

Managing Problem-Solving Iterations in Product Development

Jeffrey W. Herrmann

Abstract—Problem-solving is a useful way to view product development. In many cases, design teams iterate through a generate-test cycle until a successful design is found. The time required to find a successful design is influenced by the testing strategy. In particular, the amount of time spent generating a solution affects the probability of passing the test. This paper introduces a novel model of the generate-test cycle, develops results from the model, and presents useful insights that can be used to make better decisions about managing problem-solving iterations.

Index Terms—Research and Development Management, Product Development, Decision-making, Design Methodology.

I. INTRODUCTION

PRODUCT development processes involve a great variety of activities and can be viewed from many perspectives. Moreover, there is much diversity among product development projects even within one firm, as some projects involve quite routine redesign of a well-understood product and others require inventing completely new designs for a new market opportunity.

Problem-solving is one useful way to view product development [1], and, from this perspective, a product development project is a set of nested problem-solving cycles. Each problem-solving cycle searches for design alternatives, builds models of these, run experiments, and analyzes the result [2]. If the results are satisfactory, the cycle ends with a solution. Otherwise, the cycle must repeat.

The literature on managing product development has often looked at the problem of using testing and reviews to uncover problems that are introduced by design activities. For instance, if there is a certain amount of design work must be done, but this design work introduces random flaws, then management needs to schedule reviews to find the flaws quickly, since the penalty for a design flaw is a delay needed to repeat the work that was done since the flaw occurred [3]. The rework introduces no new flaws, but reviews take time, so too many reviews delay the project as well. In the case of sequential testing, [4] develop a policy for minimizing the

total cost of testing and redesign under the assumption that the number of problems grows as the project approaches completion. Such models are more appropriate for a routine design process that has a well-understood amount of work to do and flaws that are straightforward to fix.

However, in many cases, the amount of design activity needed is not known in advance. Instead, there are performance requirements (which may be hard to translate into engineering specifications), and the task that is assigned to the design engineer (or product development team) is to find a design that meets these requirements. This more closely matches the problem-solving perspective. Like the problem-solving cycles mentioned earlier, this type of task (repeated at different levels and for different portions of the product) can occur many times in a product development project.

For example, we studied the product development process of a firm that was developing a product for children with disabilities. A child plays with the product in certain ways designed to help the child learn new skills. The interface involves voice recognition and sensors for detecting the child's gestures and body movements, to which the product reacts appropriately. At a certain stage in the development of the product, the firm arranged testing sessions when children could play with a prototype of the product and the firm could determine if the design was satisfactory. (Note that it was not possible, ahead of time, to specify precisely what a satisfactory design should do.) After a testing session, the firm redesigned the product and then arranged another test, though they were never sure that the test would be successful since they could not test the product in the lab. The firm continued these tests until they had a product that passed. The design activities needed to reach this point were not possible to predict ahead of time, since the firm had never done anything like this before. The design activities were moving the design towards the desired state, not introducing problems.

This type of iteration has been described as the “generate-test” cycle [5]. There is a chance that the results of a cycle will be unsatisfactory, but that likelihood depends upon the amount of effort spent in the generate stage of the cycle. This raises the following question: *how much effort should be spent in each generate-test cycle?* The answer affects the probability of finding a satisfactory solution during that cycle and thus the number of cycles needed until a satisfactory solution is found. This, in turn, affects the total time needed to solve this problem.

Manuscript received July 7, 2005. This material is based on work supported by the National Science Foundation under grant number 0225863.

Dr. Jeffrey W. Herrmann is with the Department of Mechanical Engineering and Institute for Systems Research, University of Maryland, College Park, MD 20742 USA (phone: 301-405-5433; fax: 301-314-9477; e-mail: jwh2@umd.edu).

This problem is most relevant to situations where an explicit testing step is needed and managerial judgment is needed to determine when the design should be tested. This occurs in situations like the one described above, when the manager or customer cannot describe a satisfactory design but instead says “I know it when I see it.” It also occurs when there is no substitute for sophisticated tests like crash tests, explosive tests, environmental stress tests, and similar activities. For example, the design of a protective shield for a workstation that processes explosive material must undergo a destructive test to verify that it can protect the operator in case of a mishap. Such a test requires a great deal of effort to create a prototype, setup the test equipment, perform the test, and analyze the results.

The significance of this question is increasing as new technologies (including simulation, computer-aided engineering analysis, and rapid prototyping) become available and product development organizations modify their product development processes to take advantage of these tools. The benefits of rapid experimentation have been well documented [6-8]. However, no single approach can be optimal for all product development processes. Managers need insight and models for making good decisions about how to change their testing strategies.

The problem is minimized if the requirements can be explicitly stated and possible solutions can be quickly, inexpensively, and accurately checked against them. However, two possible obstacles can occur.

First, the design team may have limited ability to check a possible solution during the generate stage, especially if they depend upon models to estimate design performance (e.g., deflection of a beam, circuit board reliability, machining time, or vehicle maneuverability). These models, based on scientific theory, experimental results, and experience, may be inaccurate or imprecise. Thus, the actual performance of a design will vary from the predicted, and a design that should be adequate will fail during testing.

Second, this type of requirements satisfaction is contrary to the desire to optimize performance that is deeply ingrained in engineering. For instance, a study of Volvo engineers responsible for the final development of new engines revealed that some engineers believed their job was to make the engine meet performance specifications, others thought that they needed to resolve tradeoffs between performance categories, and a third set wanted to make the engine provide the customer with a good driving experience [9]. Design engineers who want to optimize performance are never “done” with a design; there are always additional features that can be added.

This paper presents a simple model that focuses on minimizing the expected time needed to complete a design task that iterates through a generate-test cycle until the design passes. The purpose of this analysis is to gain insight into this fundamental phenomenon that will help managers control their product development processes. A key feature of the model is that the probability of passing the test is a function of

the time spent in the generate stage, which is an independent variable.

Unlike the models described above, the model presented in this paper is more appropriate for innovative design processes that involve creativity and more technical risks. In the face of complexity or an inaccessible environment that makes learning ineffective, project managers may take a selectionist approach [10]. In this domain, a design team may choose to try multiple concepts, either in parallel or in sequence. The model presented in this paper can be used to describe this sequential strategy by considering each iteration as a separate attempt to reach the design requirements. A test can confirm that the team has found a satisfactory solution, so having a test quickly is a chance to complete the project as soon as that happens.

This article proceeds as follows. We start with a brief discussion of iteration. Section III then presents the model of the generate-test cycle. Section IV discusses the special case where each iteration is the same because there is no learning. Section V analyzes the general case, in which learning can occur from iteration to iteration. Section VI describes a policy that can be used when multiple design teams need to use the same test resource. We conclude the paper with a discussion of the insights gained from these models.

II. ITERATION

The iteration caused by the generate-test cycle is not the only type of iteration in product development. Three types of iteration have been identified [11]. *Design iteration* occurs when the same design activity is done again on the same part of the product, but at a different level of abstraction. *Behavioral iteration* occurs when the same design activity at the same level of abstraction is done again on another portion of the product. The iteration caused by generate-test cycles is *rework iteration*, because the same design activity at the same level of abstraction is done again on the same portion of the product. An example of a generate-test cycle is the design of the Ariane 5 rocket. Unfortunately, the rocket failed at launch (the ultimate test), and the rocket had to be redesigned.

The diverse causes of iteration have inspired many approaches to minimize its impact. A complete review is beyond the scope of this paper, but a few examples will be mentioned. Iteration can be caused by the decomposition that occurs in product design [12]. Determining the value of some design variables early in the process limits the freedom of design variables that are set later. This can lead to situations in which no solution is feasible or to solutions that perform poorly. The properties of sequence invariance and task invariance can be used to improve the design process. When there are many coupled design tasks that require information to be interchanged, performing each task requires the other tasks to be repeated. In this case, a work transformation matrix model can determine which tasks require the most effort [13].

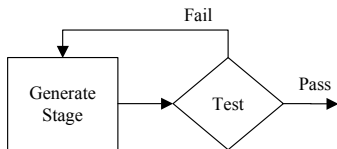


Fig. 1. The generate-test cycle model. The design team iterates through this cycle until their design passes the test.

III. THE BASIC MODEL

In this model of the generate-test cycle, the probability that a design will pass the test increases as the generate stage continues. This occurs because the design team works on the design more, finding better solutions, discovering more relevant information, double-checking their analysis, and allowing new issues to appear so that the team can resolve them before the test.

Another key feature of this model is that the test takes time. The particular activities necessary for a test depend upon the type of test, but they may include constructing a prototype, planning, obtaining approvals, conducting the test, analyzing the results, documenting the results, preparing presentation materials, and holding a review or meeting to discuss the results.

This situation presents an important question for management, who must decide how long the design team should spend in the generate stage that precedes the test. The key performance measure is the time until a successful design is found. This time is a random variable, and there can be a tension between minimizing its expected value and minimizing its variance, as we will see.

The design team designs the product during the generate stage and then proceeds to a test that determines whether the product at that point is satisfactory. If so, the design activity is done. If not, the design team must redesign the product (repeat the generate stage) and return to the test. The design team must repeat this loop until the product passes the test. Each passage through this loop is an iteration.

Let r denote the time needed to conduct a test (this is given). Let t_i denote the time spent in the generate stage during the i -th iteration. This decision variable must be determined at the beginning of the iteration. Let $p_i(t_i)$ be the probability of passing the test in the i -th iteration, which depends upon the time spent in the generate stage. Intuitively, it is reasonable that $p_i(t_i)$ is an increasing function that approaches 1 in the limit.

We assume that, if the product fails the test and the design team has to conduct another iteration, the probability of success must return to zero at the beginning of the generate stage (since the existing design is known to be bad). Now, if the test provides information from which the design team can learn, then $p_i(t_i)$ may be different.

If there is no learning from iteration to iteration, then we can assume that all of the $p_i(t_i)$ have the same form. Let this

function be $p(t)$. We can also use this model as an approximation to the more complex case.

One possible form for $p(t)$ is

$$p(t) = 1 - e^{-at}, \text{ for } t > 0 \quad (1)$$

However, other forms may be appropriate.

IV. NO LEARNING: IDENTICAL ITERATIONS

A. Minimizing Expected Total Time

We first consider the case of minimizing the expected total time. In this case, it is easy to show that the design team should spend the same amount of time in the generate stage each iteration. Given that the design team must perform the generate stage, finding a successful design does not depend upon how many iterations the design team has performed previously. That is, there is a memoryless property. The expected time needed to find a successful design is independent of what has happened before. Thus, the policy should remain the same each time.

For a specific value of t , which determines $p(t)$, the number of iterations needed is a random variable that has a geometric distribution. The expected number of iterations (including the successful one at the end) is $1/p(t)$.

The total time needed to find a successful design is $t + r$ times the number of iterations. Thus, $T(t) = (t + r)/p(t)$ is the expected total time. The shape of this curve depends upon the shape of $p(t)$. Given a test time r , minimizing the expected total time requires choosing the optimal time t^* for the generate stage. Choosing t too small leads to a large number of iterations, while choosing t too large leads to long stages.

If $p(t) = 1 - e^{-at}$, $T(t)$ is a convex function that has a single optimal solution at the point t^* that satisfies $e^{at^*} - at^* = ar + 1$. Fig. 2 shows the shape of $T(t)$ for different values of r in this case.

The test time r has a significant impact on the optimal solution in this case. As r approaches 0, t^* and $T(t^*)$ approach 0. As r increases, t^* increases and $T(t^*)$ increases. In some sense, the generate stage t is similar to a batch size that is used to avoid setups (in this case, tests). Instead of worrying only about the optimal batch size (in this case, t^*), it also makes sense to reduce the setup cost (the test time r), which allows smaller batches (quicker tests) and improves the overall performance. This result, which matches our intuition and others' observations [6-8], provides evidence of the model's validity.

If the design is a more routine project where the design team can generate a design that will certainly pass the test, we should consider a different function for the probability of test success. Suppose that there is a bound G on the time needed for the generate stage and the probability of test success equals 1 if the generate stage is this long. Moreover, the probability of test success is a linear function of t : $p(t) = t/G$. Then, the expected total time $T(t) = (t + r)/p(t) = (t + r)G/t = G(1 + r/t)$, which reaches its minimum at time $t^* = G$, independent of the test time r . The optimal expected total

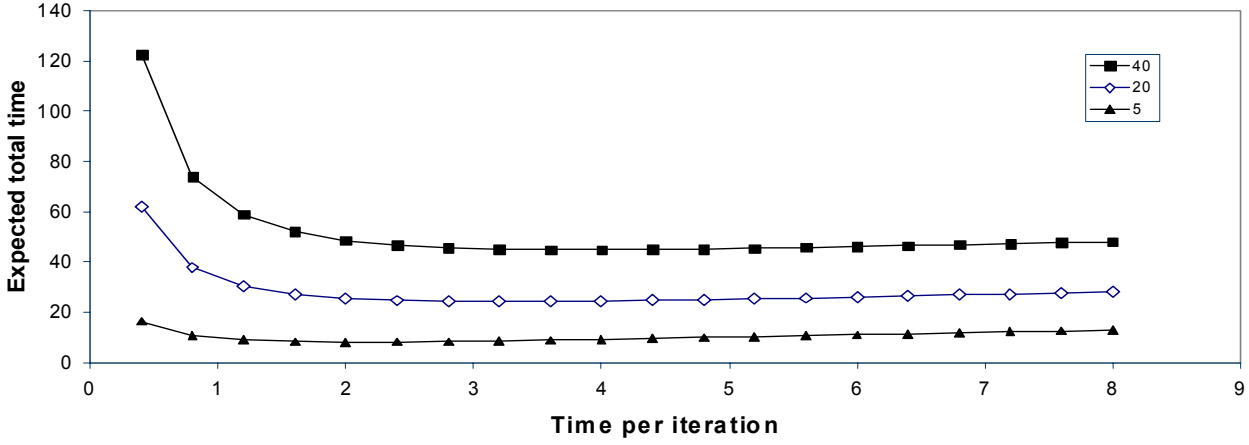


Fig. 2. Graph of expected total time versus time per iteration for different values of test time. In this case, iterations occur without learning, and the probability time curve has an exponential form. The three curves correspond to different values of test time: 5, 20, and 40 time units. These curves show that, as the test time increases, the optimal time per iteration increases, and the minimum expected total time increases.

time is $T(t^*) = G + r$, and there is no uncertainty about the value (since $p(G) = 1$). This shows that iteration is not helpful in this type of design scenario.

Risk aversion also affects the optimal policy, as the next section discusses.

B. A Risk-averse Manager

Managers are notoriously risk averse. A risk averse manager worries more about the variability of project duration than the expected duration. If there is a deadline, the risk-averse manager is most concerned about maximizing the probability that a successful design will be found by the deadline. (Ideally, this probability will equal one.)

Let D be the given deadline for the design to pass the test. The manager must decide on the maximum number of iterations to do and the time to spend in each iteration's generate stage. If the team gets to the last iteration, it is easy to show that the amount of time spent in that generate stage should be just enough to allow time for one last test before the deadline.

For example, suppose the deadline is 90 days from now, and the test takes 5 days. Then, the design team could perform one iteration, spending 85 days in the generate stage. Or the design team could spend 45 days in the generate stage in the first iteration. The first test ends 50 days from now. If the design fails the first test, the design team will have 40 days remaining, so they can spend 35 days in the generate stage in the second iteration. Or they could spend 60 days in the generate stage in the first iteration and 20 days in the generate stage in the second iteration. Or they may plan for three, four, or more iterations.

Let m be the number of planned iterations. Let t_i be the amount of time spent in the generate stage of the i -th iteration. The time for the i -th iteration is $t_i + r$. Therefore, to allow time for the m tests, the sum $t_1 + \dots + t_m$ must equal $D - mr$.

Let q_i be the probability of a successful test by the end of the i -th iteration. Then, $q_1 = p(t_1)$. For $i > 1$, $q_i = q_{i-1} + (1 - q_{i-1})p(t_i)$.

Let P_m be the maximum probability of success when the design team plans to have m iterations.

$$P_m = \max \{q_m : t_1 + \dots + t_m = D - mr\}$$

If $p(t) = 1 - e^{-at}$, it is easy to show that, for a given m , $q_m = 1 - e^{-a(D-mr)}$ for any feasible t_1, \dots, t_m .

Thus, $P_m = 1 - e^{-a(D-mr)}$. This is maximized when $m = 1$, so the design team should plan one iteration.

If $p(t) = t/G$ and $G > D - r$, then note that $1 - P_m$ equals $\min \{1 - q_m : t_1 + \dots + t_m = D - mr\}$. $1 - q_m$ is the probability of failing all m iterations, which can be expressed as follows:

$$1 - q_m = \left(1 - \frac{t_1}{G}\right) \dots \left(1 - \frac{t_m}{G}\right) \quad (2)$$

This reaches its minimum when all of the t_i except one are equal to 0. The non-zero t_i equals $D - mr$. Thus, $1 - P_m = 1 - (D - mr)/G$, so $P_m = (D - mr)/G$. As in the previous case, this is maximized when $m = 1$, so the design team should plan one iteration.

If $p(t) = t/G$ and $G \leq D - r$, then it is clear that the design team can spend G time units in the generate stage of the first iteration and certainly pass the test on the first try.

V. ITERATION WITH LEARNING

When there is learning from iteration to iteration, the function $p_i(t_i)$ is different each iteration. Because the general case is intractable, we consider the case where the function $p_i(t_i)$ converges to a function $p(t)$ after m iterations. That is, $p_i(t_i) = p(t)$ for $i > m$.

Let R_i be the expected remaining time needed to pass the test at the beginning of the i -th iteration.

$$R_i = t_i + r + (1 - p_i(t_i))R_{i+1} \quad (3)$$

Based on the results for the case with no learning, we know that, if the design team fails on the first m iterations, then, for the subsequent iterations, they should choose t to minimize $R_{m+1} = (t + r)/p(t)$. Then, we can find the optimal t_m, t_{m-1}, \dots, t_1 by minimizing R_m, R_{m-1}, \dots, R_1 using (3).

Consider, as an example, a situation with the following probability functions:

$$\begin{aligned} p_1(t) &= 1 - e^{-t} \\ p_2(t) &= 1 - e^{-2t} \\ p_i(t) &= 1 - e^{-4t} \text{ for } i > 2. \end{aligned}$$

If $r = 5$ time units, the design team can minimize R_3 by choosing $t = 0.796$ time units (for all iterations after the second), which yields $R_3 = 6.046$ time units. Then, to minimize R_2 , t_2 should be 1.246 time units, which yields $R_2 = 6.746$ time units. Then, to minimize R_1 , t_1 should be 1.909 time units, which yields $R_1 = 7.909$ time units.

VI. MULTIPLE PROBLEMS AND A SINGLE TEST RESOURCE

The above results indicate how much time a design team should spend in the generate stage before performing a test to determine if they have solved their problem. This analysis assumed that the design team's activities are independent of any other design teams. However, in practice, an organization may have limited testing resources. For instance, there may be a single facility that can perform an explosive test. For a given testing resource, there will be multiple design teams working to solve problems that require that testing resource. Then, there may exist queueing for the testing resource. If design teams test too often, that increases the queueing and the time for all teams to solve their problems. Ideally, the numerous design teams that require the testing resource should find a joint policy that is best for everyone [14]. However, such coordination, even if tractable, may be difficult to implement.

To simplify the situation, consider the expected number of tests that each problem requires to be solved. This affects the utilization of the testing resource. Maintaining a reasonable utilization is an important management objective.

Consider now a specific class of problems that requires a specific testing resource. Let λ be the rate (in problems per time unit) at which new problems in this class arrive. As before, r is the time needed to perform a test. We assume that $\lambda r < 1$ so that the testing resource has enough capacity to perform at least one test for each problem. Let n be the expected number of tests that each problem will require before it is solved. The utilization u of the testing resource equals $\lambda r n$. Since u must be less than one, this provides an upper bound on n . The system will be stable if and only if $n < 1/(\lambda r)$. Moreover, if management wants to limit utilization to at most U (where $U \geq \lambda r$) then the expected number of tests must be no greater than $U/(\lambda r)$. This rule provides a simple way to coordinate problem-solving. Each design team must choose a policy that yields an expected number of tests less than $U/(\lambda r)$.

If there is no learning, then $n = 1/p(t)$, and the design team must choose t large enough so that $p(t) > \lambda r/U$.

If there is learning, then we can use the following procedure to calculate the expected number of tests based on t_1, \dots, t_m , and t for the remaining iterations.

Let N_i be the expected number of remaining tests including the one in the i -th iteration. $n = N_1$.

$$N_i = 1 + (1 - p_i(t_i)) N_{i+1} \quad (4)$$

If the design team fails the first m iterations, then, for the subsequent iterations, they will spend t time units in the

generate stage of each subsequent iteration. Thus, $N_{m+1} = 1/p(t)$. Then, we can find the remaining N_m, N_{m-1}, \dots, N_1 . The design team should set t_1, \dots, t_m , and t so that $N_1 \leq U/(\lambda r)$.

Consider again the example presented above. If the design team follows a policy with $t_1 = 1.909$ time units, $t_2 = 1.126$ time units, and $t = 0.796$ time units, the expected number of remaining tests at each iteration can be calculated as follows: $N_3 = 1.043$. $N_2 = 1.086$. $N_1 = 1.161$.

VII. CONCLUSION

This paper presents a deliberately simple model of product development and focuses on the generate-test cycle in order to obtain some useful insights into the issue of controlling product development processes. In particular, to minimize the expected time needed to solve a design problem, the optimal strategy depends upon the nature of the relationship between the time spent developing a solution and the probability of test success. Performing more iterations can reduce the expected total time to solve a problem in some cases. Also, reducing the time needed to perform a test has a significant impact on the expected duration. Not only is there less test time per iteration, reducing the test time makes shorter generate stages (more frequent iteration) desirable.

When the design problem is routine, then iteration does not reduce the expected total time. The team should finish the generate stage to get a design that will certainly pass. Reducing the time needed to perform a test has less impact on the expected duration in this case.

Iterations do add variability to the time needed to find a successful design. In a creative design process, a risk-averse manager who wishes to meet a deadline should avoid iteration, since it does not increase the probability of meeting the deadline. Also, reducing the time needed to perform a test has a significant impact on this probability, since it means more time to perform the design stage.

If there is a shared test resource that many design teams need to use to solve problems, then design teams should follow policies that bound the expected number of tests that a team will perform. Such a strategy ensures reasonable utilization of the shared test resource. Our model provides guidance on determining the bound and evaluating testing policies against that.

There are many more issues regarding iteration and testing that are not considered in this model, and the model could be extended in many ways. For instance, the time-probability curves could vary based on the total amount of time spent in previous generate stages (not just the number of iterations already completed). Research on methods for eliciting the probability-time curves from managers and engineers and from data about previous projects would be a useful step towards providing better tools that help managers make these decisions.

ACKNOWLEDGMENT

The author appreciates the insights and help provided by Linda Schmidt, Joseph Donndelinger, my students, and many

other collaborators.

REFERENCES

- [1] Thomke, Stefan, and Takahiro Fujimoto, "The effect of front-loading problem-solving on product development performance," *Journal of Product Innovation Management*, Volume 17, pages 128-142, 2000.
- [2] Clark, Kim B., and Takahiro Fujimoto, *Product Development Performance*, Harvard Business School Press, Boston, Massachusetts, 1991.
- [3] Ha, Albert Y., and Evan L. Porteus, "Optimal timing of reviews in concurrent design for manufacturability," *Management Science*, Volume 41, Number 9, pages 1431-1447, 1995.
- [4] Thomke, Stefan, and David E. Bell, "Sequential testing in product development," *Management Science*, Volume 47, Number 2, pages 308-323, 2001.
- [5] Simon, Herbert A., *The Sciences of the Artificial*, Second edition, The MIT Press, Cambridge, Massachusetts, 1981.
- [6] Reinertsen, Donald G., *Managing the Design Factory*, The Free Press, New York, 1997.
- [7] Schrage, Michael, *Serious Play*, Harvard Business School Press, Boston, Massachusetts, 2000.
- [8] Thomke, Stefan, *Experimentation Matters*, Harvard Business School Press, Boston, 2003.
- [9] Sandberg, Jorgen, "Understanding competence at work," *Harvard Business Review*, Volume 79, Number 3, pages 24-28, 2001.
- [10] Pich, Michael T., Christoph H. Loch, and Arnaud De Meyer, "On uncertainty, ambiguity, and complexity in project management," *Management Science*, Volume 48 Number 8, pages 1008-1023, 2002.
- [11] Costa, Ramon, and Durward K. Sobek II, "Iteration in engineering design: inherent and unavoidable or product of choices made?" DETC2003/DTM-48662, ASME 2003 Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Chicago, Illinois, September 2-6, 2003.
- [12] Krishnan, V., S.D. Eppinger, and D.E. Whitney, "Simplifying iterations in cross-functional design decision making," *Journal of Mechanical Design*, Volume 119, pages 485-493, 1997.
- [13] Smith, Robert P., and Steven D. Eppinger, "Identifying controlling features of engineering design iteration," *Management Science*, Volume 43, Number 3, pages 276-293, 1997.
- [14] Herrmann, Jeffrey, "Controlling iteration in product development processes," Technical Report 2005-91, Institute for Systems Research, University of Maryland, College Park, Maryland, 2005.