INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN, ICED11
15 - 18 AUGUST 2011, TECHNICAL UNIVERSITY OF DENMARK

RULES FOR IMPLEMENTATING DYNAMIC CHANGES IN DSM-BASED PLANS

Arie Karniel[1], Yoram Reich[1]
Tel Aviv University

ABSTRACT
Planning a New Product Development (NPD) process has an evolving nature. It is repeatedly updated during the development due to changes in requirements, technology, product concept, or testing results that drive product design modification and require changes in the plan. The Design Structure Matrix (DSM) is utilized to generate a process plan that is based on the product knowledge. The translation of the DSM-based plan to a DSM net process scheme requires implementation rules. Such translation is not unique and there are implementation choices we define as business rules. This paper presents the implementation rules used for dynamic changes in the plan. The application of these rules conforms to a correctness criteria based on the soundness criteria used in Petri nets for process scheme verification.

Keywords: New Product Development (NPD), Design Structure Matrix (DSM), Product Design Process (PDP), Process Planning, Business rules

1 INTRODUCTION
New Product Development (NPD) processes are a subclass of Product development processes (PDP). They are unique, highly complex and dynamic, and iterative. They dynamically evolve due to product related changes, market requirements, technology, and conceptual changes due to test results; as well as organizational related changes such as resources and time constraints. Consequently, the scope of work needed for developing a new product cannot be fully defined nor could be *a-priori* planned [1]. Process modeling has various aspects including *process planning* in terms of the activities content, *process execution* in terms of process correctness, and *process targets* in terms of measured project properties.

The commonly used project planning techniques of GANTT and PERT charts are suited for processes that can be modeled by directed acyclic graphs (DAG). The iterative nature of PDP processes disqualifies the use of the above techniques for their planning. The scheduling literature related to process planning addresses the need to satisfy project targets such as minimum time, resources, or other objective functions [2]. However, these methods typically do not address process iterations or dynamic process planning.

In our research, the Design Structure Matrix (DSM) [3] is utilized to capture product knowledge, including activities interdependencies and process iterations (feedback loops). The DSM plan is translated to a DSM net [4]. The product knowledge evolves in the NPD process; thus, knowledge capturing, DSM-based planning, and translation to a DSM net should be dynamically updated.

The translation of DSM-based plan to a process scheme is not unique; yet it should be explicit and correct [5]. The translating rules of a static plan were described in [6] and are detailed in section 2.4. In order to use the translated process scheme model for simulating PDP processes with changing plans (due to changes in the product knowledge), the translation should be explicitly defined using run time implementation rules. These rules are described in the current article in section 3. Simulations can be further utilized for choosing the most appropriate implementation option, defined as Business Rules (BR), for the various business cases.

2 LITERATURE REVIEW
This section reviews briefly the main concepts related to process modeling that are used in this paper.

2.1 Design Structure Matrix (DSM)
The DSM is a square matrix used to signify the dependency of one element on another, typically by using off-diagonal entries (representing linkages). When modeling processes, a marking in the lower

diagonal portion represents a precedent activity relationship. Parallel or concurrent activities have no relation linkages.

A marking in the upper diagonal portion denotes iteration (a feedback link to an upstream activity). Markings vary, indicating link existence (Binary DSM), magnitude (Value), or probability. In our study, we use probability DSM to denote the probability that a change in the design of item A would cause a change in the design of item B.

Typical DSM-based reordering algorithms try to minimize the number of iterations; thus, feedback marks in an ordered DSM identify a loop of coupled activities [7], [8]. Reordering of coupled activities within a loop requires additional reordering algorithms [4], [9]. Coupled activities were defined as a Design Block (DB) [10]. The probabilities of the input links, output links, and all internal links are united. The integrated probabilities of the internal link become the probability of Self-iteration [6].

The expectation of minimum iteration marks to yield optimal processes in terms of total time and robustness to activity duration variations [3], [7], [11] is an appealing planning heuristic. However, counter examples [12], [13] contradicted this by demonstrating that shortest process time was achieved with more iterations than minimal. The resulting assertion was that simulation results should be used for DSM-based reordering (i.e., process planning).

2.2 Process verification

While the DSM can represent iterations, it does not represent any information regarding the linkages logic and required interpretation. A detailed review of DSM-based plan interpretations to processes is presented in [5], indicating that the translation of DSM-based plan to a process scheme is not unique; yet, it should be explicit and correct. The correctness criteria for NPD processes with dynamically changing process schemes are [6]:

1. As a regular project, NDP 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.
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 is applied in the definition of Workflow (WF)-nets. Requirements 2, and 3(a), echo the soundness criteria used in [14] to define the required properties of a WF-net. Requirement 2 is a modification of the original criterion; in the WF-net literature, the requirement is to reach the termination state from any state that is reachable from the start state. The requirement in [14] applies to a static process scheme; while in the current work, requirement 2 applies to 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 implications of the requirement are: (a) that 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 a quasi-static process scheme. Requirement 3(b) is unique to design processes. Requirement 4 implies the difference between following a static process scheme and a dynamic process scheme, which leads to the difference between the planned process (current or C-process) and the runtime process (RT-process). Requirement 5 implies simulation constraints that should follow reality.

2.3 DSM nets

The *DSM net* was presented in [4] as a process modeling method. It was proved equivalent to a special case of Petri net, the WRI-WF-net (Well-handled with Regular Iterations Workflow net). The WRI-WF-net is sound by construction [15], i.e., in such implementation there are no problematic process issues (e.g., no deadlocks). The build process of WRI-WF-nets ensures the soundness property during net expansion (i.e., replacing an activity in a WRI-WF-net by a subnet that is also a WRI-WF-net). Thus, the DSM net provides a validated expandable modeling method that can model dynamically expanding processes with dynamically changing plans, for serial and parallel logic (and their combinations). A DSM net has *Design Activities* (representing the activity of designing a product item), and *Logic Activities* (that represent the implementation of process logic).

Design activities may have self-iterations (e.g., the checker did not approve the design, or it failed in testing), which are typically not addressed in the DSM literature and were presented in [16].

The DSM net (that might change in time) represents the Current process scheme, i.e., the potential options that are applicable and aligned with the current process state. A runtime (RT) process scheme represent the process history and the current process state (i.e., if an activity has iterated, it will appear twice in the RT process) [17].

2.4 Implementation rules

The following interpretation rules for translating a static DSM structure to a process scheme were defined in [6]. First, we define the logic activities: Begin, End, Input logic (*IL*), and Output logic (*OL*). The following *Implementation Rules* **(**IR**)** apply:

(**IR1**): Logic activity duration is zero and by definition, it does not have self-iterations.

(**IR2**): Design activity has one forward input link (from *IL*) and one forward output link (to *OL*).

(**IR3**): 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 following Implementation Rules for parallel independent activities were introduced in [5] as a procedure to handle multiple parallel initiation and parallel termination of multiple paths simulation.

(**IR4**): The logic of Begin activity is Split-And.

(**IR5**): The logic of End activity is Join-And.

(**IR6**): If an activity has no previous source (input link), it should be linked from the Begin activity.

(**IR7**): If an activity has no target (out link), it should be linked to the End activity.

Forward links and feedback (iteration) links may have distinct logic operands; the following basic Implementation Rules are defined for both *IL* and *OL*.

(**IR8**): Forward links to design activities (lower DSM) have AND logic, on first iteration.

(**IR9**): Forward link to the End activity, have XOR (Exclusive OR) logic with the other links.

(**IR10**): Feedback (iteration) links have OR logic.

Serial activities in a design process might be the result of standardization [18], e.g., the design of a standard part A may influence the design of B, but changes in the design of B will not affect the design of A (otherwise A is not standard). The interpretation of a forward link in case of iterations is not unique. If activity *A* iterates, can *B* start in parallel, or should it wait until A completes all its iterations. The interpretation may be one of the following Business Rule options.

(**IR11**): Output logic options: sending completion signal(s) to subsequent serial design activities may follow one of the following BRs:

(**BR11.1**): Sending only once the activity has completed all its iterations.

(**BR11.2**): 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., IR9).

In the latter case (BR11.2), there might be iterations of the previous activity A that are executed in parallel to activity B. The implementation of this case requires additional rules, which are useful for other parallel cases as well. Such possibility was not studied in the existing literature.

(**IR12**): Output logic: signal to End Activity. Second (or subsequent) execution of an activity (e.g., *A*) may send signal to End Activity, while the following serial activities (e.g., B) have started execution.

(**BR12.1**): On the second (or subsequent) execution, the activity must be followed by its next serial activities.

(**BR12.2**): On the second (or subsequent) execution of the activity, the next activities may follow, or the End activity may follow (not both, subject to IR9).

Allowing early start option (BR11.2) leads to Run Time options (IR13). These options can be viewed only in the Run Time process scheme.

(**IR13**): Iterations of the same activity cannot occur in parallel:

(**BR13.1**): While the activity is executing, input link signals are directed to this activity (i.e., to its input Logic activity).

(**BR13.2**): 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).

The coupled activities case may have serialization requirements (e.g., testing activity should serially follow design activities).

(**IR14**): Coupled activity execution start:

(**BR14.1**) (serialization): coupled activity may start after its previous activity (according to DSM) has completed at least once.

(**BR14.2**) (parallel): coupled activity may start in parallel to all the other activities in the same activity loop.

In the case of coupled activities, the options described in IR12, are respectively replaced by the options of IR15.

(**IR15**): Sending signal on second (or later) completion of a coupled design activity is done according to one of the following BRs:

(**BR15.1**): A coupled design activity should link to the next activity on its completion.

(**BR15.2**): A coupled design activity may link to the End activity on its completion.

2.5 Formulating the rules

The following logic formulations (comparing DSM interpretations) were defined in [5]:

Logic indication: Split ($\Rightarrow$) for Output logic; Join ($\Leftarrow$) for Input logic.

Logic operations: + (OR); $\bullet$ (AND); $\oplus$ (XOR, Exclusive-Or).

The short form of multi-variable logic operations: Multiple-Or $\Sigma(A_i) = A_1 + A_2 + \cdots + A_n$; Multiple-And $\Pi(A_i) = A_1 \bullet A_2 \bullet \cdots \bullet A_n$; and Multiple-Xor $\otimes(A_i) = A_1 \oplus A_2 \oplus \cdots \oplus A_n$, 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 was presented for the implementation of Begin, End, IL and OL:

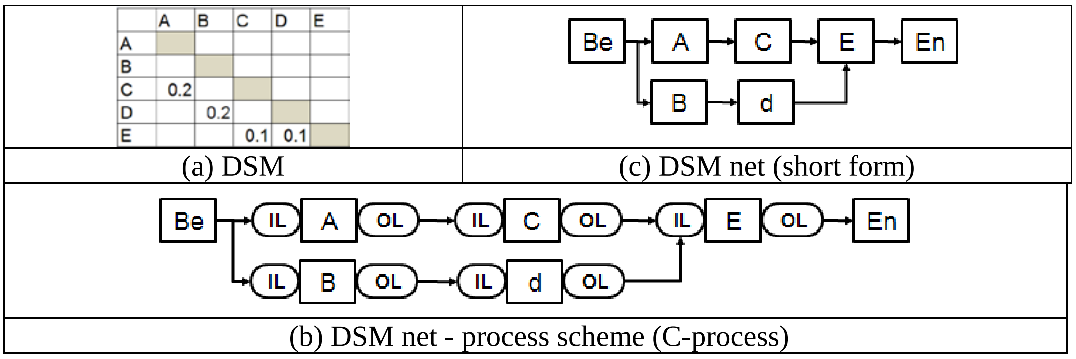
$Begin \Rightarrow \Pi(F_i)$	(1)	$OL \Rightarrow (\Sigma(I_i) \oplus \Sigma(F_i)) \oplus End$	(2)
$End \Leftarrow \Pi(F_i)$	(3)	$OL \Rightarrow (\Sigma(I_i) + \Pi(F_i)) \oplus End$	(4)
$IL \Leftarrow (\Sigma(I_i) + \Pi(F_i))$	(5)	$OL \Rightarrow (\Sigma(I_i) + \Sigma F_i)) \oplus End$	(6)
$OL \Rightarrow (\Sigma(I_i) \oplus \Pi(F_i))$	(7)	$IL \Leftarrow Begin + \Sigma(I_i) + \Sigma(F_j) + \Pi(F_k)$	(8)

Where F_j are forward links within the DB, and F_k are other forward links.

The above rules are applicable to the various combinations of business rule cases, e.g., for the case of BR11.1 and BR12.1 the applicable formulations are (3) for IL, and (4) for OL, see annex A.

2.6 Implementation examples

Simple demonstrations of the above implementation rules are depicted in Figure 1. A probability DSM is presented in Figure 1(a) of a serial case. Its transformation to a DSM net for the logic of BR11.1 and BR12.1 is presented in (b). A short graphical representation form (without *IL* and *OL* for each design activity) is depicted in (c). The short form is used in the following examples.



	A	B	C	D	E
A					
B					
C	0.2				
D		0.2			
E			0.1	0.1	

(a) DSM (c) DSM net (short form)

(b) DSM net - process scheme (C-process)

Figure 1. From a DSM to a DSM net

3 DYNAMIC IMPLEMENTATION RULES

The implementation rules, described in section 2, provide interpretation options of a DSM structure (after reordering) to a process scheme. There are no preferred interpretations; the applicable interpretation depends on the particular business case. In NPD processes, the DSM structure is expected to change according to changes in the product knowledge (activities, and dependency linkages).

3.1 Self-iteration rules

Self-iteration rules apply to the dynamics of the process, i.e., once activity iteration occurs [16]. Applying the rule option do not change the DSM net, but shapes the runtime process.

(**IR16**): *OL* and *IL* options for early or late start of proceeding activity may follow one of the BRs:

(**BR16.1**) (Late start): *IL* – Starting the activity execution once all iterations of preceding activities have completed. *OL* – Sending forward signal only once the activity has completed all its iterations.

(**BR16.2**) (Early start): *IL* – Starting execution once each preceding activity have completed at least once. *OL* – Sending forward signal once the activity has completed its first execution; sending a signal to End activity can be done only once all iterations have completed (IR9).

3.1 Process scheme changes

Ad-hoc changes of a process scheme do not necessarily follow any pattern. Their implementation is left to the user that should take care to keep process correctness. Various tools (e.g., "Woflan" system [19]) may help checking the proposed change. A change that cannot be implemented (violating correctness requirements) should be notified to the user. The current work is focuses on the more structured changes resulting from product knowledge changes via DSM interpretations. Scheme change types can be classified as follows:

(1) *Add activity*. Using IR2 and IR3, any added activity should have at least an input forward link and an output forward link to other activities.

(2) *Adding forward links* may reduce parallelism (if activities were previously not linked). Such addition can immediately be implemented, with no special process logic issues involved.

(3) Adding / removing feedback links may have implications to the definition of Design Blocks (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 a 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 an 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 Figure 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 Figure 2(a). The same logic may apply if there is only one outgoing forward link from *OL* of *A* to *IL* of activity *C*, figure 2(b).

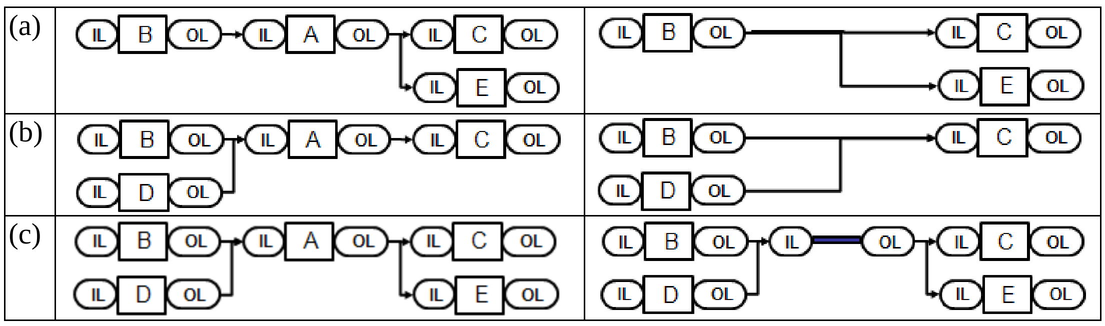


Figure 2. Design task removal

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, Figure 2(c). Therefore, replacement by dummy is the option being implemented always.

Note: if there is a self-iteration link of the dummy activity (*OL* to *IL* of *A*), it should be removed (avoiding infinite loops of process with zero duration that consume computing resources).

3.2 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 BR11.1 or BR11.2 would have different impact. According to BR11.2, 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 BR11.2 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., BR13.1 will merge to current execution, and BR13.2 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 (the link target). 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.

(5) Removal of a link when the source is inactive requires complying with IR 2 and IR 3, i.e., a single link from the design activity to an OL logic activity or from IL logic activity to a design activity cannot be removed. A link from 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 IL, it should be replaced by a link from Begin activity (IR 6). If it is a single out link from OL is should be replaced by link to the End activity (IR 7).

(6) Removal of an activity that has never been performed could be done immediately (i.e., replacing it by a dummy activity). Removal of a design activity that is currently performing is postponed until the activity has 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 be removed.

The following rule indicates the options of defining the duration of a removed activity.

(**IR17**): Duration of removed design activity:

(**BR17.1**) (As soon as possible): Activity duration is reduced to be its current duration.

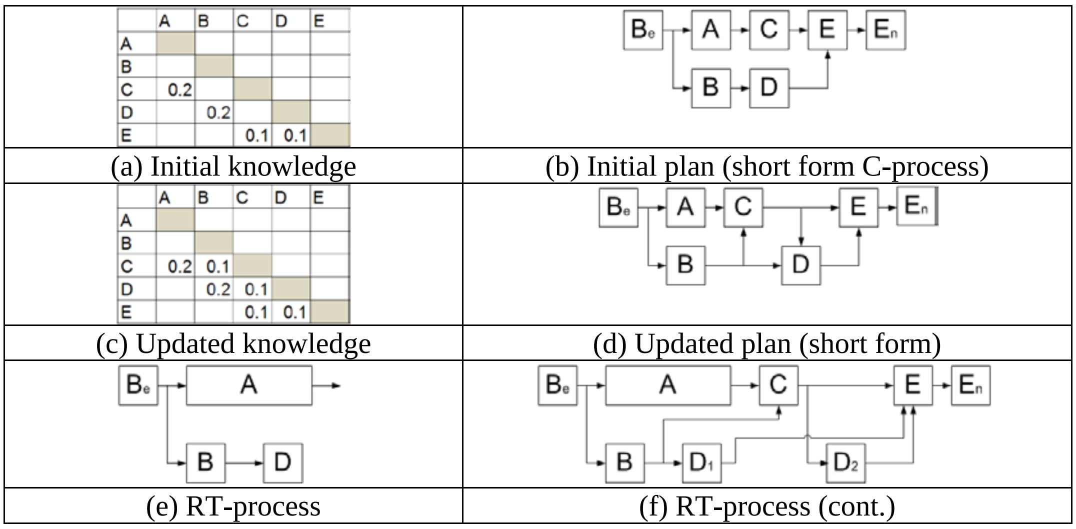
(**BR17.2**) (Minimal duration value): Activity duration is set to be the maximum of either minimal duration or current duration.

The implementation of scheme changes may cause activity iterations even if the planning does not indicate so: for example, the DSM in Figure 3(a), whose short form process plan is in (b) (repeating Figure 1(a) and (c) respectively). At a certain time, it was realized that activity *C* needs input from *B* and activity *D* needs input from *C*, see the DSM and process plan in Figure 3(c) 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 the additional knowledge and plan update are done after D has started, and if activity *A* has a long duration (e.g., D(*A*)>D(*B*)+D(*D*)), as schematically depicted in Figure 3(e) (such that the rectangle length indicates relative time), then activity *D* should start after C and therefore should iterate as shown in Figure 3(f).

Note 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 (from *C* to *D*) 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 example demonstrates the importance of the distinction made between the C-process, and the RT-process. This separation is essential for modeling the process scheme dynamics. The RT-process model keeps the interim process information, while the C-process plan is dynamically changing.



(a) Initial knowledge

	A	B	C	D	E
A	▨				
B		▨			
C	0.2		▨		
D		0.2		▨	
E			0.1	0.1	▨

(c) Updated knowledge

	A	B	C	D	E
A	▨				
B		▨			
C	0.2	0.1	▨		
D		0.2	0.1	▨	
E			0.1	0.1	▨

(b) Initial plan (short form C-process)
Be
A
C
E
En
B
D
(d) Updated plan (short form)
(e) RT-process
(f) RT-process (cont.)
D1
D2

Figure 3. Scheme changes lead to iterations

Link changes either addition/deletion or change in a value in the DSM may cause different ordering and different allocation to DBs. Implementation of changes in DB content, are 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.
(**IR18**): Changes in design block content:

(**BR18.1**) (As soon as possible): Change parallel completion constraints immediately.
(**BR18.2**) (One cycle completion): All design activities in design block should have completed at least once.
(**BR18.3**) (Current cycle completion): All parallel activities in design block should complete the current cycle.

The business rule BR18.2 (one cycle completion) is actually a combination of the other two rules. Using a counter of DB executions, it is equivalent to BR18.3 at the first execution and to BR18.1 in all subsequent iterations.

3.3 Business cases

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 (BR16.2) or Late (BR16.1) start of activity or DB.
- Merger to executing activity (BR13.1), or starting a new iteration (BR13.2).
- Exit option (jumping to the end activity) on second iteration (BR12.2 or BR15.2), or not (BR12.1 or BR15.1, respectively).[1]
- DB change implementation: ASAP (BR18.1), at least one execution (BR18.2), or by the end of current DB iteration (BR18.3).

3.4 Applying Business Rules combinations at run time

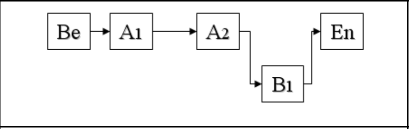
The implication of a BR on the process performance (duration and required resources) is dependent on the actual duration of the activities. The following simple examples demonstrate the various

[1] Reminder, the business rules of IR15 apply to coupled activities, while IR12 applies to serial activities.

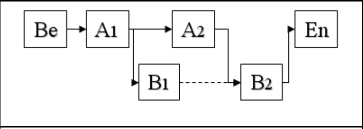
possibilities, having two activities with a serial link from *A* to *B* with self-iterations of *A*. Activity *A* executes twice, activity *B* executes twice at most. The DSM of the process is depicted in Figure 4(a).

$$\begin{array}{c|c|c} & A & B \\ \hline A & Pa & \\ \hline B & Pab & 0 \end{array}$$

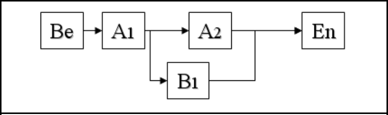
(a) DSM



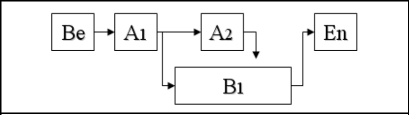
(b) Late start



(c) Early start



(d) Exit



(e) Merge

Figure 4. RT processes of various Business Rules

The run time processes are illustrated in Figure 4. The process total duration *T*, the required resources *R*, and reference to process figure are indicated in Table 1. The cases represent shortest path options, subject to iteration of *A*. The durations are represented in some time units, and resources consumption is assumed to be one resource per activity per time unit. The shortest path (*Be-A-B-En*) without iterations has duration and resources of 50 units.

The combinations of the following rules are demonstrated for two activity duration settings: first D(*A1*)=40, D(*B1*)=10; and second D(*A1*)=10, D(*B1*)=40:

- 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 (BR16.1) versus Late start (BR16.2);
- Merger to executing activity (BR13.1) versus initiating a New iteration (BR13.2); and
- Exit option of second iteration (BR12.2) versus continuing by enforcing iteration of the next serial activity (BR12.1).

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 1.

Table 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	40	10	40		1	Late	N/A	N/A	90	90	(b)
2	40	10	40	10	1	Early	N/A	Cont.	90	100	(c)
3	40	10	40		1	Early	N/A	Exit	80	90	(d)
4	40	10	20		0.5	Late	N/A	N/A	70	70	(b)
5	40	10	20	5	0.5	Early	N/A	Cont.	65	75	(c)
6	40	10	20		0.5	Early	N/A	Exit	60	70	(d)
7	10	40	10		1	Late	N/A	N/A	60	60	(b)
8	10	40	10	40	1	Early	New	Cont.	90	100	(c)
9	10	40	10		1	Early	N/A	Exit	50	60	(d)
10	10	40	10		1	Early	Merge	N/A	50	60	(e)
11	10	40	5		0.5	Late	N/A	N/A	55	55	(b)
12	10	40	5	20	0.5	Early	New	Cont.	70	75	(c)
13	10	40	5		0.5	Early	N/A	Exit	50	55	(d)
14	10	40	5		0.5	Early	Merge	N/A	50	55	(e)

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.

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 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.

3.5 Applying Business Rules combinations for DB at run time

A probability DSM with iteration is depicted in Figure 5(a). The DSM is reordered in (b) and presented as DB implementation at (c), where activities (*A*,*C*,*E*) are presented as (*ACE*) design block. The assignment of the reordered DSM into the DB form depends on the algorithm used for identifying design blocks. The self-iteration probability of the design block is calculated according to the following equation (1) in [6].

$$P_d = 1- \Pi(1-P_i) \qquad (1)$$

For the current example, the self-iteration probability is $P_d=1-(1-0.1)^4=0.344$

	A	B	C	D	E
A	▨				0.1
B		▨			
C	0.1		▨		
D		0.2		▨	
E	0.1		0.1	0.1	▨

(a) Initial knowledge DSM

	B	D	A	C	E
B	▨				
D	0.2	▨			
A			▨		0.1
C			0.1	▨	
E		0.1	0.1	0.1	▨

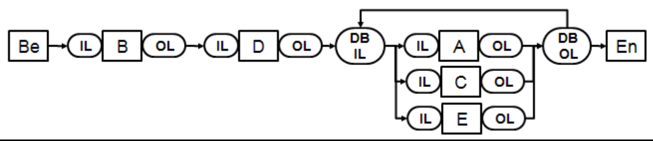
(b) Reordered DSM

	B	D	ACE
B			
D	0.2		
EAC		0.1	0.34

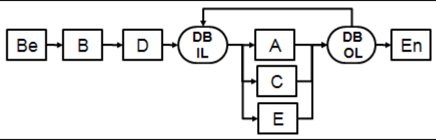
(c) Design Block implementation

Figure 5. DSM-based reordering and its conversion to a DB implementation

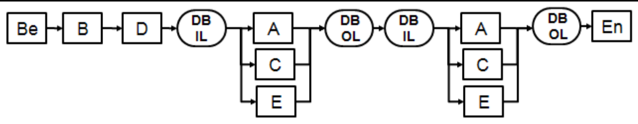
Using the implementation rules (IR1 to IR10), we can obtain the DSM net in Figure 6(a), from the DSM in 5(c), by adding the Begin and End activities, adding IL and OL activities, and adding additional *DBIL* and *DBOL* activities with a feedback (iteration) link from *DBOL* to *DBIL*. A short form description is used in Figure 6(b). The RT-process with a DB iteration is depicted in (c).



(a) DSM net (C-Process)



(b) DSM net (short form)



(c) RT-process

Figure 6. DSM net (C-process) and RT-process

The product knowledge may change during the development, e.g., linkages were identified between *C* and *D*, and also between *D* and *C*, as depicted in Figure 7(a). The DSM may be reordered to the DSM in (b) according to the procedure developed in [4] or to the DSM in (c) using the optimization sequencing in [9]. The reordering in (b) indicates one DB (*ADCE)* with an internal loop (*DC*). The reordering in (c) indicates two DB's (*DC*) and (*AE*) with a feedback link between them. The probabilities for the DB's in both cases are calculated according to Equation (1). The conversion of each option to a DSM net follows the example above.

	A	B	C	D	E
A	▨				0.1
B		▨			
C	0.1		▨	0.3	
D		0.2	0.2	▨	
E	0.1		0.1	0.1	▨

(a) Product knowledge change

	B	A	D	C	E
B	▨				
A		▨			0.1
D	0.2		▨	0.2	
C		0.1	0.3	▨	
E		0.1	0.1	0.1	▨

(b) Procedure reordering

	B	D	C	A	E
B	▨				
D	0.2	▨	0.2		
C		0.3	▨	0.1	
A				▨	0.1
E		0.1	0.1	0.1	▨

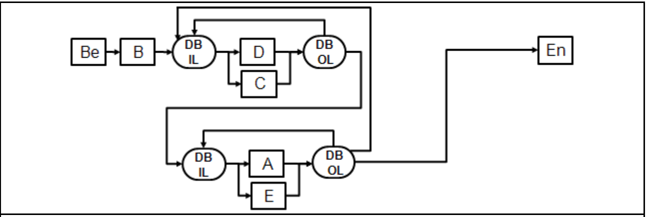
(c) Ordering by optimization

Figure 7. DSM-based reordering in case of DB

The implementation of the DSM in Figure 7(c) into a DSM net is presented in Figure 8. The assignment of the design activities into DB's is presented in Figure 8(a). The two DB's are coupled (i.e., could be implemented as one DB). The implementation of activities within a DB is done according to IR14. In this case, the rule is applied three times, one for the activities within (*DC*), once for the activities in (*AE*), and once for the relation between (*DC*) and (*AE*). If in all three case BR14.2 (parallel) was applied with BR15.1 (equation (6), see annex A); then the resulting DSM net is depicted in Figure 8(b).

	B	DC	AE
B			
D	0.2	0.44	0.1
AE		0.19	0.19

(a) DB implementation



(b) DSM net (parallel implementation)

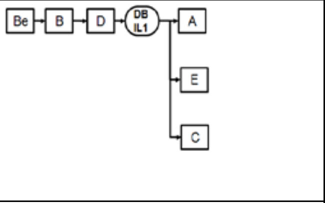
Figure 8. From a DSM with DB to a DSM net

For demonstrating a dynamic change in the process scheme (C-process) due to product knowledge, we shall define the following use case:

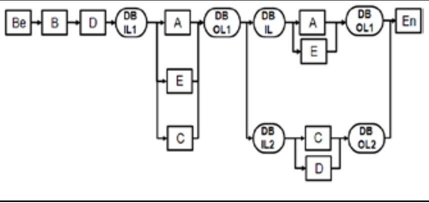
- The additional knowledge was obtained during the first execution of the (*ACE*) DB (Figure 6(c))
- We use the implementation depicted in Figure 8(b).

The implementation of the change can be done according to BR18.1 (ASAP), BR18.2 or BR18.3. As previously indicated, in this case, BR18.2 is equivalent to BR18.3. The RT-process at the time of the change is depicted in Figure 9(a). Implementation of the revised process plan according to BR 18.2 (or BR 18.3) is depicted in (b). After the completion of (*ACE)* on an additional iteration, activity *C* is part of (*DC)* and after both design blocks have completed the process reaches the End activity. Note that the additional iteration is not necessary as all activities have completed at least once.

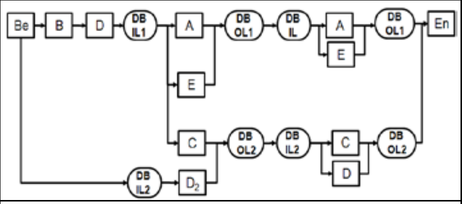
The more interesting case is the implementation according to BR18.1 that is depicted in (c). In the latter case, activity *C* starts as part of design block (*ACE*) but completes as part of design block (*DB),* while activities *A* and *E* complete as part of (*AE*). Again additional iteration of each DB was added (though it is not necessary for completing the process), Note that in the case of applying BR18.1 activity D is executed three times.



(a) RT-process at the time of the process change



(b) RT-process with changes according to BR 18.2/BR 18.3



(c) RT-process with changes according to BR 18.1

Figure 9. DB changes at run time

4. Discussion and conclusions

The design activities order of a new product is not predefined, and the product knowledge used for ordering these activities keeps changing during the development process. The DSM-based planning method has non-unique results in implementing the plan into process logic. The problem of interpreting the process logic exacerbates when the process plan is changing dynamically. Implementation Rules and the implementation options defined as Business Rules were defined in [6] for static process scheme case, and enhanced in the current article to the cases of dynamic changes of the process plan. In order to apply process changes dynamically, the process should confirm with correctness criteria. It was proved in [4] that using design block implementation (either serial, parallel, or combinations) with the defined Implementation Rules (and the BR options) keep the correctness properties through a build process. Simulation techniques are required for analyzing process objectives, regarding time, cost, rework effort [12], risk propagation [20], uncertainty and learning [21]. DSM-based simulations using the IR can be used for evaluating the process parameters.

Annex A

The relations between Implementation Rules, Business Rules, and the formulation of Input and Output Logic, are shown in Figure 10

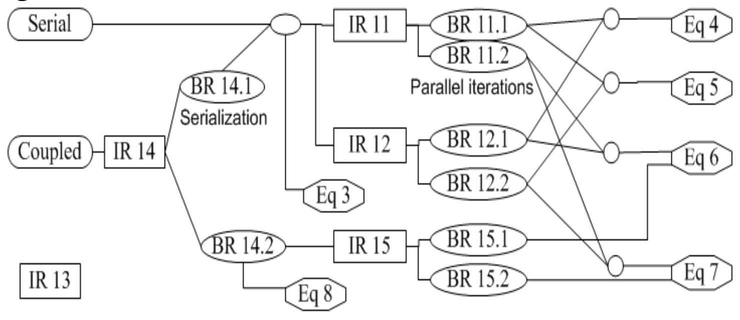


Figure 10. Business Rules and the applicable logic formulation [6]

REFERENCES

[1] Westfechtel B., *Models and tools for managing development processes*, *Lecture Notes in Computer Science*, vol. 1646, Springer, Berlin, 1999.

[2] Brucker P., Drexl A., Möhring R., Neumann K., Pesch E., Resource-constrained project scheduling: Notation, classification, models and methods, *European Journal of Operational Research*, 112:3–41, 1999.

[3] Steward D.V., The design structure system: a method for managing the design of complex systems, *IEEE Transactions on Engineering Management*, 28:71–74, 1981.

[4] Karniel A., Reich Y., Formalizing a workflow net implementation of Design Structure Matrix based process planning for New Product Development, *IEEE Transactions on Systems, Man and Cybernetics A*, in press, 2011.

[5] Karniel A., Reich Y., From DSM-based planning to design process simulation: A review of process-scheme logic verification issues, *IEEE Transactions on Eng. Management*, 56(4):636-649, 2009.

[6] Karniel A. and Reich Y., A coherent interpretation of DSM plan for PDP simulation, *in Proceedings of the International Conference on Engineering Design, ICED 07,* Paris, 2007(c).

[7] Eppinger S.D., Whitney D.E., Smith R., Gebala D., A model-based method for organizing tasks in product development, *Research in Engineering Design*, 6(1):1-13, 1994.

[8] Gebala D.A., Eppinger S.D., Methods for analyzing design procedures, *ASME 3rd Int. Conf. On Design Theory and Methodology*, pp. 227-233, 1991.

[9] Karniel A., Belsky Y., Reich Y., Decomposing the problem of constrained surface fitting in reverse engineering, *Computer-Aided Design*, 37:399-417, 2005.

[10] Karniel A. and Reich Y., Managing dynamic new product development processes, *in Proceedings of the 17th Annual International Symposium of The International Council on Systems Engineering- INCOSE'07,* San Diego, California, June 2007b.

[11] Smith R.P. and Eppinger S.D., A predictive model of sequential iteration in engineering design, *Management Science*, 43(8):1104-1120, 1997(b).

[12] Browning T.R. and Eppinger S.D., Modeling impacts of process architecture on cost and schedule risk in product development, *IEEE Transactions on Engineering Management,* 49(4): 428-442, 2002.
[13] Abdelsalam H.M.E. and Bao H.P., A simulation-based optimization framework for product development cycle time reduction, *IEEE Transactions on Engineering Management*, 53(1):69-85, 2006.
[14] Aalst W.M.P. van der, and Hee K.M. van, *Workflow Management: Models, Methods, and Systems*, MIT Press, Cambridge, MA. 2002.
[15] Ping L., Hao H., Jian L., On 1-soundness and Soundness of Workflow Nets," *in D. Moldt (Ed.): Proceedings of the Third Workshop on Modelling of Objects, Components, and Agents, DAIMI PB – 571*, pp. 21-36, Aarhus, Denmark, 2004.
[16] Karniel A. and Reich Y., Simulating design processes with self-iteration activities based on DSM planning, *IEEE Proceedings of the International Conference on Systems Engineering and Modeling - ICSEM'07*, pp. 33–41, Haifa, March, 2007a.
[17] Karniel A. and Reich Y., *Managing the Dynamics of New Product Development Processes - A new Product Lifecycle Management Paradigm*, Springer, in press.
[18] Sered Y., Reich Y., Standardization and modularization driven by minimizing overall process effort, *Computer-Aided Design*, 38(5):405-416, 2006.
[19] Verbeek H.M.W., Basten T., and Aalst W.M.P. van der, Diagnosing workflow processes using Woflan, *The Computer Journal*, 44(4):246–279, 2001.
[20] Eckert C.M., Keller R., Earl C., and Clarkson P. J., Supporting change processes in design: Complexity, prediction and reliability, *Reliability Engineering and Safety System*, 91(12):1521-1534, 2006.
[21] Cho S.H. and Eppinger S.D., A simulation-based process model for managing complex design projects, *IEEE Transactions on Engineering Management*, 52(3):316-328, 2005.

Contact: Prof. Yoram Reich
School of Mechanical Engineering
Tel Aviv University
Tel Aviv 69978
Israel
Tel: +972 3 6407385
Fax: +972 3 6407617
Email: yoram@eng.tau.ac.il
URL: http://www.eng.tau.ac.il/~yoram

Arie Karniel has a Ph.D. in mechanical engineering from Tel Aviv University. His expertise spans engineering design, product management, and system engineering in various industries. His research interests include product life cycle management, process modeling, and engineering knowledge transformation to production.

Yoram Reich is a professor at the Faculty of Engineering, Tel Aviv University. He is the Editor-in-Chief of *Research in Engineering Design* and serves on the editorial board of 5 other journals; a co-chair of the design theory special interest group of the design society, and a member of the advisory board of the society. He recently founded and is the chairman of Israel Institute for Innovation Technology