

# From DSM based planning to Design Process Simulation: A review of process scheme verification issues

Arie Karniel Yoram Reich

**Abstract**—Planning Product Design Processes (PDP) is complex and challenging. The plan should reflect the product related knowledge, specifically the influences of performing changes in one product component on the need to iteratively rework the design of other components. Given the uncertainties of design processes, the plan evaluation requires simulation tools. The Design Structure Matrix (DSM) is a known method for design process planning. However, the DSM itself does not express all the relevant information required for defining process logic.

Many logic interpretations are applicable in different business cases. However, a consistent method of transforming a DSM model to logically correct concurrent process model, in the case of iterative activities is lacking. A gap was identified between the literature concerning the activities sequencing plan, based on DSM, and the process modeling literature concerning process verification. This survey systematically classifies the approaches used in DSM based process planning and discusses their strengths and limitations along problems related to process modeling verification of iterative processes.

**Index Terms**—Design Structure Matrix (DSM), Product Design Process (PDP), Process Knowledge, Process Simulation, Process Verification, Workflow

## I. INTRODUCTION

PRODUCT development processes are widely recognized as crucial for business competitiveness. The current business environment dictates to simultaneously reduce development lead-times, reduce cost, and improve quality [1], [2]. Product Development Process (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 Design Process reflects the specific design requirements, objectives, and constraints applicable to the current product.

The interdependence between design activities makes the 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 inherent and necessary part of design [3], feedback due to

unplanned changes or problems cause an additional iterative rework that cannot be predicted [4]. Iterations are considered as major source of increased product development lead-time and cost [5], [6], [7]. Therefore, performing project activities in an appropriate sequence is critical for minimizing rework due to information interdependencies [8]. Planning and predicting the consequences of change in a product are essential in modifications to existing products, especially safety critical products [9].

The current article is focused on the transformation stage from process planning to a Process Scheme model (Workflow); specifically on the transformation of the Design Structure Matrix (DSM) [11], used as a planning method, into the logic of the resulting process. DSM does not represent any information regarding the linkages logic, thus the transformation may have various interpretations that may actually represent different business cases. Various simulation process structures and process progress types have been introduced in the literature, including serial and parallel process activities, which progress according to deterministic, Markov chain, or Monte Carlo methods.

Process verification issues, which are thoroughly discussed in the process modeling literature [12], [13], [14], are seldom mentioned in the DSM literature. One of the assumptions raised in this article is that DSM allows straightforward verification check, which can be manually done for some example cases. However, verification checks become essential once the DSM should automatically be translated to a process scheme. Such automatic translation is necessary when the product knowledge, being the foundation of the DSM, has changed. Automation might also be very useful for large scale processes.

The current survey proposes a classification model, which segregates the different approaches, and is further used for analysis of the verification consequences of the logic implementation being practiced in each case. No previous DSM-based related work has comprehensively discussed logic verification issues.

## II. LITERATURE REVIEW

The following survey starts with definitions of the terminology used. The DSM literature follows, focusing on DSM-based process simulations. Finally, process verification issues are overviewed.

### A. Ontology

Since different literature sources make use of the same term with different intent, or use different terms with the same

Manuscript received Sept 15, 2006.

Arie Karniel is with the School of Mechanical Engineering, Tel Aviv University, Tel Aviv, Israel (e-mail: ariek@eng.tau.ac.il).

Yoram Reich is with the School of Mechanical Engineering, Tel Aviv University, Tel Aviv, Israel (e-mail: yoram@eng.tau.ac.il; phone: 972-3-640-7385; fax: 972-3-640-77617).

meaning; the terms being used in this article are explicitly defined. The term *Process Scheme*, is used to describe the process structure, i.e., the model of linkages between activities, their precedence, concurrency, and logic relations. This term is equivalent to the terms: *Process Definition* [15], *Workflow Process* [16], *Process type* [17], *Workflow Type* (WF type) [18], *Workflow scheme* [12], [19], *Process Model* [20], and *Process description* [24]. The term *Design* is exclusively used in the context of the product; and the term *Planning* is used in the context of process management, and refers to *Process Scheme* creation.

A limited set of process constructs are being used to define the Input logic of an activity (pre-conditions) and its Output logic (post conditions). A comprehensive study of construct definitions (Workflow patterns) is elaborated in [21]; the definitions of this study are referred to by the following numbers in brackets. The constructs utilized are: Serial link {1}; Split-And (fork, {2}), meaning sending a termination signal through all output links; Split-Or (multi choice, {6}), sending termination signal to some of the output links; Split-Xor (Exclusive choice, {4}), sending to only one of the output links; Join-And (Synchronize, {3}), wait for all input signals; Join-Or (Multi merge, {8}), wait for first to come, execute multiple times; and Join-Xor (Simple merge, Asynchronous join {5}).<sup>1</sup>

### B. 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 not adequate for planning design processes, as they do not effectively model the design activities interdependencies and process iteration (feedback loops) [22], [23]. A comprehensive survey of methods being used for modeling design processes is given by Browning *et al.*, [24] Using multiple views of the different representation for different stakeholders is presented in [25], [26], and [27].

The Design Structure Matrix (DSM) introduced by Steward [11] and extended by Eppinger *et al.* [5], is utilized to capture the required product knowledge. In a DSM survey [28], four main categories were indicated: Component-based or Architecture DSM; Team-based or Organization DSM; Activity-based or Schedule DSM; and Parameter-based DSM. The first two are considered static, as they represent existing elements. The latter two are defined as time-based, as their ordering implies flow. Eppinger and Salminen [29] presented the interactions between these different aspects. When used to model design processes, the matrix presents process activities dependencies. DSM provides the means to identify serial, parallel, and iterative design flows; and models iterations and multidirectional information flows [5].

1) *DSM basics*: DSM is a  $N^2$  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 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 lower diagonal 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 impact a previously performed activity, which should be performed again, i.e., rework<sup>2</sup>.

Serial activities (synonyms: sequential, dependent, precedence) are linked by forward link, and indicate a serial process, Figure 1(a). Parallel activities (synonyms: concurrent, independent), have no relation linkages, Figure 1(b). Coupled activities (synonyms: interdependent, mutually dependent) have forward and feedback links, Figure 1(c). Additional relation type, Overlap activities, can be considered as a Parallel relation [31], [7]. Cho and Eppinger [32],[33] modeled two overlapping activities as parallel activities with additional lead time, and a constraint preventing the latter activity from completing before the first activity. Nevertheless, it was indicated that this model does not scale to multiple activities having multiple iteration paths.

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

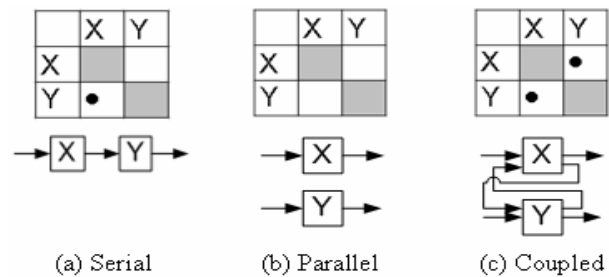


Fig. 1. Activity relations types in DSM, and corresponding process schemes adapted from [30]

Figure 2. Activity loop may have internal coupling, Figure 2 (d) and (e). The order or the activities ascribes different interpretations of the dependency links. The processes described by Figure 2(b) and (c), have the same set of links, but the link from Z to Y has a different interpretation (feedback in (b) forward in (c)); therefore it represents different types of process logic. Concurrent process interpretations are depicted in Figure 2(f) to (j), as detailed in the following sections.

2) *DSM based algorithms*: There are three main types of DSM related algorithms: Partitioning, Sequencing (includes Tearing), and Clustering. Partitioning is done by decomposing the process into clusters of interdependent activities.

<sup>1</sup> The process construct Synchronizing Merge {7} is defined at run time. It acts as {5} if only one process path is executed and as {3} if multiple paths are executed concurrently. It is not used in this study.

<sup>2</sup> Some articles use the opposite convention; upper diagonal expresses forward links (e.g., [28] and [22]).

Partitioning is the commonly used algorithm for reordering DSM activities and achieving minimum iterations. Minimum iterations marks are expected to yield optimal processes [11], [34], [35]. Partitioning methods include: the Path Searching method [36]; the Reachability Matrix method [37]; the Triangularization algorithm [38]; and the powers of the Adjacency Matrix method [39] in [40]. Partitioning results in a lower matrix or block diagonal matrix in the case of coupled activities.

Ideal sequencing with no feedback marks is unlikely to exist [41]. Thus, coupled activities within a loop require

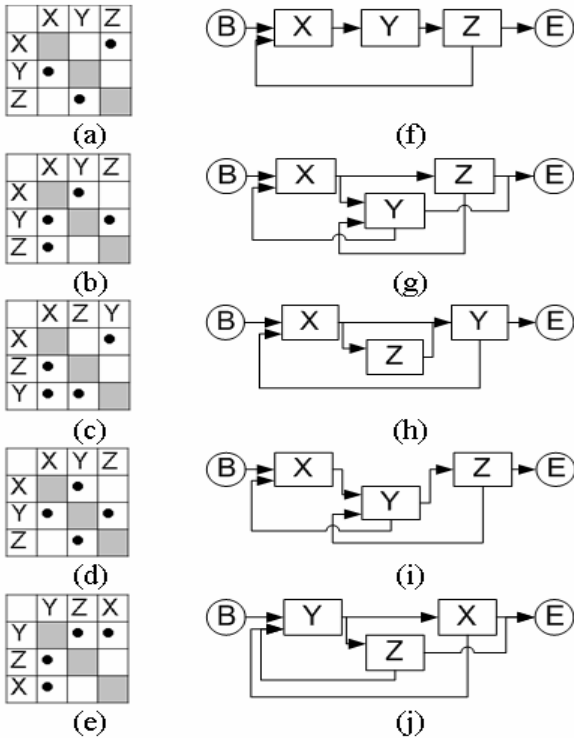


Fig. 2. Coupled activities form activity loops

additional reordering algorithms. Tearing [42] is a process of removing feedback marks. Tearing of a Component-based DSM may imply modularization or standardization of components [43]. In a parametric-based DSM tearing may imply confidence in the initial estimation of the parameter value [23]. Tearing algorithms are typically done manually based on the additional knowledge required to establish the meaning of removing the feedback.

Sequencing Optimization algorithms can be defined as algorithms that are used for both ordering coupled activities (in a manner similar to Tearing) and minimizing feedbacks (Partitioning). A detailed comparison of optimization objective functions of Partitioning and Sequencing algorithms is presented by Meier *et al.* [8]. Clustering algorithms can help identify clusters of highly connected components and are typically used in the context of Component-based DSM or Team-based DSM [28]. In Activity-based DSM context, several clustering algorithms were presented to improve work teams formation [44], [45], [7]. Another optimization algorithm that combines Clustering and Sequencing (i.e.,

Tearing and Partitioning) was developed by Karniel *et al.* [46].

3) *DSM marking*: Steward [11] used tick-marks ('x') to represent precedence relations, later, referred to as *Binary DSM* [5]. The DSM marking was extended to *Numeric DSM*, which represents a measure of the degree of relation or its importance ranking. Numeric DSM using Level numbers were proposed by Steward [42], for a manual Tearing algorithm. Dependency strength or Importance ranking of dependency degree can be used for sequencing optimization algorithms. Further refinement to *Probability DSM* was presented in [35].

### C. Process model verification

Process correctness verification is profoundly discussed in the Workflow literature [13], [18]. An established formal language model for process specification is Petri nets. Petri net is a bipartite directed graph with two node types called *places* and *transitions*; connected by directed edges (arcs). Places can contain *tokens*. A transition may transfer tokens from its input places to its output places according to *firing rules*. Process activities are modeled by transitions; places correspond to conditions. The distribution of tokens over the places is called *marking*, which represents the process state. Petri nets can be used as diagrammatic tool for modeling and analyzing distributed system behavior including sequential, conditional, parallel, and iterative routing [47].

WF nets (workflow nets), defined by Aalst [48] are a class of Petri nets, used for specifying the control aspect of workflow processes with defined starting and ending. Using the formal definitions of WF nets, one can set the required conditions for proper process specification (*Process Scheme*), using a correctness criteria named *Soundness*. WF net properties are [48]: 1) A WF net has a starting place and a final place. 2) All the activities and places are on a path from the starting place to the final place, i.e., there are no activities or conditions which do not contribute to process progress.

A proper process, defined according to the *Soundness* criteria, has the following properties. 1) From every process state (marking), which is reachable from the initial state (token at the starting place), there is a *firing sequence* (i.e., sequence of activities, could be parallel activities) that leads to the termination state. The process should terminate eventually. 2) Once the terminal state was reached there are no activities that did not perform yet; formally, there are no tokens in places other than the final place (i.e., a termination state). 3) There are no 'dead' activities (i.e., activities that could not perform due to unfulfilled pre-conditions).

Correctness criteria for Directed Acyclic Graph (DAG) are described in [49]. The processes should be free of Deadlocks and free of Lack of Synchronization. The first criterion is similar to the third requirement of Soundness; and 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. Aalst [50] presented the 'well-

handled' condition, which is the interpretation of both requirements, in terms of a WF net.

It should be noted that process characteristics that are considered non acceptable in the context of administrative process (typically addressed in Workflow literature), are very common in the Design Process context. Examples are: iterations (which solve the 'Dynamic change bug' [14]), and lack of synchronization.

### III. DSM-BASED SIMULATION - LOGIC VERIFICATION

Though DSM-based modeling helps in identifying iterative loops, and guides process planning; DSM models do not include the tasks duration, the impact of iteration or rework. Consequently the information presented in DSM is partial and insufficient for process plan evaluation. Therefore, simulation techniques are required for handling additional information such as: Time, Cost, Rework effort, or Risk. Most simulations address duration issues; resources issues were addressed by [51] and [52].

There are several issues related to assigning logic to the 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 DSM planning results in a logically correct process for specific DSM example; but changes in the DSM data might yield undetermined process scheme. Notwithstanding the importance of process logic verification, it is disregarded.

#### A. Process logic definition issues

As illustrated in the following examples, 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. Figure 1(c)). 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 Figure 2(c) (process in Figure 2(h)), activity  $Y$  should wait for  $X$  and for  $Z$  to complete. Join-Or logic may apply to multiple feedback links. In Figure 2(e), 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 (Figure 2(j)).

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 [6].

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 Figure 2(d) (process in Figure 2(i)), 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 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 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 [32], [33] 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 a Begin and End states, is implicitly supported by assuming them to be 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 [40]. 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 which can start are checked. Yet, termination of parallel activities is not explicitly defined. For certain DSM data the problem may not occur However, having different DSM data, or changing the DSM data may result in 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 [35] or the next activity according to DSM is enforced [40].

A potential logic problem of undetermined process scheme arises in the Signposting system [53]. A parameter value is the outcome of 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 linkages between required confidence levels define the precedence of tasks. The process serialization in that system has two underlying assumptions: 1) the linkages 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 Figure 3. Four coupled activities 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 3(a) presents the case of equal probabilities. Recalling that minimum iterations are 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.* [46]. In (b), the values of linkages from  $W$  to both  $X$  and  $Y$  were set to  $p=0.1$ . Setting both linkages from  $X$  and  $Y$  to  $Z$  as  $p=0.1$  would result in the DSM structure in (c)<sup>3</sup>. 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

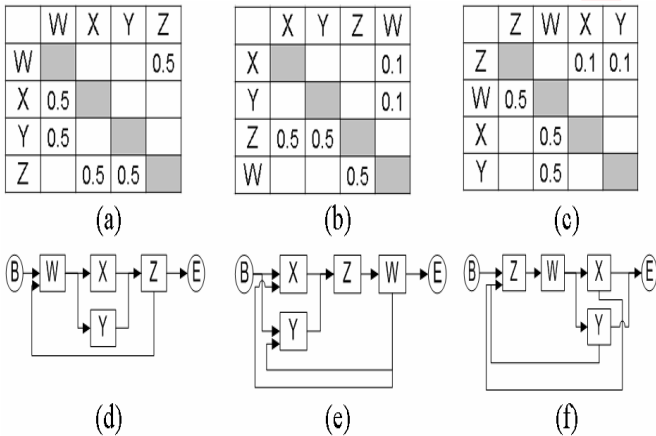


Fig. 3. Activity loops defined by DSM couple activities.

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, 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-

<sup>3</sup> Other results may occur by setting different values for the feedback and forward coefficients in [46].

And (otherwise the process may reach a termination state while not all activities have completed, see Figure 2(g) 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 Figure 3(d) to (f), respectively; and used for interpreting the process schemes in Figure 2(f) to (j).

### B. Presenting process logic in DSM

Presentation of process logic within the DSM is described in [25] and [28]. 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 demonstrates the inability of a single DSM to represent both input logic and output logic at the same time. Furthermore, Clarkson and Hamilton [25] 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 Figure 4(a). 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  $X$ ). When defining the symbols: Split ( $\Rightarrow$ ); Join ( $\Leftarrow$ ); And ( $\bullet$ ); and Or ( $+$ ); the above logic descriptions are formulated by symbolic

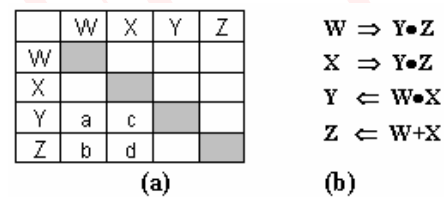


Fig. 4. Representing process logic by single DSM is inappropriate representation shown in Figure 4(b).

Actually there is no marking scheme that can convey such logic in one matrix. Since  $W$  and  $X$  have the same logic, the markings in their columns should be the same,  $a=b$  and  $c=d$ ; the input markings in the row of  $Y$  should differ from those in  $Z$ , i.e., the couples should be different  $\{a,c\} \neq \{b,d\}$ . This could be satisfied if  $a \neq b$  or  $c \neq d$ ; resulting in a contradiction.

Separating the logic presentation to two matrices, Input Logic and Output Logic will overcome the above problem. Yet, 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 Figure 2(c)). A DSM rearrangement might cause this logic to become a Join-And of a forward link and a feedback link (Figure 2(b)). In this case the resulting process may have a dead activity ( $Y$ ), as  $Z$  may not send a feedback signal, but send a link to End, see Figure 2(g).

Additional DSM limitation is described in [40]. The required logic 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.* [52] described a complex logic condition. The process described had design tasks and optimization scheduling activity. Assume that design task  $B$ , has a precedent design task  $A$ . 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 applies: If scheduling optimization activity  $S$  is performed after the completion of task  $A$ , then task  $B$  should wait for its completion; otherwise, task  $B$  may start once task  $A$  has completed. Definitely, such logic cannot be expressed in a DSM.

#### IV. CLASSIFICATION OF DSM BASED SIMULATIONS

The actual use of DSM values for simulation purposes has no broad agreement; different interpretations are used. Two differentiating parameters can be addressed: The information used, i.e., DSM type; and the procedure of choosing the next step: Simulation progress type. All DSM types were used: Binary DSM<sup>4</sup>; Probability DSM, where probability figures are used for simulation; and direct use of Numeric DSM with explicit number of iterations was proposed once [40].

The Work Transformation Matrix (WTM), first proposed by Smith and Eppinger [54], is used to calculate the remaining work to be done. Combination of Numeric and Probability DSM were used in [32], [33].

Using Probability DSM for estimating iterations probabilities has two main challenges: a) estimating the probability values; b) interpreting the figures at simulation time. The use of Probability DSM varies. Sered and Reich [43] used the values directly, indicating the probability to proceed from activity to another (either forward or feedback). Smith and Eppinger [35] used the feedback probabilities directly and have a more complex derivation of forward probabilities due to process serialization. Browning and Eppinger [6] 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 progressing simulation method and stochastic (Monte Carlo) choice of other simulation parameters. Additional parameters which may be added to the simulation for contributing 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

<sup>4</sup> 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.

effect of the changes in the process structure. The inclusion of such changes further complicates the system behavior because

TABLE I  
DSM BASED SIMULATIONS MAPPING

Author	Simulated Process type	Serialization \ Concurrency	DSM type
Abdelsalam and Bao [40]	Deterministic	Serialized	Numeric
Smith and Eppinger [54]	Deterministic	Fully Parallel (Coupled activities)	Numeric (to WTM)
Huberman and Wilkinson [55]	Deterministic	Fully Parallel (Coupled activities)	Numeric (to WTM)
Yassine et al. [56]	Deterministic	Fully Parallel (Coupled activities)	Numeric (to WTM)
Smith and Eppinger [35]	Markov chain	Serialized	Probability
Sered and Reich [43]	Markov chain Random walk	Serialized (Coupled activities)	Numeric to Probability
Melo and Clarkson [58]	Markov chain	Serial choice (Parallel)	Binary, Triangular
Coates et al. [52]	Actual performance (Monte Carlo equivalence)	Parallel	Binary, Triangular +Logic
Lévárdy and Browning [3]	Similar to Markov chain	Serial choice	Numeric (Symbols)
Browning and Eppinger [6]	Monte Carlo	Parallel	Probability
Yassine [23]	Monte Carlo	Parallel	Numeric to Probability
Cho and Eppinger [33]	Stochastic parallel discrete-event (Monte Carlo)	Parallel	Probability (+ Numeric overlap)

their effect is highly process scheme dependent [55].

Table I summarizes the classification of surveyed DSM-based simulation tools according to the above parameters. Some of the surveyed tools were described in many publications. The representative articles present the simulation assumptions. 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.

Abdelsalam and Bao [40] used explicit assignment of the required number of iterations. The main assumption is 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 (e.g., [54], [55], and [56]), referred to a case of a clique. In this case, all activities are coupled, 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.* [56] mentioned sub processes that may have internal order. Huberman and Wilkinson [55]

indicated that the analysis done may apply to each sub process 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 [56] or fluctuations in the interaction values [55]. In the latter case, diagonal elements are used to indicate the autonomous completion rate of the activity.

Smith and Eppinger [35] described a serialization of coupled activities. A Markov chain process is suggested by which the process can 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 [43] 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 [57] 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 process simulation. After each activity, the next one might be any linked activity either forward or feedback according to direct use of the assigned probabilities. The proceeding logic is therefore different from the one utilized in [54], where an activity can be activated only if its previous one has completed.

Melo and Clarkson [58] 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. Thus, 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.

In [3], the process progressed one activity at a time, through choosing the most applicable activity (including activity iterations) according to the target function. Activity iterations are 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, thus the process simulation is not fully equivalent to a parallel process simulation.

Coates *et al.* [52] described real time coordination and scheduling system, optimizing resource utilization. The resources perform computational analysis tasks, agent communication, resource management, and tasks management and optimization, in an integrated approach. The analysis tasks have triangular precedence interdependencies (i.e., lower diagonal 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 [6] defined the feedback probabilities (upper diagonal) logic as being first-order iteration, the forward probabilities (lower diagonal) 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 [23], for estimating the probability figures assigned to numeric values for validating the transformation between numeric to probability DSM.

Cho and Eppinger [32], [33] 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 (Learning curve, stochastic influences), and resources constraints, is used for process simulation and improving the process plan.

## V. LOGIC COMPARISON

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

The simulation logic of the above approaches is compared according to the following logic issues presented in section 3: Parallel Start (**PS**) indicating the initiation of parallel activities; Parallel End (**PE**) indicating the logic required for completing parallel activities (e.g., Figure 3(c)); Input Logic in case of Multiple Forward links (**IL MF**); Input Logic in case of Multiple Iteration<sup>5</sup> (feedback) links (**IL MI**); The separation of Forward links from Iteration (feedback) links at Input Logic (**IL F/I**) required for preventing deadlocks (cf. section 3.1); Output Logic in case of Multiple Forward links (**OL MF**); Output Logic in case of Multiple Iteration (feedback) links (**OL MI**); and Output Logic

<sup>5</sup> The term Iteration link temporarily replaces the term Feedback link, for distinguishing the initials.

TABLE II  
MAPPING THE DSM-BASED LOGIC

Author	PS	PE	IL MF	IL MI	IL F/I	IL formulation $IL \Leftarrow$	OL MF	OL MI	OL F/I	OL formulation $OL \Rightarrow$
Abdelsalam and Bao [40]	N/A	N/A	J-X	J-X	J-X	$(\otimes(F_i)) \oplus (\otimes(I_i))$	N/A <sup>(d)</sup>	S-X <sup>(f)</sup>	S-X	$(\otimes(I_i)) \oplus F_{next}$ <sup>(g)</sup>
Smith and Eppinger [54]; Huberman and Wilkinson [55]; Yassine <i>et al.</i> [56]	S-A	J-A	J-O	J-O	J-O	$(\Sigma(F_i)) + (\Sigma(I_i))$	S-A	S-A	S-A	$(\Pi(I_i)) \bullet (\Pi(F_i))$
Smith and Eppinger [35]	N/A	N/A	J-X	J-X	J-X	$(\otimes(F_i)) \oplus (\otimes(I_i))$	S-X <sup>(e)</sup>	S-X	S-X	$(\otimes(I_i)) \oplus (\otimes(F_i)) \oplus F_{ready}$ <sup>(h)</sup>
Sered and Reich [43]	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 [58]	Imp <sup>(a)</sup>	UD <sup>(b)</sup>	J-A	N/A <sup>(c)</sup>	N/A <sup>(c)</sup>	$\Pi(F_i)$	S-A	N/A <sup>(c)</sup>	N/A <sup>(c)</sup>	$\Pi(F_i)$
Coates <i>et al.</i> [52]	S-A	J-A	J-A	N/A <sup>(c)</sup>	N/A <sup>(c)</sup>	$\Pi(F_i)$	S-A	N/A <sup>(c)</sup>	N/A <sup>(c)</sup>	$\Pi(F_i)$
Lévárdy and Browning [3]	Imp <sup>(a)</sup>	UD <sup>(b)</sup>	J-A	J-O	J-X	$(\Pi(F_i)) \oplus (\Sigma(I_i))$	S-A	S-X	S-X	$(\otimes(I_i)) \oplus (\Pi(F_i))$
Browning and Eppinger [6]; Yassine [23]	S-A	Imp	J-A	J-O	J-X	$(\Pi(F_i)) \oplus (\Sigma(I_i))$	S-A	S-O	S-X	$(\otimes(I_i)) \oplus (\Pi(F_i))$
Cho and Eppinger [33]	S-A	J-A	J-A	J-O	J-O	$(\Pi(F_i)) + (\Sigma(I_i))$	S-A	S-O	S-X	$(\Sigma(I_i)) \oplus (\Pi(F_i))$

Notation: 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; UD: undetermined;

Logic formulation:  $F_i$ : forward links;  $I_i$ : iteration (feedback) links; Split ( $\Rightarrow$ ); Join ( $\Leftarrow$ ); + (OR);  $\bullet$  (AND);  $\oplus$  (XOR);

$$\Sigma(A_i) = A_1 + A_2 + \dots + A_n \quad \Pi(A_i) = A_1 \bullet A_2 \bullet \dots \bullet A_n \quad \otimes(A_i) = A_1 \oplus A_2 \oplus \dots \oplus A_n$$

- a Parallel activities are implicitly started in parallel, using serialized check
- b Parallel completion is undefined. Problem could be avoided by linking parallel activities to the last activity
- c No iteration (feedback) links
- d Next activity only (other potential activities with forward links are disregarded)
- e Forward links are available at stages; e.g., Figure 2(e),  $X$  becomes ready for advancing from  $Y$  only after  $Z$  has completed
- f The case of multiple iterations is not strictly defined, thus the simulation might be undetermined in such case
- g The process may forward only to next activity ( $F_{next}$ )
- h The process may forward to the ready activity ( $F_{ready}$ ) (see e), or any activity which has already completed once, or iterate

separation of Forward and Iteration (Feedback) links (**OL 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);  $\bullet$  (AND);  $\oplus$  (XOR, Exclusive-Or); the short form of multi-variable logic operations: Multiple Or:  $\Sigma(A_i) = A_1 + A_2 + \dots + A_n$ ; Multiple And:  $\Pi(A_i) = A_1 \bullet A_2 \bullet \dots \bullet A_n$ ; and Multiple Xor:  $\otimes(A_i) = A_1 \oplus A_2 \oplus \dots \oplus A_n$ .  $A_i$  represents a link signal, which can be a forward link  $F_i$  or Iteration (feedback) link  $I_i$ .

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.

For example,  $IL \Leftarrow (\Pi(F_i)) \oplus (\Sigma(I_i))$  [4], [7], 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 [6], 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. In [33], an activity can get signals both from upstream or downstream activities. A second forward link signal to  $B$  may follow an iteration of  $A$ .  $IL \Leftarrow (\Pi(F_i)) + (\Sigma(I_i))$

The output logic indicates a decision process, i.e., sending a signal that the current activity has completed. For example,  $OL \Rightarrow (\Sigma(I_i)) \oplus (\Pi(F_i))$  in [6] and [33] implies that a signal is either sent through 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).

## VI. DISCUSSION

A symbolic formulation was defined for describing different logic types and defining logic requirements that cannot be expressed by the DSM structure. This formulation can describe the logic of control signals; therefore it is effective in comparing the different logic used in different DSM studies. However describing complex logic is more demanding: in [52] a dynamic DSM definitions at run time is required (i.e., different schemes if optimization activity is executed or not); in [6] the logic changes between the first execution and following iterations of the design activity (i.e. dependent on run time status).

The examples in Table II are roughly ordered in an increasing potential of Run Time processes variety. The process logic circumscribes the diversity of process schemes under the assumption of stochastic activity duration, i.e., how many different process schemes could be created at run time, using the defined logic, for a given DSM. The deterministic definition in [40] would always yield the same run time process scheme (same order of activities), for a given DSM. The logic defined in [33] would yield many different run time 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.

However, this 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 [54], [55], [56], implies that all activities start together and the process completes once all have completed. Such logic is unacceptable once there is a defined serialization of the coupled activities, e.g., in the case of testing activity [3], or specific serialization requirement [40]. 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.

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 to the end 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.

## VII. CONCLUSION

This article focused on utilizing DSM for planning Product Design Processes. The comparative study reveals that the interpretation of DSM results is not unique. Moreover, there is no single or best interpretation as different interpretations may be applicable for describing different business processes. Unlike the workflow literature which emphasizes the need for process verification, in the DSM literature, process logic is defined, but is not verified. In some cases this may lead to process schemes that might be applicable only to a limited set of DSM cases.

The workflow literature typically assumes process scheme being defined by process experts. The definition requires extensive verification to assure its correctness. Using DSM reordering techniques may have appealing characteristics for process definitions, e.g., the automatic distinction of forward and feedback links in complex cases

(such as iterative design processes). Further research is required to bridge the gap between DSM planning and Process definition and verification; especially once dynamic processes, based on dynamic changes in DSM data, are to be considered.

## REFERENCES

- [1] K. Ulrich and S.D. Eppinger, *Product Design and Development*, 2nd ed. New York: McGraw-Hill, 2000.
- [2] Kusiak A., "Integrated product and process design," *J. Eng. Design*, vol. 13, no. 3, pp. 223-231, 2002.
- [3] V. Lévárdy, T.R. Browning, "Adaptive test process – designing a project plan that adapts to the state of a project," *15th Annu. Int. Symp. of the Int. Council On Sys. Eng. (INCOSE)*, July 2005.
- [4] B. Westfechtel, "Models and tools for managing development processes," *Lecture Notes in Computer Science*, vol. 1646, 1999.
- [5] S.D. Eppinger, D.E. Whitney, R. Smith, and D. Gebala, "A model-based method for organizing tasks in product development," *Res. Eng. Design*, vol. 6, no. 1, pp. 1-13, 1994.
- [6] T.R. Browning, and S.D. Eppinger, "Modeling impacts of process architecture on cost and schedule risk in product development," *IEEE Trans. Eng. Manage.*, vol. 49, no. 4, pp. 443-458, 2002.
- [7] R.I. Whitfield, A.H.B. Duffy, L.K. Gartzia-Etxabe., "Identifying and evaluating parallel design activities using the design structure matrix," *Int. Conf. Eng Design, ICED 05*, Melbourne, Aug. 2005.
- [8] C. Meier, A. Yassine, T. Browning, "Design process sequencing with competent genetic algorithms," *PDRL working paper # PDL-2006-03*, Mar. 2006.
- [9] C.M. Eckert, R. Keller, C. Earl, and P. J. Clarkson, "Supporting change processes in design: Complexity, prediction and reliability," *Reliability Eng. and System Safety*, 2006, to be published.
- [10] M. Weske, W.M.P. van der Aalst, H.M.W. Verbeek, "Advances in business process management," *Data and Knowledge Engineering*, vol. 50, no. 1, pp.1-8, 2004.
- [11] D.V. Steward, *Systems Analysis and Management: Structure, Strategy, and Design*, Petrocchi Books, NJ, 1981.
- [12] S. Rinderle, M. Reichert, P. Dadam, "Flexible support of team processes by adaptive workflow systems," *Distributed and Parallel Databases*, vol. 16, no. 1, pp. 91-116, July 2004.
- [13] S. Sadiq, M.E. Orłowska, W. Sadiq, C. Foulger, "Data flow and validation in workflow modeling," *Proc. Int. Conf. in Research and Practice in Information Technology, ADC'2004*, Dunedin, 2004.
- [14] W.M.P. van der Aalst, "Exterminating the dynamic change bug: a concrete approach to support workflow change," *Inf. Sys. Frontiers*, vol. 3, no. 3, pp. 297-317, 2001.
- [15] WFMC, "Workflow management coalition terminology and glossary," *Tech. report WFMC-TC-1011*, issue 3, Workflow Management Coalition, 1999.
- [16] H.M.W. Verbeek, W.M.P. van der Aalst, "XRL/Woflan: Verification and extensibility of an xml/petri-net-based language for inter-organizational workflows," *Inf. Tech. and Manage.*, vol. 5, pp. 65-110, 2004.
- [17] S. Rinderle, B. Weber, M. Reichert, W. Wild, "Integrating process learning and process evolution – A semantics based approach," in W.M.P. van der Aalst et al., Eds.: *BPM 2005, Lecture Notes in Computer Science*, vol. 3649, pp. 252-267, Springer-Verlag, 2005.
- [18] S. Rinderle, M. Reichert, P. Dadam, "Correctness criteria for dynamic changes in workflow systems - A survey," *Data and Knowledge Engineering*, vol. 50, no. 1, pp. 9-34, 2004.
- [19] P. Mangan, S. Sadiq, "A constraint specification approach to building flexible workflows," *J. Res. and Practice in Inf. Tech.*, vol. 35, no. 1, pp. 21-39, 2003.
- [20] M. Adams, D. Edmond and A.H.M ter Hofstede, "The application of activity theory to dynamic workflow adaptation issues," *7th Pacific Asia Conf. on Inf. Sys.*, Adelaide, South Australia, July. 2003.
- [21] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, A.P. Barros, "Workflow patterns," *Distributed and Parallel Databases*, vol. 14, no. 1, pp. 5-51, 2003.
- [22] V. Lévárdy, M. Hoppe, T.R. Browning, "Adaptive test process – An integrated modeling approach for test and design activities," in *The Product Development Process, Proc. of DETC'04ASME 2004 Design*

- Eng. Tech. Conf. and Computers and Inf. in Eng. Conf.*, Salt Lake City, Utah, September 28 - October 2, 2004.
- [23] A. Yassine, "Investigating product development process reliability and robustness using simulation," *PDRL working paper # PDL-2005-01*, Jan, 2005. *J. Eng. Design*, 2006, to be published.
- [24] T.R. Browning, E. Fricke, and H. Negele, "Key concepts in modeling product development processes," *Sys. Eng.*, vol. 9, no. 2, pp. 104-128, Wiley, 2006.
- [25] P.J. Clarkson, and J.R. Hamilton, "Signposting, A Parameter-driven Task-based Model of the Design Process," *Res. Eng. Design*, vol. 12, no. 1, pp. 18-38, 2000.
- [26] T.L. Flanagan, C.M. Eckert, R. Keller, P.J. Clarkson, "Bridging the gaps between project plans and reality: The role of overview," in *Proc. of Tools and Methods of Competitive Eng. (TMCE 2006)*, vol. 1, pp. 105-116, Ljubljana, Slovenia, April 2006.
- [27] R. Keller, T.L. Flanagan, C.M. Eckert, P.J. Clarkson, "two sides of the story: visualising products and processes in engineering design," in *Proc. of the 10th Intl. Conf. on Information Visualisation (IV 2006)*, IEEE Computer Society, London, July 2006.
- [28] T.R. Browning, "Applying the design structure matrix system to decomposition and integration problems: A review and new directions," *IEEE Trans. Eng. Manage.*, vol. 48, no. 3, pp. 292-306, 2001.
- [29] S.D. Eppinger, V. Salminen, "Patterns of product development interactions," *International Conf. on Eng. Design, ICED 01*, Glasgow, August 21-23, 2001.
- [30] DSM web site at MIT. <http://web.mit.edu/dsm>, accessed on September 1, 2005.
- [31] A. Yassine, D. Braha, "Complex concurrent engineering and the design structure matrix method," *Concurrent Eng. Res. and Applications*, vol. 11, no. 3, pp. 165-176, 2003.
- [32] S.H. Cho, S.D. Eppinger, "Product development process modeling using advanced simulation," *ASME Conf. on Design Theory And Methodology (DECT 2001/DTM)*, Pittsburgh, PA, September 2001.
- [33] S.H. Cho, S.D. Eppinger, "A simulation-based process model for managing complex design projects," *IEEE Trans. Eng. Manage.*, vol. 52, no. 3, pp. 316-328, 2005.
- [34] A. Kusiak, J.R. Wang, D.W. He, C.-H. Feng, "A structured approach for analysis of design processes," *IEEE Trans. on Components, Packaging, and Manufacturing Technology - Part A*, vol. 18, no. 3, pp. 664-673, 1995.
- [35] R.P. Smith, S.D. Eppinger, "A predictive model of sequential iteration in engineering design", *Manage. Sci.*, vol. 43, no. 8, pp. 1104-1120, 1997.
- [36] D.A. Gebala, S.D. Eppinger, "Methods for analyzing design procedures," *ASME 3rd Int. Conf. Design Theory and Methodology*, 1991.
- [37] J.N. Warfield, "Binary matrices in system modeling," *IEEE Trans. Syst., Man and Cyber.*, vol. 3, pp. 441-449, 1973.
- [38] A. Kusiak, N. Larson, J. Wang, "Reengineering of design and manufacturing processes," *Computers and Industrial Eng.*, vol. 26, no. 3, pp. 521-536, 1994.
- [39] W. P. Ledet, D.M. Himmelblau, "Decomposition procedures for the solving of large scale systems," *Advances Chemical Eng.*, vol. 8, pp. 185-254, 1970.
- [40] H.M.E. Abdelsalam, H.P. Bao, "A simulation-based optimization framework for product development cycle time reduction," *IEEE Trans. Eng. Manage.*, vol. 53, no. 1, pp. 69-85, 2006.
- [41] A. Yassine, D. Whitney, J. Lavine, T. Zambito, "Do-it-right-first-time (DRFT) approach to design structure matrix restructuring," *Proc. of the 12th International Conf. on Design Theory and Methodology*, Baltimore, Maryland, USA, Sept. 10-13, 2000.
- [42] D. V. Steward, "The design structure system: A method for managing the design of complex systems," *IEEE Trans. Eng. Manage.*, vol. 28, pp. 71-74, 1981.
- [43] Y. Sered, Y. Reich, "Standardization and modularization driven by minimizing overall process effort," *Computer-Aided Design*, vol. 38, no. 5, pp. 405-416, 2006.
- [44] C. Fernandez, "Integration analysis of product architecture to support effective team co-location," *SM Thesis*, Massachusetts Institute of Technology, 1998.
- [45] D. Sharman, A. Yassine, "Architectural valuation & optimization using the dependency structure matrix and real options," *PDRL working paper # PDL-2006-05*, Univ. of Illinois at Urbana-Champaign, May 2006.
- [46] A. Karniel, Y. Belsky, Y. Reich, "Decomposing the problem of constrained surface fitting in reverse engineering," *Computer-Aided Design*, vol. 37, pp. 399-417, 2005.
- [47] W.M.P. van der Aalst, K.M. van Hee, *Workflow Management: Models, Methods, and Systems*, MIT Press, Cambridge, MA, 2002.
- [48] W.M.P. van der Aalst, "The application of Petri nets to workflow management," *The Journal of Circuits, Systems and Computers*, vol. 8, no. 1, pp. 21-66, 1998.
- [49] W. Sadiq, M. E. Orłowska, "Applying graph reduction techniques for identifying structural conflicts in process models," *Lecture Notes in Computer Science*, vol. 1626, pp.195-209, 1999.
- [50] W.M.P. van der Aalst, "Finding control-flow errors using Petri net based techniques," in *W.v.d. Aalst. J. Desel and Oberweis A., Eds., Business Process Management, Lecture Notes in Computer Science*, vol. 1806, pp. 161-183, Springer-Verlag, 2000.
- [51] B.D. O'Donovan, P.J. Clarkson, C.M. Eckert, "Signposting: Modeling uncertainty in design processes," *Intl. Conf. on Eng. Design (ICED 03)*, Stockholm, August 19-21, 2003
- [52] G. Coates, A.H.B. Duffy, I. Whitfield, W. Hills, "An integrated agent-oriented approach to real-time operational design coordination," *Artificial Intelligence for Eng. Design, Analysis and Manufacturing*, vol. 17, pp. 287-311, 2003.
- [53] P.J. Clarkson, A.F. Melo, C.M. Eckert, "Visualization of routes in design process planning," in *Proc. of the Fourth International Conf. on Information Visualisation (IV2000)*, IEEE Computer Society, pp. 155-164, London, 2000.
- [54] R.P. Smith, S.D. Eppinger, "Identifying controlling features of engineering design iteration," *Manage. Sci.*, vol. 43, no. 3, pp. 276-293, 1997.
- [55] B. A. Huberman, D.M. Wilkinson, "Performance variability and project dynamics," *Computational & Mathematical Organization Theory*, vol. 11, pp. 307-332, 2005.
- [56] A. Yassine, N. Joglekar, D. Braha, S.D. Eppinger, D. Whitney, "Information hiding in product development: The design churn effect," *Res. Eng. Design*, vol. 14, no. 3, pp. 131-144, 2003.
- [57] J.L. Rogers, C.L. Bloebaum, "Ordering design tasks based on coupling strengths," *AIAA*, paper no. 94-4326, 1994.
- [58] A.F. Melo, P.J. Clarkson, "Design process planning using a state-action model," in *13th Intl. Conf. Eng. Design (ICED 01)*, Glasgow, 2001.



Arie Karniel is a PhD student in the Faculty of Engineering, Tel Aviv University, Israel. He received his BSc and MSc degrees in Mechanical Engineering from the Technion, Israel, in 1989 and 1991 respectively. He received his MBA, information systems, from Tel Aviv University in 1997. Arie has practiced engineering design, product management, and system engineering in aerospace, mechatronics, and software industries. His research interests include: design processes, collaborative design, reverse engineering, workflow management, product life cycle management, and engineering knowledge management.



Yoram Reich is an Associate Professor in the Faculty of Engineering, Tel Aviv University, Israel. He received his BSc (Summa Cum Laude) and MSc (Magna Cum Laude) in Mechanical Engineering from Tel Aviv University in 1980 and 1984, respectively. Before obtaining the PhD degree in Civil Engineering from Carnegie Mellon University in 1991, he practiced engineering design for over 7 years in the audio, structures, and marine industries. Yoram Reich has authored or co-authored over 150 papers and is a member of the editorial boards of the journals *Advanced Engineering Informatics*, *International Journal of Mass customization*, *Research in Engineering Design*, and *Journal of Engineering Design*. His research focuses on several interrelated topics: product design, conceptual design, collaborative design, knowledge management, and decision support systems.