chapter.

12.1.2 Formal Verification

Petri net formulation is an accepted method for process implementation. The *soundness* correctness criteria defined for WF-nets is a proved useful tool for verification of process schemes. If a process is *sound*, then it has some required characteristics. WRI-WF-nets are formally defined to be sound by build; thus, allow to hierarchically build processes that conform to the (general) soundness criteria (see Definition 12).

A formal method was used (in Sect. 9.4) for developing a formal proof of the translation of the reordered DSM (using partitioning) to DSM net (being the current process scheme). The resulting DSM nets were proved to be equivalent to WF-WR-nets, and conform to the soundness criteria for typical DSM-based process cases. The cases include serial, fully parallel, and block diagonal DSM-based processes. In addition, the case of activity cycle with sub-cycles, using the enhanced reordering algorithm, is also proved to have implementation translations to DSM nets that are equivalent to WRI-WF-nets.

12.1.3 Logic Verification

While the structural conversion of DSM can be formally verified for the specified cases, the logic of the equivalent WRI-WF-net is limited. The resulting logic can be applied only to a limited set of DSM logic interpretations, out of all the potential logic interpretations that could be applied according to the BRs (described in Chap. 1), that is implemented through the logic activities. The rationale for the BR was demonstrated by counter examples. A thorough

investigation of all the potential combinations was conducted for the simple case of two activities with two executions at most. This limited check supports the concept, but due to the iterative nature of the process, checking all the business rule options for all cases is a combinatorial task; and a formal proof is left for future work.

12.1.4 Simulation

The concepts were tested using the system as a simulation tool. The example presented describes a potential dynamic NPD process for a simplified industrial case study. The whole framework is presented through the case study, which demonstrates the planning, conversions to model, model execution, and simulation results analysis as an aid for decision-making. The process progress of the design activities was done stochastically, i.e., process changes and occurrence of iterations are probabilistically chosen. The choices were recorded to allow replaying the process progress (bug tracking or conceptual problem tracking). Executing many trials, through which the new concepts (e.g., BR) were applied, suggest a strong evidence (though not a full verification) of the correctness of resulting processes according to the correctness criteria.

12.2 The Example Product

The following example illustrates the application of the various DnPDP framework stages to a case study, demonstrating dynamic process planning for the design process of a laser direct imaging (LDI) plate/image-setter for the prepress market industry. The example demonstrates: (1) the framework stages of process propagation (according to Fig. 8.1); (2) the conversion of product knowledge to required process plan (Figs. 8.3, and 8.4); (3) the *transient process* that arises due to the different process states at the time that new product knowledge is generated and incorporated into the process plan; and finally, (4) the simulation-based decision-making process used for controlling the process (Fig. 8.2).

The data input for the design of laser direct imaging (LDI) plate/image-setter in this case study was given by a leading engineering organization that took an active part in the design and development of prepress imaging systems for market-brand names such as Creo-Scitex Inc. (now HP Israel). The same product example was described in Sered and Reich (2006) for the analysis of modularization and standardization of component in a product family.

This imaging system is based on external drum technology applying direct exposure of high-laser power on plate panels and films, to convert digital input into finished panels ready to be used in standard press-printing process. A metal drum cylinder revolves about its axis allowing a fast motion-imaging axis. A laser

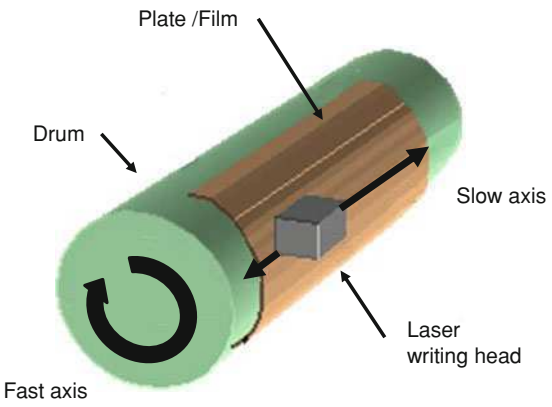


Fig. 12.1 External drum LDI plate/image-setter technology

exposure head, allows a slow-motion imaging axis. It is adjacent to the drum perimeter, driven by a smooth and precise mechanism, parallel to the drum axis, see product scheme in Fig. 12.1 (copy of Fig. 4.1). The flexible plate/film is held around the drum by a registration system using edge clamps and device for punching edge holes. Load/unload system draws a plate from an automatic cassette unit or a manual tray. After the imaging terminates, it unloads the plate to a special fitting in the back of the machine. A computerized control system manages the process.

Various future products were planned ranging from a plotter designated for small printing shops to a product that meets the large high-quality press houses. The plotter changes from a small plate size to twice the size in future markets. The imaging quality and accuracy evolves from 1,600 dpi in current market to 4,000 dpi in future products; this is done mainly by changing the writing head from visible red-laser diode 650 to 830 nm thermal-imaging diode. These enhancements are enablers of using high-quality and high-durability plates. Another change increases the productivity from 10 to 25 full format plates per hour in future markets; this is also achieved by going from semiautomatic setting to fully automated machine. The above changes affect the system's components and may require their redesign.

The data presented was not collected during the development process, yet it is utilized in the example as if it was collected during such process. The product knowledge revealed in each process stage is limited; thus, simulating the knowledge evolution during a new product development process.

12.3 Development Process Setting

Initially, the development process (and the simulation) started with a pre-defined (best practice) process scheme that can potentially be retrieved from a process repository. The high-level process in Fig. 2.4 was used. In this process, Specification and Conceptual design stages were done in parallel.

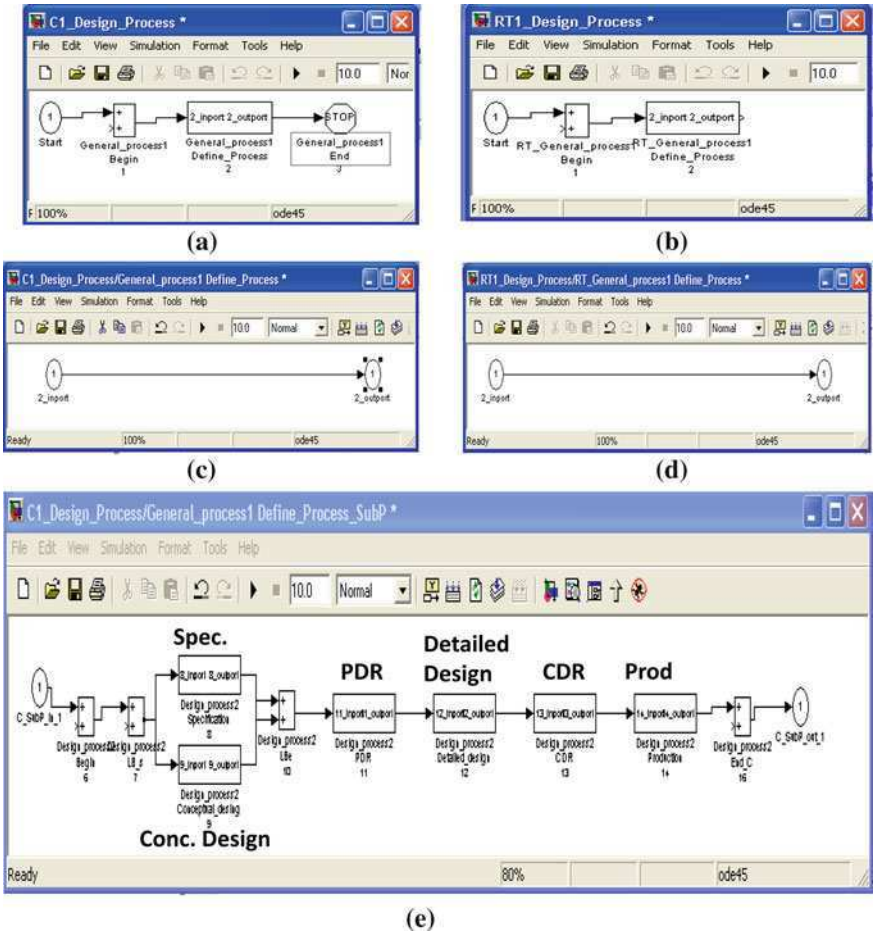


Fig. 12.2 Setting up high-level process model. **a** General process. **b** Initial content of subprocess. **c** RT-process following. **d** Change to C subprocess content. **e** New subprocess content: High-level development process

The actual implementation[1] used the following steps: first a general process was defined, including start, a logic activity Begin, a general activity (being dynamically replaced by subprocesses), and finally, End activity(C-process, Fig. 12.2a). The content of the subprocess (activity #2 in (a)—the general process) was initiated as null (shortcut of input to output), see Fig. 12.2b.

[1] The implementation of a dynamic process scheme was done using Matlab (by Mathworks). The process presentation employs Simulink (within Matlab). Simulink could not be used for simulation since the Simulink engine (like most workflow engines) cannot stop the process, change its scheme, and continue.

The RT-process followed the current process plan (C-process). As it reached the general activity, Fig. 12.2c, a predefined activity assignment was performed: choosing a high-level process. The choice might be manual (by the user from a process repository), or for simulation purposes, setting a specific choice. The content of the general subprocess changed in the C-process accordingly, Fig. 12.2d, to the required high-level process Fig. 12.2e.

Notes:

- The upper level iterations of PDR and CDR were not implemented in this example; their implementation is straightforward by adding logic activities and feedback links
- The upper level process in (a) and (d) looks the same, but the subprocess has a different content. The subprocess of activity 2 in (a) has only one link (b) versus the full subprocess in (e), being updated at run time.
- Each subprocess is a process, therefore has its own Begin and End logic activities.
- The implementation of logic (e.g., parallel execution of activities), Fig. 12.2e, requires additional logic activities (before and after the parallel activities Specification and Conceptual Design).

The example above represents the mode of dynamically expanding the process by replacing the content of an activity by a subprocess. This mode is further used as the process is further elaborated. This expansion allows refining the process hierarchically to any required granularity. This option also allows adding predefined administrative subprocesses within any design activity (e.g., opening a new part number). The latter option was not demonstrated, as its implementation is straightforward.

According to Definition 15, expanding a WRI-WF-net by another WRI-WF-net generates a WRI-WF-net. Yet, since such property is not inherent to a predefined process, it should be either checked or enforced. Typically, predefined processes are acyclic and are WRI-WF-nets, so we shall assume it was enforced.

12.3.1 Following High-Level Process to Conceptual Design

Once the RT-process reached the conceptual design activity (or during that activity), new product knowledge has been generated. The product data collection within the conceptual design activity is assumed relatively simple, yet it follows the main steps described in Sect. 8.5.

For the purpose of demonstration, the following data was assumed (representing a limited product knowledge state).[2] The product is decomposed into four main components: W- frame, X- drum, Y- laser subsystem, and Z- control system. It is assumed that relations at that stage were identified as binary values; the relative impact figures were not yet determined. This initial product knowledge is depicted in Fig. 12.3.

[2] The illustrative data at this state do not represent the data given by the company.

Fig. 12.3 Planning the Conceptual Design phase

	W	X	Y	Z
W - frame		1		1
X - drum	1			
Y - laser	1			
Z - control		1	1	

DSM

Fig. 12.4 Planning the Conceptual Design phase (cont.). **a** Probability DSM. **b** Design Block. **c** Serialized DSM

	W	X	Y	Z
W - frame		p		p
X - drum	p			
Y - laser	p			
Z - control		p	p	

(a)

	DB1
DB1	*P*

(b)

	W	X	Y	Z
W - frame		p		p
X - drum	1			
Y - laser		1		
Z - control			1	

(c)

Process planning according to the above DSM is used as refinement (through hierarchy enhancement) of the Conceptual Design activity. This process detailing is a dynamic expansion (Class IV, Sect. 2.3) of the process, as the product knowledge was not available when the project started.

12.4 Conceptual Design Process Planning

The next step of planning the conceptual design subprocess according to the knowledge gained was to transform the DSM to a probability DSM (Sect. 8.5). Since all the influence values are equivalent, the resulting probability DSM was obtained by replacing the '1' marks with probability p, see Fig. 12.4a.

Optimal DSM ordering result was a single DB Fig. 12.4b, (with an internal order of the activities being XWYZ). The DB implementation requires four resources, working in parallel. The feedback probability P is calculated according to Eq. 4.3, $P = 1-(1-p)^6$.

A serialized implementation of the process is depicted in Fig. 12.4c. The forward probability links are removed, and replaced by the '1' markings.

The variable p is the maximal expected probability of iterations and should be estimated. Additional simulation parameters are not represented by the DSM, and should be estimated. These may include the duration of each activity $D(ai)$, i being activity index, $D(DBj)$–Duration of forming and managing the DB, i.e., cost of communications (meetings, power point presentations, etc.); where in this case, the index $j = 1$, as there is only one DB. If resources were considered, additional simulation data were required.

For comparison, we evaluated both time and resources. It was assumed that all resources are the same and their cost is proportional to the activity duration. Furthermore, resources are assumed to be available as needed and a resource which is not required has other things to do (i.e., does not spend time and cost for waiting). The relaxation of these assumptions can be easily implemented, but comparison for various setting is more complex to obtain.

12.4.1 Conceptual Design Planning: Implementation

Implementation of the process can be done using various logic options. Three logic options were compared (according to Table 7.2); serialized process (e.g., having one resource) in Smith and Eppinger (1997a); parallel process using the logic in Browning and Eppinger (2002); Yassine (2007); and the coupled activities logic (one DB) described in Smith and Eppinger (1997b); Huberman and Wilkinson (2005); Yassine et al. (2003).

Reducing the number of options, e.g., by setting a constraint of maximum two iterations (i.e., activity *W* may be performed three times at most), and restricting activity *Z* to have late start (i.e., completion of all iterations of precedent activities, BR 16.1) the RT-process cases could be fully analyzed and calculated.

Subject to the above restrictions, there were 37 logically correct RT-process paths for parallel implementation; seven paths for serialized implementation; and three paths for the DB logic.

A correct path means that

1. each activity is performed at least once; and
2. for parallel implementation, activity Z can be executed only if activities X and Y have been executed at least once (this requirement is always true for serialized implementation).

The distinct paths of parallel logic and their relative probabilities (for $p = 0.2$) are depicted in Table 14.3 (annex C, Sects. 14.5 and 14.6). The serialized paths included similar cases to the parallel implementation (cases {1, 10, 25, 37} in Table 14.3, Sects. 14.5 and 14.6), and some unique paths {WXWXYZ, WXWXWXYZ, and WXWXYZWXYZ). The DB implementation included only the cases {1, 10, and 37} in Table 14.3.

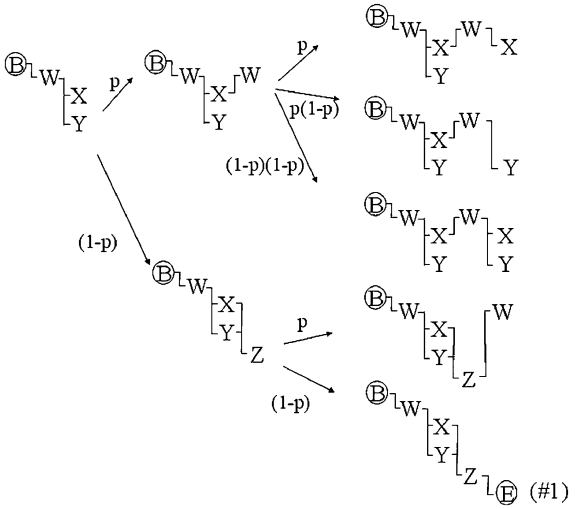
The probabilities were calculated using probability tree. The initial part of the tree for the parallel logic case is depicted in Fig. 12.5.

After the first execution of *W,* both *X* and *Y* activities start in parallel (common to all paths). The order of checking the process options influences the process probabilities. After the first execution of *X* and *Y*, activity *X* may have feedback (to activity *W* with probability *p*). If activity *X* did not send feedback [with probability $(1-p)$], activity *Z* may have feedback to activity *W* with probability *p*. The probability to complete the path without any feedback (path #1 in Fig. 12.5, and in Table 14.3) is $(1-p) * (1-p) = 0.64$, for $p = 0.2$.

In a similar manner, on iterations of *W*, the process can continue by executing only *X*, or only *Y* or both (default). Since checking the "only *X*" option is done first, the branches probabilities are: forward by *X* only with probability *p*, forward by *Y* only with probability $(1-p) * p$, or the default decision to perform both with probability $(1-p) * (1-p)$. The three options presented are actually aggregation of two decisions, and could be presented by two decision steps.

The probability of performing two feedbacks from *X* to *W*, where after *W* iteration only *X* iterates (case #5) is $p * p * p * p = 0.0016$. Making one

Fig. 12.5 Probabilities tree



feedback on *Z*, and proceeding by *Y* only (case #9) has the probability $((1-p) * p) * ((1-p) * p) * ((1-p) * (1-p)) = 0.021504$.

The transformation of the DSM to DPM and then to a process scheme model is demonstrated for the parallel case in Fig. 12.6a. The DPM of the DB is depicted in Fig. 12.6b (note: the activities order within the DB is according to the optimal solution; yet, it has no practical meaning in the simulation). The DPM of the serialized implementation is very similar to the parallel; except that the parallel probability marks are removed, and instead, the '1' marks are added (from *OLw* to *ILx*, *OLx* to *ILy*, and *OLy* to *ILz*).

In the DB implementation, the actual optimal order is *XWYZ*, Fig. 12.6b. Since the order of the activities (*XWYZ* or *WXYZ*) has no impact on the implementation, or the simulation results, the simulation can be implemented as if the order was *WXYZ*.

The process scheme (C-process) of the single design-block process model is depicted in Fig. 12.7a. All activities started together, performed in parallel, and completed together. The design-block self-iteration merged probability is assigned according to Eq. 4.3. In a case of no self-iterations, the links exist in the model but the probability to use the link is set to $p = 0$. Each activity is implemented as part of a DB (having two logic activities prior to the design activity and two after it), with one activity.

The implementation of DBs requires additional extension of the DPM for DB logic activities *DBIL* and *BDOL* (respectively, *IL* and *OL* of a DB, each with internal *IL* and *OL*, Sect. 11.11). These are depicted in Fig. 12.7a, by the two logic activities before and the two activities after the DB activities (WXYZ). The first *IL* of the *DBIL* is used to implement the logic of incoming links to the DB. The *OL* in *DBIL* is merely Split-And to all the activities in the DB.

	B	ILw	W	OLw	ILx	X	OLx	ILy	Y	OLy	ILz	Z	OLz	E
B														
ILw	1						*p*						*p*	
W		1												
OLw			1											
Ilx				*p*										
X					1									
OLx						1								
ILy				*p*										
Y								1						
OLy									1					
ILz							*p*			*p*				
Z											1			
OLz												1		
E													1	

(a)

	B	ILD	X	W	Y	Z	OLD	E
B								
ILD	1						*P*	
X		1						
W		1						
Y		1						
Z		1						
OLD			1	1	1	1		
E							1	

(b)

Fig. 12.6 DPM of Conceptual Design implementation options. **a** DPM of a parallel process. **b** DPM of a design block

The *IL* of the *DBOL* is a Join-And of all the activities. The *OL* of the *DBOL* is used for implementing the out logic of the DB, and is linked by a self-iteration link to the *IL* of the *DBIL*.

The diagrams of the serialized and parallel implementation cases become more complex since each activity is implemented as a DB. In order to simplify their representation (Fig. 12.7b and c, respectively), the DB logic activities, self-iteration links were eliminated and a clear distinction between forward and feedback links was made by dashed lines

The RT-process examples of the single DB serialized process are presented in Fig. 7.4d and f, respectively. The parallel process example depicted in Fig. 7.5a represents process path #1 in Table 14.3 (Sects. 14.5 and 14.6); both Fig. 7.5b and c represent path #3. The differences may indicate different duration of activity *Y* (if $D(Y) < D(X) + D(W)$ then the process in (b) is applicable, and the process in (c) is not); or different BR logic [BR 13.2 in (b), and BR 13.1 in (c)].

Some additional non-trivial RT-process examples are depicted in Fig. 12.8 and Fig. 12.9, indicating the option of performing only activity *X,* or *Y* on the second execution (*Z* requires at least one execution of its precedent activities).

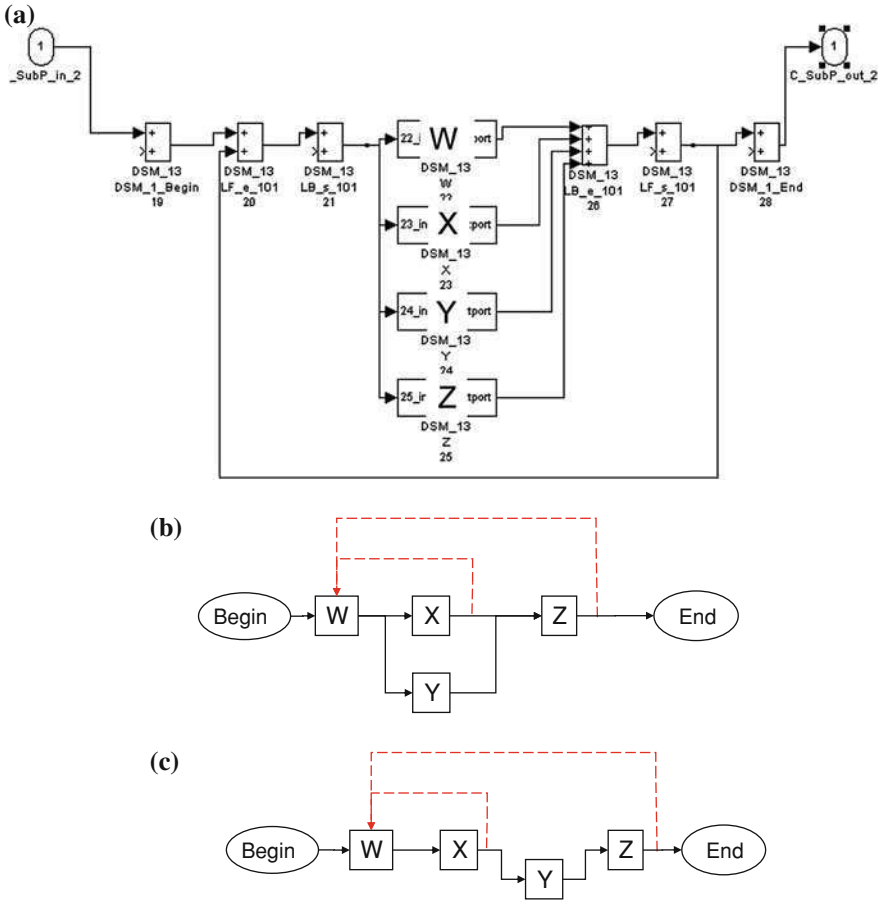


Fig. 12.7 Conceptual Design C-process

Path #2 in Table 14.3, is depicted in Fig. 12.8a. Activity *X* sends a feedback link signal (Z cannot start, as *X* may have additional iterations), but on the second execution of *Y* activity *Z* may start since *X* has completed once (and there are no further expected iterations of *X*). If *Z* had early start (BR 11.2), it could start once *X* and *Y* have completed their first execution (in parallel to *W*). In such case, the second completion of activity *Y* could reinitiate *Z* (depending on the relative duration of the activities and the applied logic). Such process options (not being part of Table 14.3) are depicted in Fig. 12.8b and c. Figure 12.8 b shows the case when *Z* completes before the second execution of *Y*, or if BR 13.2 was applied (start next execution); and Fig. 12.8c shows the case when *Z* did not complete and BR 13.1 was applied.

Path #6 in Table 14.3 (Sects. 14.5 and 14.6) is depicted in Fig. 12.9. In this case, *Z* starts on completion of *X* and *Y*. Iteration to *W* is done only when *Z* completes (i.e., it is not in parallel), therefore the other options described for path #2 are not applicable in this case. Path #7 is similar to #6, only replacing the second execution of *X* by *Y*.

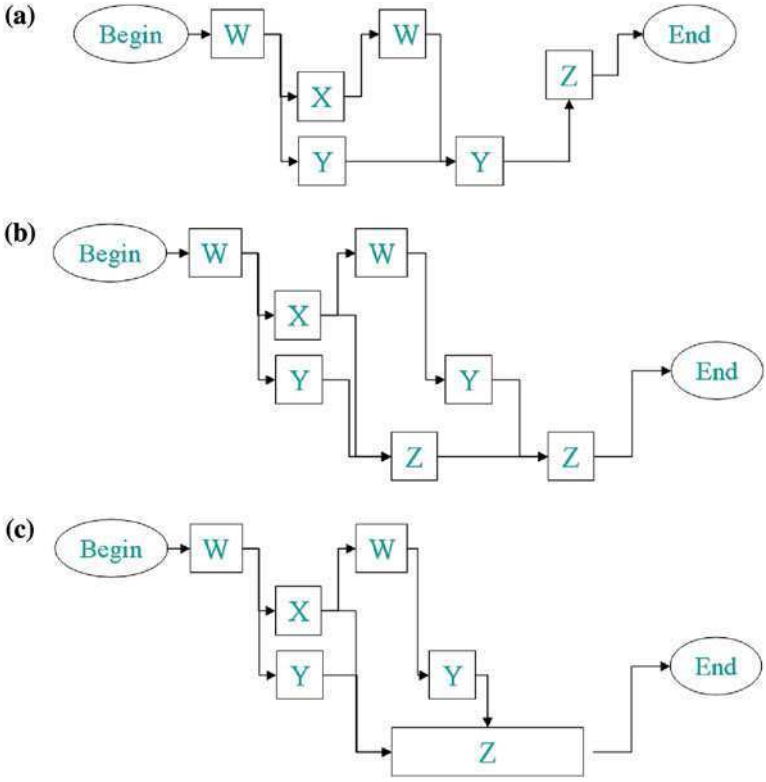


Fig. 12.8 Conceptual Design RT-process examples—case #2

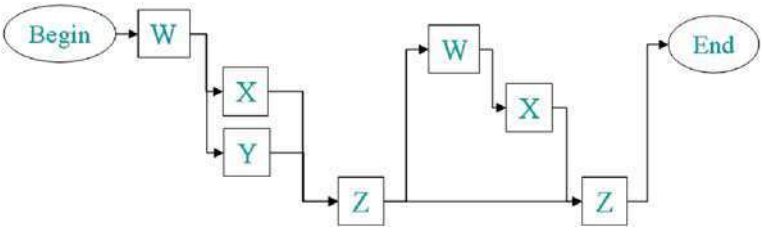


Fig. 12.9 Conceptual Design RT-process—path #6

12.4.2 Conceptual Design Planning: Logic Considerations

The coupled activities in the implementation of the DSM (in Fig. 12.4a) are not interpreted as serialized or as a DB, but as a general process of parallel activities with iterations. The implementation of that logic is interesting and provides rich process phenomena as presented in the previous examples. Applying self-iterations would have enabled more variants of RT-process options (as demonstrated in Sect. 10.7.3).

The logic used was: OL⇒(Σ(*Ii*))⊗(Π(*Fi*)) as in (Cho and Eppinger 2005) and IL⇐(Σ(*Ii*)) + (Σ(*Fi*)) (not implemented in the surveyed articles). Such logic combination of OL and IL was defined in Sadiq and Orlowska (2000), as leading to lack of synchronization; or equivalently, not satisfying the 'well-handled' conditions (van der Aalst 2000). In the context of administrative processes, lack of synchronization is unacceptable since the activity needs to start over again. However, it is acceptable in the case of the iterative NPD process. The merger rule or the "start new iteration" rule (BR 13.1 and BR 13.2, respectively) explicitly specifies the logic types applicable in such BR case.

Formal proofs for the serialized process (Proposition 18) and DB case (Proposition 20) were presented in Karniel and Reich (2011); yet a formal proof of the applicability of parallel logic is still an open issue. It is expected that the current foundation of the basic proofs in Sect. 9.4, with extensions using the Petri net definitions of Siphons and Traps might provide the required basis for proving the additional cases in future research.

12.4.3 Simulation Results

The simulation parameters are the activities duration $D(a_i)$, where $a_i \in \{W,X,Y,Z\}$, and the probability p is as depicted in Fig. 12.4. For the coupled activities (DB), the duration is $D(DB) = \max(D(a_i))$, and the self-iteration merged probability is $P(DB) = 1-(1-p)^6$ (i.e., all probabilities of coupled activities are contributing to the probability of self-iteration of the DB (Karniel and Reich 2007b).

The measured results were the total process time T and the required resources R. For simplicity, one resource per activity time unit was assumed; thus, the required resources of serial and parallel process implementations were equivalent to the duration of the serialized process. For the coupled activities, it was assumed that the resources are the sum of activity durations (i.e., no cost of waiting). Activity duration was not reduced with iterations, i.e., no learning occurs.

Results are presented in Table 12.1. The table has three sections: simulation parameters, statistics, and examples of practical performance questions. The process time and resources due to iterations have skewed distributions (like the distributions in Fig. 11.6). The statistics presented includes minimal process time and its probability, maximal time and its probability, average time and standard deviation (which is relatively large), and the average and standard deviation of resources.

The performance questions presented were the probability to complete the project in the given limited time (deadline); or the given resources (Williams 2003). The probabilities are calculated for two limitations: process time, $T < 20$, and process resources, $R < 32$. Four cases of simulation parameters are described, three cases with different iteration probabilities, $p \in \{0.01, 0.1, 0.2\}$, and an additional case with different activity durations.

The results were sensitive to the parameters, and especially the performance results were highly sensitive to the choice of limits. In general, a parallel process

Table 12.1 Simulation results for various DSM-based Logic options

Simulation parameters				General statistical results				Performance	
#	*P*	*D*(*W*,*X*,*Y*,*Z*)	Simulation type	Time min (prob. %)	Time max (prob. %)	Time Avg./ std	Resources Avg./ std	Prob. (%) T < 20	Prob. (%) R < 32
1	0.01	3,7,2,1	Serial	15 (0.98)	45 (0.01)	15.26/1.88	15.26/1.88	98.01	99.96
			Parallel	12 (0.98)	36 (0.01)	12.23/1.65	15.26/1.88	98.02	99.96
			Coupled	9 (94.1)	27 (0.34)	9.55/2.29	15.93/3.82	99.65	99.65
2	0.1	3,7,2,1	Serial	15 (80)	45 (0.64)	17.90/6.42	17.90/6.42	80.80	96.11
			Parallel	12 (80)	36 (0.81)	14.64/5.73	17.90/6.42	81.60	96.11
			Coupled	9 (53.1)	27 (21.9)	15.19/7.27	25.32/12.1	78.04	78.04
3	0.2	3,7,2,1	Serial	15 (0.98)	45 (1.44)	21.39/9.12	21.39/9.12	62.40	84.96
			Parallel	12 (0.98)	36 (2.60)	17.96/8.27	21.39/9.12	64.80	84.96
			Coupled	9 (26.2)	27 (54.44)	20.54/7.67	34.23/12.7	45.56	45.56
4	0.1	6,5,1,1	Serial	13 (80)	39 (0.64)	15.77/5.99	15.77/5.99	80.80	96.11
			Parallel	12 (80)	36 (0.81)	14.68/5.77	15.77/5.99	81.60	96.11
			Coupled	6 (53.1)	18 (21.9)	10.12/4.85	21.94/10.51	100.00	78.04

Table 12.2 Evolving Product Knowledge - Components

Initial product decomposition	Current decomposition
X (drum)	B (drum body)
	C (drum rotation system)
Y (Laser)	D (Laser writing head)
	E (Laser driving system)
W (Frame)	A (Chassis and Cover)
	F (Load/Unload unit)
	G (Registration & punch unit)
Z (Control system)	H (Control system)

has shorter time than the serial process, and according to the above assumption use similar resources; hence, could be considered as better practice. Comparison of a parallel process to the coupled activities process is dependent on the case-specific details. The average time of the parallel process was better than the coupled activities process in cases {2, 3} and worse in cases {1, 4}. Fewer average resources were required for the parallel (and serial) process in all cases.

The presented results (Karniel and Reich 2009) demonstrate the sensitivity of process performance indicators to the specific simulation parameters. Therefore, there are no general rules of thumb for decision-making. Decisions should be made according to the specific case parameters, and preferably use statistical analysis, as discussed in Sect. 11.9, and presented in the next sections.

12.5 Planning the Detailed Design Stage

By the end of the conceptual design stage, it is expected to gain more knowledge regarding the product. This product knowledge can be used for planning the design stage (the predefined detailed design activity in Fig. 12.2e). The high-level predefined design activity has no predefined content. The actual content depends on the product knowledge. Another simplified model of the example product (Fig. 12.1), being extended (from four) to eight main subsystems (components) of the LDI plotter system was considered. The simplified assumption of 1:1 linkage between product components and process activities (Sect. 8.5, Step (I) sub stage (A)] is used again.

12.5.1 DSM Data Collection

Mapping the evolution of the product data knowledge between the development stages is described in Table 12.2, indicating more elaborated decomposition.

Additional knowledge regarding the links between components, (i.e., the influence parameters) provided estimation of the influence "magnitude" {high, medium, low}. An example of the knowledge gained is represented in Table 12.3, using component names defined in the previous table. The ranking values used were suggested in Sect. 4.1.

Table 12.3 Evolving Product Knowledge—Links

From	To	Parameter Description	Influence	Value
B	A	Weight	Med	3
B	A	Size	High	9
B	C	Weight	Low	1
B	C	Moment	High	9
B	G	Size—Number/force of clumps	Med	3
B	G	Size—Number of holes	Med	3
C	A	Weight	Low	1
C	A	Size	Med	3
C	A	Max velocity	Low	1
C	B	Inertia moment	Low	1
C	H	Rotation accuracy	Low	1
C	H	Resolution	Low	1
C	H	Acceleration	Med	3
D	C	Power	Low	1
D	C	Exposure time	Low	1
D	E	Weight	Low	1
D	E	Size—surface area (near drum)	Med	3
D	E	Size—distance	Med	3
D	E	Exposure time	Med	3
D	H	Diodes type	Low	1
D	H	Exposure time	Low	1
E	A	Weight	Low	1
E	D	Moving Speed	Low	1
E	D	Size of moving mechanism	Med	3
E	H	Resolution	Med	3
F	A	Weight	Low	1
F	A	Size	Med	3
F	G	Plate load speed	Low	1
F	H	Load/unload rate	Low	1
F	H	Load/unload time	Low	1
G	B	Weight	Low	1
G	B	Punch force	Low	1
G	B	Beam pressure	Low	1
G	C	Gripper force—radial	Med	3
G	C	Gripper force—contact area	Med	3
G	C	Beam pressure	Med	3
G	H	System type	Med	3
H	D	Processing time	Low	1
H	E	Processing time—location	Med	3
H	E	Processing time—velocity	Low	1
H	E	Processing time—acceleration	Low	1

Fig. 12.10 Matrix representation of linkage values

		Chassis & Cover	Drum	Drum Rotation system	Laser head	Writing head drive sys	Load /unload unit	Registration & punching unit	Control system
		A	B	C	D	E	F	G	H
Chassis & Cover	A		Weight 3 Size 9	Weight 1 Size 3 Max velocity 1		Weight 1	Weight 1 Size 3		
Drum	B			Inertia Moment 1				Weight 1 Punch force 1 Beam pressure 1	
Drum Rotation system	C		Weight 1 Inertia Moment 9		Power 1 Exposure time 1			Gripper force 6 Beam pressure 3	
Laser head	D					Speed 1 Size 3			Processing time 1
Writing head drive sys	E				Weight 1 Size 6 Exposure time 3				Processing time 5
Load	F								
Registration & punching	G		Size 6 Surface 1 Roughness				Plate load speed 1		
Control system	H			Rotation accuracy 1 Resolution 1 Acceleration 3	Diodes 1 Exposure time 1	Resolution 3	Load/Unload rate 1 Load/Unload Time 1	System type 3	

The values were summarized in a matrix form in Fig. 12.10. As previously discussed, when using a summation method, it does not matter if summation is done over aggregated parameters (which may have internal structure) or over distinct parameters.

The parameters values in the above example are aligned with the values in Sered and Reich (2006); indicating that values other than {1, 3, 9} could be interpreted as sum of influences of a more detailed consideration. For example, the link *H* to *E* is the summation of processing time of location, velocity, and acceleration (from Table 12.3); thus, the aggregated influence value of the parameter process time is 5.

Since the overall influence figure is dependent on the number of parameters being considered, links with better knowledge might be overestimated (due to assessment of more parameters); therefore in addition to the bottom-up approach, a top-own approach should be used to balance the overall results. Further research is required in order to define better methods for assigning influence figures in complex cases.

Setting the parameters and the influence values is an engineering knowledge-core process [Sect. 8.5, stage (B)]. Once the assignment is done, the next stage (C) is technical. The influence values in each cell were summed, creating the Impact DSM, with numeric values, depicted in Fig. 12.11a.

An additional impact figure on the diagonal was assigned to activity *F*. Indicating self-iteration influence may imply the confidence (or actually the uncertainty) assigned to the design of a component, e.g., component that was never designed before, or which incorporates new technology. This assignment will be later translated to self-iteration probability, and could be assigned either at the current stage or directly at the next stage.

	A	B	C	D	E	F	G	H
A	0	12	5		1	4		
B		0	1				3	
C		0	0	2			9	
D				0	4			1
E				10	0			5
F						5		
G		7				1	0	
H			5	2	3	2	3	0

(a)

	A	B	C	D	E	F	G	H
A	0.00	0.52	0.22		0.04	0.17		
B		0.00	0.04				0.13	
C		0.17	0.00	0.09			0.39	
D				0.00	0.17			0.04
E				0.43	0.00			0.22
F						0.22		
G		0.30				0.04	0.00	
H			0.22	0.09	0.13	0.09	0.13	0.00

(b)

Fig. 12.11 Impact DSM and Probability DSM. **a** Impact DSM. **b** Probability DSM

In the next step, the impact (or influence) DSM was converted to probability DSM using linear conversion, where the highest probability is assigned to the highest influence value. As previously discussed, the assessment of the maximal probability value requires more research. In the following example, the maximum probability was set to $pmax = 0.52$ (Yassine 2007); resulting in the probability DSM in Fig. 12.11b.

Self-iteration probabilities (or influence assessment if done in the previous influence assessment stage) do not change the ordering (in next section), but influence the simulation. Self-iterations could be interpreted as an implicit duration expansion of an executed activity. If such could be done in parallel to other activities, it could be interpreted as a simulation of overlapping activities. Yet, the explicit definition of self-iterations allows for a better distinction of the cases; it allows the explicit implementation of learning curve; and allows the definition of the aggregated DB self-iteration merged probability, demonstrated previously.

In the current example, the self-iteration probability assigned to activity *F* followed the same linear conversion rules (of the assessment in previous stage), or could have been directly set based on direct assessment of the probability to have the iterations of the activity. Such probability might also be influenced by the choice of resources, e.g., an inexperienced engineer is assumed to require more design iterations than an experienced engineer does.

12.5.2 DSM Reordering

The reordering result of the example according to the cost function (Eq. 4.1) is depicted in Fig. 12.12. Using a one-to-one correspondence of components and design activities, the DSM structure indicated that design activity of component *F* (Load/Unload unit) was not affected by changes of any other subsystem; therefore, it becomes the first activity (by partitioning). Design activity for subsystem *A* (cover) was influenced by other activities, but was not influencing other activities; therefore, it should wait until the other activities completed. All other activities were coupled, forming an activity loop (*BGCDHE*).

Fig. 12.12 DSM optimal reordering

	F	B	G	C	D	H	E	A
F	0.22							
B		0.00	0.13	0.04				
G	0.04	0.30	0.00	0.00				
C		0.17	0.39	0.00	0.09			
D					0.00	0.04	0.17	
H	0.09		0.13	0.22	0.09	0.00	0.13	
E					0.43	0.22	0.00	
A	0.17	0.52		0.22			0.04	0.00

Table 12.4 Design block calculations

One design block (*BGCDHE*)	6*(*F**(0.3 + 0.17 + 0.39 + 0.13 + 0.22 + 0.09 + 0.43 + 0.22) + *C**(0.13 + 0.04 + 0.09 + 0.04 + 0.17 + 0.13)) + *F**(0.04*3^*q* + 0.09*6^*q* + 0.17*8^*q* + 0.52*7^*q* + 0.22*5^*q* +0.04*2^*q*)	518.47
Two design blocks (*BGC*) (*DHE*)	3*(*F**(0.3 + 0.17 + 0.39) + *C**(0.13 + 0.04)) + 3*(*F**(0.09 + 0.43 + 0.22) + *C**(0.04 + 0.17 + 0.13)) + *F**(.04*3^*q* + 0.09*6^*q* + 0.13*4^q + 0.22*3^q + 0.17*8^*q* +0.52*7^q + 0.22*5^*q* + 0.04*2^*q*) + C*0.09*2^*q*	412.22

The activity loop was decomposed into two DBs (*BGC*) and (*DHE*) with a feedback link. The decomposition is an optimization between the cost of large groups and the cost of separation. Using Eq. 4.1 the calculation results for $F = 3$ (forward constant), $C = 64$ (closed-loop constant), and $q = 2.32$, are demonstrated in Table 12.4. Parametric analysis shows that for $F = 3$ the separation into two DBs will occur with $q = 2.32$, and any $C > 3.012$; or $q = 3.19534$ and any C > 3. The latter result is very close to the value of q in the simple cycle case, in Sect. 4.2.

Design Block Internal Ordering

It could be assumed that the internal ordering of a DB is inherently derived from minimization of its feedback link, i.e., minimizing the first component in Eq. 4.1. A specific case would be the complementary links (*D, E*) and (*E, D*). In the latter case, it could be assumed that the link with lower probability would be the feedback link, such that the sum of link values will be reduced.

A modification of the above DSM is presented in Fig. 12.13a, where the value of link (*D, E*) was replaced by the probability value P(*D, E*) = 0.17, and (*E, D*) by P(*E, D*) = 0.43, while all other link values remain the same. Two DSM configurations are presented; in Fig. 12.13a, it is the same configuration as in Fig. 12.12 (i.e., *DHE*). An optimal reordering of the DSM yields the DSM in Fig. 12.13b, where the internal order of the second DB has changed. In the current example,

	F	B	G	C	D	H	E	A
F	0.22							
B		0.00	0.13	0.04				
G	0.04	0.30	0.00	0.00				
C		0.17	0.39	0.00	0.09			
D					0.00	0.04	0.17	
H	0.09		0.13	0.22	0.09	0.00	0.13	
E					0.43	0.22	0.00	
A	0.17	0.52		0.22			0.04	0.00

(a)

	F	B	G	C	H	E	D	A
F	0.22							
B		0.00	0.13	0.04				
G	0.04	0.30	0.00	0.00				
C		0.17	0.39	0.00			0.09	
H	0.09		0.13	0.22	0.00	0.13	0.09	
E					0.22	0.00	0.43	
D					0.04	0.17	0.00	
A	0.17	0.52		0.22		0.04		0.00

(b)

Fig. 12.13 Design block reordering **a** Design block (DHE) **b** Design block (HED)

Table 12.5 Design block calculations—internal order

(*DHE*)	$[F^*[0.09 + 0.17 + 0.22] + C^*(0.04 + 0.43 + 0.13)]$ $+ F^*(0.09^*6\hat{}q + 0.13^*4\hat{}q + 0.22^*3\hat{}q + 0.04^*2\hat{}q)$ $+C^*(0.09^*2\hat{}q)$ $= C^*1.0494 + 37.451$
(*HED*)	$[F^*[0.22 + 0.04 + 0.43] + C^*(0.13 + 0.09 + 0.17)]$ $+ F^*(0.09^*5\hat{}q + 0.13^*3\hat{}q + 0.22^*2\hat{}q + 0.04^*3\hat{}q)$ $+C^*(0.09^*4\hat{}q)$ $= C^*1.54123 + 23.18654$

there is no ordering by which all the pairs of complement links are ordered such that $P(a_i, a_j) > P(a_j, a_i)$, where $i > j$ (i.e., forward link value larger than feedback link). The value $P(D, H) = 0.09$ in Fig. 12.13b, is higher than the value $P(H, D) = 0.04$. Any other reordering will make some pair of links to have the higher value in the feedback position.

Using the expressions in Table 12.4, and comparing the cost functions (using Eq. 4.1) we can calculate that the optimal ordering would have been (*HED*) for closed-loop constant $C < 29.0025$ in Fig. 12.13b. Ignoring the fixed parts of the DSM that are the same, we need to calculate the contribution of the links within the block and the links to and from other blocks. The calculations of both cases are presented in Table 12.5, where $F = 3$ and $q = 2.32$.

Using similar calculation in the "main" example (Fig. 12.12) the order (*HED*) would have been preferred over (*DHE*) for $C < 15.6394$. In that example, the order (*DHE*) was preferred since the value used was$C = 64$

As previously indicted, the internal order of a DB has no influence on the simulation when using parallel logic, where all activities within the DB start and complete together. However, changes in DB content, do affect the simulation.

12.5.3 Probability DSM of Design Blocks

Using DB implementation, the block is treated as a compound activity with self-iteration probability and duration (Karniel and Reich 2007a). The self-probability

Fig. 12.14 Probability DSM of design blocks

	Begin	F	BGC	DHE	A	End
Begin	0.00					
F	1	0.22				
BGC		0.04	0.70	0.09		
DHE		0.09	0.32	0.72		
A		0.17	0.63	0.04	0.00	
End					1	0.00

was calculated according to Eq. 4.3. For the current example, the DSM with DBs (represented as one activity each) is depicted in Fig. 12.14.

As previously indicated, a DB may have one activity. Activity *F* being implemented as a DB has self-iteration like any other DB; thus, its management is not unique. Activity *A* being a DB has a feedback link that is implemented as part of the process scheme model, and its probability at run time is zero (other zero values in the DSM are not modeled in the process scheme). Begin and END logic activities have no self-iteration links.

DBs (*BGC*) and (*DHE*) formed coupled activities loop; therefore, the implementation rules and the business rule options applied to coupled activities (Sect. 10.5) were applicable in this case as well.

12.5.4 DSM Conversion to Process Scheme

The DSM was converted to a DPM. The conversion was done in the same manner as previously presented for the WXYZ process example (depicted in Fig. 12.6). A parallel implementation of the DBs is depicted in Fig. 12.15a. Activities *F* and *A* were also implemented using the DB logic mechanism, since in general, the content of a DB may change during the process, as demonstrated in the next section. A simplified C-process implementation is presented in Fig. 12.15b, where solid lines indicate forward links, and dashed lines indicate feedback links between DBs. Such simplified diagrams will be used to further present C-processes and RT- processes.

12.5.5 RT-Process Using Design Blocks

In order to check the implication of simulation parameters, the process was simulated in batches of 1,000 runs each. An example of run time process is depicted in Fig. 12.16. The process is described using a non-scaled time line (representing relative activity duration). In this example, DBs (*BGC*) and (*DHE*) started in parallel, applying BR 14.2 for the parallel DBs. DB (*BGC*) had a self-iteration, and the second execution was in parallel to (*DHE*).

When DB (*DHE*) completed, a feedback link was activated causing another execution of (*BGC*). This third execution could not be in parallel to the second

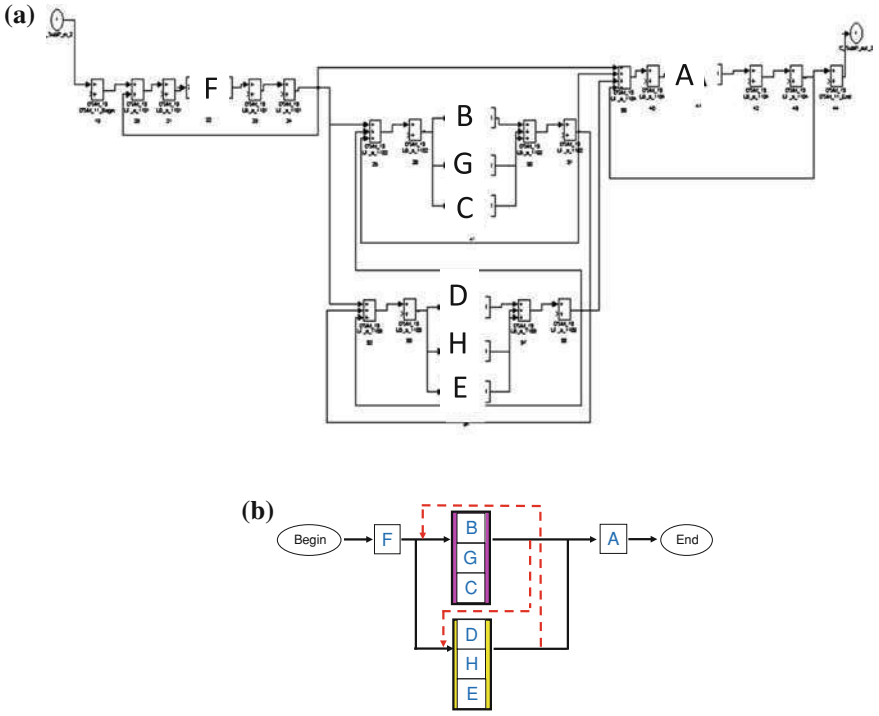


Fig. 12.15 Coupled Design blocks. **a** Parallel design blocks. **b** Parallel design blocks simplified

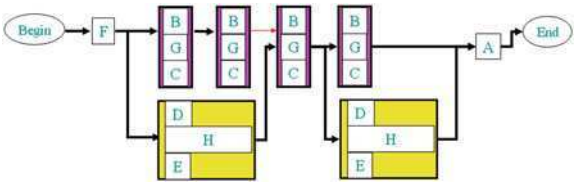


Fig. 12.16 RT process of Coupled Design blocks

execution of (*BGC*). Note: the duration of *H* in this example was D(*H*) = 10, and the duration of DB (*BGC*) was D(*BGC*) = 4; thus, DB (*BGC*) completed two executions before the completion of (*DHE*). Applying BR 13.2 would have caused a similar result even if D(*H*) < 8. However, in the case of BR 13.2, if D(*H*) < 4, then the second execution of (*BGC*) would have been the result of the feedback link from (*DHE*) and not a self-iteration. Using BR 13.1 the executions can merge.

When (*BGC*) completed its third execution (= second iteration) a feedback link to (*DHE*) and a self-iteration link were assigned, causing iteration of (*DHE*) and additional iteration of (*BGC*). Both are considered feedback links since they are within coupled activities.

Activity *A* had a late start (BR 16.1), i.e., after all its precedent activities have completed. When (*BGC*) had completed its second execution, it linked to *A*,

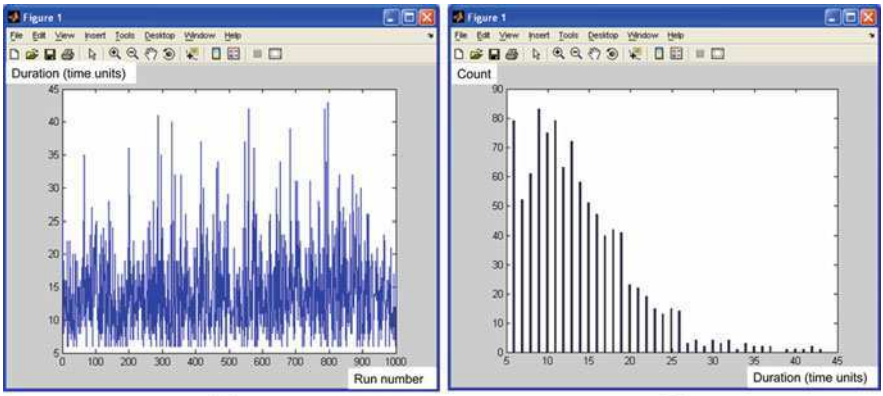


Fig. 12.17 Coupled Design blocks process duration D(H) = 1. **a** RT-process duration. **b** RT-process duration distribution

according to BR 16.1. Since there were additional iterations of (*BGC*) due to a feedback link of (*DHE*), the link to *A* was removed and replaced by a link to the next iteration (third execution) of (*BGC*); such link is marked by a thin (red) line, indicating it is not an actual self-iteration link. Using that link indicates that the next iteration waits for the completion of the previous one.

For demonstrating the implication of relative duration, the duration of activity H was changed D(H) = {1 to 10}. The duration of the other activities were set as follows: D(B) = D(G) = D(C) = 4; D(D) = D(E) = 2; and D(A) = D(F) = 1. The duration time units are not specified (e.g., days).

For each duration value of D(H), 1,000 simulation runs were performed. The process durations T, of 1,000 runs, for D(H) = 1, are presented in Fig. 12.17a. Note: the fixed duration of high-level activities (Fig. 12.2), was subtracted from the total, i.e., the presented figure indicates only the duration of the design subprocess.

The minimum process duration was 6 (time units). Resources calculations assumed one resource unit per one time unit of an activity. It also assumed that resources within a DB, which are not active, are not kept idle. With these assumptions, the minimal resources required were 19. The maximal process duration for D(H) = 1 was 43, and the maximal resources utilization was 157. The minimal duration remained 6 until D(H) = 4 (i.e., until both DBs had the same duration); then, it linearly increased with D(H) as activity H became part of the critical path. Several sample results are presented in Table 12.6.

The process duration distribution is presented in Fig. 12.17b. The distribution shape is right skewed (positive skew) with skewness measure $\gamma = 0.77$.[3] It has

[3] Skewness measure is given by $\gamma = \frac{\sqrt{n(n-1)}}{n-2}\sqrt{n}\sum_{i=1}^{n}(x_i-\bar{x})^3 \Big/ (\sum_{i=1}^{n}(x_i-\bar{x})^2)^{3/2}$(Whitley 1994).

Table 12.6 Run time results of parallel design blocks

Simulation parameter	Process duration			Resources		
D(*H*)	Min	Max	Avg., std	Min	Max	Avg., std
1	6	43	13.85, 6.30	19	157	51.21, 22.52
4	6	50	14.32, 6.46	22	193	53.41, 24.63
10	12	60	21.05, 7.31	28	202	57.60, 25.43

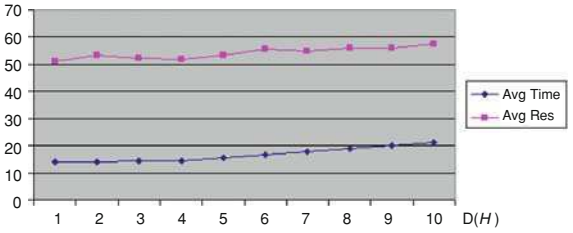


Fig. 12.18 Coupled Design blocks process duration

high-standard deviations. As D(*H*) increased, the standard deviation increased (Table 12.6) and the distribution becomes closer to the case of one DB in Fig. 11.6. Graphs of the average time and average resources as functions of D(*H*) are depicted in Fig. 12.18.

The average time increased and so did the difference between the average time and the minimal duration. The minimal duration (no iterations) increased linearly with D(*H*), from D(*H*) = 4, once its duration becomes part of the critical path. There was a slight increase of average process time due to iterations, but the basic result remained the same until D(*H*) = 4. From D(*H*) = 4 to D(*H*) = 10 as the minimal duration increased linearly, the average process duration increased in a linear manner as well. The results in this case, indicated that the minimal duration had a major influence on the overall process duration.

There was a slight increase of the average resource utilization. The deviation was high and the fluctuations seemed to be greater than the average increase. The average resource utilization increased, but given the high variation, there is no statistical difference between the results. If resources were idle during DB performance (waiting to other activities in the DB to complete), then the increase in resource utilization was aligned with the increase of the average process time as D(*H*) increases.

12.5.6 Business Rules Modifications

Using the same example with an early start of activity *A* (BR 13.2) would result in multiple executions of *A*, depicted in Fig. 12.19. The total time of the process (with the same feedback and self-iteration decision) is similar to the RT-process of the former example, but requires more resources, as *A* executes more times.

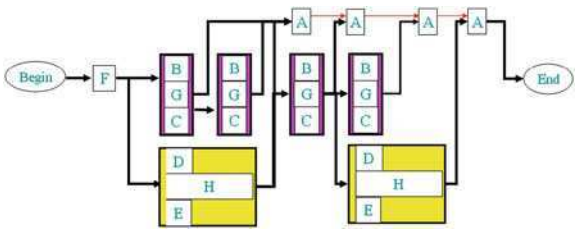


Fig. 12.19 Early start RT

In this example, D(A) = 1. Activity A completes each time before its next iteration and links to the End logic activity. Once activity A has another iteration, the link to End should be removed (End always waits for all iterations to complete); and replaced by a link to next activity iteration (thin red line), as in the previous example.

In terms of process performance, the process duration was the same as the previous process; however, more resources were required. Hence, for this example the conclusion was that early start did not contribute. However, if in addition to using BR 16.2 (early start), BR 12.2 is also used (i.e., on second or later iteration the activity can link to the END activity and not necessarily to the next serial activity), then activity A may execute only once. In such case, resources are not increased, and process duration may decrease.

12.5.7 Additional DSM Implementation Options

The DBs in the example are part of an activity cycle. Besides the parallel implementation of activities within the DBs, the DBs can be serialized. The DB (*DHE*) has to wait for the completion of (*BGC*) and can iterate to (*BGC*) through the feedback link. Serial implementation of the DBs is depicted in Fig. 12.20. Solid lines indicate forward links, and dashed lines indicate feedback links.

The performance of serialized DBs implementation was inferior to parallel DBs in terms of duration. Following the conclusions of previous section, a late start logic was applied, with a merging rule (BR 13.1, i.e., iterations of (*BGC*) do not necessarily cause iterations of (*DHE*) for large D(H) values, D(H) > 4). The results of DB serialization had small (statistically insignificant) reduction of resource utilization. Early start logic was not beneficial since in the serialized process, the second DB (*DHE*) always followed (*BGC*).

Finally, DSM-based plan can be implemented without DBs. Such implementation has the following structure, Fig. 12.21. Activities F, B, and D, which have no other preceding activities, start in parallel from the Begin activity. The other activities get forward/feedback links accordingly, the actual order is as if the order was (*F,B,D,G,C,H,E,A*), Fig. 12.21a. The C-process is depicted in Fig. 12.21b, feedback links are dashed (red) lines.

The comparison to the case of no DBs implementation, Table 12.7, is more interesting. The process has more serial structure; thus, as expected the duration is

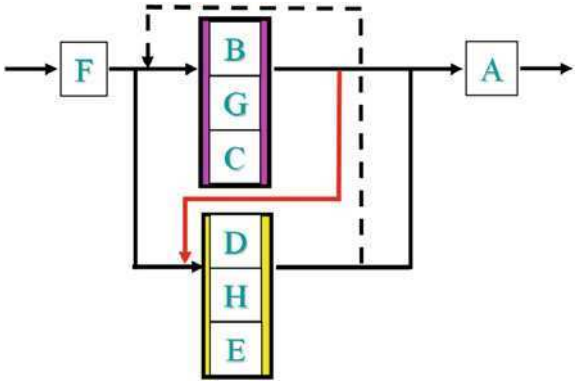
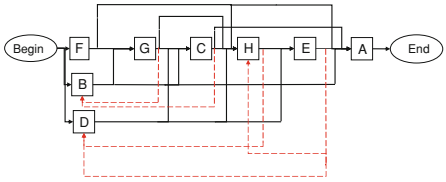


Fig. 12.20 Serializing Coupled Design blocks

	F	B	D	G	C	H	E	A
F	0.22							
B		0.00		0.13	0.04			
D			0.00			0.04	0.17	
G	0.04	0.30		0.00	0.00			
C		0.17	0.09	0.39	0.00			
H	0.09		0.09	0.13	0.22	0.00	0.13	
E			0.43			0.22	0.00	
A	0.17	0.52			0.22		0.04	0.00

(a)



(b)

Fig. 12.21 No design blocks. **a** DSM no design blocks. **b** C-Process no design blocks

Table 12.7 Run time results—no design blocks

Simulation parameter	Process Duration			Resources		
D(*H*)	Min	Max	Avg., std	Min	Max	Avg., std
1	16	39	21.42, 3.02	19	57	26.85, 3.85
4	19	46	22.37, 3.06	22	62	30.82, 4.24
10	25	49	29.92, 3.10	28	64	36.97, 4.88

longer. It should be indicated that while minimum duration was significantly longer, the simulated maximum duration was shorter. Additionally, the deviation was much smaller, since it is not in blocks. If consistency of process time is valuable, this fact may be significant (e.g., for manufacturing processes).

Resource utilization was decreased, since it only required resources that are utilized, unlike DB implementation, where all activities iterate.

12.5.8 Learning

So far the simulations were done assuming that activity (or DB) iterations have the same duration. Learning implies reduction of the activity duration, as only parts of

Table 12.8 Run time results—learning impact

Simulation parameter	Process duration			Resources		
D(*H*)	Min	Max	Avg., std	Min	Max	Avg., std
1	600	1,926	867, 261	1,900	9,362	3,175, 832
4	600	2,441	932, 327	2,200	10,731	3,471, 930
10	1,200	3,812	1,628, 438	2,800	11,472	4,160, 1,024

the activity have to be re-executed, or due to the experience gained, the execution of activities is shorter. The learning model applied assumed work reduction by a constant factor *LR* (learning ratio); such that the duration is reduces by that factor on each iteration compared to the previous execution.

$$\mathrm{D}(ai, k) = \mathrm{D}(ai) \cdot LR^{\mathrm{k}}, \tag{12.1}$$

where *ai* is the activity, D(*ai*) is its initial duration, *LR* is the learning ratio, and *k* is the number of iterations of that activity.

The activity duration was re-scaled such that each time unit (as used in Table 12.6), became 100 time units in order to get meaningful results, since the simulation progresses with integer time units.

A case of no learning ($LR = 1$) simulation results using 100 time units were similar to those in Table 12.6, with close (scaled) results of average and standard deviation.

The results of applying learning model, with $LR = 0.5$ are presented in Table 12.8.

Comparison with Table 12.6 (with the appropriate scaling) yields that applying a learning model reduced the maximal time, the average time, and the deviation. With learning, the additional iterations have less impact on the average duration since their added time is reduced, respectively, the impact of the minimal duration becomes more significant. The total time factor of infinite iterations of an activity is an infinite sum. In the case of $LR = 0.5$ it is $1 + 1/2 + 1/4 + \ldots + \ldots$, which converges to 2 (i.e., the total time would be twice the initial duration). In general, this sum converges to $1/(1 - LR)$.

Another type of learning model involves reduction of the probability for next iteration occurrence (that could be combined with duration reduction). Such model would have similar effect to duration reduction because it reduces the occurrence of multiple iterations.

12.6 Simulation Continues

The high-level process status at this point is described (in a non-formal way) in Fig. 12.22. The Conceptual Design and Specification activities were completed (marked by thick, green outline), and during the Conceptual Design, the plan for Design activity was set. During the Conceptual Design, it was also decided to

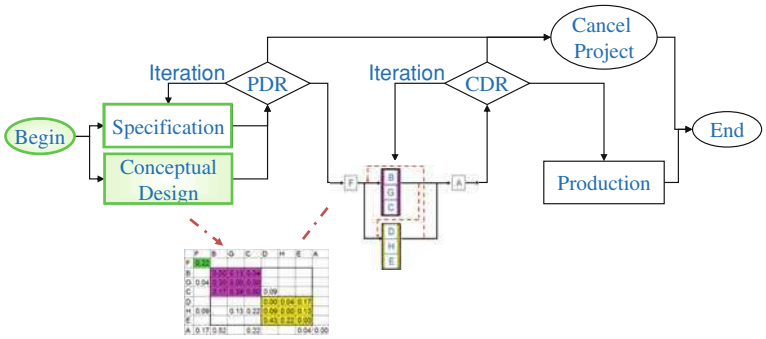


Fig. 12.22 High-level process status view

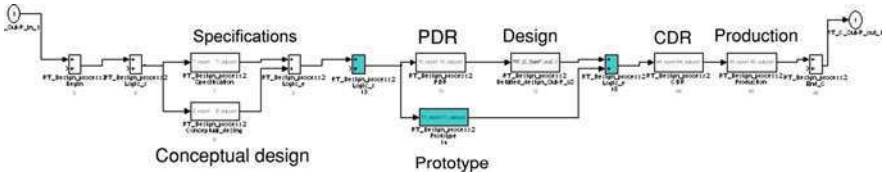


Fig. 12.23 Adding Prototype activity by ad-hoc change

add a Prototype activity in order to check the implementation of the new laser diode technology. This was planned to start once the Conceptual Design and Specification activities have completed. The implementation of such change (which is not derived from the product structure), was done by *ad-hoc* change of the process.

The process change implementation depends on the process planners/managers decision regarding when the activity should start and which activity should wait for its completion. For example, it was decided to perform it in parallel to the PDR and Design activity, and it should be completed before the CDR (other options include immediate implementation in parallel to Conceptual design or completion before the PDR activity). The implementation of the first option is depicted in Fig. 12.23. Adding the Prototype activity required addition of logic activities (marked in grey/blue) that implement Split-And and Join-And, respectively. Note: immediate implementation of a prototype activity can be done by linking it to the Begin activity (having Split-And logic).

12.7 Dynamic Design Block Changes

Dynamic changes of the product structure imply dynamic changes of the process activities. Adding or deleting activities would yield process changes as described in Sect. 11.2. The implementation of changes in the process scheme, e.g., due to

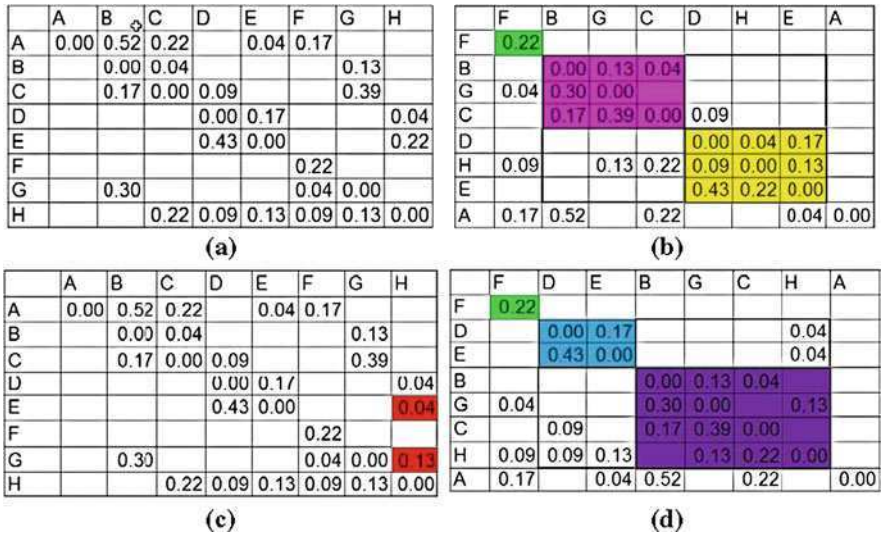


Fig. 12.24 Design blocks changes. **a** Probability DSM—case I. **b** Reordered DSM—case I. **c** Probability DSM—case II. **d** Reordered DSM—case II

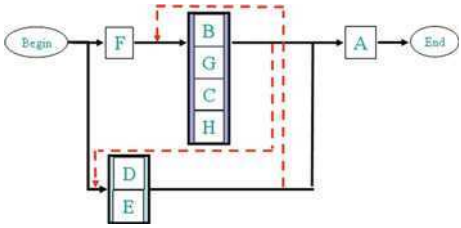


Fig. 12.25 C-process of new plan

additional links may add iterations, and is dependent on the process status, as it was demonstrated in Fig. 11.3. In a similar manner, the implementation of DB content changes is dependent on process status and the BR used.

An example of a change in the DB content (presented in Sect. 11.5) is depicted in Fig. 12.24. The probability DSM (a) and reordered DSM (b) are copies of Fig. 12.11b and Fig. 12.12, respectively. In (c), a small modification of the DSM was done: the value of the link H to E changed from 0.22 to 0.04; and a link with value 0.13 was added from H to G. The reordering result Fig. 12.24d defined modifications of the DB content, being (*DE*) and (*BCGH*) instead of (*BCG*) and (*DHE*). This change is later referred as product knowledge change.

Changes of DB content (without adding or removing activities from the process) may create interesting completion patterns of the DB depending on the process status. Assuming parallel implementation of the DBs, the C-process of the new plan is depicted in Fig. 12.25. However, it is important to note that this

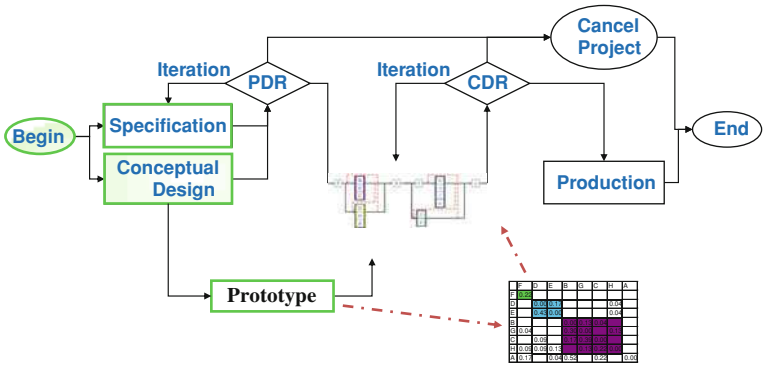


Fig. 12.26 Prototype—change in product knowledge

would have been the implementation of a new process if we were to start the process from scratch.

In this plan (*DE*) starts in parallel to *F*, and both DBs have feedback links to each other (marked as red dash lines). The structure of two parallel cyclic DBs echoes the example in Sect. 9.9. The implementation of this process plan is not a WRI-WF-net. Implementation by WRI-WF-net would be a united design block containing two parallel elements that are the two DBs (*DE*) and (*BGCH*) in a similar manner to Fig. 12.15.

12.7.1 Change of Design Blocks at Run Time

An informal illustration of a process change, due to changes of product knowledge during prototype is depicted in Fig. 12.26. The prototype started in parallel to the PDR continued in parallel to the design activity, and the information gained could alter the planning of the design activity subprocess. The subprocess started according to the first plan [e.g., DBs (*BGC*) and (*DHE*), Fig. 12.15b], and continued with a new plan [DBs (*DE*), and (*BGCH*)].

If only activity *F* has completed, when the change of plan occurred, the change could have been implemented immediately since the change is applied to a process section which is sequentially linked (not performed yet) and is not a part of a cycle (no iterations specified). Therefore, no special considerations might postpone implementing the change. Yet, the resulting process is not similar to the new plan. The DB (*DE*) cannot have started in parallel to activity *F* (which has already completed).

Defining the more complex options (demonstrated in the following examples), requires to identify the *IL* and *OL* of the DB. Changes of DB content cause the DB to change its identity; thus, a design activity might start as part of one DB, and complete in another DB. Accordingly, at run time, an identified DB may start, but not end; or might end without having an indication of starting. The importance of utilizing DB input logic and output logic is demonstrated, as they keep the DB's identity.

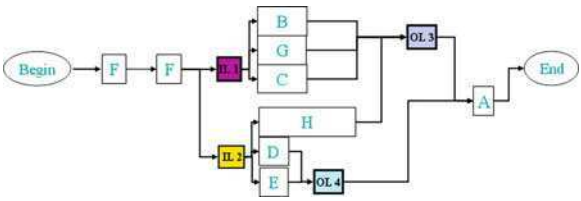


Fig. 12.27 RT-process, change at T = 3

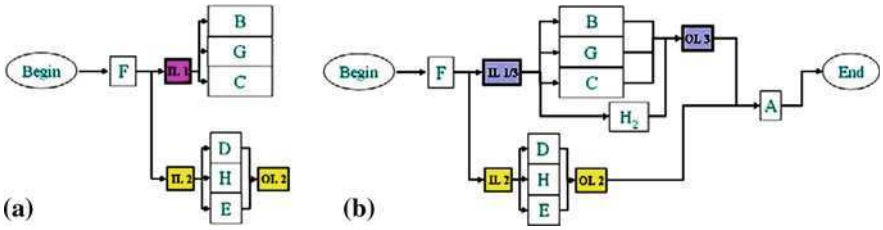


Fig. 12.28 RT-process, change at T = 3 with H iteration. **a** RT-process until T = 3. **b** Process completion

Given the activity durations (Sect. 12.5.5) including D(*H*) > 2. If the knowledge change would have been applied at process time *T* = 3, i.e., activities *D*, *E* completed and activities *B,G,C* did not; then according to the new C-process scheme, block (*DE*) could complete immediately, and activity *H* could join the other activities and complete as part of the DB (*BGCH*). An example of the resulting RT-process is depicted in Fig. 12.27. In this case, activity *F* iterated; yet, if there were only one execution of *F* the process would have been similar. Activities *B*, *G*, and *C*, started as part of *Db1* (*BGC*), and completed as part of *Db3* (*BGCH*). Activities *D*, and *E*, started as part of *Db2* (*DHE*), and completed as part of *Db4* (*DE*); H started as part of *Db2* and completed as part of *Db3*.

If D(*H*) = 2, and there were no iterations of *F,* then *Db2* (*DHE*) had completed at *T* = *3* (when the change occurred), as depicted in Fig. 12.28a. Adding activity *H* to *Db3* (*BGCH*) requires iteration of *H*, marked as H_2, in (b). The input logic of *Db1* is replaced by the input logic of *Db3* (marked as IL 1/3), since it links to the iteration of *H*, and the output logic is OL3 (of *Db3*).

The calculation of the DB duration becomes more complex. It should take into account the start time of the DB, the "internal time" Ti(*H*) in which activity *H* has joined the DB, and the duration of *H*.

The duration is D(*Db3*) = max({D(*ai*)}, {Ti(*bi*) + D(*bi*)}); where *ai* are the activities that initially started as part of the DB, Ti(*ai*) = 0; and *bi* are the activities being added during its execution. Ti(*bi*) are the internal time (time from the start of the DB) at which these activities were added. Ti(H_2) = 3−1 (time of change minus the time when the DB started); D(*Db3*) = max(D(*B*), D(*G*), D(*C*), {Ti(H_2) + D(H_2)}) = max (4, 4, 4, 2 + 2) = 4.

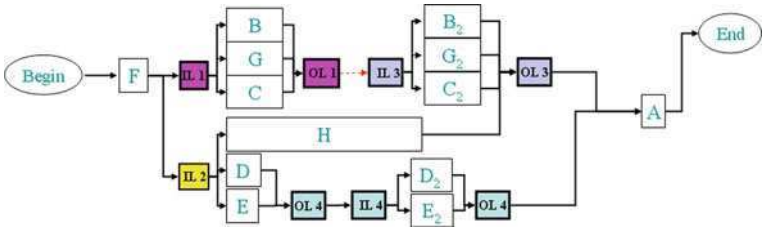


Fig. 12.29 RT-process, change at $T = 6$ with *BGC* iteration

For a change at $T = 6$, $6 < \mathrm{D}(H) < 8$ there was another process change type,[4] depicted in Fig. 12.29. In this case, DB (*BCG*) has completed; *H* was still executing; thus, iteration of activities (*B*, *C*, *G*) was required. Since (*BCG*) has completed, it had a link to A. Once *(BCG*) repeated due to the iteration enforced, the link was canceled, and replaced by a link to the new DB (dashed, red line). Activity *A* had late start logic; therefore, it could start only after the completion of the second execution of (*BGC*). In this example, an additional iteration of (*DE*) was done according to the new definition.

The time line in this figure is totally out of scale due to the logic activities (duration zero); and deviations of activity box size on the second execution (longer box due to longer title).

12.7.2 Process Reaction Types to a Change of Design Block Content

For the simple case of one change, moving an activity from "source" DB to a "target" DB, there are four types of "*process reactions*"to this DB content change:

a. Immediate implementation, if the DBs did not start (e.g., while activity *F* is iterating), or if both DBs are currently inactive.
b. Change while the activity is executing and the new "target" DB is still executing. The activity starts as part of one DB and completes in the other, Fig. 12.27.
c. Change after completion of the activity, while the new "target" DB is still executing. The activity being added to the executing DB iterates, Fig. 12.28.
d. Change while the activity is executing, and after completion of the new "target" DB. The design activities in that DB iterate, Fig. 12.29.

[4] In the current example $\mathrm{D}(H) = 7$. When using different time scale (as done for simulating learning in Sect. 5.4.8), then the inequality becomes $600 < \mathrm{D}(H) < 800$.

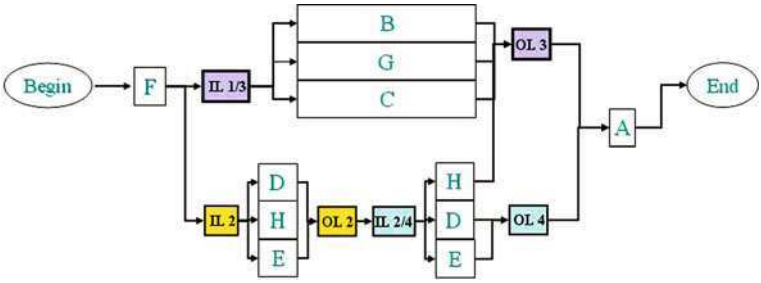


Fig. 12.30 RT-process, change at 2nd execution of H

Additional subcases are adding a new activity to a DB, removing activity from a DB to a new DB with no other content, or completely removing the activity from the process. These cases are included in the above types.

The difference of the C-process model from the RT-process model becomes significant as the RT-process may follow interim process structures that partially relate to different C-processes; consequently, the resulting RT-process does not resemble any of the C-processes (in a C-process an activity starts and completes in the same DB).

12.7.3 The Logic of Changing the Design Blocks: Internal Change

The above example demonstrated a change at some fixed process time. Such change represents an external source of knowledge change, e.g., new customer requirement. The change is considered external since its timing is not aligned with the process progress or the knowledge gained through the design. Another knowledge change type is an internal change due to the product knowledge gained during the process, e.g., assigning the change to the *second* completion of activity *H*.

The completion time of the first execution of activity *H* is dependent on its durations [D(*H*)], and the number of iterations of *F* activity (given fixed duration). The second completion of activity *H* (if *H* iterates) is influenced by the number of iterations of *F*, iteration of (*DHE*), feedback from (*BGC*), and the relative duration of *H*.

The following results used only runs with a second execution of *H*; other cases were filtered out.

Assuming a learning process with $LR = 0.5$, and given the activity duration in Sect. 12.5.5, a scaled timing was used (i.e., the minimal duration of the first execution of *H* is D(*H1*) = 100). Assuming a self-iteration of (*DHE*), the process for D(*H1*) < 266 [i.e., D(*H1*) + D(*H2*) < 400], is depicted in Fig. 12.30. The minimal total process duration was $T = 600$, and the minimal required resources were (activities in brackets):

$R = 100\{F\} + 1{,}200\{BGC\} + 400\{D1E1\} + 100\{H1\} + 200 \quad (D2E2\} + 50\{H2\} + 100\{A\} = 2{,}150$. In such cases where H completed the second time while (BGC) was executing, the process had a type "(b)" process reaction (start in one DB and complete in another). If the only iteration was of (DHE) and $D(H) > 266$, then the process reaction would have been of type "(d)" (iteration of activities in the target DB).

12.7.4 The Value of Early Knowledge

So far, the use of the system for monitoring the evolving process was presented and the simulation capabilities were demonstrated by presenting the impact of various cases on the process progress. In the following example, we present the system utilization as a decision-making aid (during the process) to managers, by utilizing the simulation capabilities of the system.

Four decision criteria are presented: averages comparison, performance indicators, pairwise comparison and finally, a decision based on a difference function using confidence interval (described in Sect. 11.7.1). The competences and limitations of the decision-making procedures are discussed in the context of the NPD process.

Different timing of gaining knowledge was demonstrated to have different process reaction types. The following simple case study is related to the decision: how many resources to invest in order to expedite gaining the product knowledge (assuming that the required resources could be estimated). The typical convention is that the sooner the knowledge is gained the better.

The example demonstrates the impact of gaining the new product knowledge at time $T = 3$ versus gaining it at time $T = 6$ for $D(H) = 5$. The BRs used were: no learning, late start, no exit option, but unlike the "long" option (Sect. 11.6) with merger and immediate implementation of DB change. Study of the impact of each BR and their combinations, and the application of DBs was presented in Sects. 11.8 and 11.9, respectively.

This example had the following process reaction type (Sect. 12.7.2) options:

If the knowledge is gained at $T = 3$ and F activity iterates and executes three times, the new process is directly implemented (type a), otherwise, type (b) is applicable.

For gaining the knowledge at $T = 6$, if activity F executes once, and there are no iteration of (DHE) and (BGC), type (a) is applicable.

For $T = 6$, if activity F executes once, and there are no iteration of (DHE) and (BGC) iterates, type (c) is applicable.

For $T = 6$, if activity F iterates, or (DHE) iterates, and (BGC) does not iterate, type (d) is applicable.

For $T = 6$, if activity F iterates, or (DHE) iterates, and (BGC) iterates, type (b) is applicable.

The simulation results are presented in Table 12.9. The average duration of the first case was shorter, requiring fewer resources in average. Using the average

Table 12.9 Gaining the knowledge early

Simulation parameter	Process duration			Resources			Performance (% runs satisfying condition)	
Change at	Min	Max	Avg., std	Min	Max	Avg., std	T < 18	R < 60
$T = 3$	7	22	12.52, 4.15	23	67	42.86, 14.89	95.8	76.9
$T = 6$	9	21	13.42, 2.74	25	67	49.22, 9.41	87.8	81.3

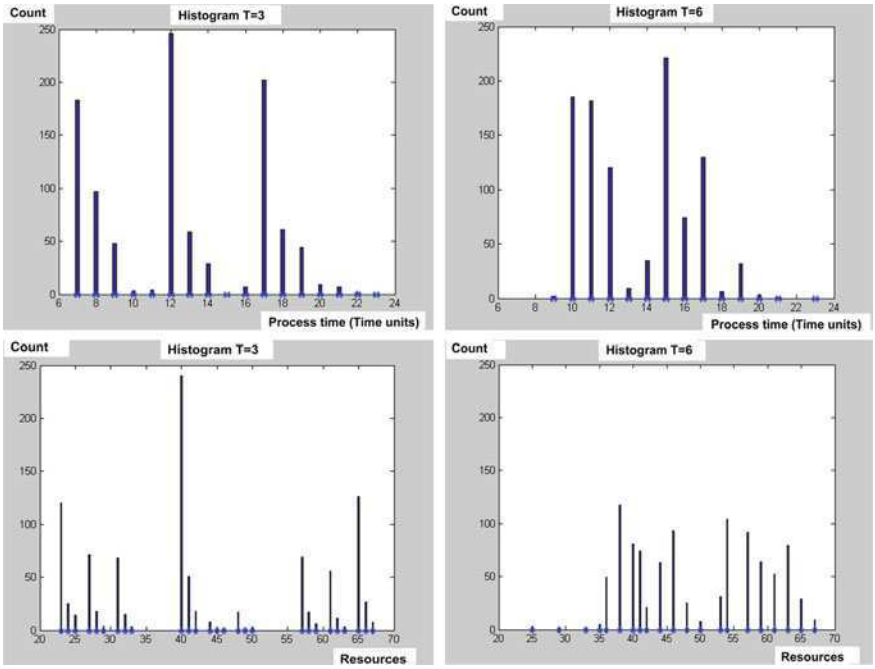


Fig. 12.31 Results Distribution D(H) = 5, no learning. **a** Process time—change at T = 3. **b** Process time—change at T = 6. **c** Process resource—change at T = 3. **d** Process resource—change at T = 6

results of duration, and the resources results it is easy to conclude that early knowledge provides better performance. Using such comparison for decision-making is the most common practice in DSM literature; however, it has no statistical significance.

The performance indicators (T < 18, R < 60) were less decisive: better time performance for the first case; but, better resource performance (probability to complete the design with limited resources) on the latter case.

In order to understand the performance indicators results, a more detailed investigation was required, and histograms of the process were drawn, Fig. 12.31. The resulting distribution looks periodic, but was not consistent. Analysis of the results indicated that the choice of the threshold values (for duration and resources)

Table 12.10 Gaining the knowledge early

Pairwise comparison		Difference function—time decision: T($T = 3$) < T($T = 6$)		Difference function—resources decision: R($T = 3$) < R($T = 6$)	
T($T = 3$) < T($T = 6$)	R($T = 3$) < R($T = 6$)	Avg., std	Confidence Interval (5%)	Avg., std	Confidence Interval (5%)
50.7%	60.8%	-0.906, 4.894	0.3033	−6.36, −17.28	1.071

had a major impact on the results comparison; therefore, such decision-making method might be too sensitive to the choice of the threshold.

For example, choosing a time threshold of T < 16.5 would yield for ($T = 3$) 67.6% and for ($T = 6$) 82.8%, i.e., for this threshold the latter case has better duration performance. On the other hand for R < 57 and ($T = 3$) we got 67.7%, and for ($T = 6$) 67.5%, i.e., the first case has better resource performance.

A pairwise comparison is depicted in Table 12.10. Each couple of runs is compared counting the number of cases where (Time/Resources) of the first case type (getting the knowledge at $T = 3$ and making the change decision at that time) is better than the changing at $T = 6$. The results indicated that the total process time was better in the first case (50.7%), and fewer resources were required (60.8% of the runs) (for 1,000 runs).

The results of using the statistical analysis of the difference function distribution (described in Sect. 11.7.1) are also depicted in Table 12.10. The run was done using a scale of 100 time steps per time unit, and scaled back (i.e., the simulation progresses with time steps of 0.01 time units).

Results of the average, standard deviation of the difference-function distribution, the confidence interval, and the decision are presented. The average time difference is −0.906 (i.e., the average time of the first case is shorter), and is out of the confidence interval (which is [−0.3033 to 0.3033]). Therefore, the difference is statistically significant, and we can reject the null hypothesis that both cases are equivalent.

The average resources difference is −6.36, and is out of the confidence interval, which is [−1.071, 1.071]. Again, the results are statistically significant.

The results indicate that with a confidence level of 5% we can decide that the first case has better performance than the second case. Decision based on early knowledge has better process performance.

Using the statistical analysis we can be more convinced that the first option is better, i.e., making changes earlier in this case is more beneficial. Furthermore, a parametric search helps estimating that the addition of up to 0.60 time units, or up to about 5 resource units to the first process, would still make that process more beneficial.

12.7.5 Change of Knowledge Without Change of Process Scheme

The next example demonstrates a ‘what if’ scenario. The system is utilized for checking what would be the impact of a change in knowledge, if it were not

Table 12.11 Decision-making—changing the process or ignoring new knowledge

Simulation parameter	Process Duration			Resources			Performance	
	Min	Max	Avg., std	Min	Max	Avg., std	T < 2,000	R < 4,800
Scheme Change	16.00	33.27	17.57, 1.57	35.00	52.29	43.29, 3.46	89.2%	92.1%
No change of scheme	16.00	37.41	18.78, 2.29	35.00	53.93	44.87, 3.87	71.7%	83.8%

Table 12.12 Decision-making—pairwise and statistical analysis

Pairwise comparison		Difference function—time			Difference function—resources		
T(change) < T(No)	R(change) < R(No)	Avg., std	Confidence interval (5%)	Decision	Avg., std	Confidence interval	Decision
59.7%	58.3%	-1.32; 2.77	0.172	T(Change) < T(No)	-1.60; 5.30	0.329	R(Change) < R(No)

followed by a change of the process scheme. The probabilities were calculated according to the updated knowledge, Fig. 12.24d, but the process continued according to the "old" plan, Fig. 12.20. In this case, the time and resources were scaled such that each simulation step represents 0.01 time/resource units. In Table 12.11, results for the "short" case[5] of DBs with D(H) = 1,000 are presented; 1,000 simulations were performed.

There is a difference in the average of duration and resource utilization between the two cases. Both duration and resource utilization are better in the case of implementing the knowledge changes, when the information is available. In this example, both process indicators are better in the case of implementing the scheme change.

Comparison of the pairwise results, in Table 12.12, yields that shorter duration of the process with scheme changes were obtained in 59.7% of the cases, and fewer resources were required in 58.3%. Statistical analysis of the difference-function distribution results are aligned with the comparison results. Both difference averages are out of the confidence interval, thus the difference is statistically significant.

The results indicate that updating the process plan according to updates in the product knowledge can improve process performance.

Changing the process scheme has cost (the time and resources required for forming new groups). If there are estimations of such additional costs, a decision can be made (or implemented into the simulation) whether to implement such change or not.

[5] Maximal parallelism, minimal impact of multiple iterations, and minimal process time, see sect. 11.6

According to parametric analysis of the difference function, we can add about 1.15 time units and 1.27 resource units and still have better time and resource performance. Therefore, any additional duration and resources for performing the process change that are less than the above justifies performing the change.

12.8 Example Outline

To recap, the detailed example demonstrated the dynamic evolution of a development process. Starting with the most general process having one process activity (within a frame of logic activities Begin and End), the process was hierarchically expanded to a best practice high-level process being the updated C-process. RT-process model, which keeps the process history and the current process status, followed the C-process.

Once initial knowledge regarding the product was gained, the stages of converting that knowledge into a process plan and then to a process scheme model were followed. These stages were applied to planning the conceptual design activity (one activity in the high-level skeleton). After reordering the DSM, various business rules (the options of serialized, design block, and general parallel options) were applied for the transformation from the DSM structure to the process scheme. Using the simulation capabilities for decision-making, the results of applying the BRs were compared.

Using *ad-hoc* changes, the C-process can be modified to (unexpected) local needs. In the presented example, it was used for adding a prototype stage that enhances the product knowledge (detailing the product structure and evaluating the links more carefully, i.e., giving values instead of binary estimation); hence, modifying the design process scheme.

The additional product data was used for the creation of probability DSM. Its reordering and its conversion to Design blocks were demonstrated, followed by conversion stages to the process model. The implementation of the various process structures at run time were presented, while checking the implications of various process parameters and different business rules. Again, simulation was used for aiding decision-making by comparing the potential implementations. A comparison of potential decision schemes was presented that manifested the properties of using statistical analysis method.

Finally, dynamic changes of the scheme were presented, discussing the various change types and using the simulation for addressing process management issues such as quantifying the value of early knowledge; or quantifying the cost of performing a process change versus not changing the process.

References

Browning TR, Eppinger SD (2002) Modeling impacts of process architecture on cost and schedule risk in product development. IEEE Trans Eng Manag 49(4):428–442

Cho SH, Eppinger SD (2005) A simulation-based process model for managing complex design projects. IEEE Trans Eng Manag 52(3):316–328

Huberman BA, Wilkinson DM (2005) Performance variability and project dynamics. Comput Math Organiz Theory 11:307–332

Karniel A, Reich Y (2007a) Simulating design processes with self-iteration activities based on DSM planning. IEEE Proceedings international Conference Systems Engineering Model—ICSEM'07, March, 33–41, Haifa

Karniel A, Reich Y (2007b) Managing dynamic new product development processes. In: Proceedings of the 17th annual international symposium of the international council systems engineering INCOSE'07, June, San Diego, California

Karniel A, Reich Y (2009) From DSM based planning to design process simulation: A review of process scheme logic verification issues. IEEE Trans Eng Manag 56(4):636–649

Karniel A, Reich Y (2011) Formalizing the implementation of DSM-based process planning for NPD. IEEE Tran Sys Man Cybern Part A 41(3):476–491

Sadiq W, Orlowska ME (2000) Analyzing process models using graph reduction techniques. Inf Sys 25(2):117–134

Sered Y, Reich Y (2006) Standardization and modularization driven by minimizing overall process effort. Comput Aided Des 38(5):405–416

Smith RP, Eppinger SD (1997a) Identifying controlling features of engineering design iteration. Manag Sci 43(3):276–293

Smith RP, Eppinger SD (1997b) A predictive model of sequential iteration in engineering design. Manag Sci 43(8):1104–1120

Smith RP, Morrow JA (1999) Product development process modeling. Des Stud 20(3):237–261

van der Aalst WMP (2000) Finding control-flow errors using Petri net based techniques. In: Aalst Wvd , Desel J, Oberweis A (eds) Bus Process Management, Lecture Notes in Computer Science, vol 1806. Springer, Berlin, pp 161–183

Whitley D, (1994) A genetic algorithm tutorial. Stat Comput 4(2):65–85

Williams T (2003) The contribution of mathematical modelling to the practice of project management. IMA J Manag Math 14:3–30

Yassine A (2007) Investigating product development process reliability and robustness using simulation. J Eng Des 18(6):545–561

Yassine A, Joglekar N, Braha D, Eppinger SD, Whitney D (2003) Information hiding in product development: The design churn effect. Res Eng Des 14(3):131–144

Chapter 13
Summary

13.1 Overview

To date, the automated monitoring of static fixed-scheme processes is managed by workflow tools; yet, a corresponding engine for management and simulation of iterative dynamic and complex processes such as NPD is lacking. The presented research therefore responds to an existing business need. Studying and evaluating the implications of dynamic changes in a process plan should be academically interesting; however, such studies are scarce. The framework proposed in this research provides methods and tools for further studying dynamic process-related aspects that were not fully addressed. These issues are discussed in the following sections.

The main contribution of this research was the composition of the DnPDP framework model that enabled supporting the requirements of NPD-like processes, including process management (planning, converting to model, model implementation, and execution), and process decisions aid tools (through simulations and statistical analysis). The framework incorporates product knowledge evolutions, and is used for process planning updates that are repeatedly converted to process model and implemented in a changing process scheme. The process is proceeding through the hierarchically evolving scheme from an initial pre-defined best-practice process model to the actual iterative "on the fly" defined process. The evolving changes required formal specification to ensure the properties of the resulting process. Such formulation using WF-nets is established for static scheme processes. The concepts adapted from WF-nets are utilized for the dynamic case by requiring that each of the process schemes implemented incorporates the current process state (i.e., keeping continuity) and that state in the current (changing) process scheme is able to reach the terminal state of the updated process scheme. A process engine was developed that implements the execution of dynamically evolving process scheme (during the process). The same engine is used for simulation of potential process evolution during the process.

A. Karniel and Y. Reich, *Managing the Dynamics of New Product Development Processes*, DOI: 10.1007/978-0-85729-570-5_13,

These simulations are analyzed and the analysis results can support decision-making during the process progress.

A test case of developing a simplified product demonstrated the capabilities of the framework to implement and analyze a dynamically evolving development process. Even for the simplified product model presented, the process is too complex to be evaluated without simulation. The decision-making is highly dependent on process parameters in a nonlinear complex manner.

The implementation of the framework in a real setting and its validation by hands-on experience are still to be accomplished. The main challenge is collecting the "on line" information that is required for the repeated DSM calculations. Existing PLM tools can provide support for data collection during the design process, therefore integration to such tools is required. Technically, such integration is feasible; but commercial and organizational obstacles are to be resolved in order to achieve it.

The research contributions include enhancements and extensions of existing concepts and models, and the development of new concepts. Existing methods were typically developed under assumptions that do not comply with NPD process requirements (e.g., having static process or assuming a validated manual plan). The main enhancements are summarized in Sect. 13.2 that follows.

13.2 DnPDP Framework

Solution constituents were identified and integrated into a framework—the DnPDP. Each of the identified resolution components may have different implementation options (methods); thus, the framework implementation presented in this study is only one of many potential implementation solutions of that framework. The choice of the specific method used was instrumental in seeking for a sufficient solution; but was not optimized (i.e., looking for an effective solution and not necessarily an efficient one). Furthermore, choosing between the various options of each component and choosing combinations would require hands-on experience with implemented solutions (that are not available currently).

A summary of the existing methods integrated into the framework, their current state of the art, and the modifications done in the current work is depicted in Table 13.1 (Karniel 2009).

13.2.1 Contribution to DSM Concepts

Several extensions and contributions to the DSM methods were proposed in the research. The novel self-iteration concept was defined and demonstrated. The self-iteration concept provides the foundation for the Design block concept, which is augmented with a proposal for calculating the self-iteration merged probability.

Table 13.1 Framework components (methods)

Component	Existing method used	Other potential methods	Main enhancements and modifications
Process planning	DSM		Temporal design blocks
			Self-iterations
			DSM reordering algorithm for the sub cycle case
			Optimization algorithm
Automated conversion to process model (process generator)		Manual	Automated conversion process from DSM-based plan to an executable process model
Process model of dynamic scheme process	Task net	Pi-calculus	DSM net with Logic activities having modifiable logic assigned at run time
Process verification	Petri nets	Pi-calculus	Correctness criteria of dynamically changing scheme
			Proofs for DSM net
Process execution (process engine)	Enhanced Task net	GERT	Execution of processes with dynamic process scheme
		Petri nets	Distinction between C-process and RT-process
		Pi-calculus	
Simulation	Enhanced task net	DSM-based simulations (+executables)	Using the same executable process (process engine) to simulate business rules
Decision-making	Simulation statistics	Statistics	

Methodological contributions of this study include (a) the assertion indicating that definition of both input and output process logic cannot be done by marking of a single DSM in the case of parallel execution of activities. (b) The assertion indicating that minimum feedback optimization methods are applicable for Boolean DSM, but may not suit DSM with variable figures (numeric or probability DSM) to determine optimal process results. The latter assertion (being previously demonstrated in the literature, but with no explanation of its rationale) emphasizes the need for simulation as a guiding tool for process evaluation. These assertions should guide further DSM-related research.

13.2.2 Partitioning and Optimized Sequencing Algorithms

The existing partitioning algorithm completes once the DSM activities are reordered to activity cycles. An enhancement was developed that continues the reordering such that the activities within a cycle are further reordered to a simplified cycle with one feedback link.

An optimization algorithm was developed for sequencing (performing an equivalent of partitioning, tearing, and clustering) and reordering the activities to sub cycles that are defined as DBs.

13.2.3 DSM Conversion to Process

A new analysis method was proposed for analyzing the various logic options used in the literature for converting the DSM structure to the process logic. The analysis provides a common ground for comparing logic options. Yet, the method is limited and cannot cope with all the logic options; specifically, it cannot compare process-status-based logic that is defined during run time.

The conversion correctness was identified as a key element for automating the conversion process, where conversion automation is required for repeatedly updating the dynamic changes of the DSM data. The gap between the DSM literature and other process implementation methods such as Petri nets was indicated. New correctness criteria were defined, along with definition of the special type of DSM nets.

Using Petri nets results (specifically WRI-WF-nets), it was proved that DSM nets are equivalent to WRI-WF-nets for some specified cases. DSM nets can be used in a hierarchical and extending manner, thus providing the foundation for dynamic changes.

Defining dynamic changes required the definition of business rules (defining applicable logic implementation) that adds an abstraction layer to the DSM data. New logic operators were defined enabling the implementation of dynamic changes to the process logic (and simplifying the DSM net structure under changes).

13.3 Future Development

Several issues discussed within the DnPDP framework require further development. These issues are grouped by the main related topics.

13.3.1 DSM-Related Issues

The DSM data is assumed available (e.g., collected by PDM/PLM tools), yet setting the actual figures on which the further calculations are established is not sufficiently addressed by DSM-related literature, and should be further examined.

1. DSM reordering according to the optimization cost function in Eq. 4.1 enforces feedback values to be less or equal to forward values in a simple activity loop (note: this may not be the case in a loop with sub loops). This assertion indicated that the accurate value of the link (i.e., the assigned influence) has no importance on the activities reordering algorithm; only its relations to other values is important.

   Top-down approach using only three values could be enough for reordering purposes. Yet, bottom-up approach that has more granularity and larger variety of values may have benefit once tearing (to DBs) is considered.

2. Probabilities calculations (forward, feedback, and self-iteration) using Eq. 4.3 for merged activities in a DB are based on calculating probability occurrence of independent events. This calculation approach is appealing since it allows creating DB probability estimations bottom-up; and is computationally appealing since it keeps commutativity and associativity. However, it needs to be validated by measurements. Another option that could be applied is using only forward links for the computation of forward link merger, and calculating only feedback and self-iteration links probabilities for merged links. Yet, commutativity and associativity are not kept for the merged forward link being calculated.
3. It is assumed that the accuracy of probability values may not be important for using the proposed framework as a decision-making aid (by comparison of options) and that the relative probability values are good enough. Yet, this assumption requires further testing and validation.
4. The algorithm for reordering sub cycles (Lemma 2) might be useful in reducing the search space of the optimal reordering algorithm. The integration of both should be further studied. Limiting the search space is appealing from a computational aspect; however, the sub cycle reordering is typically addressing minimum iterations, which might not be the optimization target.

13.3.2 Converting DSM-Based Plan to a Process Model

1. The formal definitions in Sect. 9.4 provide the foundation for proving correctness of various process types, which are represented by WRI-WF-nets. The processes include serial processes, fully parallel processes, block diagonal processes, and parallel blocks (cycles with sub cycles), where activity cycles (or sub cycles) are defined as DBs, and are implemented as serialized or parallel activities within the DB. Further work is required for defining the properties of process structures, which result from applying the business rules. Process examples were presented by demonstrating processes that are structurally correct, but cannot be extracted from WRI-WF-nets.

   It is required to prove that such processes can be built with inherent correctness using DSM net (or some other logic representation). Such process schemes were defined without using the Siphon concept (van der Aalst and van Hee 2002).

2. Research is required for automation of process planning where relations between different domains are used (instead of the simplified one-to-one relation between product and process); e.g., using the Domain Mapping Matrix (DMM) (Danilovic and Browning 2007), or the extended MDM (Maurer 2007). Such automation could be used for implementing the applicable changes during the process.

13.3.3 Process Scheme Verification

The structural conversion of DSM to a DSM net was verified for specific process cases (serial, fully parallel, block diagonal DBs, and parallel (subcycles) DBs); yet not all the potential logic interpretations that could be applied according to the BRs and their combinations (described in Chap. 10) were formally proved. Few example cases demonstrated the existence of additional subprocess structures that are applicable for implementing logic options, which are not WRI-WF-nets.

Checking all the business rule options for all cases seem to be a combinatorial task; unless there is some way to prove building block cases and prove that their combinations apply as well. Such work is left for future research.

13.3.4 Estimating Simulation Parameters

On top of estimating the DSM data there are other process variables that should be set and be estimated for simulation purposes.

1. The duration and cost parameters are typically estimated for any project planning; their estimation is therefore well established.

2. The conversion to probabilities figure (being previously discussed) or direct estimation of such figures should be accurate if the simulation results of duration (resources) are to be used for actual scheduling (not only for choosing between options). More research is required to validate that duration results based on probability estimations, yielding from the calculations, are giving meaningful values. The article by Yassine (2007) addressed the estimation of probability figures, but validation of simulation results was not reported in the DSM literature.
3. The measurement and estimates of the Learning Ratio (LR) for design activities might be somewhat tricky since the content of the activity may differ in each iteration. Thus, a naïve direct measure of two (or even multiple) occurrences of the iterated activity may not provide a good estimation. There is a need for a model that estimates the content of the activity in each iteration, then used for normalizing the activity duration for estimating the actual LR. Browning and Eppinger (2002) used a model that included learning curve, rework, and rework impact for simulation. However, it was used with assumed values, and not for estimating the values, given actual measures.

13.3.5 Resources Scheduling

Resources issues such as scheduling subject to resource constraints were not addressed in the current research. This issue is intensely discussed for static process scheme (and typically for acyclic processes), but is rarely discussed in the context of dynamic processes such as NPD.

13.3.6 Models Comparison

In Chap. 12, we demonstrated the utilization of the DnPDP framework for comparing business rules, while implementing a DSM-based process plan. This line of research can be further expanded to check additional cases. Moreover, the framework can be utilized for comparing different methods of planning (e.g., using different objective functions for DSM reordering) by replacing the *product-based process scheme generator* in Fig. 8.1. To date there is no common ground for performing such comparison.

13.3.7 Decision-Making

This research presented a statistical hypothesis method for aiding decision-makers, giving results for decisions between two optional business rules. This can be

extended to multiple options statistical analysis. Yet, other decision algorithms might emerge and be tested using concepts of control theory. The system in Fig. 8.2 is presented as a closed-loop process with the decision-making block as a controller. Such research might be interesting for the control community, as the system represented has a varying structure with varying parameters.

13.3.8 Practical Implementation

As previously indicated the main obstacle in implementing the proposed framework in practice is the collection and estimation of the required data. Such collection is a burden over process resources, and probably will not be done in an industrial setting until the method benefits are proved in some combined academic-industry settings.

Academic research can utilize the presented approach and the simulation tools for studying the development process in a monitored context. The preferred setting might be the development of the same new concept or new product done (in parallel) by several groups of students in some project-based learning setting (Reich et al. 2006). In such setting, if the number of groups is sufficiently large, the actual values could be evaluated statistically (e.g., actual learning curve ratio, and actual iteration probabilities in comparison to estimated values). Additionally, the various process conditions could be checked and compared. Comparison is typically not applicable in an industrial case study as any NPD is unique.

13.3.9 Commercial Implementation in PLM

Over the years, the complexity of PLM tools increased, when integrating modules supporting different aspects of the life cycle of products systems. New tools or their new versions are released continuously with new features to account for new emerging business and engineering needs and new technologies. In spite of these integrations or extensions efforts, the fundamental issues presented and discussed require fundamental rethinking of PLM principles.

Product data management of PLM systems could be improved by incorporating capabilities to model dependencies between information items and a capability to use them to drive development process planning (e.g., by using DSM-based planning methods). The integration between product and process management is based on the assumption that existing knowledge is available in a timely manner. Enhancements to PLM tools are expected to support the collection and management of the required information. Workflow engines could be enhanced to cope with evolving product information by incorporating capabilities to implement process changes during run time (e.g., by using DSM nets). On top of such workflow capabilities, a service for simulating plausible product-related

information changes could be developed as presented in this book. Exploiting the full benefit of the new PLM paradigm requires implementing all these capabilities.

13.4 Epilogue

The integration of a product-based planning method (the DSM), and process modeling concepts bridges a gap between two PLM roles: the product management and process management roles. The implementation of WF-net concepts through DSM net for the interpretation of DSM-based plan is beneficial in the context of simulating product design processes; but is essential in the NPD case of dynamically changing process schemes.

The dynamic process scheme requirement being enforced as part of the DnPDP framework enables implementation of hierarchically evolving process, starting from a high-level predefined best-practice design process and further refining it as the product knowledge evolves. This reflects the way that development processes evolve manually in actual practice.

One of the main results of the current study was the conclusion that no business rule is better than another is; different rules are better suited for different cases. In iterative and continuously changing process, general rules of thumb can no longer support decision-making, and therefore simulation-based tools are required for supporting the decision-makers. Simulation of various parameters revealed that different process setting yielded different preferences (in regard to choosing most applicable business rules). Consequently, the necessity for simulating the specific case context was emphasized. This has to be translated into PLM service to be available to engineers. By implementing the connection between product data and process planning but without such simulation service, process planning would be overwhelmed with consistent complex information without a way to use it effectively to drive correct decisions.

We argue that the improvement in the foundation of PLM systems in the way described in this book is critical for supporting NPD projects; thus, making sure that the major investment in PLM systems is fully exploited to drive product innovation.

References

Browning TR, Eppinger SD (2002) Modeling impacts of process architecture on cost and schedule risk in product development. IEEE Trans Eng Manag 49(4):428–442

Danilovic M, Browning TR (2007) Managing complex product development projects with design structure matrices and domain mapping matrices. Int J Project Manag 25(3):300–314

Karniel A (2009) Managing the dynamics of product development processes for new product development. Dissertation, Tel Aviv University, Israel

Maurer M (2007) Structural awareness in complex product design. Dissertation, Technischen Universität München, Germany

Reich Y, Kolberg E, Levin I (2006) Designing contexts for learning design. Int J Eng Edu 22(3):489–495

van der Aalst WMP, van Hee KM (2002) Workflow management: models, methods, and systems. MIT Press, Cambridge

Yassine A (2007) Investigating product development process reliability and robustness using simulation. J Eng Design 18(6): 545–561

Chapter 14
Annexes

14.1 Annex A: pi-calculus Comparison to Petri Net

The pi-calculus was reviewed in the current research as a formal foundation for modeling the Dynamic new-Product Design Process (DnPDP). While the dynamic modeling properties of the pi-calculus are appealing, its complex semantics interpretation was estimated as a high overhead, and the simpler Task net was utilized for process modeling.

The pi-calculus has two main entities: *names* and *agents*, where agents are (sub) processes defined by process expressions, and names indicate links (communication channels), variables, prefixes, or restrictions. Restrictions localize the scope of a name, while input-prefix is a name-binding operation utilized as a placeholder for a name that might be received as input.

The syntax is summarized in Table 14.1 (adapted from Milner 1999).

Structural congruence of two processes $P \equiv Q$ implies that they are identical up to structure, i.e., one can be obtained from the other by replacing names. The structural congruence allows identifying all the agents that represent the same system. The definition of structural congruence enables us to obtain finite state representation for classes of agents.

Transition, or reduction relation, $P \to P'$, indicates that a process after performing a transition (activity) has changed from state P to state P'.

It was proved that if $P \equiv P'$ and $P' \to Q'$, where $Q' \equiv Q$, then also $P \to Q$. This rule states that processes that are structurally congruent have the same reductions.

In a Labeled Transition Systems (LTS) of sequential processes, defined by the summation $P = \sum_{i \in I} \alpha_i \cdot P_i$, for each $j \in$ I (I finite indexing set), we get $P \xrightarrow{\alpha j} Pj$.

The latter definition will be used for describing a scheduler and demonstrating pi-calculus definition versus Petri net definition. The Petri net example was developed in the current research using a java-based Petri net simulator (Esser 1998).

A. Karniel and Y. Reich, *Managing the Dynamics of New Product Development Processes*, DOI: 10.1007/978-0-85729-570-5_14,

Table 14.1 pi-calculus syntax

Operation	Syntax	Meaning
Concurrency	$P1 \mid P2$	$P1$ and $P2$ are acting in parallel
Sequence prefix	$a \cdot P$	Activity a is followed by process state P
Communication input-prefix	$x(y) \cdot P$	The process is waiting for message through channel x and binds the name received to y
Communication output-prefix	$\bar{x}y \cdot P$	Output the name y through output channel x and behave as P. The output-prefix is synchronized with the input-prefix of the same channel
Silent prefix	$\tau \cdot P$	Performing silent action (i.e., synchronization), then process P
The nil process	0	Deadlock, the process stops
New name	$(\nu x)P$	Assignment of fresh names to distinguish between free names
Match	$[x = y]\ P$	P is enabled if names x and y coincide
Summation	$P1 + P2$	The process behaves like one or the other
Defining equation of agent	$A(x_1,\ldots,x_n)$	Parametric equation of the free names of the agent

The scheduler problem was defined (Milner 1999) as setting agents to perform certain task repeatedly. Each agent Pi wishes to perform repeatedly, and the scheduler is required to initiate the agents in cyclic order and ensure that a task can be repeated only after finishing its cycle. The labeled transition system of the scheduler is given by

$$\text{Scheduler} \stackrel{\text{def}}{=} \text{Sched}_{1,\phi}$$

$$\text{Sched}_{i,X} \stackrel{\text{def}}{=} \begin{cases} \sum_{j \in X} b_j \cdot \text{Sched}_{i,X-j} & i \in X \\ \sum_{j \in X} b_j \cdot \text{Sched}_{i,X-j} + a_i \cdot \text{Sched}_{i+1,X \cup j} & i \notin X \end{cases}$$

where $X = \{\text{started agents}\}$; $X \cup \{i\} = X \cup i$, $X - \{i\} = X - i$; $i + 1$ is modulo $\mathrm{N}\,[i+1]_N$.

The transitions (activities) are a_i (start the agent task) and b_i (complete the task).

A Petri net solution (developed for three agents) is depicted in Fig. 14.1. In each full cycle, any agent performs once. Each agent has two transitions: a_i—start the action; b_i—complete the action. All agents may perform their task in parallel, but agent $i + 1$ (cyclic) can start only after agent i has started and after he (agent $i + 1$) has finished its former task.

The process state described in 125 implies that agent 1 is performing his task; agent 2 has completed its task; and agent 3 is performing its task. Once agent 1 will complete its task (transition b1) it will be able to start again (transition a1). Immediately afterwards, agent 2 (that has already completed) will start again.

Understanding the process is quite intuitive when using the Petri net representation (and even easier with the help of the simulation). The process states map is depicted in Figure 14.2. Process states are explicitly defined by the process

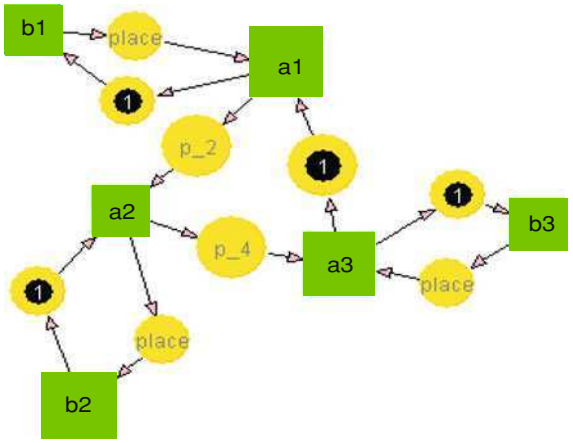


Fig. 14.1 Petri net of a scheduler with three agents

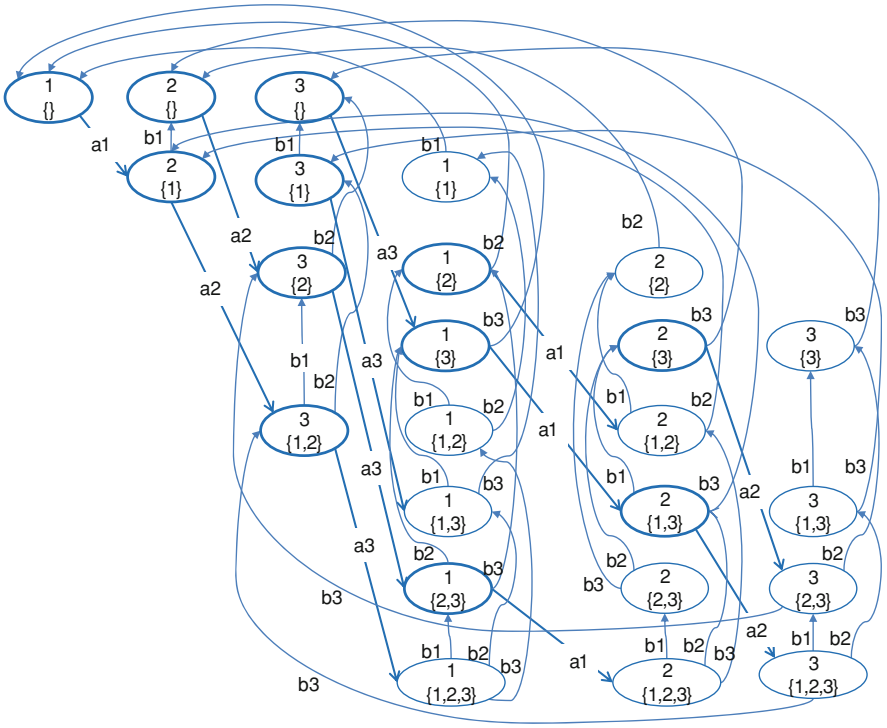


Fig. 14.2 Scheduler process states

equations of pi-calculus and are implicitly represented by the Petri net tokens. The process equations are very compact and efficient (same set is applicable for every number of agents), yet understanding the process is quite complicated.

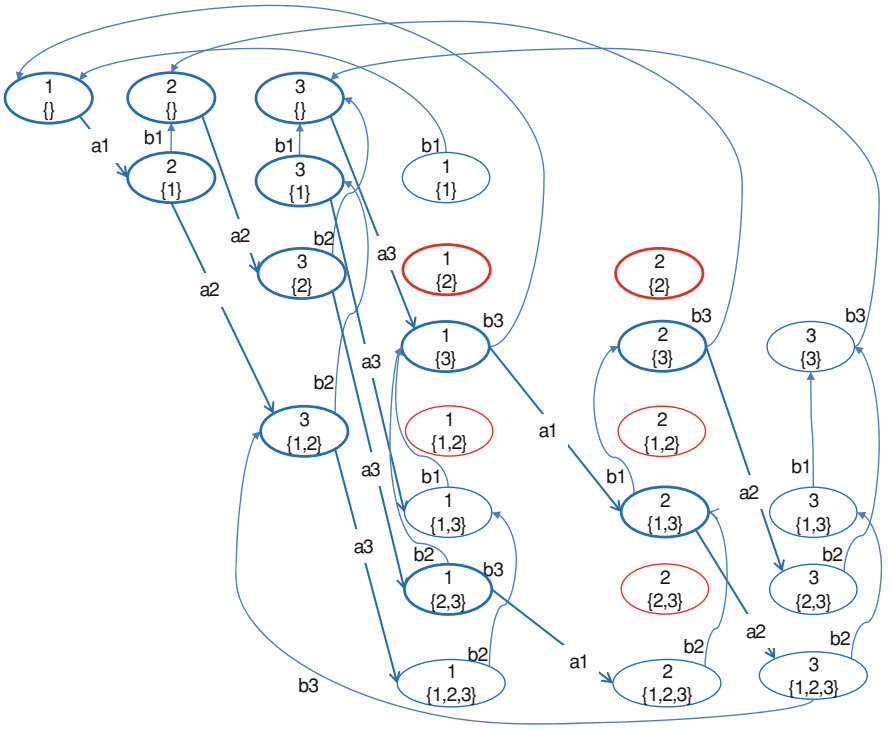


Fig. 14.3 Scheduler—unreachable process states

The states indicate the agent that can start now according to the upper cycle and the list of currently active agents. For example, on the upper left side, agent 1 is waiting to start, and no agents are active (empty list {}); once transition a1 executes then agent 2 can start and agent 1 is active. If b1 happens next (before a2), then activity 2 is waiting to start and no agent is active.

In both representations, there might be states that are unreachable due to performance (e.g., relative duration), and the system will never get there. It should be further investigated if the explicit mapping of the process might reveal such cases analytically in the case of pi-calculus. An example of such case is a case where agent 1 is slower (or has a larger job) than agent 2 (i.e., b2 is always executed before b1); and both are executing before b3. In such case, not all the process states are applicable.

The resulting process states map is presented in Fig. 14.3. States that can never be reached have no links (marked in red).

Another example relates to changing the number of agents during the process. Such change does not require changes in the pi-calculus model, but does require changes of the Petri net. Assuming that at certain point in time, agent 3 is no longer required and is removed from the cycle; nevertheless, if the decision to remove it is made while the agent is executing, it may still finish the task (transition b3).

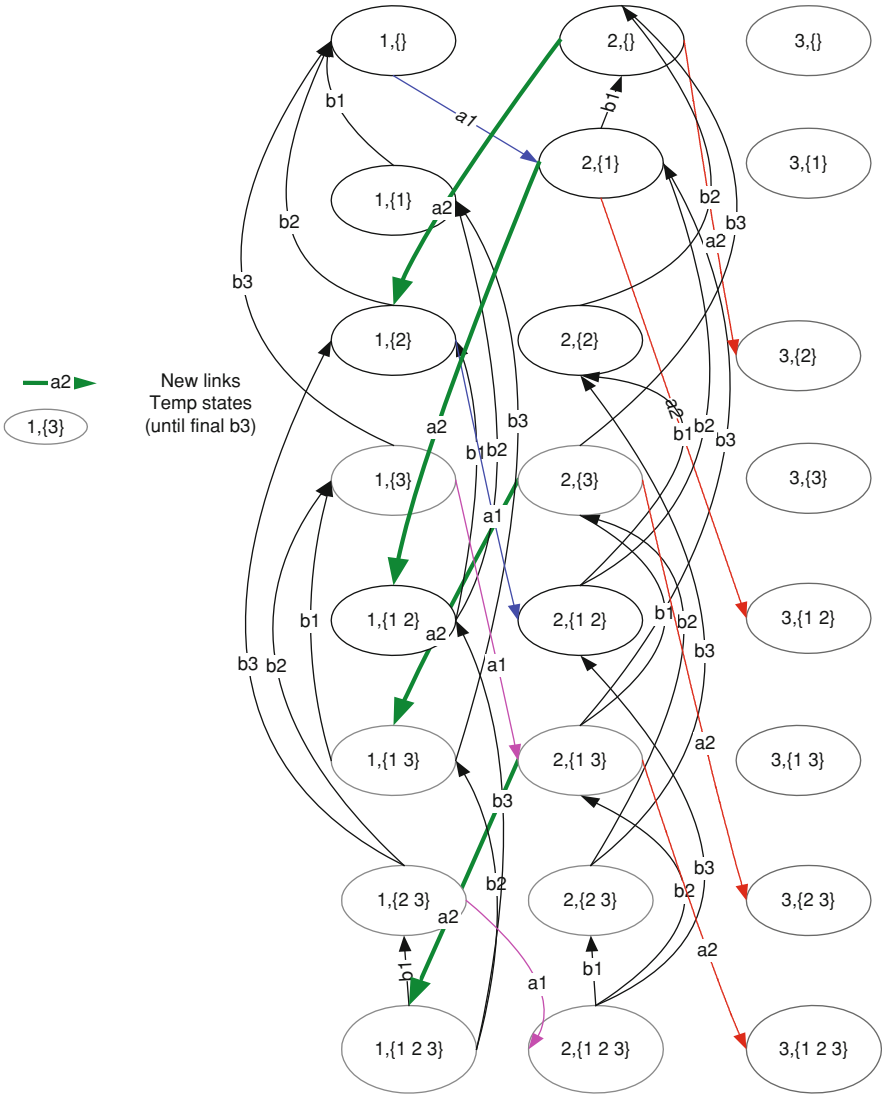


Fig. 14.4 Scheduler—decreasing process states

The modulo function of the pi-calculus equation inherently replaces the link from $A2$ to $A3$ by a link from $A2$ to $A1$; and the state map is reduced as described in Fig. 14.4. A Petri net in such case is required to perform a complex change while checking if the agent has completed or not.

Once the decision is made the links state where agent 3 should start to become unreachable (indicated as red, with dashed a2 input links). The new a2 links

between states are marked by thick green lines. Yet, while b3 did not execute (indicating completion of agent 3 task), there are temporal stages with agent 3 still in the list (applicable states are marked in purple).

14.2 Annex B: Statistics of Stochastic Simulated Process

The following section describes the statistical formulation in (Asmussen and Glynn 2007), for analyzing stochastic simulation results.

Distribution Definitions: Probability cumulative distribution P of a probability density function $f(x)$ is defined by $P(x<X) = \int_{-\infty}^{X} f(x)\,dx$.

In the discrete case, we define the discrete distribution P of a probability mass function $p(x)$ by $P(x<X) = \sum_{x_i<X} p(x_i)$.

Process Definitions:

1. A Markov Chain is a process $\{X_n\}$ with finite or countable state space, where the next state depends only on the previous state. Examples: a process defined by the transition probabilities $pij = P(X_{n+1} = j \,|\, X_n = i)$, i.e., pij is the probability to move from state i to state j; as the serial process in (Sered and Reich 2006), using probability DSM. Autoregressive process $X_{n+1} = aX_n + \varepsilon_n$, with ε_n being an independent identically distributed variable, is another example. The Markov chain is time independent (Gilks et al. 1996).
2. A Markov process is time dependent, with finite or countable state space. For example, the time of making the transition is exponential with an intensity matrix $\Lambda = \lambda ij$ and the process holding time at state i is exponential with rate $T = \exp(-\lambda ij)$, and the next state j is chosen with probability $\lambda ij/\lambda ii$.

The main property of simulation process is the ability to generate large number of examples (i.e., as large as required given processing availability). This ability is used according to the Law of Large Numbers (LLN).

For a stochastic random variable W_n and the function $f(W_n)$; if for $n \to \infty$ the steady state $W_\infty < \infty$ (i.e., W_∞ is a random variable with a limited distribution), then we can calculate the expected value of the function for $N \to \infty$, using Eq. B.1. We get an asymptotic convergence to the expected value $E[f(W_\infty)]$ for the random variable W_∞.

$$\frac{1}{N}\sum_{n=0}^{N-1} f(W_n) \to E[f(W_\infty)],\ N \to \infty \tag{B.1}$$

The expected value $z = E[Z]$, where z is not available analytically, but Z can be simulated. Using the Monte–Carlo method, we simulate R replicas $Z_1,\ldots,Z_R$ of Z and estimate the expected value z by the statistic z_R

$$z_R = \frac{1}{R}\sum_{r=1}^{R} Z_r \tag{B.2}$$

Assuming the variance $\sigma^2 = \mathrm{Var}[Z] < \infty$, the Central Limit Theorem (CLT) states that the distribution converges to normal distribution, as $R \to \infty$.

$$\sqrt{R}(z_{\mathrm{R}} - z) \xrightarrow{D} N(0, \sigma^2),\ R \to \infty \tag{B.3}$$

where $\xrightarrow{D}$ indicates distribution convergence.

The result can be interpreted as

$$z_R \overset{D}{\approx} z + \frac{\sigma V}{\sqrt{R}} \tag{B.4}$$

where $\overset{D}{\approx}$ is interpreted as "has the same distribution as", and V has a standard normal distribution $V \sim N(0,1)$; i.e., z_R is distributed as z plus an error. The error for large R is approximately normally distributed, and the approximation has a slow convergence rate $\sqrt{R}$.

In practice, σ^2 is unknown and should be estimated. The estimate is the sample variance s^2 defined by

$$s^2 \overset{\mathrm{def}}{=} \frac{1}{R-1}\sum_{r=1}^{R}(Z_r - z_R)^2 = \frac{1}{R-1}\sum_{r=1}^{R} Z_r^2 - R\, z_R^2 \tag{B.5}$$

The use of $(R - 1)$ follows the standard statistical tradition for making this estimate unbiased, though for large R the difference is minor.

Using the CLT we can define the confidence interval I_α for z. Using the normal distribution cumulative function $\Phi(Z_\alpha) = \mathrm{P}(Z < Z_\alpha) = \alpha$

$$I_\alpha = z_{\mathrm{R}} \pm \frac{Z_{1-\alpha/2}\, s}{\sqrt{R}} \tag{B.6}$$

i.e., z $\in I_\alpha$ with confidence level 1 – α. For the typical choice $\alpha = 5\%$, the corresponding interval is $z_{\mathrm{R}} \pm 1.96\, s/\sqrt{R}$.

For setting a required accuracy, we can make a two-stage procedure. First stage will be a small simulation (e.g., $R = 50$) for estimating the variance s^2 and estimating R accordingly, then simulating R occurrences.

When z is a vector with dimension d of random variables $z = (z1\, z2 \ldots zd)$, where each is the expected value variable $zi = \mathrm{E}[Z(i)]$ of a random variable $Z(i)$, we want to compute an explicitly known and smooth function $f(z)$. As previously defined $Zr = (Z_r(1) \ldots Z_r(d))$ are simulated replicates of Z, and $z_{\mathrm{R}} = (Z1 + Z2 + \cdots + ZR)/R$. The estimate is $f(z_{\mathrm{R}})$, and if $f(z)$ is continues then consistency is guaranteed.

The confidence interval and the convergence rate can be assessed using CLT and deriving the Taylor expansion. The $f(z_R)$ is a biased estimate, with bias that converges as o(1/R).

Another example, the probability distribution $z = P(W_n > x)$ can be computed by the sample of the proportion of W_{rn} that are greater than x, using the estimator z_R

$$z_{\mathrm{R}} = \frac{1}{R}\sum_{r=1}^{R} 1(W_{rn} > x) \tag{B.7}$$

where 1 () is the indication function (i.e., equals 1 if the condition holds). The LLN guarantees that the algorithm converges as the number of replicas R tends to ∞.

In general, the estimator for computing a quantity $\varphi(F)$, where F is the distribution of an underlying random variable Z and φ is a real value function on the probability distribution. For example, estimating $z = E[Z]$ is the case of $\varphi(F) = \int x\, F(dx)$. By drawing R replicates of Z from F, the estimator of $\varphi(F)$ is $\varphi(F_R)$, where F_R distribution is defined by

$$F(dx) = \frac{1}{R}\sum_{r=1}^{R} \delta Z_r(dx) \tag{B.8}$$

where $\delta Z_r(\bullet)$ is a unit point mass distribution at Z.

For $Z \in \mathbb{P}$ the cumulative distribution is

$$F_R(x) = \frac{1}{R}\sum_{r=1}^{R} 1(Z_{\mathrm{R}} \le x) \tag{B.9}$$

For $R \to \infty$, and assuming that all functions are defined and differentiable, then the function $\varphi(F) \to \varphi(F_R)$, and according to CLT, we get a distribution convergence.

$$\sqrt{R}(\varphi(F_R) - \varphi(F)) \xrightarrow{D} N(0, \sigma^2) \tag{B.10}$$

The function $\varphi(F_{\mathrm{R}})$ is distributed as $\varphi(F)$ with an error Y, which has a converging normal distribution, where σ^2 depends on the derivative $d\varphi(F)$ at F.

$$\varphi(F_R) \overset{D}{\approx} \varphi(F) + \mathrm{Y},\ \mathrm{Y} \sim N(0, \sigma/\sqrt{R}) \tag{B.11}$$

Sectioning is division of the R repetitions to M sections of length K, i.e., $R = MK$. Calculating the estimate $F_{m,\,K}$, $m = 1,\ldots,M$, for each section we get

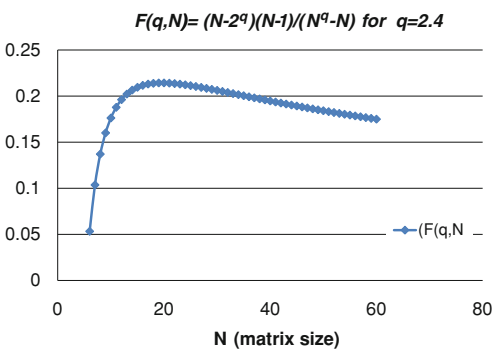
$$F_{m,K}(x) = \frac{1}{K}\sum_{r=(m-1)K+1}^{mK} 1(Z_R \le x) \tag{B.12}$$

The estimator $\varphi(F_{m,\,K})$ is constructed from section m, and we get for large K

$$\varphi(F_{m,K}) \overset{D}{\approx} \varphi(F) + \sigma/\sqrt{K}Y_m, Y_m \sim N(0,1) \tag{B.13}$$

From which, using Taylor expansion we get that the difference of the estimate from the average estimated converges to zero for $R \to \infty$.

$$\sqrt{R}\left(\varphi(F_R) - \frac{1}{M}\sum_{m=1}^{M} \varphi(F_{m,K})\right) \xrightarrow{P} 0 \tag{B.14}$$



q	N max	F(q,N)
2.1	49	0.614203
2.2	28	0.421764
2.3	23	0.298519
2.4	20	0.214215
2.5	17	0.154674
2.6	16	0.111828
2.7	16	0.080669
2.8	15	0.058006
2.9	15	0.041460
3	15	0.029412

Fig. 14.5 Parametric study of Eq. 4.2 **a** $F(N)$ graph for $q = 2.4$ **b** *Max* $F(N)$ table for q values

Following the above, we get for fixed M

$$\frac{\sqrt{M}(\varphi(F_R) - \varphi(F))}{\sqrt{\frac{1}{M-1}\sum_{m=1}^{M}\left(\varphi(F_{m,K}) - \varphi(F_{\mathrm{R}})\right)^2}} \xrightarrow{D} \frac{\sum_{m=1}^{M} Y_m}{\sqrt{\frac{1}{M-1}\sum_{m=1}^{M}\left(Y_m - \frac{1}{M}\sum_{j=1}^{J} Y_{\mathrm{j}}\right)^2}} \tag{B.15}$$

The right-hand side is free of the estimation of σ^2 and has the t-test (Student's test) standard deviation, with $M-1$ degrees of freedom. M is typically small (as for $M > 30$ the t-test distribution converges to normal distribution).

The confidence interval I_α is

$$I_\alpha = \varphi(F_{\mathrm{R}}) \pm t_{\alpha/2}\frac{\widehat{\sigma}}{\sqrt{M}}, \quad \widehat{\sigma}^2 = \frac{1}{M-1}\sum_{m=1}^{M}\left(\varphi(F_{m,K}) - \varphi(F_{\mathrm{R}})\right)^2 \tag{B.16}$$

where $t_{\alpha/2}$ is the t-test percentile for confidence level α (the distribution is symmetric).

The t-test distribution of function distribution estimation by sections can be used in building hypothesis check for decision-making.

14.3 Annex C: Parametric Study of Function $F(q, N)$

The right-hand side of Eq. 4.2, in Sect. 4.2, is defined by the function $F(q,N) = (N - 2^q) * (N - 1)/(N^q - N)$. Following is a parametric study of the function. $F(q, N)$ is a bounded function for each $q > 2$ and is decreasing toward zero for large N. The function is depicted in Fig. 14.5a, for $q = 2.4$. The maximal value is reached at $N = 20$, $F(2.4, 20) = 0.2142$. For other values of q we get other maximal values, each at different N. The table in Fig. 14.5b describes some q values, the respective $N_{\max}$ (where the maximum of the function $F(q, N)$ is reached), and the value of the

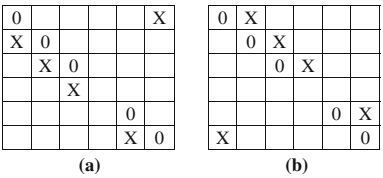


Fig. 14.6 Equation 4.2—ordering without clustering **a** Minimum feedback marks **b** feedback marks close to diagonal

Table 14.2 Optimal ordering and clustering

		Non Clustered solutions			Clustered solutions		
#	*F*, *C*, *q*	w-x-y-z	z-x-w-y	w-x-z-y	[wxyz]	[zxwy]	[wxzy]
1	3, 64, 2.32	384.60	367.58	495.00	201.96	282.48	140.96
2	3, 64, 1.8	239.40	252.40	261.00	201.96	282.48	140.96
3	3, 64, 1.1	128.7	153.22	114.75	201.96	282.48	140.96
4	3, 7, 2.32	60.01	60.19	68.55	30.96	36.24	26.96
5	3, 5, 1.1	16.07	16.63	15.37	24.96	27.6	22.96

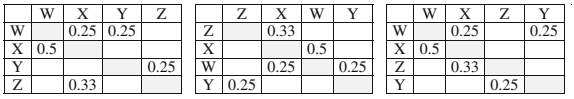


Fig. 14.7 Ordering configurations (a) (b) (c)

function at that point. As q increases the maximal value of $F(q, N)$ decreases. For $q = 2$ the maximal function value is reached at infinity $F(2,\infty) = 1$.

14.4 Annex D: DSM Optimization Without Clustering

The optimization criterion in Sect. 4.2 has interesting results when not used for clustering (i.e., no DBs, the first element in Eq. 4.1 is not used). The two cases in Fig. 14.6 can be analyzed in a similar manner to the analysis made in Sect. 2, in Fig. 4.4 all probability markings are the same.

The cost for the first case is $\text{cost}(a) = ((N-1) \times F \times 2^q + C \times N^q)) \times X$ and respectively $\text{cost}(b) = (F \times N^q + C \times (N-1) \times 2^q) \times X$.

If we want cost(a) < cost(b), then after some reordering we get $((n-1) \times 2^q - N^q). \times F < ((n-1) \times 2^q - N^q) \times C$. Since $F < C$, this implies $((n–1) \times 2^q – N^q) > 0$ or $q < \lg_{N/2}(N-1)$, for $N > 2$. As N increases q decreases toward 1. Otherwise, option (b) in Fig. 14.6 (i.e., multiple feedback marks, close to the diagonal) will be the optimal results.

When the criterion is used for ordering without clustering, it does not create minimum feedback marks results, depending on the values of F, C, and q, as depicted in Table 14.2. The cost of six configurations according to clustering (first three without clustering), and order (depicted in Fig. 14.7), using the probabilities

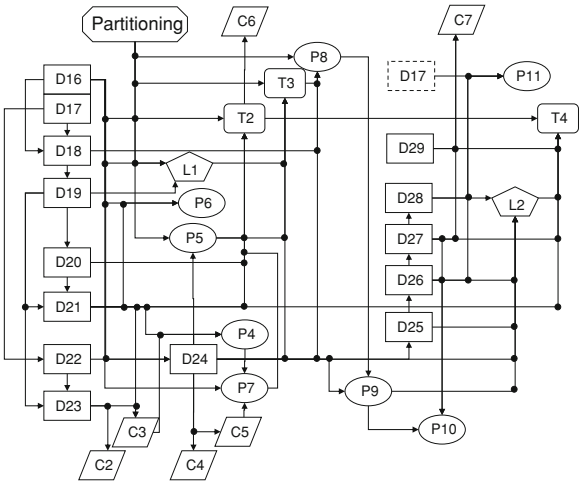


Fig. 14.8 Links between proof items

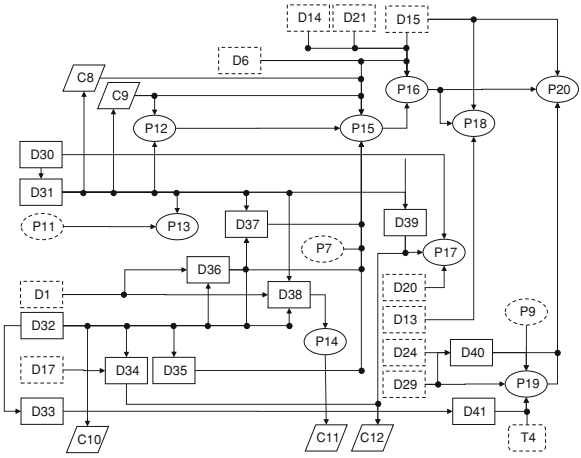


Fig. 14.9 Links of proof items diagram—continued

are presented in Fig. 4.3. The optimal configuration in each case is marked in green. The best non-clustered solution is marked in yellow (as applicable).

The non-clustered configuration (z-x-w-y) in line #1 is the optimal solution for a wide range of parameters for $q > 2$. Changing q or C changes the optimal solution (e.g., in cases #2 and #4). For $q = 2.32$ the non-clustered solution (z-x-w-y) is preferred for $C = 8$.

- Once q is "small enough", the non-clustered solution is always preferred. When all probabilities are similar, for $N = 4$, we get $q = \log_2 3 = 1.585$. Using the example probabilities (Fig. 4.3), the thresholds were: $q = 1.2793$ (for $F = 3$, $C = 64$) and $q = 1.511$ (for $F = 3$, $C = 5$).

Table 14.3 Parallel paths probabilities

#	Process path	x-feedbacks	z-feedbacks	Prob.
1	*WXYZ*	0	0	0.64
2	*WXYWXZ*	1	0	0.0256
3	*WXYWYZ*	1	0	0.02688
4	*WXYWXYZ*	1	0	0.08192
5	*WXYWXWXZ*	2	0	0.0016
6	*WXYWXWYZ*	2	0	0.00128
7	*WXYWXWXYZ*	2	0	0.00512
8	*WXYZWXZ*	0	1	0.02048
9	*WXYZWYZ*	0	1	0.021504
10	*WXYZWXYZ*	0	1	0.065536
11	*WXYWXZWXZ*	1	1	0.00128
12	*WXYWXZWYZ*	1	1	0.001024
13	*WXYWXZWXYZ*	1	1	0.004096
14	*WXYWYZWXZ*	1	1	0.001024
15	*WXYWYZWYZ*	1	1	0.000819
16	*WXYWYZWXYZ*	1	1	0.003277
17	*WXYWXYWXZ*	1	1	0.00512
18	*WXYWXYWYZ*	1	1	0.004096
19	*WXYWXYWXYZ*	1	1	0.016384
20	*WXYWXYZWXZ*	1	1	0.004096
21	*WXYWXYZWYZ*	1	1	0.003277
22	*WXYWXYZWXYZ*	1	1	0.013107
23	*WXYZWXWXZ*	1	1	0.00128
24	*WXYZWXWYZ*	1	1	0.001024
25	*WXYZWXWXYZ*	1	1	0.004096
26	*WXYZWXYWXZ*	1	1	0.004096
27	*WXYZWXYWYZ*	1	1	0.003277
28	*WXYZWXYWXYZ*	1	1	0.013107
29	*WXYZWXZWXZ*	0	2	0.001024
30	*WXYZWXZWYZ*	0	2	0.000819
31	*WXYZWXZWXYZ*	0	2	0.003277
32	*WXYZWYZWXZ*	0	2	0.000819
33	*WXYZWYZWYZ*	0	2	0.000655
34	*WXYZWYZWXYZ*	0	2	0.002621
35	*WXYZWXYZWXZ*	0	2	0.003277
36	*WXYZWXYZWYZ*	0	2	0.002621
37	*WXYZWXYZWXYZ*	0	2	0.010486

- When a non-clustered solution is preferred, it has minimum feedback marks.

Clustered solutions are preferred when q is "high". Some notes regarding clustered solutions:

- The criterion value for a clustered solution is not affected by the choice of q.
- For a clustered solution, minimum feedback marks are always preferred.

14.5 Annex E: Proofs Linkages

The relations between proofs items in Sect. 9.4 are presented in the following diagrams. The diagrams include all the definitions, proposition, lemmas, theorems, and corollaries that are contribution of the current study, and the applicable ones from the Petri net literature (Sect. 6.7).

The proof links of Sect. 9.5 are presented in Fig. 14.8, using the initial letter and number of the item. The links are elbow type and are in general from left to right, except the links between definitions in the first column. In order to clarify link splits, a split or a join of links are marked by small black circles. Lines that cross each other with no circle indicate that there is no connection.

A dashed frame indicates a reference item; either items from the literature, repeated items (to reduce lines), or items from Fig. 14.8 that repeat in Fig. 14.9.

In Fig. 14.9 are the links of the items in Sects. 9.7 and 9.8.

14.6 Annex F: Probabilities of Parallel Paths

The following Table 14.3 presents the probabilities of the parallel path (Sect. 12.4.1); for the conceptual design planning in Fig. 12.3a, using $P = 0.2$.

References

Asmussen S, Glynn PW (2007) Stochastic simulation algorithms and analysis. Springer, New York

Esser R (1998) Petri net tool. Adelaide university. http://www.cs.adelaide.edu.au/users/esser/forkjoin.html. Accessed July 2010

Gilks WR, Richardson S, Spiegelhalter DJ (1996) Markov chain Monte–Carlo in practice. Chapman and Hall/CRC, Boca Raton

Milner R (1999) Communicating and mobil systems: The π–calculus. Cambridge University Press, Cambridge

Sered Y, Reich Y (2006) Standardization and modularization driven by minimizing overall process effort. Compt.-Aided Design 38(5):405–416

Commercial Web Sites

1. ARIS, http://www.ids-scheer.com/en/ARIS_Software_Software/3730.html
2. CPN, http://www.daimi.au.dk/CPnets/
3. Dassult, http://www.3ds.com/products/enovia/welcome/
4. Fujitsu, http://www.fujitsu.com/global/services/software/interstage/bpm/integrate.html
5. IBM (was Cognos-BAM) http://www.cognos.com/products/business_intelligence/event_management/
6. IBM (was Filenet-BPM), http://www-306.ibm.com/software/data/content-management/filenet-business-process-manager/
7. IBM, http://www-306.ibm.com/software/info/bpmsoa/index.html
8. Oracle (BPM), http://www.oracle.com/technologies, bpm/index.html
9. Oracle (was Agile-PLM), http://www.oracle.com/applications/agile/index.html
10. PTC, http://www.ptc.com/products/windchill/
11. SAP Netweaver, http://www.sap.com/platform/netweaver/pdf/BWP_NetWeaver_BPM.pdf
12. Siemens (was UGS), http://www.plm.automation.siemens.com/en_us/
13. Tibco (was Staffware-BPM), http://www.tibco.com/solutions/bpm/default.jsp
14. Tibco (BAM)—http://www.tibco.com/software/business_activity_monitoring/
15. Ultimus, http://www.ultimus.com/products/ultimus_adaptive_bpm.htm

All sites accessed on March 2008.

A. Karniel and Y. Reich, *Managing the Dynamics of New Product Development Processes*, DOI: 10.1007/978-0-85729-570-5,

Index

A. Karniel and Y. Reich, *Managing the Dynamics of New Product Development Processes*, DOI: 10.1007/978-0-85729-570-5,