Proceedings of DETC'03
ASME 2003 Design Engineering Technical Conferences and Computers and Information in Engineering Conference
Chicago, Illinois USA, September 2-6, 2003

DETC2003/DTM-48662

ITERATION IN ENGINEERING DESIGN: INHERENT AND UNAVOIDABLE OR PRODUCT OF CHOICES MADE?

Ramon Costa, Durward K. Sobek II
Montana State University
Mechanical & Industrial Engineering Dept.
Bozeman, Montana, 59717-3800
Tel: 406 994 7140
Fax: 406 994 6292
ramoncosta@montana.edu, dsobek@ie.montana.edu

ABSTRACT

Iteration in design has different meanings, ranging from simple task repetition to heuristic reasoning processes. Determining the need to iterate is important to improve the design process on cost, time, and quality, but currently there is no categorization of iterations conducive to this goal. After exploring the possible causes and attempts to address them, we propose to classify iterations as rework, design, or behavioral. This framework suggests that design teams should try to eliminate rework iterations, perform design iterations without skipping abstraction levels, and do behavioral iterations in parallel.

Keywords: Engineering Design, Iteration, New Product Development, Design Process Attributes, Set-based Concurrent Engineering.

INTRODUCTION

Iteration is a term frequently used to describe design. It is commonly accepted that design is iterative in nature, but what do we mean by iteration? And if we are not completely sure of what iteration is, could we iterate more than needed? Can we reduce engineering time by identifying unnecessary iterations and eliminating their root cause, or by performing some iterations in parallel rather than sequentially? Because iterations shape the outcomes of design in terms of cost, time, and quality, exploring the nature of iteration and identifying its causes and proposed countermeasures may help improve design methods. Terwiesch, Loch, and De Meyer [1] compared iterative approaches to design with set-based concurrent engineering approaches [2], and concluded that iterative approaches tend to increase both development lead time and cost (as tooling rework and engineering hours increase). Krishnan, Eppinger, and Whitney [3] found that sequential decision-making degrades solution quality because previous decisions constrain future decisions and limit a person's ability to make a decision that meets criteria (in the extreme case sequential decision-making might make it impossible to meet the design criteria). To prevent degrading design quality, Krishnan et al. propose to repeat activities or to decide non-sequentially.

After reviewing definitions and causes of iteration from the literature we conclude that a need exists for a new categorization of iterations that helps answer the question of need. Is all iteration inevitable, or can some of it be avoided using different design strategies? Does the nature of different instances of iteration suggest new and more effective approaches to design problems? For this purpose, we developed a framework that classifies iteration as rework, design, or behavioral iteration.

In the pages that follow, we briefly summarize current definitions of iteration extant in the design theory literature, and then present our framework. Ensuing is a discussion of key root causes of iteration, approaches to minimize the negative impact of iteration, and how our proposed framework can potentially address some the issues raised. We conclude with hypotheses to be tested in future research.

DEFINITIONS OF ITERATION

A common approach in the literature is to consider iteration as repeating design activity. For example, Ulrich and Eppinger [4] formally define iteration as repeating an already completed task to incorporate new information (such as performing finite element analysis followed by design revision and repeating the analysis incorporating feedback from the revision). A second approach defines iterations more broadly

as a heuristic reasoning process. Adams and Atman [5], for example, describe a broad cycle of gathering information, processing that information, identifying possible design revisions, and executing those revisions in pursuit of a goal. This perspective of iteration is less concerned with repeating design activity, and more focused on the thought process that justifies the need to perform the activity in the first place. Further expanding this definition, Ullman, Wood, and Craig (as cited in [6]), define iterations as the cognitive processes that the designer uses when performing activities that change the design state. Emphasis is on how the designer reasons about the design as opposed to how she performs specific activities. Thus definitions of iteration focus on designer behavior, and range from the repetition of activities to more abstract patterns of designer behavior.

Based on these definitions researchers have developed taxonomies for iteration to help better understand the nature of the phenomenon. For instance, in conjunction with the designer behavioral definitions of iteration, Adams and Atman [6] differentiate between diagnostic iterations that define and evaluate design tasks, and transformative iterations that synthesize new information. Other approaches classify iterations based on information/task interdependence, such as Smith and Eppinger's sequential [7] versus parallel iterations [8], or Denker, Steward, and Browning's [9] interdependent task cycles.

Applicability to the Question of Need

The definitions undoubtedly help understand different aspects related to iteration. But, at the same time, they also increase the meanings associated with the term and potentially transform it into an umbrella for all necessary and unnecessary, value-adding and wasteful, iterations. Further, because these definitions and categories were attempting to answer other questions, they complement the image of a particular occurrence of iteration but do not seem conclusive on the need to iterate.

A common implicit assumption in these definitions is that iteration is unavoidable and should be planned into design processes even though there are some drawbacks, such as increased development time. Iteration, it seems, is seen as a necessary evil in design and is not challenged. Such a perspective may systematize inefficiencies associated with unnecessary iterations.

An important assumption made in some definitions is sequential decision-making [7-9]. For instance, parallel iterations do not indicate simultaneous iterations, but rather simultaneous execution of interdependent or dependent tasks that influence each other's inputs and thus generate task repetition. Both the sequential and parallel categories assume sequential decision-making. In dealing with information interdependency, Denker et al. [9] propose breaking the interdependency cycle by assuming a value for a parameter and then sequentially repeating the dependent tasks alternately until reaching a final value for the different task outputs. It is important to note this since sequential decision-making is one of the key inducers to iteration [3].

Proposed Framework

Our research on engineering design identifies design process attributes along two dimensions, activities and abstraction levels [10]. We categorize *activities* into four areas: problem definition, idea generation, engineering analysis, and design refinement. And we distinguish three *levels of abstraction*, namely concept, system, and detail, to indicate the progression of design work from ambiguous to specific, with a middle step focused on system architecture. In addition to activity and abstraction level, we've begun to also include a third process attribute, *scope*. Within this framework, scope refers to the (sub)problem at which a given design activity is targeted (e.g. the designer might be working on concept design for overall problem, or the concept design of a subsystem). Table 1 summarizes activities, abstraction levels, and scope.

Table 1: Activity, Abstraction Level, and Scope

Activity
Problem Definition
Idea Generation
Engineering Analysis
Design Refinement
Abstraction Level
Concept
System
Detail
Scope
Defined by product architecture and design requirements

To illustrate the three design process attributes, consider a design team meeting to brainstorm ideas for the communications network of a motor vehicle's electrical system (e.g., using a Controller Area Network or a Local Interconnect Network). In our framework, we would classify the activity as idea generation and the abstraction level as concept, and label the scope as communications network.

Now, if we consider that repetition of an activity is the basis for defining iteration, it is possible to develop three iteration types based on whether the abstraction level or scope change, as shown in Table 2.

When the designer repeats the activity at the same abstraction level on the same object it is probably *rework iteration* –the three process attributes remain the same when the design task is repeated. The most common reason for repeating an activity at the same scope and abstraction level is to correct an error. For instance, when in 1996 the European Space Agency reused software from Ariane 4 in the Ariane 5 project, an undetected design error resulted in the loss of the rocket plus the half a dozen commercial communications satellites it was supposed to set in orbit [11]. The design work to replace the rocket would be considered rework iteration because neither scope (same mission) nor abstraction level (a working rocket) has changed, yet the design task was repeated.

Table 2: Iteration type defined by design process attributes compared to first task performance.

Iteration Type	*Design Process Attribute*		
	Activity	**Abstraction Level**	**Scope**
Rework	Repeats	Repeats	Repeats
Design	Repeats	Changes	Repeats
Behavioral	Repeats	Repeats	Changes

Rework iteration should change the design state in the same manner in the second task execution as it did in the first, so if a designer executes the activities correctly, there is no need for this type of iteration. Rework iteration, then, should receive the same treatment as errors that produce defects in a manufacturing environment would receive: their causes should be prevented or detected at the source so errors do not become defects, or in this case iterations that cost money.

However, if the activity's abstraction level is not the same as in the first execution, then we consider this a sign of *design iteration*, because as design activity progresses through abstraction levels [5] the design evolves toward the desired final state. As an analogy, consider the problem of finding directions to a specific address in Los Angeles. Consulting a map of the U.S. will provide information on L.A.'s location, but is not very helpful for finding directions within the city. Repeating the activity (looking for directions) at a lower abstraction level, or from a lower height for this analogy, will provide different information useful to find the address of interest. An L.A. street map will ultimately help reach the address, but it will be of no use if we do not know where L.A. is. Similarly, design iteration repeats activities to generate meaningful information at different abstraction levels. Design iteration indicates the designer is progressing through different abstraction levels, defining and refining a solution while moving from the initial concepts to a detailed final design. The activities repeat on the same scope, but task content differs significantly as it carries the solution to a different resolution level.

Finally, *behavioral iteration* means proceeding through the same activity at the same abstraction level, but applied to a different scope. In other words, the behavior repeats but on a different (sub)problem. Designers often divide a problem into pieces and proceed with a similar pattern of design activity performed on each of the subdivisions as object. For example, if a team works on the design of a vehicle's power train system while a different team develops the air conditioning system, the teams may perform similar design activities but on different scopes.

To illustrate the potential usefulness of this framework, consider a common occurrence in product development: changing customer requirements [12]. Product developers often assume that customers know exactly what they need, which creates the notion that design is searching for the right solution for the perceived customer need. Though changing requirements is quite common, planners often do not consider it a likely event and therefore do not plan for it, so that when product specifications change, customers are blamed for triggering rework. But the proposed framework helps us think about it differently.

Imagine a designer presenting the customer with a detailed design of a multipurpose knife that satisfies the customer's need as the designer perceived it, with the unstated but very real hope that the customer will accept it as-is. After reviewing the design, the customer points out that, against requirement, the blades bind when more than one tool is already unfolded. The observation results in a rework iteration to address the lack of compliance with requirements. In contrast, consider the same situation (designer presenting the customer with the newly designed multipurpose knife) but now, after seeing what a possible solution looks like, the customer decides that so much functionality is confusing and sends the designer back with a shorter list of functional requirements and an additional requirement for a more user-friendly folding mechanism to prevent injury when transitioning between tools. The designer goes back to work thinking the customer holds back information just to make life miserable, without realizing that maybe the customer did not understand the implications of the design requirements until exposed to a possible solution. The designer has to iterate but under different circumstances: in the second scenario the scope has changed and triggered a behavioral iteration.

The process of generating solutions and sharing them with the customers can trigger the redefinition of the problem [13], which will drive the designer to accommodate the new information by repeating the same tasks at their corresponding abstraction level on the modified problem definition. It is important to differentiate between the two instances of iteration because we have no control over the customer changing requirements upon gaining new information (i.e., about the solution space), but we can avoid faults in our own design process. Failure to include information that constrains the design and is available to the designer does not trigger behavioral iteration because it is neither a subdivision of the design space nor a requirement change. For example, ignoring restrictions on materials use (e.g., lead or asbestos) and having to repeat design tasks to include this information falls in the category of rework iteration, not of behavioral iteration. This differentiation is often used in product development to define responsibility for the increase in development or product costs due to design changes – negligence comes at a price.

The proposed framework helps identify an alternative approach to working with customers. Instead of presenting a single, near-complete design, showing the customer alternative designs at a higher abstraction level earlier in the process might help accelerate problem and solution definition, with less disruption. This approach incorporates design iteration combined with parallel execution of behavioral iteration at a given abstraction level.

This classification of iteration also helps identify necessary iterations in design. Based on the assumption of repeating an activity, rework iteration does not move the design to an increased state of completion because there is no evolution on either abstraction level or scope, and thus might not be necessary. Design iteration represents design evolution towards completion through abstraction level, while behavioral iteration portrays this evolution across scope. Design and behavioral iteration are performed in tandem to evolve the design to its final state, as shown in Figure 1.

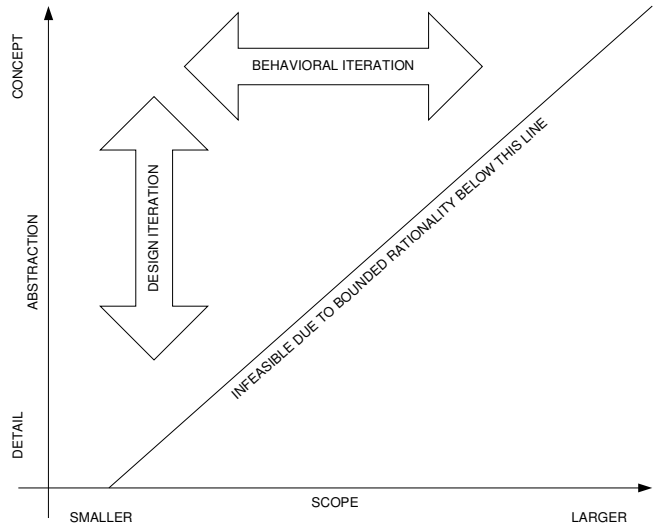

Figure 1: Direction of design and behavioral iteration on abstraction – scope plane

CAUSES OF ITERATION

It is necessary to identify the causes of iteration because while categorizing helps understand the phenomena it does little in directing action to improve the design process. There are two approaches to consider: information characteristics and design process. Certain characteristics of information have been found to influence iteration, such as new information that surfaces during design [4], stability or probability to change and information precision [1], information interdependency that results in interdependent tasks [9], information overload due to human bounded rationality or cognitive limitations that do not allow a designer to consider all design details at once [14], and finally information that is wrong due to design error. The last characteristic is the only one triggering rework iteration.

The causes for the other two types of iteration, design and behavioral, also have their roots in information characteristics. In particular both iteration types can be seen as strategies to deal with information overload. The cognitive limitations that do not allow human designers to process all relevant information at all abstraction levels motivates design iteration – one cannot find out how to get to L.A. and to the address within it at the same time. As Ullman (as quoted in [15]) indicates, it is necessary to develop solutions at different abstraction levels because the alternative solutions are too complex for human cognition to cope with at a detail level. Designers often differentiate the design scope to make the design manageable by breaking it up into sub-problems and then performing behavioral iteration on each of them. Problem and solution co-evolution that triggers changes in the definition of the scope, and its corresponding behavioral iteration, seem to also derive from bounded rationality.

The literature identifies information characteristics as either important to understand iteration or directly as its cause, but the design process can still influence iteration by shaping the nature of the information. Some design processes might create an environment conducive to behavioral iteration because they impose a sequential decision-making approach, as in communicating information that is precise but unstable [3]. Or, the design team may take a depth-first approach characterized by diving into the details of the design early in the process and resolving interface issues as they arise at the detailed level [15]. Behavioral iteration might also occur by not developing the system's architecture before diving into details, or by planning design activities without taking into account information dependencies and interdependencies.

MINIMIZING THE IMPACT OF ITERATION

To counterbalance the effects of unstable information, Krishnan, Eppinger, and Whitney [16] define iterative overlapping as the problem of interchanging preliminary information to reduce lead time. To perform interdependent tasks simultaneously, the authors suggest starting with value intervals (less precise, more stable information [1]) and iteratively narrow the range until reaching a final value. Krishnan et al. [3] present two characteristics of design parameters to be taken into account to reduce iteration: task and sequence invariance. The objective is to iterate on parameters that are dependent on both the task performed and task sequence.

Krishnan et al. [3] suggest an alternative countermeasure to iteration in dealing with loss of quality due to sequential decision-making, which is to decide in a way other than sequentially. There are several process countermeasures that deal directly with the assumption of sequential decision-making.

Sobek, Ward, and Liker [2] argue that Toyota's development system follows three principles that differentiate it from some of its competitors: they map the design space,

integrate by interception of feasible spaces, and establish feasibility before commitment. Based on these principles, set-based concurrent engineering (SBCE) breaks from the paradigm of deciding sequentially in a point-to-point solution search. In SBCE, engineers refrain from making design decisions at a detail level before exploring alternatives at higher abstraction levels to determine whether the solutions are feasible when considered against downstream task constraints [19]. This evolution along abstraction levels to determine solution feasibility seems to indicate design iteration. At the same time, SBCE also involves behavioral iteration in that the sets of alternatives allow for the simultaneous evaluation of a range of problem definitions, which with our framework we could potentially identify as sets of scopes. The authors also indicate that a set-based approach reduces backtracking the design, i.e., rework iteration.

Similarly, Ball, Evans, and Dennis [15] argue that taking a breadth-first approach, in which we perform all design activity at all abstraction levels (design iteration with behavioral iteration at each abstraction level), reduces rework iteration because it allows for a better exploration of interactions at higher abstraction levels (Anderson, as quoted in [15]). The breadth-first approach performs behavioral iteration at different abstraction levels considering scope only as the subdivision of the design into modules. Both SBCE and breadth-first approach involve the execution of behavioral and design iterations, avoiding sequential decision-making that leads to performing behavioral iterations sequentially.

Another approach to minimizing the impact of iteration is to reduce information interdependence between modules at their interfaces. The designer needs to perform system level activity in order to define the boundaries for the modules. Standardizing and making these interfaces explicit can aid in keeping the information transfer across boundaries low, as well as synchronizing or integrating the modules frequently [17], thus possibly extending the limits to concurrency as Hoedemaker et al. present [18].

Denker et al. [9] present the use of design structures matrix to limit the number of iterative cycles through better task planning and sequencing. Smith and Eppinger [8] extend the model to include strength of coupling, and present advice on how to better plan activity execution based on the information obtained. In essence, the results indicate that designers should not execute a task until all the tasks that strongly influence it are finished, and should perform long tasks later so there is less chance of repeating them.

FUTURE WORK

The framework this paper presents helps address the question of need because it induces the designer or manager to think about the causes of iteration and how they affect progression on either scope exploration or abstraction level. The next step is to validate framework. We have collected engineering design journals from students who keep them as part of their requirements for engineering capstone design courses [20]. Recognizing iteration types in the design journals and associating them to time spent on the design (as a measure of development time or cost) and on the quality of the outcome design might provide insight into the validity of the framework.

In the context of our research effort, the proposed framework intends to help answer three related questions. First, what is the relationship between design process and project success as measured by achieving a quality outcome at a low cost? The framework provides attributes for coding design journals and grouping them under similar process attribute profiles to then determine the association between design process, quality of the design, and hours spent on the project (a proxy for engineering development cost).

Second, we hypothesize that accelerating the parallel definition of problem and solution spaces leads to a higher quality fit between problem and solution space. It seems that the complete definition of a problem entails the complete definition of the solution. Furthermore, decisions made while designing often determine which problems are encountered in the future. Thus the problem space changes as decisions are made. If design decisions at least partially define the problem space, how we inform these decisions ultimately affects solution feasibility and complexity. For example, converging on a solution without determining first the feasibility or complexity of alternative solutions may lead to selecting and pursuing an infeasible solution, or an unnecessarily complex one. Because of the interdependence of problem and solution spaces, it seems that the focus should be on exploring design space as a combination of both spaces, similar to focusing on the coin rather than one of its sides, before committing to a single solution.

After acknowledging the phenomenon of problem/solution co-evolution, the third question deals with how to accelerate it. In what ways do representations of ideas and information affect designers' ability to explore the design space before committing to a solution? Thorough exploration of the design space might better inform decisions and prevent executing behavioral iterations sequentially, which increases lead time. Exploration should occur at abstraction levels that provide enough ambiguity to evaluate feasibility of alternative solutions without expending resources to develop them completely. As a tool for this exploration, representations might provide the ambiguity needed to design at different abstraction levels [21].

CONCLUSION

The extant literature on iteration provides insightful definitions and categorizations useful to better understanding the phenomenon. However, the taxonomies do not seem very helpful in answering the question of whether iteration is absolutely necessary, and if so under what conditions and with what consequence. In an effort to further explore iteration, this article presents a new classification to directly address this question that delineates rework from design and behavioral iteration types. It also sheds new light on the causes of iteration, which potentially could result in guidelines to improve design practice.

Rework iteration does not help the design evolve towards the intended goal because it focuses on recovering from previous design errors. Design iteration focuses on the

evolution of the design through abstraction levels while behavioral iteration explores design space through alterations on scope. A possible recommendation is avoiding the causes of rework iterations (identifying errors at source) and performing both design and behavioral iterations to evolve both in terms of design definition and breadth of design space exploration, but performing behavioral iterations in parallel to reduce design lead time.

ACKNOWLEDGMENTS

The National Science Foundation grant entitled "CAREER: The Role of Representation in the Synthesis Process" (NSF Award # REC-9984484) provides funding for this research.

REFERENCES

[1] Terwiesch, C. Loch, C.H. De Meyer, A. 2002. Exchanging Preliminary Information in Concurrent Engineering: Alternative Coordination Strategies. Organization Science. July-August, 13(4): 402-419

[2] Sobek II, D.K., Ward, A. Liker, J.K. 1999. Toyota's Principles of Set-Based Concurrent Engineering. Sloan Management Review, Winter, 40(2): 67-82.

[3] Krishnan, V. Eppinger, S.D. Whitney, D.E. 1997. Simplifying Iterations in Cross-Functional Design Decision Making. Journal of Mechanical Design. December. 119: 485-493

[4] Ulrich, K. Eppinger, S. 2000. Product Design and Development. 2nd Ed. Irwin McGraw-Hill, Boston.

[5] Adams, R.S. Atman, C.J. 1999. Cognitive Processes in Iterative Design Behavior. Proceedings of the 29th ASEE/IEEE Frontiers in Education Conference, Session 11a6. November 10-13, San Juan de Puerto Rico.

[6] Adams, R.S. Atman, C.J. 2000. Characterizing Engineering Student Design Processes – An Illustration of Iteration. Proceedings of the ASEE Annual Conference, Session 2330. June 18-21, St. Louis, MO.

[7] Smith, R.P. Eppinger, S.D. 1997a. A Predictive Model of Sequential Iteration in Engineering Design. Management Science. August. 43(8): 1104-1120

[8] Smith, R.P. Eppinger, S.D. 1997b. Identifying Controlling Features of Engineering Design Iteration. Management Science. March. 43(3): 276-293

[9] Denker, S. Steward, D.V. Browning, T.R. 2001. Planning Concurrency and Managing Iterations in Projects. Project Management Journal. September. 32(3): 31-38

[10] Sobek II, D.K. 2002. Preliminary Findings from Coding Student Design Journals. Proceedings of the 2002 American Society for Engineering Education Annual Conference. Montreal, Canada.

[11] Press release from the European Space Agency's Inquiry Board at ESA's official web site: http://www.esa.int/export/esaLA/Pr_33_1996_p_EN.html. Also find in http://ravel.esrin.esa.it/docs/esa-x-1819eng.pdf the official report.

[12] Karlsson, C. Nellore, R. Soderquist, K. 1998. Black Box Engineering: Redefining the Role of Product Specifications. Journal of Product Innovation Management 15: 534-549

[13] Busby, J.A. Lloyd P.A. 1999. Influences on Solution Search Processes in Design Organizations. Research in Engineering Design. 11: 158-171

[14] March, J.G. Simon, H.A. 1958. Organizations. New York, John Wiley

[15] Ball, L.J. Evans J. St. B. T. Dennis, I. 1994. Cognitive processes in engineering design: a longitudinal study. Ergonomics. 37(11): 1753-1786.

[16] Krishnan, V. Eppinger, S.D. Whitney, D.E. 1995. Accelerating Product Development by the Exchange of Preliminary Product Design Information. Journal of Mechanical Design. December. 117: 491-498

[17] Cusumano, M. 1997. How Microsoft Makes Large Teams Work Like Small Teams. Sloan Management Review. Fall, 9-20.

[18] Hoedemaker, G.M. Blackburn, J.D. Van Wassenhove, L.N. 1995. Limits to concurrency. INSEAD R&D Working Papers. January, 95(27): 1-21

[19] Ward, A. Liker, J.K. Cristiano, J.J. Sobek II, D.K. 1995. The Second Toyota Paradox: How Delaying Decisions Can Make Better Cars Faster. Sloan Management Review, Spring, 36: 43-61.

[20] Sobek II, D.K. 2002. Use of Journals to Evaluate Student Design Process. Proceedings of the 2002 American Society for Engineering Education Annual Conference. Montreal, Canada.

[21] Busby, J.S. 2001. Practices in Design Concept Selection as Distributed Cognition. Cognition, Technology & Work. 3:140-149

