Reducing Rework by Applying Set-Based Practices Early in the Systems Engineering Process

Authors: Brian Kennedy, Durward Sobek, & Michael Kennedy

Reducing Rework by Applying Set-Based Practices Early in the Systems Engineering Process

Brian M. Kennedy,*[,1] Durward K. Sobek II,[2] and Michael N. Kennedy[1]

[1] *Targeted Convergence Corporation*, 1420 *Valwood Parkway Suite* 100, *Carrollton*, TX 75006
[2] *Department of Mechanical and Industrial Engineering*, *Montana State University*, *Bozeman*, MT 59717-3800

Abstract

Rework that occurs late in the product life cycle is dramatically more expensive than design work performed early in the cycle. However, shifting traditional design work earlier in the design process so as to avoid rework later is difficult. A number of product development practices that have been characterized as a shift from developing a single-point design to developing a set of possible designs have proven effective at reducing development rework. This paper refines the definitions of such "set-based" devel-opment practices, which are aimed at early development phases, and shows how they can be applied to the systems engineering process in order to reduce or eliminate the root causes of rework. Examples from the Wright Brothers, Toyota, and several other companies are presented.

1. INTRODUCTION

Systems engineers and development managers are all too familiar with the frustration of seeing development teams revisit decisions made earlier in their projects, and wincing at the ripple effects of violated assumptions, associated design changes, and reworked plans, analyses, and designs they know are coming as a result. For example:

- During prototype testing, the software team discovers that two of the design specifications are incompatible, meaning one of them must be revised and all the code developed to date reworked; or,
- The product management team realizes to their surprise that customers do not like the trade-offs made among competing objectives, so key product features must be changed late in the game at significant cost to the organization, or face lower than projected sales; or,
- In developing the manufacturing process, the manufacturing engineering team learns that the current equipment is not capable of manufacturing an important product feature, resulting in the tough decision either to redesign the feature thereby delaying product delivery or to invest much more than planned in updating manufacturing capability; or,
- After market introduction or delivery to the customer, a latent design flaw results in warranty claims that eat up most of the profitability of that product, possibly even blowing up into a product recall.

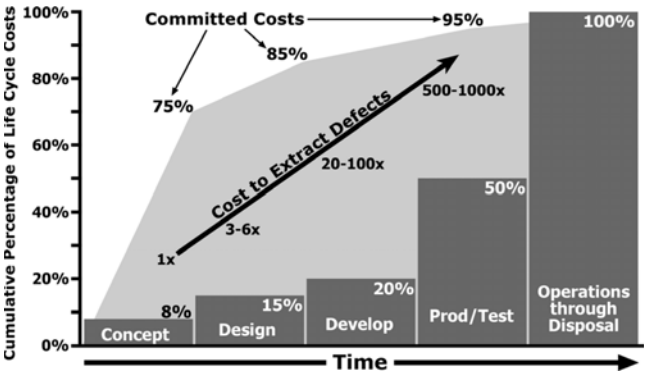

Figure 1. Costs are committed very early, even if they are not incurred until much later in the project cycle, adapted from INCOSE [2011], credited to the Defense Acquisition University, 1993.

The bottom line impact of unfortunate but common occurrences such as these is staggering. The cost to extract such defects can increase from 3 to 1000 times the original development costs depending on when they are discovered, as shown in Figure 1. Further, such defects squander market opportunities, damage brand reputation, and otherwise take huge bites out of a firm's return on investment and growth potential. Often missed are the compounding effects of frustrated employees, cross-organizational conflicts, and related inefficiencies.

As shown in Figure 1, it is widely reported that 60 to 75% of the life cycle costs of a project are determined by decisions made by the end of the concept phase [e.g., Hari et al., 2008; Smith and Reinertsen, 1991: 225; Anderson, 2010]. Further, as much as 85% is committed before the core development process (or detailed design) starts. However, the dilemma systems engineers face is that often very little is known about the full ramifications of those decisions when they are being made. Thus, we should not be surprised that such decisions are frequently revisited, and the associated development work redone.

We use the term *rework t*o specifically mean the work that occurs when a prior decision that was assumed to be final for that project is changed because it was later found to be defective. (A decision is considered "final" in the sense that the team does not have any reason to believe that the decision would need to change and therefore expects that development work can proceed assuming that decision will stand for the remainder of the project.) Such rework is distinct from design iterations performed for rapid learning, where the design decisions are understood to be experimental, and thus other development work does not proceed assuming those to be the final decisions. It is also distinct from establishing rapid project cycles to accelerate customer feedback. However, rework may occur within a design iteration or rapid project cycle if a poor decision would otherwise result in the iteration or project cycle failing to achieve its goals.

Some of the larger companies that we work with have measured (with unconfirmed accuracy) that 70 to 80% of their development effort is spent on loopbacks such as the examples in the bulleted list above. Other projects have reported similar numbers, with rework commonly ranging from 30 to 70% [Osborne, 1993; Reichelt and Lyneis, 1999: 146; Chucholowski et al., 2012]. And those numbers are commonly reported despite the fact that concealment of rework is often standard practice [Ford and Sterman, 2003b].

Rework has become so commonplace that most development managers seem to consider it *inevitable* [Fricke et al., 2000; Thomke, 2003: 162, 165; Clark and Fujimoto, 1991; Forsberg et al., 2005: 267], a necessary evil. So they schedule multiple prototype build-and-test cycles into their project plans, create elaborate engineering change processes to manage multiple layers of rework [Jarratt et al., 2011], and pad schedules to account for the unknowns. Such practices treat the symptoms as if they were unavoidable, rather than address the root causes so that the rework can be eliminated [Forsberg et al., 2005: 265–270].

We assert that much of the rework that product development organizations experience is *not* inevitable. Development organizations *can* eliminate rework from their product development processes. But to do so requires a very different way of thinking about the systems engineering of new products and technology; this, in turn, mandates a very different way of managing development projects and development organizations, all starting with the front end of the product development process.

2. PREVENTING REWORK

A number of models and tools have been proposed to reduce the need for rework to some degree. We summarize a few of them in this section, but go on to argue that these approaches will have limited impact if they do not address the root causes of that rework. For that, we suggest a different paradigm is needed.

2.1. Existing Approaches

One of the earliest approaches to reduce the size, cost, and frequency of rework is concurrent engineering, where development teams attempt to avoid rework by performing dependent design tasks (such as product design and manufacturing process design) concurrently [Winner, 1988]. Since then, various forms of concurrent or simultaneous engineering have been proposed and attempted, with varying degrees of success and frustration [Componation et al., 1999]. However, in practice, concurrent engineering tends to be limited to engaging the engineers who will perform the later dependent tasks as advisors to the earlier tasks in order to minimize the number of defective decisions (and thus reduce the amount of rework), often under the label of "Design for X" [White, 1998]. It has even been argued that such concurrency increases rework [Ford and Sterman, 2003a].

The Design Structure Matrix (DSM) can be used to map the dependencies between different parts of the design, or between various development tasks, enabling project teams to sequence design tasks optimally [Steward, 1981; Eppinger, 1991; Browning, 2002; Meier et al., 2007; Eppinger and Browning, 2012]. Sequencing places dependent tasks as close

as possible to each other in time, and clusters interdependent tasks. This lowers the chances of defective decisions resulting from doing tasks out-of-sequence, and reduces the time to discover defective decisions made in another task. However, Yassine et al. [2000: 2] found in some cases "that sequencing and partitioning did not help in improving the process (i.e., the new DSM sequence did not change much from the old sequence)." They further observe that rework often occurs "because the as-is process has the structure of 'generate and test.'" Their solution is to employ domain experts to reengineer the tasks to "create a process that starts out with enough accurate information that the chance of rework is reduced or eliminated, or the time consumed by the rework is reduced" [Yassine et al., 2000: 2].

Gated methodologies introduce "gates" (reviews) between development phases to help identify defective decisions before additional investment is made based on incorrect assumptions [Cooper, 1986; McGrath, 1996].

The Vee Model (shown in Fig. 2) is another approach to reducing rework by carefully and systematically cascading design requirements from the system level to the component level, and doing design verification and validation at the component and subsystem levels before doing it at the system level. Forsberg and Mooz [1991: 61] stress the importance of the "off-core" activities that happen on the left side of the Vee: "The detailed evaluation of operational feasibility, identification of driving technologies, development of software and hardware feasibility models, and identification of system risks must be done by or perceptively reviewed by technical experts. These tasks are the off-core activities… key to the success of this project cycle concept." They depict those off-core activities roughly as in Figure 3.

The off-core activities suggest that it is often necessary to delve into the lowest levels of the design to fully identify and resolve the critical issues that may affect the design decisions in the early phases (see Fig. 3). This requires teams to pull some of the on-core development earlier. For example, the diagram shows that functional allocation should be done off-core for each alternative concept so as to enable identification and resolution of the critical issues for each of those alternatives.

Other models intended to reduce rework include Spiral Development, which emphasizes the gradual refinement of the design to avoid premature decision-making (and the associated rework) [Boehm, 1988], and Iterative Development where design and development of a new product occurs in a series of short phases that repeat similar development activities, but adding more detail and resolution with each successive iteration [Larman and Basili, 2003]. The short cycles reduce the time from when decisions are made to when the defective ones are detected, thus minimizing the size and cost of rework.

On the surface, the high-level Vee Model seems to lack gradual refinement and iterative phases of these last two models. However, Forsberg et al. [2005] demonstrate how both the Spiral Model and the Iterative Development Model can be mapped onto the off-core activities of the Vee Model.

Trying to shift problem identification and resolution to earlier phases of development by "front-loading" projects also potentially reduces rework [Fricke et al., 2000]. Smith and Reinertsen [1991: 224] discuss a particular case of this, "working on the product and process simultaneously typically results in superior design trade-offs that avoid substantial downstream delays that result from poor decisions," and then discuss the many difficulties in making that happen. Thomke and Fujimoto [2000] suggest that front-loading can be accomplished by enhancing knowledge transfer from previous projects, and conducting rapid problem-solving cycles in early phases. But exactly how to most effectively do these front-loading activities is still an open field. Advancements in CAD, CAE, and product simulation technology make it possible, in theory, to conduct problem-solving cycles using virtual rather than hardware models. In practice, though, these technologies still require high-resolution models and therefore are difficult to use in the early development phases. In their discussion of front-loading, Fricke et al. [2000: 174] note that "Unfortunately, an earlier detection of changes does not automatically lead to a front-loading of these changes." We

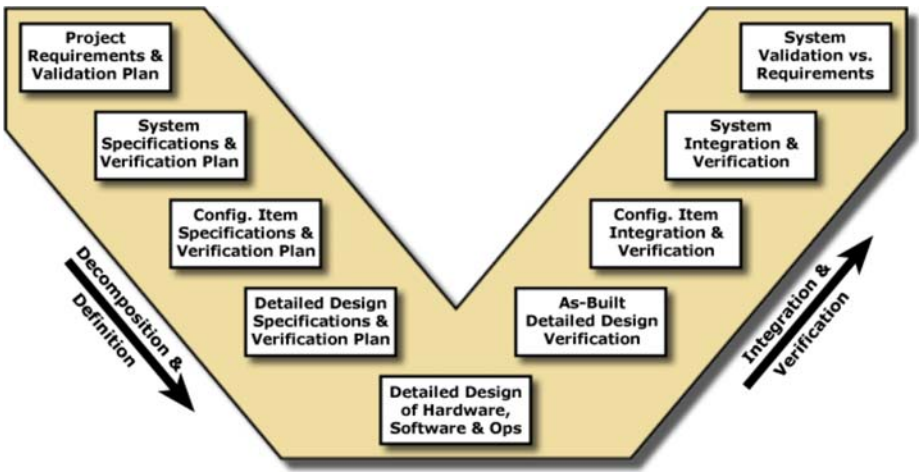

Figure 2. The "Vee" Model of the project cycle, adapted from Forsberg and Mooz [1991]. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

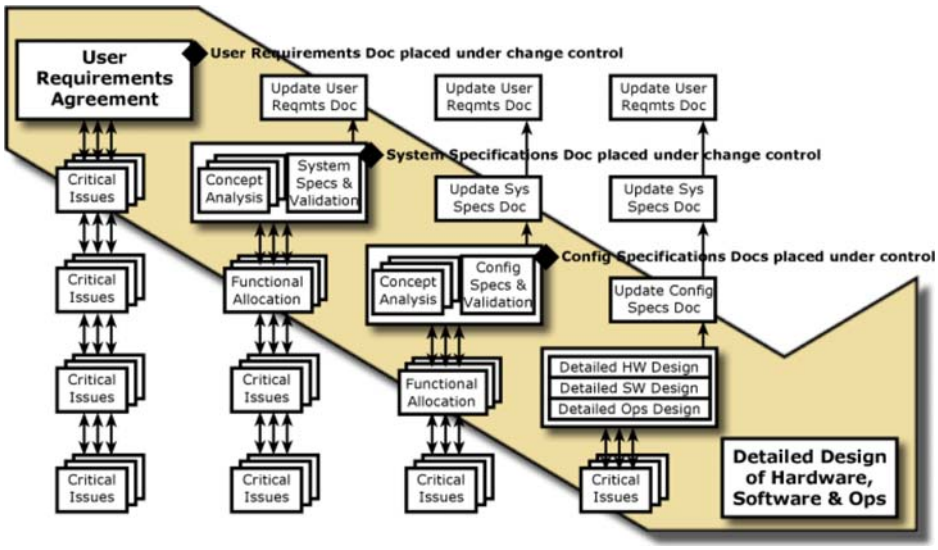

Figure 3. The "off-core" activities of the "Vee" Model, adapted from Forsberg and Mooz [1991] and Forsberg et al. [2005]. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

suggest that front-loading is an important remedy for rework, but can only be achieved consistently by understanding the root causes of the rework (without such understanding front-loading is often difficult or ineffective).

Traditionally, the requirements and specifications developed in the core of the Vee (or any of the alternative processes) have been managed with documents. While those documents generally contain models, the "models have not been integrated into a coherent model of the overall system" [Friedenthal, et al., 2012: 20]. By replacing those documents with models, Model-Based Systems Engineering (MBSE) "integrates system requirements, design, analysis, and verification models to address multiple aspects of the system in a cohesive manner" [Friedenthal et al., 2012: 20]. Such a coherent model provides many benefits to the systems engineering processes that should reduce rework through better informed decisions. However, those models are typically not designed to change the relative timing of the key decision-making in the development process (the importance of that will become clear in the next section).

Despite the many strategies and tools that have been developed to minimize the occurrence of rework, rework still consumes a great deal of product development time and resources. While it is certainly possible that engineers are not using the aforementioned techniques properly, we suggest that there are other reasons why rework remains so common.

2.2. Why Do We Have Rework?

We have visited dozens of companies (ranging in size from 35 engineers to over 10,000 engineers, from a variety of industries including automotive, aerospace, defense, construction equipment, medical devices, semiconductor, electronics, and consumer products), and have had in-depth conversations about the causes of rework within their development organizations. Most of the more significant causes can be classified into one of three general categories:

1. The development team learns something critical late in the development process that invalidates prior assumptions or otherwise causes the team to revisit a prior decision.

Example: When engineers claim that most of their rework is caused by the customer or marketing changing requirements late in the process, it usually means the team has learned something critical (in this case, what the customer really wants or what the market needs really are) late in the development cycle.

2. The development team makes critical decisions too early in the project, before they have the knowledge needed to make a reliable decision.

Example: To get projects under way, teams often make key decisions early so that plans can be formulated and assignments given, in spite of general consensus that they are just guessing this early in the project.

3. Development team members with one expertise inadvertently make decisions that overly constrain those of another expertise.

Example: Marketing or the customer requiring product specifications that are technically infeasible, or product engineering designing products too difficult to manufacture or service.

Certainly, causes outside these three general categories exist; for example, Lévárdy and Browning [2009] suggest five causes of rework in product development: poor activity sequencing, missing activities, poor communication, input changes, and mistakes. But our experience is that focusing on and addressing these three categories will reduce or eliminate much, if not most, of the rework in product development. Fewer and less severe rework efforts mean more reliable on-time performance, faster time to market, lower develop-

ment costs, fewer manufacturing problems, more innovation, better quality and customer satisfaction, and higher returns on investment.

The fundamental difficulty with conventional systems is that the knowledge needed to make good decisions in the early phases can only be generated through detailed design work. But detailed design requires that concept and systems design decisions are already made. Generating this knowledge before making those key decisions, and thus breaking the circular dependency, requires a different way of thinking.

2.3. A Different Paradigm: Set-Based Concurrent Engineering

Discovery of a different paradigm began with Ward's work to develop an automated mechanical design compiler [Ward and Seering, 1993]. His initial effort following the traditional development process proved futile: it was not possible to guarantee convergence using the conventional design-validate cycles. If it failed to converge, the software would loop endlessly without reaching a satisfactory solution. Ward then moved to a fundamentally different "set-based" approach, in which he could either guarantee convergence or rapidly determine that convergence was impossible.

With the realization that set-based design is so much more effective than traditional point-based design, Ward began searching for companies that were using set-based design in their development processes, expecting to find superior results (which he did at Toyota and its suppliers) [Ward et al., 1995]. Subsequent research expanded the theory and its application in industrial settings, resulting in a novel set of development practices termed *set-based concurrent engineering* [Sobek, 1997; Sobek et al., 1998]. In brief, the practices observed within Toyota and its suppliers include:

- Initial requirements represented as "sets" or ranges rather than point values, which are subsequently refined over the course of the development project.
- Explicit characterization of trade-offs and design limits before making key design decisions, including choice of concept solution.
- Development of sets of discrete design alternatives. The sizes of the sets are gradually reduced as the granularity of the solutions increases.
- Parallel subsystem and manufacturing development, with cross-functional information sharing to inform design decisions.

In 1997, the National Center for Manufacturing Sciences (NCMS) launched a project with a consortium of companies that "represent the mainstream of American understanding about product development." Representatives of participating companies expressed frustration with the large investment expended for improving product development that seldom reaped the promised rewards. The goal was to "search for and validate better approaches for product development improvement" [NCMS, 2000: 1-1].

To the surprise of most involved, other than terminology and superficial differences, the practices of almost all the companies were essentially the same, and they achieved similar results. Toyota, however, was an exception. Toyota's practices seemed backwards from the generally accepted best practices [Ward et al., 1995; McDevitt et al., 2004]. "Some of Toyota's practices appear irrational, such as excessive prototyping, delayed decisions, multiple design alternatives carried throughout the process, and final specifications not set until close to actual product launch" [NCMS, 2000: 3-2]. And yet, Toyota's product development performance exceeded that of their competitors by a wide margin [NCMS, 2000: B-1; Morgan and Liker, 2006; Clark and Fujimoto, 1991]. The consortium found that the set-based concurrent engineering paradigm at work at Toyota had many significant advantages.

2.4. Contribution

The key to understanding how Toyota practices generated such dramatically better results lies in understanding how they come together as a system to transform the early phases of product development. Prior work describes set-based principles as observed at Toyota and its suppliers. Here, we demonstrate how those set-based principles, when applied to the early development phases, directly address the three causes of rework articulated in Section 2.2. We show that this more effective "front-loading" of the development projects to address those causes is justified by the elimination of late-cycle design changes and rework.

Furthermore, we extend the high-level theory from prior work by defining the set-based strategies in methodological terms, making it more applicable to systems engineering practice. We then use the set-based methodology to refine a widely used systems engineering model. The result is a substantially different front end to the systems engineering process.

We now turn our attention, in Sections 3, 4, and 5, to how set-based practices can be used to remedy the three causes of rework described earlier: late learning, premature design decisions, and poor cross-functional coordination, respectively. Then, in Section 6 we propose a set of modifications to the systems engineering Vee Model, and conclude in Section 7 with a summary and managerial implications.

3. REMEDY 1: ACCELERATED LEARNING

The first cause of rework, acquiring new information late in the process that invalidates earlier decisions, can be addressed in part with set-based principles that accelerate learning. Prior work found that Toyota engineers seek to understand the design limits of new technologies before incorporating them into a new product, and seek to understand technical trade-offs for product subsystems early in the design cycle before settling on subsystem requirements. Furthermore, they extract the learning acquired from each development cycle into design guidelines that are used in the development of next-generation products.

While Toyota is a valuable example, demonstrating that these nontraditional practices can be applied successfully to complex systems, we look to a much earlier and better documented example to gain more detailed insight into how set-

based practices can accelerate learning and eliminate late-phase rework.

3.1. How to Design the First Airplane and Live to Tell about It (and Profit from It)

In the late 1800s, many talented engineers across the globe tried to build machines capable of sustained, manned, heavier-than-air flight. For these engineers, the need for rework was not only expensive, it could actually be fatal; and for some, it was.

German inventor Otto Lilienthal developed a safer small engine that gave him the financial freedom to focus on aviation, his lifelong passion. He began testing glider designs in 1885. He developed 18 gliders and one powered airplane, and made over 2000 test flights. The last, in 1896, ended in his death. In the 25 years from 1872 to 1897, with backing from the French government, engineer and inventor Clement Ader spent $120,000 on the development of designs inspired by the flight of birds and bats [Kane, 2003]. In England during the 1890s, engineer Hiram Maxim invested over $200,000 of the proceeds from his invention of the Maxim machine gun in the development of a flying machine [Eppler, 2004]. In the United States, talented and award-winning scientist and engineer Samuel Langley began experimenting with powered flight in 1887, developing dozens of model airplanes. Based on those, he was awarded nearly $70,000 in grants from the War Department, Smithsonian, and others to develop a manned airplane. Although some of his models did fly, Langley was never able to learn enough from those to design a working manned airplane [Anderson, 2002; Tobin, 2003].

In contrast, Orville and Wilbur Wright lacked engineering degrees or even high school diplomas, had little aeronautical experience, and had essentially no budget (the Wrights estimated they spent less than $1000 [Tobin, 2003]). They did know how to build machines quickly, as they owned a bicycle shop. But they succeeded whereas everyone else to that point had failed spectacularly; and they did it with a fraction of the resources and in much less time. How did they do it?

The Wright brothers shifted to a fundamentally different approach to development. The Wrights' predecessors (in some cases, competitors) approached development as a series of design–build–test cycles [Tobin, 2003]. Each test failure would be analyzed, the design modified, and the test repeated until they ran out of ideas, ran out of money, or died. Such an approach is remarkably reminiscent of approaches used in many of today's product development organizations.

The Wright brothers encountered similar failures in 1900 and early 1901 as they conducted various tests of manned and unmanned glider designs. However, in contrast to other would-be inventors of manned powered flight, the Wright brothers broke from the traditional design–build–test cycle. Through analysis of their initial results, they identified three critical knowledge gaps [Wright, 1901: 99]:

- "the construction of the sustaining wings"
- "the generation and application of the power required to drive the machine through the air"
- "the balancing and steering of the machine after it is actually in flight."

Figure 4. The Wright brothers' wind tunnel [Smithsonian Institution, SI Neg 2003-12979].

For the wings, they determined from their early failures that the airfoil data that most were using, created by Lilienthal, were erroneous. They further observed that the majority of their predecessors had spent thousands of hours designing their machines, but only a few seconds testing (before it crashed). The solution, in Wilbur's words: "We thought that if some method could be found by which it would be possible to practice by the hour instead of by the second there would be hope of advancing the solution of a very difficult problem . . . and without any serious danger" [Wright, 1901: 103].

So, rather than design and build a new flying machine in late 1901, the Wrights instead designed and built a wind tunnel (Fig. 4) and associated lift balance (Fig. 5) that would allow them to take accurate measurements of the effects of different wing designs. They would be able to learn how wing span, wing chord, angle of incidence, aspect ratio, surface area, and airfoil shape each independently affect the lift and drag, and how various combinations would work together. Their focus was on learning first via careful testing of a variety of alternative wing designs. Then based on that knowledge,

Figure 5. The Wright brothers' lift balance [Smithsonian Institution, SI Neg 2003-12980].

Figure 6. The Wright brothers' first flight [Wright and Daniels, 1903]. Public domain.

they would be able to design a wing for their flying machine for which they could accurately predict the lift/drag ratio.

The design–build–test cycles that were common then [Tobin, 2003] and remain common today [Wheelwright and Clark, 1994; Yassine et al., 2000] typically involve engineers rushing to the next full design in the hope that it will be the last. This wishful thinking results in inherently long learning cycles. Furthermore, because many design factors are changing in each learning cycle, the amount of learning in each is frequently minimal (the effects are confounded [Antony, 2003]). The Wright brothers were able to run hundreds of tests (dozens in a single day [Tobin 2003: 128]) with isolated design changes, such that the cause-and-effect relationship became clear. The result was much higher rates of learning.

That learning proved to be almost immediately reusable. After determining that the understanding of propellers from the marine industry would be as useless as their predecessors' airfoil data, the Wrights realized that a propeller is essentially a wing rotating in a vertical plane. And since the dynamics near the center of the propeller are different than at the tips, the ideal wing shape would vary from center to tip. With the knowledge of wings they had developed, they were able to build a far more efficient propeller than their competitors, one that is quite efficient even by today's standards. With a more efficient propeller, they calculated that they could use an inexpensive, lightweight, 12-hp engine (versus the 50-hp engine that Langley spent much of his budget developing for his flying machine).

In 1903, they designed a powered flying machine based on their learning during the prior 2 years (of part-time work, since their paying job was their bicycle shop), focused on solving the three knowledge gaps they identified in 1901. Their first test failed as the pilot (Wilbur) overcompensated in the use of their new control system. But in that failure, they had proven that they had solved all three of their knowledge gaps. Wilbur wrote home, "The machinery all worked in entirely satisfactory manner and seems reliable. The power is ample, and but for a trifling error due to lack of experience with the machine and this method of starting, the machine would undoubtedly have flown beautifully. There is now no question of final success" [McFarland, 1953: 393]. At about the same time, Orville sent a telegram to his father (with punctuation added): "Misjudgment at start reduced flight to hundred and twelve. Power and control ample. Rudder only injured. Success assured. Keep quiet" [McFarland, 1953: 393].

Three days later, with no design revisions (just a few repairs), the Wright Flyer flew 852 feet in 59 seconds (Fig. 6). With their new approach to product development, their first powered manned airplane design had succeeded. And they did so in a mere 22 months (spread over 4 years), with a budget of under $1000, and a staff of three (a bicycle shop employee did much of the work on the 12-hp engine). In contrast, Langley had generated numerous failed designs and much rework over the course of 16 years, spending well over $70,000 [Tobin, 2003], and consuming many engineering hours while putting his test pilots at great risk. And his manned airplane designs never flew. The contrast was not so much the individuals, but the approach: less than 3 years prior, while still operating in the traditional paradigm, Wilbur had asserted that "men would not fly for fifty years" [McFarland, 1953: 934].

The key takeaway from this story: the Wright brothers fundamentally transformed the front end of development into a sharply focused learning process, and thereby eliminated the late-process rework in which their competition was stuck. And this takeaway remains relevant today, as almost all companies that we ask answer that their current development process is more like Langley than like the Wright brothers. The transformation was accomplished through two innovations.

3.2. Limit Curves and Set-Based Knowledge

The first innovation in systems engineering approach was to synthesize and visualize their experimental data into reusable knowledge [Ward, 2007]. The Wrights created limit curves (such as the ones shown in Fig. 7) that efficiently integrated their learning into an actionable form for application to their design problems. To design a wing, now, they simply needed

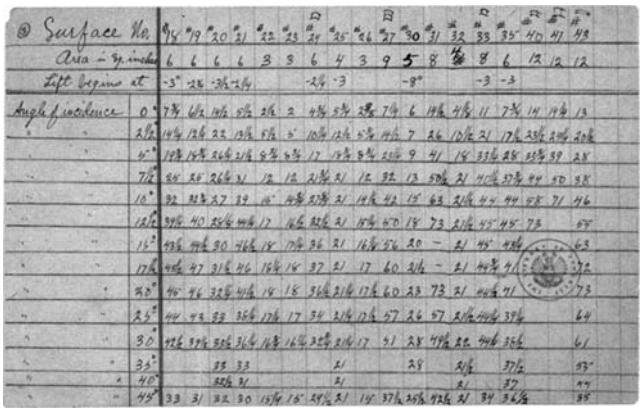

Figure 7a. Example of the detailed lift vs. angle of incidence limit curves for different surfaces measured by the Wright brothers [Wright and Wright, 1902: 32]. Public domain.

to determine the lift/drag coefficients, then "look up" the appropriate combinations of camber, wing span, wing cord, wing surface area, and so on to arrive at a known, workable design quickly and with absolute reliability. As they moved onto propellers, they added slip, throwdown, torque, thrust, and others into their list of concerns.

To better understand the power of this innovation, let us consider a hypothetical design-and-test cycle on a muffler. The goal is to minimize both the noise emitted and the back pressure that it loads on the engine, reducing horsepower. The development team works out a new design (Prototype 2.1 in Fig. 8) that is substantially better in both, but fails in reliability testing.

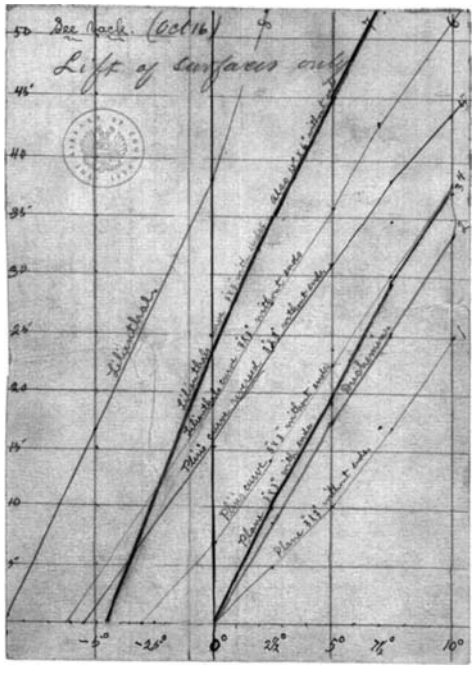

Figure 7b. Example of the detailed lift vs. angle of incidence limit curves for different surfaces drawn by the Wright brothers (including some measurements by Lilienthal, for comparison) [Wright and Wright, 1901: 72]. Public domain.

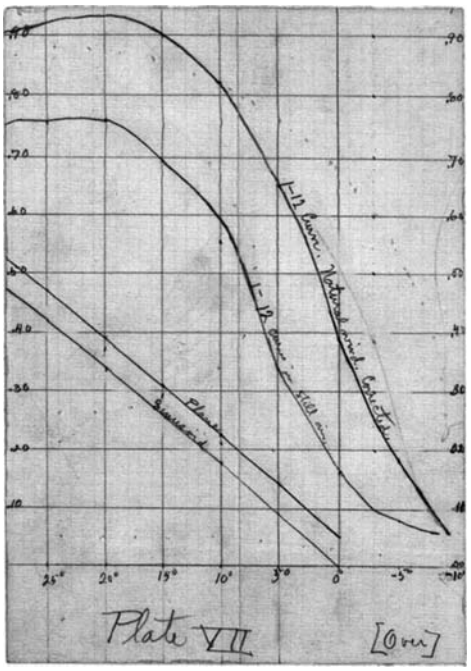

Figure 7c. Example of the detailed lift vs. angle of incidence limit curves for different surfaces drawn by the Wright brothers (showing a wider range than Fig. 7b, outside the linear region) [Wright and Wright, 1901: 76]. Public domain.

To fix reliability, they make some modifications (focusing on the reliability issues, largely ignoring the noise and back pressure issues), resulting in losing some of the noise reduction and much of the back pressure reduction (Prototype 2.2 in Fig. 8). Was that the best trade-off for the customer? Late in the project with a failing product, that question was probably not even considered!

On the next muffler project, the team shoots for improvements (Prototype 3.1 in Fig. 9), but not as much of a stretch as the failed prototype from the last project. But realistically, the team is still just guessing at what might work using the

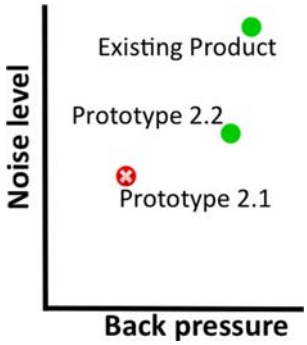

Figure 8. Point-based learning on the second-generation project. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

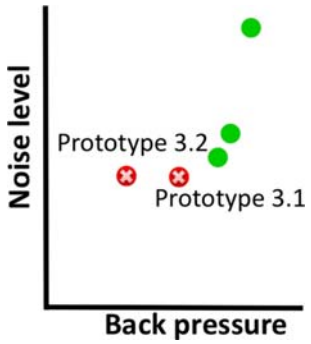

Figure 9. Point-based learning on the third-generation project. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

traditional design-then-test approach, so it is not surprising when it again has reliability issues. The next prototype (3.2 in Fig. 9) achieves the needed reliability, losing some of the gains, but still making a small improvement in both.

What did the team learn from those two projects? Clearly the three points established in the first project helped point the way to a better result for the second. But was that result the best trade-off for the customer? And if the next project calls for a different trade-off, will those five points help the team make a better design?

From those five points, should the engineering team conclude that they have reached a lower limit on noise level (that noise below the horizontal line in Fig. 10a is not feasible)? Or should they conclude that they have reached a limit on back pressure (no less than the vertical line in Fig. 10b)? More likely, the real answer is somewhere in between, like the line in Figure 10c. But what is the slope of the line, and is it linear?

The fact is the team simply does not know. They have three projects under their belt, but very little knowledge about the products they have designed! We call this approach "point-based design" because the development team is simply moving from one "point" solution to the next in search of one that will work a little better [Sobek et al., 1999].

Alternatively, the team could take a fundamentally different approach. Following the Wright brothers example, they could innovate an inexpensive way to systematically test different muffler designs to understand the true limit. If this is done on the first project, subsequent projects would not need to repeat that testing. They could simply choose where they want to be relative to the curve based on what would most satisfy the target customers for that particular product (e.g., allow more back pressure to get less noise for a luxury sedan).

By taking just enough test points to identify the limit curve, the team has available to them an infinite number of points, a "set" of possible designs, from which to choose. Any point in the safe region of Figure 11 is a valid design. Identifying limit curves for each of the key design decisions changes the development process from guess-and-test then rework (typical of "point-based design"), to reliably selecting designs that are within the known sets of what is possible (i.e., "set-based design") [see also Ward et al., 1995; Ward, 2007]. By sys-

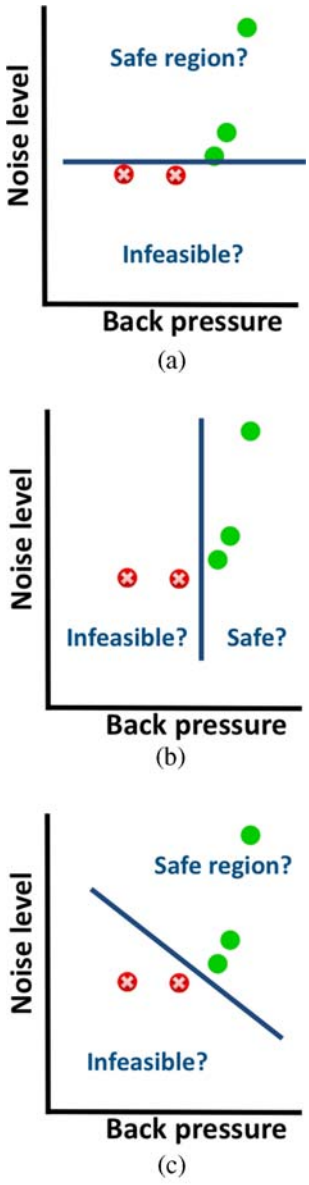

Figure 10. Possible interpretations of the five point designs tested. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

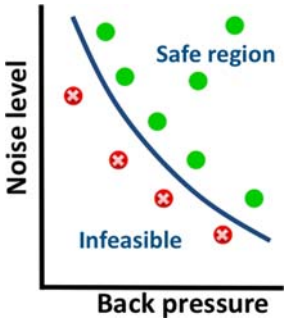

Figure 11. Testing to find the limits reveals the true shape of the limit curve. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

tematically identifying the limits of product technologies, manufacturing process capabilities, user preferences, market conditions, and so forth *before* making key decisions on the final design, critical information is far less likely to be uncovered late in the project.

Teledyne TapTone applied this approach to the design of a new automated leak detection machine for high-speed bottling operations. The planned timeline for a product development project of this nature was 11 months from project kickoff to start of production. However, they never met that timeline—projects always took at least 7 months longer than planned due to rework. When the company president decided to try a test-then-design strategy on a new project, members of the development team (especially sales and marketing) were skeptical, particularly since the target trade show was only 11 months away. (To the objections, the president argued that they were never on time anyway.)

They started by identifying the customer interests for this product, and then characterizing which of their design decisions affected those interests. Much of this learning was done through inexpensive testing using the current machine and modifying it, such as adding clamps to increase stiffness, or swapping sensors of varying sensitivities. Eight months into the project (two-thirds of the normal planned timeline), the engineering team did not have a single drawing because they had spent all of their time up to that point learning.

This unorthodox approach made the marketing and sales members of the team very nervous about making the trade show, but the company leadership stayed the course. After having closed all the known knowledge gaps, the engineering team finally began creating production drawings. The team lead observed, "There was no fear. We knew as fact what our customers needed and what our capabilities were" [Kennedy et al., 2008: 248]. It took just 4 weeks to complete the set of drawings due to the amount of knowledge they now possessed, and the team experienced no significant rework as the project entered the beta testing and production release phases. As a result, they made the trade show with a higher-quality machine than ever before, and then started full production 2 months later. This timeline was significantly faster than any project of a similar magnitude at that company. And, since the team focused so much on understanding their customers' interests, the product outsold all projections, and became one of the most profitable in the company's portfolio.

Better yet, the next development project targeting leak detection in cans rather than plastic bottles was able to leverage the reusable knowledge to complete in only 9 months (half the previously typical 18-month cycle time).

Although we have illustrated the idea of limit and trade-off curves using x–y plots of two variables, the concept can be extended to multidimensional relationships (surfaces). This can be accomplished through mathematical functions, through meaningful combinations of variables that consolidate several variables into one, or through multiple connected 3D charts.

3.3. Systematic, Innovative Testing

The second innovation of the Wrights' systems engineering approach was quick and effective means to test before design, which was needed to generate the data for the trade-off curves just described. The Wrights tested over 100 wing designs in a 4-week period [Tobin, 2003: 128]. From the carefully tabulated data relating decision variables to performance measures, they were able to design and build an accurate and reliable wing on the first try. On reflection, Wilbur Wright stated, "…as famous as we became for our 'Flyer' … it all would never have happened if we had not developed our own wind tunnel and derived our own correct aerodynamic data" [American Institute of Aeronautics & Astronautics, 2012].

Of course, to do the testing, they needed airfoil designs to test. But they were not designing wings to meet a set of airplane specifications or customer requirements. Rather, they were systematically varying key design parameters within a wing design to cover a large range of possibilities, testing each in isolation. The goal was to understand the effect that each design decision would have on the performance characteristics of the wing. The amount of design effort involved in creating the next wing design to test was minimal. In the end, they had a body of knowledge from which to design a wing to meet any given set of specifications (or know quickly that the specifications were infeasible).

This approach is more like "design of experiments" (DOE) than traditional product design. The goal of DOE is to achieve the maximum learning with the least amount of testing effort. And the means to that end is being able to determine the isolated effects of particular design changes by leveraging statistical methods rather than brute force one-factor-at-a-time testing [Antony, 2003; Tucker and Dagli, 2009]. Thomke [2003] elaborates on the many potential benefits of front-loading projects through intentional and systematic experimentation. The key is a focus on maximizing learning rather than on hopefully satisfying the final specifications.

The most common objections to the test-before-design strategy are that there is not enough time or that it will be too expensive. But the Wright brothers had minimal resources, a small team, no existing wings available for purchase, no prior working products to modify, and no test equipment. Few development organizations can claim such an austere environment in which to operate. Yet they designed, fabricated, and tested more than 200 wing samples in just 2 months while simultaneously keeping their bicycle shop running [Anderson, 2002: 107; Jakab, 1990].

The key is to focus on designing the minimal tests that will yield sufficient data needed to close the identified knowledge gaps. This does not mean designing, building, and testing full system prototypes (which *is* time-consuming and expensive) but rather innovating ways to test (via prototype, simulation, or analysis) the critical elements of a system quickly and inexpensively. Thomke [2003] recommends rapid low-fidelity experiments in early phases to learn about the consequences of design decisions before making them. Multivariate methods such as DOE or Taguchi methods can also be used to identify the minimal set of tests that will yield the desired information. Often the testing required is surprisingly small.

For example, Teledyne TapTone needed to understand the impact of rigidity on the leak detection performance. Rather than designing numerous alternative machines with different rigidity, and building prototypes or simulations to test (which

would be time-consuming and costly), they fashioned a test using parts on-hand and some $14 carpenter clamps to vary the rigidity of an existing leak-detecting machine. In a few hours of testing, they had limit curves that showed they did not need nearly the level of rigidity that they thought they needed, resulting in a less expensive design with a smaller footprint.

Leveraging the latest software technology can be another source of innovation in early testing. Mathematical and simulation tools can allow rapid construction of tests that are much quicker and lower cost to conduct than hardware tests. For example, consider how much more can be learned from a crash simulator configured to allow quick variation of the design being simulated versus relying on physical crash tests [Thomke, 2003].

3.4. Proving "Success Is Assured!" as Early as Possible

Numerous success criteria exist for new product development projects, and the most suitable for a given project depend on the project strategy and company's business strategy [Griffin and Page, 1996]. Regardless of the criteria used, teams would like to know as early as possible that their design approach will succeed. For many projects, supplying the customer with a reliable solution to their needs while earning a profit would define success. The question then becomes, at what point do development teams *know* their product design will work reliably, satisfy the target customers consistently, and be producible profitably? In other words, at what point are they certain that rework to fix earlier decisions will not be necessary?

The systems engineering Vee Model highlights the need to ensure that the verification and validation testing at the back end of the project align with the system specifications and user requirements (respectively) identified at the front end [Forsberg and Mooz, 1991; Forsberg et al., 2005]. Project success is assured, then, at the back end, which we assert is far too late in the process.

We are, of course, not the first to assert this. Gated models of development include early reviews or gates that are intended to identify projects doomed to failure as early as possible. However, whether intended by those models' original designers or not, it is clear that most modern applications of those gated product life cycles do not insist on any level of proof that success is assured until verification and validation testing. Rather, those earlier gates are geared to identify risks and kill projects where the risks are too high.

Given how costly late-phase rework is, we feel every company should be transforming their product development process such that they can add "Is success assured?" to their gate criteria before entering the detailed design phase when the rate of investment increases considerably. This is effectively accomplished if design teams know the critical design limits and trade spaces of the design alternatives being considered.

4. REMEDY 2: DELAYING CRITICAL DECISIONS UNTIL THE KNOWLEDGE IS LEARNED

For insight into how to deal with the second major category of rework causes, making critical decisions with insufficient knowledge, we look at some of the earliest decisions in the systems engineering process. Traditional systems engineering begins with requirements documents and then proceeds to development of specifications documents, starting at the system level, and then allocating down to the lower levels, eventually feeding into detailed design at the lowest levels (as depicted graphically on the left side of the Vee Model) [Forsberg and Mooz, 1991; INCOSE, 2011: 7, 27–32; Marchant, 2010: 3–4].

The typical goal in this work is to get the requirements and specifications nailed down as early as possible, with sufficient precision that they can be used in the verification and validation testing on the right side of the Vee Model. The process allows for those requirements and specifications to be modified as more is learned, but the preference is that they not be modified. And once approved early in the process, those documents are recommended to be put under change control [Forsberg and Mooz, 1991], which (intended or not) will tend to inhibit change (and thus inhibit learning).

However, tightly specifying requirements early in the project means that some of the most critical decisions are made very early. If they are made with too little knowledge of what customers really want or what is technically possible, then rework is inevitable. On the other hand, starting a development project with no clear direction is a recipe for disaster. So what is a systems engineer to do?

As discussed earlier, Forsberg and Mooz [1991] encourage the systems engineers to actively resolve the "critical issues" so as to ensure the requirements and specifications documents are accurate as early as possible. But set-based practices suggest an alternative, potentially complementary, approach.

4.1. Set-Based Requirements Definition

One of Toyota's "irrational" practices was their propensity to delay decisions, in particular delaying the setting of requirements and specifications. While all other companies scramble to nail down requirements and specifications as early as possible, Toyota actively tries to delay those decisions, making them in stages [Sobek, 1997].

A common technique for dealing with uncertainty in what the customer wants or needs is to express requirements as ranges of values, such as a minimum "acceptable" level and a "goal" level. Set-based practices extend this idea more broadly and generally.

Design requirements should embody a deep understanding of both the customers' interests and the business's interests such that designers can properly evaluate the trade-offs between requirements. We use the term *customer interests*, rather than *requirements*, because high-performing design teams should be knowledgeable about what the customer wants from the product, whether it is possible for the product to fulfill it or not. The project may target only a subset of those interests, but that does not change what the customer is interested in.

Business interests are equally important. Target profit margin, use of existing manufacturing lines, competitive positioning, and so on must be explicitly captured and well-understood, or it will not be possible to evaluate the trade-offs with the customers' interests. Projects should not simply produce a product design; they should design a profitable operational value stream for delivering that product [Ward, 2007].

Finally, set-based requirements may not be adequately expressed as a list of numbers or even ranges, but rather must provide the knowledge needed to properly evaluate trade-offs. Since the nature of the trade-offs emerges as the design develops, the convergence of set-based requirements will necessarily be an ongoing learning process.

Hence, operating set-based dramatically changes the dynamics between engineering and the customer and/or marketing. We frequently hear engineers complain of late changes from the client or marketing. In response, we often ask, "So, if marketing or product management learns more about your customer late in the process, are you saying you want to ignore that learning?"

Both sides should recognize that the learning comes from both directions. Set-based requirements enable engineers to learn about technical trade-offs. They can then present the trade-offs to the customer or marketing to learn how the customer would want to make those trade-offs. These interactions, combined with engineering design effort, reveal the most important trade-offs that must be learned in order to converge the set-based requirements into the final requirements that will be used to drive the system validation at the end of the project.

For example, Toyota development teams, who were operating set-based, put together targets for fuel economy, vehicle weight, acceleration, overall vehicle dimensions, and cost in the early concept phase of development [Sobek et al., 1999]. Their objective was to express the range of what is appealing to customers, what will sell, but with minimal constraint on the development team and on the competing customer interests. This gave the team flexibility to learn about the trade-offs associated with various decisions. As they learned, they narrowed the target ranges to the final requirements at a point in the process when they had the knowledge to make decisions that would not have to be revisited later [Sobek, 1997].

Gradual convergence of requirements allows for deeper learning about some of the finer details that may be overlooked until the customer can be presented a more concrete picture. Returning to the Teledyne TapTone example, after achieving "success is assured" on the original set of customer interests, the development team decided that it was worth a second visit to their customers for additional learning (despite marketing's objections that there was too little time prior to the upcoming trade show). Feedback on the graphical mockups yielded new and more detailed requirements, such as using rounded stainless-steel legs versus rectangular legs made from cast iron to make it easier to clean, and metric-sized parts for global deployment. Had the team been in detailed design, such changes would have required tremendous rework. However, given that no solid models, engineering drawings, or tooling had been created yet, the new customer interests could be accommodated easily [Kennedy et al., 2008].

4.2. Set-Based Specifications Definition

A similar set-based approach can be applied to the specifications of the system and its hierarchy of configuration items (subsystems, assemblies, subassemblies, and parts) that it designs or outsources. For example, Toyota and their supplier Denso communicate specifications initially as ranges rather than points, and set the final specifications based on the learning gained from investigating the alternatives within those ranges. The interfacing developers can then feel comfortable working within those design windows, free to choose any point in that window that works best, given the issues being addressed at that next level of detail.

An example of this can be seen in the functional build of car bodies. In the interface between product design and manufacturing in the development of a car body, Toyota styling and body engineering teams send sheet metal part design data to manufacturing without tolerances on most of the dimensions [Majeske and Hammett, 2000]. The die design team then designs the dies to produce the body parts as close to nominal as possible. But since stamping is not an exact science, the parts produced deviate from the nominal dimensions. They then assemble the parts to see how the body looks (including visually consistent spacing on all seams, since that is a customer interest), and make the least expensive changes to meet the design intent. It does not matter if a given part dimension differs from nominal by 2 or 3 mm as long as the overall system looks and performs as intended. In this case, the design teams are placing the minimum amount of constraint on their interfacing team (die engineers in this case), implicitly acknowledging that a set or range of solutions is acceptable.

In contrast, the traditional approach would place tight tolerances on all part dimensions. Parts would be forced to conform to the tolerances before the dies would be accepted, typically incurring much rework to meet tolerances that were not really required to achieve the design intent. Then the parts are assembled and more changes made (more rework) when they do not fit quite right. The end result is a much lengthier and more costly die development process, with no improvement in achieving the real design intent.

With outsourced parts that are engineered and produced by a supplier, development teams can similarly communicate initial specifications as ranges in their request for proposals [Ward et al., 1995]. They ask the supplier for trade-off data among the various design parameters over the ranges of interest (and they make sure the engineering knowledge is there to back it up!). With that, they can then converge to the final specifications with the knowledge of what trade-offs are involved with the decisions, avoiding rework late in the project.

The move from point-based to set-based specifications may be the most significant and valuable transformation to traditional systems engineering. It eliminates much risk by defining windows of feasibility within which to work. It increases the level of innovation and optimization through increased design freedom. It allows teams to accommodate changing market requirements and other (dreaded) scope

creep later in the process if the changes do not stray out of the sets. Ultimately, "success is assured" once the set-based specifications are supported by limit curves such that, if the team stays within the ranges of the set-based specifications, the team *knows* that the design will work. Once that is established, teams are free to explore and optimize without fear of late design changes.

Hence, "set-based specifications" are specs for a system or configuration item that are defined in terms of ranges or lists of options that together define a set of designs that is being evaluated as a whole set, *rather* than a set of point-designs that are each being evaluated as individual points. And "set-based requirements" are the target ranges that the customers and the business are interested in the product satisfying, coupled with their preferences for the trade-offs that currently exist between those various requirements, as driven by the emerging set-based design.

4.3. Set-Based Management of Major Alternative Concepts

Not all decisions are amenable to limit curves, nor are they a matter of degree. Development teams also face qualitatively different approaches, such as choice of technology (e.g., wireless versus cable) or material (e.g., metal versus polymer), that lead to substantial differences in the design being analyzed and therefore substantial differences in the lower-level requirements.

A common approach to the "concept selection" problem is to evaluate the alternatives, often using an evaluation matrix, then to select the best perceived concept for the systems and detail-level design phases. This produces rework as this critically important choice is frequently made without a clear understanding of the technical, business, manufacturing, and supply chain implications. It is, by its very nature, a high-risk strategy. However, acquiring that understanding with traditional development practices would require launching full development projects for each alternative, which would be far too expensive in almost all cases.

Set-based practices offer a more efficient and robust decision process. Rather than trying to select the best alternative, development teams seek data that will eliminate the worst alternatives. For this to work economically, teams should seek out the fastest, quickest way to disqualify an alternative (e.g., do back-of-the-envelope calculation before developing a product simulation model, or perform quick subsystem tests on modified production units rather than build system-level prototypes). Alternatives that survive this process have demonstrated some level of robustness. Teams can then learn the design limits and trade-offs of the alternatives using the approaches discussed earlier (including test-before-design to thoroughly understand each alternative as quickly as possible, and set-based specifications to allow room to explore, innovate, and make wise trade-offs) in order to optimize the design if one alternative remains; or to continue eliminating the weaker alternatives if more than one is in contention.

Automotive companies often face this decision in the styling design of their vehicles, a very important customer interest. For major vehicle programs, the Toyota styling team generates many ideas as sketches and renderings. They then eliminate alternatives and combine alternatives into new designs, and the subset is further refined and developed via 3D solid modeling or scaled clay models. After evaluation of those, typically at least two alternatives are selected for full-scale modeling before the final design is chosen. Elimination decisions are made in stages to give time for other groups, such as marketing, product engineering, and manufacturing, to do the required design and analytical work to understand the ramifications of alternative styling designs on their own subsystems (is there a feasible design for their item that fits within the styling constraints?). In this way, development converges to a final design that works from all perspectives, minimizing rework [Sobek, 1997].

The vice president of engineering of Nexen, a small manufacturer of industrial motion control and power transmission devices, actually encouraged sets of alternatives through a program he called 4-2-1 [Hein, 2007]. For each new development program, he required solid models of four distinct design alternatives, then prototypes of two alternatives before selecting the final alternative. One of the challenges in instituting this program was getting his teams to generate four viable, competitive ideas. But after some training in creativity techniques (and insisting that projects would not move forward without four distinct, viable alternatives), the development teams overcame the mental blocks. In the first few months of implementation, the engineering organization saw a measurable dip in productivity. But within 18 months, their productivity soared as they began to create more innovative products and as rework decreased. In addition, they soon found that alternatives not selected in one project were often useful on future projects. The VP credits this set-based program with helping engineering achieve a 50% increase in productivity over a 2-year period.

Parallel exploration of alternatives, which is similar to selectionism [Sommer and Loch, 2004], can require additional resources upfront [Gil and Beckman, 2007]. That investment is likely worth it if the uncertainty as to which alternative is most desirable is high, the learning from parallel exploration is high, and the cost of rework is high [Sommer and Loch, 2004]. Also, the investment is even more likely to pay off if the learning produced is likely to be reusable on future projects, based on a vision of where the market is going (as illustrated in the Nexen example).

5. REMEDY 3: SET-BASED CONCURRENT ENGINEERING

We now turn our attention to the third major cause of rework: development team members of one expertise overly constraining those of another. There has been considerable interest in concurrent engineering since the early 1990s, when engineering was characterized by "throw-it-over-the-wall" design practices, particularly between product engineering and manufacturing process engineering [Chapman et al., 1992]. Rework was common across that interface because it was a point at which both the available expertise changed and dominating customer interests changed, meaning that the design work to that point usually did not reflect the expertise and concerns that followed. Many cost overruns and ineffi-

ciencies in manufacturing were caused by product engineers who, unaware of the limiting constraints in manufacturing, made design decisions that painted the manufacturing engineers into artificial corners. Worse, due to the extreme expense of rework that late in the process, the result was often manufacturing inefficiencies that would live on throughout the production life cycle.

The split in organizations and expertise also gave that rework a lot of attention as fingers would often be pointing in frustration when projects encountered delays. The problems showed up while the ball was in the manufacturing process engineers' hands, but due to decisions made earlier in product engineering.

The original solution to this mess was to break down the walls between functional groups, putting in place "concurrent engineering" practices where all departments started working on their piece of the development project early, such that their concerns and expertise would get equal weight in influencing the course of development. In theory, it would allow higher-quality designs, much faster, and with less costly rework.

Unfortunately, given point-based specifications and design-then-test processes on point solutions, such concurrent engineering has typically been far too inefficient. Downstream teams, such as the manufacturing engineers, quickly tire of redoing their design work in response to the many design changes within the product engineering effort, for example. The early design decisions are simply too unreliable and rework to change those decisions too common. As a result, most concurrent engineering efforts degenerate into simply involving representatives from other areas as advisors, to hopefully avoid the larger design mistakes. That certainly helps, but has resulted in only a slight reduction in the frequency and impact of rework.

The set-based practices described above offer the opportunity to completely revolutionize concurrent engineering. First, if one area identifies the key limits that others must respect, and represents them visually in a way that others can easily apply, that knowledge can then be communicated early in the project so that it is taken into account at a point when the development team has more flexibility. While it may require deep expertise to identify and determine all the key sets and limit curves that need to be considered, the sets and limit curves can be applied by team members with less expertise and in a fraction of the time, particularly if they are approaching their development tasks in terms of sets and limit curves anyway.

For example, in analyzing paint adhesion problems, the manufacturing engineers at a small manufacturer determined that more expensive and/or slower painting equipment would be needed to get the paint into sharp corners. By testing those limits, they determined the trade-offs between the sharpness of corners and the expense and speed of the painting equipment. Armed with that knowledge, the upstream product designers were often able to reduce the sharpness of such corners, allowing for much lower cost manufacturing, without trying to concurrently design the product and painting process, or otherwise needing to directly involve the manufacturing designers.

Second, by moving from point-based to set-based specifications and designs, the specifications going over the "walls" are now the widest sets that the team could determine would satisfy their customer interests, leaving as much design space as possible for the downstream team. Rather than artificially overspecified designs forcing expensive, nonoptimal decisions, they are providing clear, though bounded, flexibility within which the interacting team can optimize their part of the development work for the good of the whole, as was illustrated with the earlier die design example.

Finally, if rework does occur, rather than finger-pointing, both sides should recognize that they have identified a knowledge gap. Thus, they should cooperate on a design fix for the current project, and define new sets or revise limit curves to guide future development work so that such rework will never occur again.

Thus, "set-based concurrent engineering" is the collaborative set-based design work to converge set-based specifications and requirements based on the set-based knowledge learned by all (in and out of engineering) to prove that "success is assured" for any design in the remaining set.

6. A SET-BASED FRONT END FOR SYSTEMS ENGINEERING

The remedies for late-phase rework have significant implications for the front end of systems engineering. We illustrate those implications by proposing a set of concrete changes to the front end of the systems engineering Vee Model described in Section 2. The changes are concentrated in the "off-core" activities associated with the left side of the Vee.

The start of detailed design is an important boundary in any project cycle. First, detailed design necessarily involves moving into the world of CAD/CAM/CAE tools, SPICE-based simulators, and so on which require highly detailed models. Second, the start of detailed design marks a significant increase in development costs. Taken together, this means that most of the critical design decisions cannot be delayed beyond that point. Like it or not, most if not all will have to be made to allow detailed design to proceed. Thus, the traditional specifications documents that feed the traditional detailed design and back end of the Vee can be required to emerge by that point, since most of the learning needed to be complete by then anyway. (Of course, some decisions can be allowed to be made later, extending set-based practices into the back end, but that is beyond the scope of this paper.)

Unfortunately, the traditional use of point-based requirements and specifications before that point (the vertical line in Fig. 12) can encourage or even force decisions earlier than ideal (before the knowledge exists to make those decisions properly). We suggest augmenting the process model with a clear set-based front end feeding more traditional specifications to the detailed design phase, just in time, such that the left side of the Vee (or a linear process model) becomes a gradual (not to be confused with slow) convergence process.

To show this more concretely, we modify the Vee off-core activities (shown in Fig. 3) [Forsberg and Mooz, 1991] to depict specific learning processes for the front end. These learning activities start with set-based requirements and set-based specifications that progressively converge until the start of detailed design (as shown in Fig. 13). In this way, rather

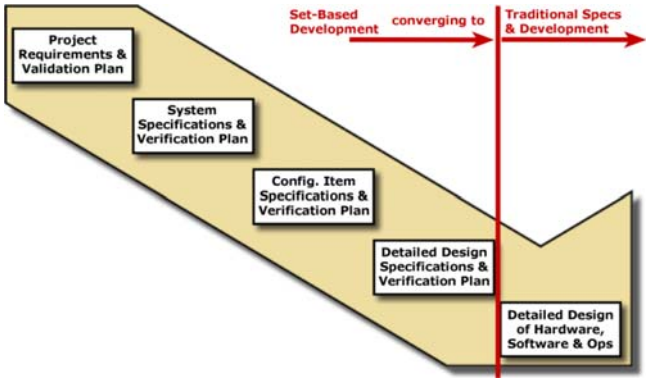

Figure 12. A natural transition point from predominately set-based learning to more traditional development work. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

than dealing with the complexities of the change review processes used with point-based specification documents, the set-based requirements and specifications naturally pull the behaviors that create deep learning for better design decision-making.

When operating the front end set-based, each phase of the process converges the design at the upper levels based on continued learning at the lower levels. The core activities on the Vee Model indicate the level at which decisions are *beginning* to converge, a process that may continue until shortly before detailed design. Hence, the boxes from the core to the right are primarily convergence activities as narrowing decisions are made.

The boxes to the left of the core represent learning activities, pulled by the needs of the convergence decisions above. For each decision to be made, the team must understand how that decision will impact the trade-offs between the competing customer and business interests (as discussed earlier). To make those connections, the possible impacts (which depend on the various design alternatives) must be identified. The base limits that constrain each design decision will ultimately impact the customer and business interests; thus, those impacts must be discovered and understood. In complex, real-world projects, those impacts tend to be broad and diverse, necessitating strong modeling tools (such software tools have been emerging recently). Toyota has used their engineering checksheets to capture those impacts and ensure they are not forgotten on future projects.

In conventional systems engineering practice, the requirements and specifications documents are reviewed as if they are final. Once approved, the documents are put under change control, even though changes are expected. Revising and reapproving such "hoped to be final" documents is inefficient and cumbersome, which becomes a strong impediment to continued learning and innovation.

In contrast, in Figure 13 final requirements and specifications emerge or get approved just before entry to detailed design (the red line). Prior to that event, set-based requirements and specifications are used to encourage and facilitate ongoing learning and innovation in order to discover the best trade-offs in requirements.

Note that, unlike Figure 3 where the "Critical Issues" boxes do not connect horizontally to later phases, the boxes in Figure 13 do connect horizontally, indicating the ongoing evolution of the set-based knowledge at each level as the design decisions continue to converge at each level, influenced by the new knowledge learned from below and the new trade-offs made from above, reflecting customer feedback on the trade-offs just learned. Figure 13 depicts a continuous learning process!

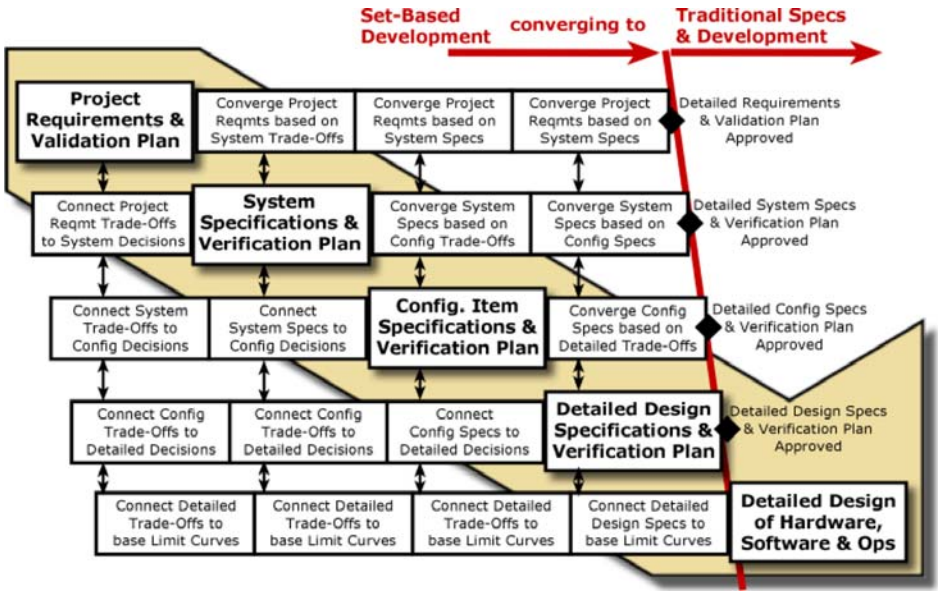

Figure 13. Set-based front end on the Vee Model. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

That seamless flow to later phases is only possible by virtue of the continued applicability of set-based knowledge (such as that depicted in Fig. 11) to the set-based specifications as they converge. In contrast, knowledge of point solutions is often of little use in later phases if the specifications have changed; its applicability is inherently unclear (as depicted in Fig. 10).

Finally, gradual convergence of the requirements and specifications increases the effectiveness of concurrent engineering. The bottom two levels of boxes are actually multiple rows on each level as the system is decomposed into multiple configuration items (segments, subsystems, assemblies, subassemblies, parts, and components) and on down to the lowest-level functional teams (hardware, software, and operations), where the deep expertise lives. Using conventional approaches, those experts are asked to create designs for specifications that will almost surely change. With the set-based specifications of Figure 13, those experts are being asked to prove that they know how to design a solution that can cover the full set specified. If they cannot do so with confidence, then either they will ask for time and resources to do more early learning or they will ask to reduce the set (eliminating what they are not confident that they can do).

This difference between point-based and set-based collaboration has been observed in the dialog at Toyota. At other automotive companies, the systems engineers tend to come up with allocated specifications and ask the lower-level teams or suppliers, "Can you do this?" In contrast, Toyota systems engineers tend to express what they are trying to accomplish and ask the functional teams or suppliers, "What can you do for me?" The answer to the latter question is naturally a set of possibilities that the systems engineers know how to leverage in their decision-making. The answer to the former is simply a judgment of a point, which either initiates rework or not.

7. ELIMINATING THE CAUSES OF REWORK

In summary, adopting set-based practices in a systems engineering process that encourages learning first (before committing to early decisions) can eliminate rework at the root cause. Further, since the set-based knowledge generated is often reusable on future projects, the positive benefits often multiply several times over. Thus, the remedies for each of the three causes of rework identified earlier can be summarized as follows:

1. Cause: Teams learn something critical late in the development process.

Remedy: Replace design-then-test with test-before-design to accelerate learning in the early phases.

Remedy: Learn the sets of possibilities rather than single points, capturing set-based knowledge (such as limit curves) that covers the full range of possible designs.

2. Cause: Teams make critical decisions too early in the design process, before they develop the knowledge needed.

Remedy: Specify customer and business interests as target ranges, giving the development teams room to explore, innovate, and find the most appealing trade-offs for both the business and customer interests.

Remedy: Use set-based specifications in the early phases of development, and allow the final specifications to emerge from the learning in the convergence process.

Remedy: Investigate alternative ideas in parallel when uncertainty is high or when teams must select from among fundamentally different technologies or design approaches, but do so with a focus on quickly identifying and eliminating the weak alternatives.

3. Cause: Designers with one expertise paint designers of another expertise into corners that will not work.

Remedy: Leverage set-based knowledge to communicate the key issues from one area of expertise to another, breaking down the walls, without requiring heavy involvement.

Remedy: Leverage set-based specifications to minimize the restrictions on later design phases, maximizing the design windows within which they can optimize without risking later rework.

Putting the remedies into action can be challenging because it represents a radical change in thinking. The entire organization (including marketing, product management, engineering, manufacturing, and management) must let go of the traditional point-based thinking that forces premature decision-making. However, as an old adage states, "It is easier to act your way into a new way of thinking than to think your way into a new way of acting." Below we recommend a set of actions that systems engineering teams can take to implement the remedies above. Note that these actions are not independent; they build on one another to form a new system for the front end that is a rigorous managed learning process.

- Stop using point-based requirements and specifications as the controlling document for development projects. Rather, get everyone collaborating on a set-based specification document that is the *outcome* of the front end.
- After identifying the targeted customer interests, list the key knowledge gaps that must be closed in order to assure success. Teams should include gaps in their knowledge of the customer interests as well as technical knowledge gaps.
- Encourage teams to innovate ways to quickly and efficiently close the identified knowledge gaps; innovation in learning should be as valued as innovation in design.
- Capture the learning through limit curves and other devices that allow development teams to visualize and understand the trade-offs over a range of decision variables.
- Have teams create project plans that show how and when they will generate the knowledge needed to make good decisions. Plan backwards to find the latest possible convergence date for each decision, and use those key convergence dates to pull the development process by establishing due dates for that knowledge.
- Do not approve decisions where there is insufficient knowledge to back them up, and resist the urge to make decisions in order to "move things forward" when there is no grounding in knowledge.

The lessons learned from the Wright brothers and Toyota point toward a vastly superior way of approaching the research and development of new products and services. These ideas have been successfully applied across industries, and at companies of varying sizes. Every development team should be able to say, even before the start of detailed design, "We have the knowledge to show that this design will satisfy these customer interests in this timeframe. No fear—this project's success is assured."

REFERENCES

American Institute of Aeronautics & Astronautics (AIAA) Wright Flyer Project, http://www.wrightflyer.org/WindTunnel/testing1.html. Last accessed September 20, 2012.

D.M. Anderson, Design for manufacturability and concurrent engineering: How to design for low cost, design in high quality, design for lean manufacture, and design quickly for fast production, CIM Press, Cambria, CA, 2010.

J.D. Anderson, The airplane: A history of its technology, American Institute of Aeronautics and Astronautics, Reston, VA, 2002, pp. 64–78.

J. Antony, Design of experiments for engineers and scientists, Butterworth–Heinemann, Burlington, MA, 2003.

B.W. Boehm, "A spiral model of software development," in R.H. Thayer (Editor), Tutorial: Software Engineering Project Management, IEEE Computer Society Press, Washington, DC, 1988, pp. 128–142.

T.R. Browning, Process integration using the design structure matrix, Syst Eng 5 (2002), 180–193.

W.L. Chapman, T. Bahill, and A.W. Wymore, Engineering modeling and design, CRC Press, Boca Raton, FL, 1992, p. 326.

N. Chucholowski, S. Langer, M.G.G. Ferreira, F.A. Forcellini, and A. Maier, Engineering change management report 2012: Survey results on causes and effects, current practice, problems, and strategies in Brazil, Universidade Federal de Santa Carina report, 2012, http://orbit.dtu.dk/fedora/objects/orbit:111066/datastreams/file_7873060/content. Last accessed: December 7, 2012.

K.B. Clark and T. Fujimoto, Product development performance: Strategy, organization and management in the world auto industries, Harvard Business School, Boston, 1991.

P.J. Componation, D.R. Utley, and R.L. Armacost, Prioritizing components of concurrent engineering programs to support new product development, Syst Eng 2 (1999), 168–176.

R.G. Cooper, Winning at new products, Addison–Wesley, New York, 1986.

S.D. Eppinger, Model-based approaches to managing concurrent engineering, J Eng Des 2 (1991), 283–290.

S.D. Eppinger and T.R. Browning, Design structure matrix methods and applications, MIT Press, Cambridge, MA, 2012.

M. Eppler, The Wright way: Seven problem-solving principles from the Wright brothers that can make your business soar, Anacom, New York, 2004, pp. 19, 66.

D.N. Ford and J.D. Sterman, Overcoming the 90% syndrome: Iteration management in concurrent development projects, Concurr Eng Res Appl 11 (2003a), 177–186.

D.N. Ford and J.D. Sterman, The liar's club: Concealing rework in concurrent development, Concurr Eng Res Appl 11 (2003b), 211–219.

K. Forsberg and H. Mooz, The relationship of system engineering to the project cycle, Proceedings of the National Council for Systems Engineering (NCOSE) Conference, Chattanooga, TN, 1991, pp. 57–65.

K. Forsberg, H. Mooz, and H. Cotterman, Visualizing project management, 3rd edition, Wiley, New York, 2005, pp. 245–247.

E. Fricke, B. Gebhard, H. Negele, and E. Igenbergs, Coping with changes: Causes, findings, and strategies, Syst Eng 3 (2000), 169–179.

S. Friedenthal, A. Moore, and R. Steiner, A practical guide to SysML: The systems modeling language, Morgan Kaufman OMG Press, Waltham, MA, 2012, p. 20.

N. Gil and S. Beckman, Design reuse and buffers in high-tech infrastructure development: A stakeholder perspective, IEEE Trans Eng Manage 54 (2007), 484–497.

A. Griffin and A. L. Page, PDMA success measurement project: Recommended measures for product development success and failure, J Prod Innov Manage 13 (1996), 478–496.

A. Hari, S. Shoval, and J. Kasser, Conceptual design to cost: A new systems engineering tool, 18th Annual International Symposium of INCOSE, 6th Biennial European Systems Engineering Conference, Utrecht, Holland, June 17, 2008.

D. Hein, 4-2-1 set-based at Nexen, unpublished presentation, Minneapolis, MN, June 14, 2007.

INCOSE systems engineering handbook: A guide for system life cycle processes and activities, International Council on Systems Engineering, San Diego, CA, 2011.

P.L. Jakab, Visions of a flying machine: The Wright brothers and the process of invention, Smithsonian Institution, Washington, DC, 1990.

T.A.W. Jarratt, C.M. Eckert, N.H.M. Caldwell, and P.J. Clarkson, Engineering change: An overview and perspective on the literature, Res Eng Des 22 (2011), 103–124.

R.M. Kane, Air transportation, Kendall Hunt, Dubuque, IA, 2003, p. 48.

M.N. Kennedy, J.K. Harmon, and E.R. Minnock, Ready, set, dominate: Implement Toyota's set-based learning for developing products and nobody can catch you, Oaklea Press, Richmond, VA, 2008, pp. 247–248.

C. Larman and V.R. Basili, Iterative and incremental development: A brief history, IEEE Comput 36 (June 2003), 47–56.

V. Lévárdy and T.R. Browning, An adaptive process model to support product development project management, IEEE Trans Eng Manage 56 (2009), 600–620.

K.D. Majeske and P.C. Hammett, The functional build approach to tolerance development, IEEE Trans Eng Manage 47 (2000), 493–496.

A.B. Marchant, Obstacles to the flow of requirements verification, Syst Eng 13 (2010), 1–13.

C.A. McDevitt, E.C. Cahill, and C. Stambaugh, System-level application of the evolutionary product development process to manufactured goods, Syst Eng 7 (2004), 144–152.

M.W. McFarland (Editor), The papers of Wilbur and Orville Wright, McGraw–Hill, New York, 1953.

M.E. McGrath (Editor), Setting the PACE in product development: A guide to product and cycle-time excellence, Butterworth–Heinemann, Boston, 1996, pp. 37–44.

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

J.M. Morgan and J.K. Liker, The Toyota product development system: Integrating people, process, and technology, Productivity Press, New York, 2006.

NCMS, Product development process—Methodology and performance measures: Final report, Project number 130120, National Center for Manufacturing Sciences, Ann Arbor, MI, 2000.

S.M. Osborne, Product development cycle time characterization through modeling of process iteration, M.S. thesis, Massachusetts Institute of Technology, Sloan School of Management, 1993.

K. Reichelt and J. Lyneis, The dynamics of project performance: Benchmarking the drivers of cost and schedule overrun, Eur Manage J 17 (1999), 135–150.

P.G. Smith and D.G. Reinertsen, Developing products in half the time, Van Nostrand Reinhold, New York, 1991, pp. 224–240.

Smithsonian Institution, Wright Brothers wind tunnel, SI Neg 2003-12979 (available from http://www.flickr.com/photos/publicresourceorg/494018125; last accessed January 14, 2013).

Smithsonian Institution, Wright Brothers wind tunnel test apparatus, lift balance, SI Neg 2003-12980 (available from http://www.flickr.com/photos/publicresourceorg/494018051; last accessed January 14, 2013).

D. Sobek II, Principles that shape product development systems: A Toyota-Chrysler comparison, Ph.D. dissertation, University of Michigan, Ann Arbor, 1997.

D.K. Sobek II, J.K. Liker, and A.C. Ward, Another look at Toyota's integrated product development, Harvard Bus Rev 76 (July–Aug 1998), 36–49.

D. Sobek II, A.C. Ward, and J.K. Liker, Toyota's principles of set-based concurrent engineering, Sloan Manage Rev 40 (1999), 31–40.

S.C. Sommer and C.H. Loch, Selectionism and learning in projects with complexity and unforeseeable uncertainty, Manage Sci 50 (2004), 1334–1347.

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

S.H. Thomke, Experimentation matters: Unlocking the potential of new technologies for innovation, Harvard Business School Press, Boston, 2003.

S.H. Thomke and T. Fujimoto, The effect of "front-loading" problem-solving in product development performance, J Prod Innov Manage 17 (2000), 128–142.

J. Tobin, To conquer the air: The Wright brothers and the great race for flight, Free Press, New York, 2003, pp. 23, 192.

A.A. Tucker and C.H. Dagli, Design of experiments as a means of lean value delivery to the flight test enterprise, Syst Eng 12 (2009), 201–217.

A.C. Ward, Lean product and process development, Lean Enterprise Institute, Cambridge, MA, 2007.

A. Ward and W. Seering, Quantitative inference in a mechanical design compiler, ASME J Mech Des 115 (1993), 29–35.

A.C. Ward, J.K. Liker, J.J. Cristiano, and D.K. Sobek II, The second Toyota paradox: How delaying decisions can make better cars faster, Sloan Manage Rev 36 (1995), 43–61.

S.C. Wheelwright and K.B. Clark, Accelerating the design-build-test cycle for effective product development, Int Market Rev 11 (1994), 32–46.

K.P. White, Jr., Systems design engineering, Syst Eng 1 (1998), 285–302.

R.I. Winner, The role of concurrent engineering in weapons system acquisition, Institute for Defense Analysis, Alexandria, VA, 1988.

O. Wright and J.T. Daniels, First flight, 120 feet in 12 seconds, 10:35 a.m.; Kitty Hawk, North Carolina, Library of Congress, Washington, DC, LC-W861-35, LOT 11512, 1903.

O. Wright and W. Wright, Octave Chanute papers: Special correspondence, The Wilbur and Orville Wright papers, Manuscript Division, Library of Congress, Washington, DC, mwright-06003, 1901, images 72, 76.

O. Wright and W. Wright, Octave Chanute papers: special correspondence, The Wilbur and Orville Wright papers, Manuscript Division, Library of Congress, Washington, DC, mwright-06004, 1902, image 32.

W. Wright, Some aeronautical experiments, lecture to the Western Society of Engineers on September 18, 1901; as published in M.W. McFarland (Editor), The Papers of Wilbur and Orville Wright, McGraw–Hill, New York, 1953, pp. 99–118.

A.A. Yassine, D.E. Whitney, J. Lavine, and T. Zambito, Do-it-right-first-time (DRFT) approach to design structure matrix (DSM) restructuring, Proceedings of ASME 2000 International Design Engineering Technical Conferences, September 10–13, Baltimore, p. 2.

Brian M. Kennedy is cofounder and CTO of Targeted Convergence Corporation (TCC) and is responsible for the software tools being developed to support the Set-Based, Learning-First Product Development practices that TCC is actively teaching corporations. After 8 years of software development at Texas Instruments, Brian spent 12 years at i2 Technologies where he served as the Chief Architect of their Supply Chain Planner and Demand Fulfillment applications, applying Toyota lean manufacturing, Theory of Constraints, and advanced optimization to the planning and scheduling of the larger supply chain, helping to establish a new market space (Supply Chain Management). Brian became the first i2 Fellow and holds a dozen patents on the inventions that were the basis for those software systems. Brian is an INCOSE Certified Systems Engineering Professional (CSEP).

Durward K. Sobek II is Professor and Program Coordinator of Industrial Engineering at Montana State University, and currently serves as Board Chair of the Lean Product & Process Development Exchange. He has been researching lean product development for over a decade, being one of the first researchers to be given access to directly observe and interview inside Toyota's product development organization. His work has appeared in numerous publications, including *Sloan Management Review*, *Harvard Business Review*, and *IEEE Transactions on Engineering Management*; and he is coauthor of the Shingo Prize-winning book *Understanding A3 Thinking: A Critical Component of Toyota's PDCA Management System* (Productivity Press).

Michael N. Kennedy is cofounder and CEO of Targeted Convergence Corporation (TCC) and is responsible for actively carrying Set-Based, Learning-First, and other Lean Product Development concepts to companies. As a thought leader in this space, Michael is on the Board of Directors of the Lean Product & Process Development Exchange, is actively working with AME to help expand their Lean content to cover Product Development, and has been asked to speak at and keynote numerous other conferences on Lean and/or Product Development. Before TCC, Michael worked for Texas Instruments Defense Systems & Electronics Group for 31 years in product development, in manufacturing, and in systems development—in both individual contributor and midlevel manager positions. At TI, he was a leader in the creation and adoption of improved product development processes. His efforts helped enable TI to win the coveted Malcolm Baldrige award for process excellence. After retiring from TI, Michael wrote his first book, *Product Development for the Lean Enterprise*, while he was working jointly with Allen Ward on the NCMS project where they met and on other joint efforts. Michael worked with the rest of the TCC Team to write his second book, *Ready, Set, Dominate*, to build on the concepts from the first book, explaining the many lessons learned from implementing those concepts at companies.