Managing Model-Based Design

Roger Aarenstrup

Managing Model-Based Design

Roger Aarenstrup

ISBN-13: 978-1512036138
ISBN-10: 1512036137

Printed in the United States

Printed on 30% post-consumer waste materials

Published by The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760
mathworks.com

This book is dedicated to all those who have confronted silent or vociferous resistance to change, and prevailed.

About the Author

Roger Aarenstrup is a MathWorks consultant who has spent a decade actively helping engineering teams and organizations adopt Model-Based Design. In previous roles he performed large-scale modeling and simulation for the Swedish Defense Research Agency, designed motion controllers for surface-mounting robots at MYDATA Automation, and developed real-time operating systems for ENEA. He holds an M.A. in computer science and engineering, with minors in automatic control and robotics, from the Luleå University of Technology, and an M.A. in business administration (e-business) from the University of Gävle.

Acknowledgements

I would like to thank everybody who has participated in interesting discussions on the subjects addressed in this book and provided constructive criticism. Many clients and colleagues have given different perspectives and shed new light on old questions. A very special thank you to Rosemary Oxenford for her hard work reviewing and editing the material. And not to forget my daughter, Alva, who gives me strength.

Contents

Managing Change, Complexity, and Innovation

To succeed in today's competitive marketplace, engineering organizations must adapt to rapid technological change and satisfy a continuous demand for new products and technologies. Whether the end product is a mobile phone, a car, an airplane, or a wind power plant, innovation is not simply a desirable goal; it is a necessity.

In one respect, creating innovative products and features has never been easier. Inexpensive hardware, computational power, and sophisticated design tools are readily available, making possible unprecedented innovations. In another respect, however, innovation has never been more challenging.

In what has become known as Moore's Law, Intel cofounder Gordon E. Moore observed that roughly every two years, the number of transistors on integrated circuits doubles. According to a 2020 report by Deloitte, electronics make up 40% of the cost of a new car.

[1] *"Electronics Account for 40 Percent of the Cost of a New Car."* Car and Driver, *May 2, 2020. https://www.caranddriver.com/features/a32034437/computer-chips-in-cars/.*

Driven by customer requirements, tightening safety and environmental regulations, and market competition, the number of components in each product continues to rise. Making all the components work together becomes increasingly difficult, hindering the design and implementation of innovative features.

The key challenges of innovation and complexity place pressure on engineering organizations from every perspective—not only technical, but also organizational, administrative, and cultural. Engineers must design systems comprising many parts so that all the parts work together. Often they must do so within shrinking development schedules, working with geographically scattered teams, and using development methodologies rooted in an Industrial Age culture, with its bureaucratic corporate structure and hard boundaries between departments.

Some organizations tackle system complexity by removing features or by simply accepting lower performance. In other words, requirements are changed to fit what has been made, and innovations are deferred or canceled. Other organizations tackle the problem by hiring more engineers. This approach can only work to a degree. As an organization grows there are new challenges, such as communication and knowledge sharing between departments and groups.

Ikujiro Nonaka points out that "In an economy where the only certainty is uncertainty, the one sure source of lasting competitive advantage is knowledge."[2] Managing change, system complexity, and innovation requires, first, a shift of emphasis from production to learning and knowledge management. Second, it requires a holistic approach in which organizational structure, development methodologies, and knowledge management are recognized as interdependent, and are considered together.

In the past 20 years, organizations seeking to manage complexity while staying innovative and competitive have increasingly turned to Model-Based Design. Twenty years ago, the questions asked were, "What is Model-Based Design?" "Does Model-Based Design work?" "Is it efficient?" "Is it safe?" Today, the main question is, "How do we adopt it in our organization?"

[2] *Nonaka, Ikujiro. "The Knowledge-Creating Company." hbr.org/2007/07/the-knowledge-creating-company/es. Accessed on September 9, 2014.*

Successful adoption of Model-Based Design requires careful management of the change process, a thorough understanding of how Model-Based Design works, and the ability to communicate its value to key decision-makers.

Efforts to adopt Model-Based Design often start with one or a few engineers who see the benefit of Model-Based Design and want to convince the rest of the organization to adopt it as well.

This book provides arguments and background information to enable those engineers to champion Model-Based Design within their organization. It also serves as a guide for managers to take the lead in making their organizations more efficient, effective, and innovative through Model-Based Design. It provides a road map to the major concepts of Model-Based Design, and shows how these concepts, used together or individually, can help make any organization more efficient and better prepared to meet the challenges of change, complexity, and innovation.

1

Inside Model-Based Design

This chapter provides an overview of Model-Based Design and defines its eight core concepts.

Model-Based Design is a model-centric approach to the development of control, signal processing, communications, and other dynamic systems. Rather than relying on physical prototypes and textual specifications, Model-Based Design uses a model throughout development. The model includes every component relevant to system behavior—algorithms, control logic, physical components, and intellectual property (IP). Once the model is developed (elaborated), it becomes the source of many outputs, including reports, C code, and HDL code. Model-Based Design enables system-level and component-level design and simulation, automatic code generation, and continuous test and verification (Figure 1.1).

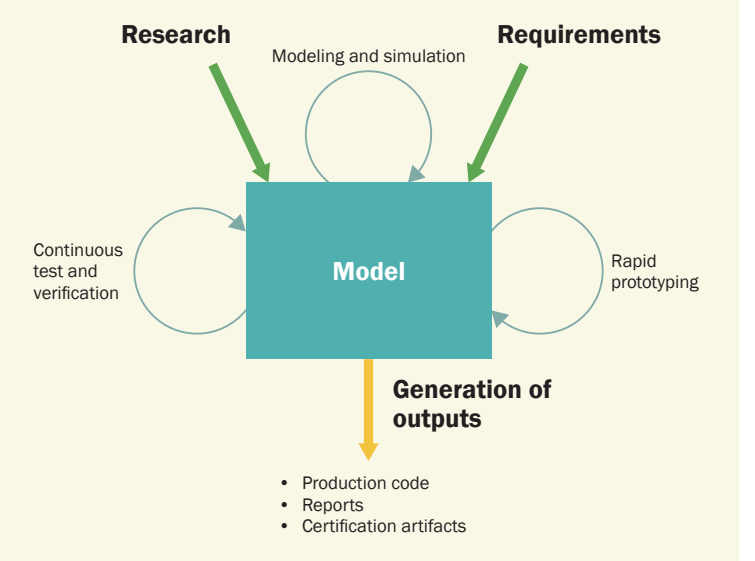

Figure 1.1. Workflow for Model-Based Design.

Model-Based Design can support virtually any organizational type, and it has been implemented successfully within many different development workflows. How you implement it depends on the size, structure, and culture of your organization, as well as the systems being developed and the demands of your target market. You might choose to adopt Model-Based Design enterprise-wide, transforming your entire development process. Alternatively, you might apply it selectively to address a specific challenge, such as a workflow bottleneck, a sudden change in design requirements, or increased system complexity.

"Three years ago, SAIC Motor did not have rich experience developing embedded control software. We chose Model-Based Design because it is a proven and efficient development method. This approach enabled our team of engineers to develop the highly complex HCU control logic and complete the project ahead of schedule."

– Jun Zhu, SAIC Motor Corporation

Why Model-Based Design?

Model-Based Design provides a path to streamlining many aspects of development. For example, organizations report that Model-Based Design enables them to:

- Manage complex systems
- Automate time-consuming and error-prone tasks
- Quickly explore new ideas
- Create a common language that fosters communication and collaboration
- Capture and retain intellectual property
- Increase product quality
- Reduce risk

Example

Building an ECU with Model-Based Design

A team of automotive engineers sets out to build an engine control unit (ECU) for a passenger vehicle. Using a workflow employing Model-Based Design, the engineers begin by building a model of the entire system—in their case, a four-cylinder engine. This high-level, low-fidelity model includes simplified representations of the portion of the system that will be implemented in software (the ECU) and of the environment (the plant and the conditions under which the engine will operate).

The team performs initial system and integration tests by simulating this high-level model under various scenarios to verify that the system is represented correctly and that it properly responds to input signals. Issues such as ambiguous requirements are often detected at this early stage, when they are easy and relatively inexpensive to fix. The model becomes an executable specification that is used to verify textual requirements.

After running the first high-level system simulation, the team adds detail to the model, continuously testing and verifying the

system-level behavior against requirements and standards through simulation. If the system is large and complex, the engineers can develop and test individual components independently but still test them frequently in a full system simulation.

Ultimately, the team builds a detailed model of the system and relevant parts of the environment within which it operates. This model captures the accumulated knowledge about the system (the IP). The engineers generate code from the model of the control algorithms for software testing and verification. Following real-time hardware-in-the-loop tests, the team downloads the automatically generated code onto production hardware for testing in an actual vehicle.

The Core Concepts of Model-Based Design

Model-Based Design is founded on eight core concepts:

- Executable specification
- System-level simulation
- What-if analysis
- Model elaboration
- Virtual prototyping
- Continuous test and verification
- Automation
- Knowledge capture and management

Executable Specification

An executable specification is a model that encapsulates all design information, including requirements, system components, IP, and test scenarios. It can be a model of the environment with use cases that the embedded software needs to manage, or a high-level algorithm model that specifies the implementation's exact behavior.

An executable specification offers the following advantages over text-based specifications:

- A model typically includes more information than a text document.

- Models are unambiguous, and do not require interpretation the way a text document does.
- Because it includes unambiguous information, a model enables clear, efficient communication between team members and with customers and suppliers.
- An executable specification can be used to validate textual requirements—the requirements are modeled to ensure their consistency and accuracy.

System-Level Simulation

In system-level simulation, a model of the entire system is simulated to investigate system performance and component interactions. You can use system-level simulation to validate requirements, check the feasibility of a project, and conduct early test and verification. Simulation provides a way to verify complex, multidomain systems that are more than the sum of their parts.

Other benefits of system-level simulation include the following:

- Design problems and uncertainties can be investigated early, long before you build expensive hardware.
- Simulations are safe—there is no damage to hardware or other hazards if the design does not work.

What-If Analysis

What-if analysis is a simulation method used to test ideas and learn about the system. You can perform what-if analysis to test a single component or to investigate the interactions of all components in the system.

What-if analysis brings the same benefits as system-level simulation. In addition, it enables you to:

- Quickly explore and evaluate multiple design ideas
- Generate new knowledge about the system (see "Knowledge Capture and Management")

Model Elaboration

Model elaboration is an iterative process that uses simulation to turn a low-fidelity system model into a high-fidelity implementation. Model

elaboration begins once you have simulated the high-level system model to verify requirements. When the model yields the desired results, details and refinements are added, and the model is simulated again. Common refinements include converting from floating point to fixed point, converting from continuous time to discrete time, replacing a behavioral actuator model with a detailed actuator model, and adding signals for diagnostics.

Model elaboration enables the entire system to be continuously tested. When there is an adequate level of detail, the parts describing the embedded software can be used to generate code for rapid prototyping and hardware-in-the-loop testing. With even more detail, the model can be used for production code generation.

Virtual Prototyping

Virtual prototyping is a technique that uses simulation to validate a design before hardware is available. In cases where the plant and environment are not yet fully known or understood, such as a mechanical construction, it may be necessary to use a hardware prototype for experiments to build the model. The knowledge acquired from these experiments is then stored in the model, where it can be transferred to other developers, departments, suppliers, and customers.

Virtual prototypes save development time because building a model is usually much faster than building a physical prototype. Virtual prototypes also reduce cost and increase innovation because they enable a team to quickly and safely try out new concepts. In many situations, a model can replace a test rig. Using a model reduces development bottlenecks, since test rigs are often a scarce resource.

Continuous Test and Verification

Continuous test and verification is the practice of simulating a design at every stage of development. It is used to identify faults as soon as they are introduced into the design.

Continuous test and verification can take different forms, and it can be conducted at different levels, depending on the complexity of the system and the stage of development. For example, it can be any of the following:

- Open-loop testing—testing a single component with predefined inputs and specified checks for the outputs
- Closed-loop testing—testing a component or design with a model of the environment and a plant model
- Rapid prototyping—generating code from the controller model to test the software part of an embedded system against the actual plant and environment
- Hardware-in-the-loop (HIL) simulation—generating code from the model of the environment to test an actual embedded system against a simulated environment in real time

Benefits of continuous test and verification include:

- Early identification of errors, reducing cost and development time
- Error reduction, increasing software quality
- Reduced risk, providing a cost-efficient, safe way to test scenarios that could damage expensive hardware
- Increased understanding of the system

Automation

Automation is the practice of using scripts and tools to perform repetitive tasks or tasks that are error-prone when performed manually.

Common automations within Model-Based Design include the following:

- Generating production code
- Developing targets to customize generated code for specific target hardware
- Generating reports, such as design descriptions and test results
- Conducting model checks to ensure that the model conforms to guidelines
- Connecting to system databases for interface checks and setup
- Formally proving system properties
- Formally proving code correctness

- Automatically building and testing an entire system, including component tests

Automation in Model-Based Design brings the following benefits:

- The team can focus on design instead of implementation details.
- Faster development cycles make it easier to handle requirement changes.
- Complex systems are easier to manage.

Knowledge Capture and Management

In Model-Based Design, models are the primary source of project information. That knowledge includes not only design specifications and details about the system under development, but also product knowledge, team members' design expertise, past experience, and design best practices.

The models become a common language for the transfer of information within teams and with customers and suppliers. Because the models can be executed or simulated, the knowledge they contain increases as understanding of the system grows.

Using models for knowledge capture and management helps to:

- Improve communication
- Preserve intellectual property
- Build a culture of knowledge sharing
- Improve project and business relationships

Key Takeaways from This Chapter

- Model-Based Design is a model-centric approach to the development of embedded systems.
- Rather than relying on physical prototypes and textual specifications, Model-Based Design uses a system model as an executable specification throughout development.
- Model-Based Design enables system-level and component-level design and simulation, automatic code generation, and continuous test and verification.
- Model-Based Design is founded on eight core concepts:
  - Executable specification
  - System-level simulation
  - What-if analysis
  - Model elaboration
  - Virtual prototyping
  - Continuous test and verification
  - Automation
  - Knowledge capture and management
- Model-Based Design can support virtually any organizational type, and it has been implemented successfully within many different development workflows.

2

Implementing Lean Development Principles with Model-Based Design

This chapter provides an overview of lean development and explains how the core concepts of Model-Based Design support lean principles.

Originally developed by Toyota for the Toyota Production System, lean development is a methodology based on specific principles and core values. Lean development calls on management to invest in its employees and establish a culture of mutual respect and continuous improvement. Fundamental to lean development are managers who have a lean mindset and who coach the workforce as mentors.

The terms "lean development" and "lean production" are often used interchangeably. The basic difference between lean development and lean production is that lean development focuses on outlearning the competition while lean production focuses on outimproving the competition. Of course, the two concepts are not mutually exclusive. An organization that outlearns creates knowledge with a value for the customer faster. The organization can use this new knowledge to deliver higher-quality products with more innovative features.

Like the core concepts of Model-Based Design, lean development principles can be applied within any organizational structure. At the same time, implementing lean principles has the effect of shifting a bureaucratic, hierarchical structure to one that is more decentralized and organic.

Seven Principles of Lean Software Development

Poppendieck and Poppendieck (2003) define seven lean principles and show how they can be used to improve software development practices:

Eliminate waste. Avoid incomplete work; knowledge scatter or loss; task switching or interruptions; software defects; underutilized skills, insights, or ideas; overproduction of features or elements; and unnecessary movement of people or materials. Adding unproductive layers of management is also considered waste.

Amplify learning. Provide opportunities for developers to learn more about the application domain. Implement short development cycles that give immediate feedback on the design. Solicit feedback from customers.

Minimize bureaucracy. Simplify procedures and decision-making. Reduce administrative overhead. Remove superfluous management layers.

Decide as late as possible. Take the time to gather facts. Base decisions on these facts rather than on hastily formed assumptions. The more complex the system, the more flexibility should be built in. A flexible architecture makes it possible to delay many implementation decisions.

Deliver as quickly as possible. Ensure that your product meets current customer needs, not what the customer needed some time ago.

Build integrity into the process. Make the components of a system work as a coherent entity. Ensure design consistency. Build perceived integrity by helping the customer understand how the system is used, delivered, or deployed.

See the whole. Recognize that complex systems are more than the sum of their parts. Create well-defined interfaces and standardize components to ensure that components work together. Build a strong communication network with vendors and subcontractors.

Two Lean Development Core Values

Lean development is supported by two core values: respect for people, and continuous improvement.

Respect for People

Focus on building a strong work culture, boost employee morale, and reinforce customer relationships by applying the following best practices:

- Reduce trouble for customers—do not make them wait, do not send them defects, do not blame them for issues.
- Develop and invest in staff—teach and coach rather than direct.
- Lead by example.
- Develop cross-functional teams.
- Share knowledge and best practices rather than enforce processes.

Continuous Improvement

Ensure continuous improvement by rigorously applying concepts and techniques such as the following:

- The five whys—find the root cause of a problem by asking "why?" five times
- Fishbone (Ishikawa) diagrams mapping cause and effect
- The "go see" principle—go and see for yourself to thoroughly understand an issue or situation
- Quality leaps—take a large step, or adopt new methods or ways of thinking that allow for further improvements

Implementing Lean Development with Model-Based Design

Model-Based Design supports the principles and core values of lean development, and it can be a valuable method for implementing lean principles in a development organization. In many cases, the tools and concepts of Model-Based Design are a direct response to the need for a lean approach.

The core concepts of Model-Based Design support all seven lean development principles and both core values.

Eliminate waste

- With Model-Based Design, the entire development process is completed in a single, integrated environment. Working in one environment means less task switching. For example, it is more efficient to do control design in the same environment that is used for plant modeling.
- System-level simulation makes it possible to try different solutions in a cost-efficient way and quickly eliminate the unworkable ones.
- Continuous test and verification ensures that errors are removed early in development.
- By automatically generating model coverage metrics, you can find unused parts of a model and remove unnecessary code and superfluous functionality.
- By automatically generating code from the model, you eliminate the error-prone step of manually translating designs into code.
- Using models helps to reduce knowledge scatter or loss. Models can be used to capture project details as well as engineers' knowledge.
- Models can work as specifications and are never ambiguous because they can be simulated. If documents are required, they can be linked to a model to make requirements clearer.

Amplify learning

- System-level simulation and what-if analysis provide a fast and efficient way to experiment, test ideas, and learn.
- Models store much more information—and more accurate information—than documents do.
- Organizations can use models to help new employees get up to speed quickly, while more experienced employees can use them to experiment and create new knowledge. When an expert creates a model of his or her view of the system, that expert's knowledge is preserved even if the expert leaves or is transferred.
- System-level simulation provides instant feedback about a design. Modeling the environment increases your understanding of the system.

Minimize bureaucracy

With Model-Based Design, you can reduce the impact of bureaucracy on productivity by automating bureaucratic steps such as conducting design reviews, checking standards and guidelines, and generating reports.

Decide as late as possible

Models provide a way to delay certain decisions without compromising productivity. For example:

- Modeling and simulation make it inexpensive to try different solutions and enable set-based development in which several solutions are developed in parallel.
- Models are generally independent of the target implementation. As a result, you can select the target hardware (such as a DSP, FPGA, or ASIC) later in the development process, or you can change the hardware target without losing the implementation.

Deliver as quickly as possible

- With an environment or plant model, algorithm development can take place before hardware is built.
- Automation streamlines the workflow and reduces delay. For example, automatic code generation eliminates the time-consuming and error-prone step of manually translating designs into code.
- With models, testing does not have to wait for a full implementation. You can perform tests continuously during development using techniques such as rapid prototyping and hardware-in-the-loop simulation.
- Your supplier's component model can be simulated as part of your larger system simulation. Component performance can then be verified long before actual hardware or parts are delivered or even manufactured.
- Simulation and code generation give faster output and results, which helps reduce cycle times.
- Simulation makes feasibility studies faster and less expensive, enabling your team to investigate more options without slowing development.

Build integrity into the process

- You can give external customers an early opportunity to steer development by sharing simulation results with them.
- Models foster cooperation by showing team members how their roles are interconnected and by enabling them to combine and test their work. For example, a software engineer might work on integrating algorithms in a software platform by customizing the code generator and developing a suitable, flexible platform. An algorithm developer develops the algorithm in a model. Their combined effort can instantly be tested by code generation and code integration.
- With automation, employees can more easily share best practices and model guidelines.
- Model guidelines with automatic checks can ensure consistency in the model design.
- With a system architecture and breakdown where each component is designed as a separate model, there is strong component integrity. The component can be tested and implemented as a separate entity, but it can still be used with other models for full system simulation. Each model has a "hard" interface, which means that the interface and behavior during execution are the same regardless of whether it is executed alone or as part of a larger simulation.

See the whole

- Building up simulation models of systems brings knowledge about how a system works, the relationship between the different parts, and which details are important and which ones can be ignored.
- Full system simulations are a key capability for managing the complexity of a system. If components are divided and implemented as separate models, they have well-defined interfaces. You may choose to hide the contents in a model sent to a third party, but it is still possible to use the model as a part of a larger simulation to evaluate performance or to see how it fits into a larger system.

Respect for people

- Using models for communication can strengthen the network of suppliers, customers, and other stakeholders.
- Simulations let you present examples and user interfaces to your customers frequently and early in the development cycle. You can give customers accurate feedback about details or system performance.
- Development teams using Model-Based Design can include people with environment (plant) modeling knowledge, algorithm developers, test engineers, and software engineers. Using models makes cooperation easier since the different tasks, although separated, are still closely connected. The connections among work roles are well defined, which makes cooperation easier.
- Models are better than text documents for knowledge sharing and including other people's ideas. Models are also more engaging than text documents, so they increase motivation.

Continuous improvement

- Standards and guidelines can be implemented as checks that are automatically performed on the model. In this way, formal implementation makes continuous improvement possible.
- Formalizing best practices as an implementation means that the practices themselves can undergo continuous improvement.
- Simulation supports quality leaps by building confidence and providing an inexpensive and safe way to learn and reduce risk. The ability to include legacy code in the simulation environment ensures that existing functionality matches or works with new designs.

Example

Diagnosing a Controls Error Using Lean Principles and Model-Based Design

A team of engineers is using Model-Based Design to develop controls for a high-performance conveyor (Figure 2.1). Driven by a DC motor with a ball screw, the conveyor moves circuit boards back and forth to enable a robot to pick up and place components accurately. A prototype of the conveyor is currently under test.

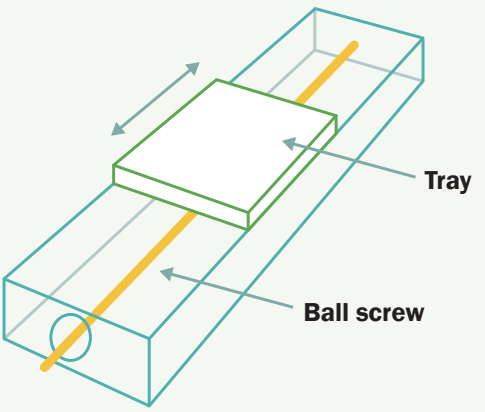

Figure 2.1. Ball screw conveyor.

The test engineer is about to add new test cases when he is interrupted by a phone call. He goes to take the call, leaving the conveyor turned on but not executing any tests. When he returns about 15 minutes later, the conveyor is making an odd humming sound. The developer reports the problem to the project leader.

Identifying the Cause of the Problem with Lean Principles

Following the lean core value **continuous improvement**, the project leader begins with **stop, go see**. She and the team meet and try to reproduce the problem in the lab. They power up the conveyor but do not give it any directions for movement. The humming sound recurs, confirming that there is an issue.

The next step is to **take the time to gather facts**. They conduct a root cause analysis, beginning with the "Five Whys," and incorporate the results into an Ishikawa diagram (Figure 2.2).

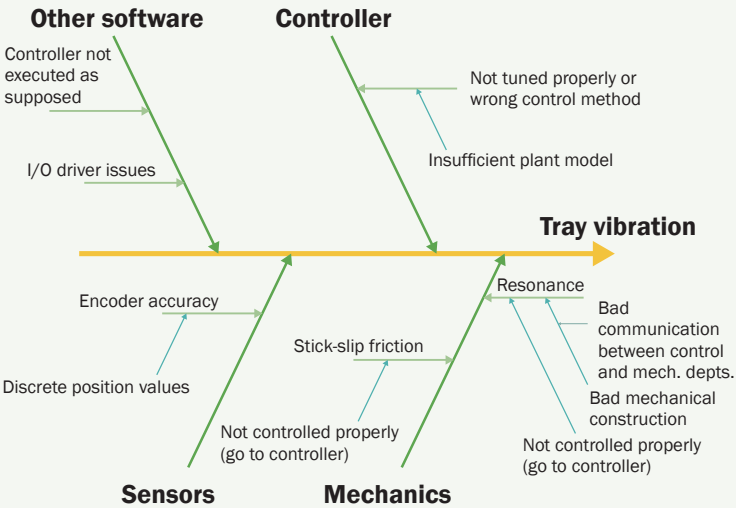

Figure 2.2. Ishikawa diagram for conveyor problem.

Two possible causes of the problem are identified: stick-slip friction and discrete position values from the encoder.

The project leader divides the team into two self-directed groups, each responsible for investigating one cause. By allowing this autonomy, she is **amplifying learning** (providing opportunities for team members to learn more about the application), **eliminating waste** (avoiding the addition of unproductive layers of management), and **minimizing bureaucracy** (simplifying procedures and decision-making).

Solving the Problem with Model-Based Design

Both groups begin by updating the plant model. One team adds the details for the discrete encoder, and the other adds stick-slip friction to the mechanics. They then use **system-level simulation** and **what-if analysis** to investigate the effects of the updates.

The group investigating the discrete encoder identifies the issue. They add simple quantization of the encoder signal. Test cases run in the simulation model reveal no tray vibration. However, when the team sets the desired position manually, they see flickering in the control signal whenever a position cannot be met by the discrete encoder. Because of the discrete steps in the encoder, a control error remains, causing the integral part of the controller to grow. After a long time, it is strong enough to move the tray to a position that is still not entirely correct.

The team quantizes the input to the controller so that the encoder can always represent the desired position. They also add test cases to check for this problem. They generate code from the updated controller, compile it, and download it to the actual machine. They check for the newly found issue, but this time there is no noise. The product can now be released without delay.

The team fixed the problem in a timely manner. But perhaps even more important, they learned something new about the system. They captured the knowledge gained in the plant and controller models, where it can be applied to future projects. The main point is that they learned to build a better system.

Key Takeaways from This Chapter

- Lean development is a methodology based on clearly defined principles and core values.
- Like Model-Based Design, lean development can help turn a bureaucratic, hierarchical structure into one that is more decentralized and organic.
- Model-Based Design is a direct response to the need for lean development.
- The core concepts of Model-Based Design foster lean development principles.

3

Improving Development Methodologies with Model-Based Design

This chapter provides an overview of common development methodologies and explains how Model-Based Design core concepts support these methodologies. It then explains how Model-Based Design can help you adapt any methodology to the needs of your project.

From the late 1970s to the early 1990s, the waterfall was the standard software development methodology. Its rigid, stepwise approach fit well into the bureaucratic organization at the time. However, the advent of the Internet and other technological advances in the mid-1990s accelerated the pace of change in software projects, requiring more flexible approaches.

Engineering organizations today use a wide range of development methodologies. Ideally, the project manager selects the methodology that best fits the nature of the project and the system being developed. In reality, however, this is not always the case, and a team might be required to use the same methodology for every project—not because it is optimal, but because this is how the work has always been done.

From Construction to Evolution: Common Development Methodologies

Most development methodologies lie on a spectrum between the classical, plan-based "construction" approach and the more flexible and iterative "evolution" approach.

The following methodologies are the most common:

- Waterfall
- V-model
- Iterative and incremental development (IID)
- Spiral
- Scrum
- Extreme programming (XP)

Waterfall

The waterfall methodology breaks development down into five steps or phases: requirements, design, implementation, verification, and maintenance (Figure 3.1). Each step must be completed before the next begins.

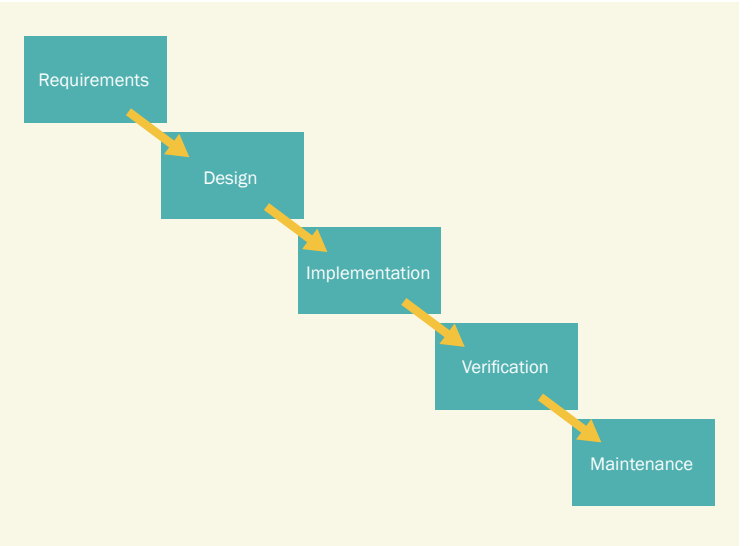

Figure 3.1. The waterfall methodology.

However, to be effective, waterfall requires team members to have solved similar problems before. Moreover, its stepwise approach is too rigid to handle changes in requirements and does not encourage innovation.

Model-Based Design Within the Waterfall Methodology

Model-Based Design enhances the waterfall methodology in these ways:

- Automation makes it easier to deal with change. It's easier to redo a step when tasks such as report generation are automated.
- System-level simulation makes it easier to manage complexity by showing the interactions between components.
- What-if analysis fosters innovation by enabling team members to try out new ideas quickly and without risk.

V-Model

The V-model development methodology is common in automotive organizations. Instead of proceeding through the steps in a linear fashion, like waterfall, it bends upwards after the implementation (coding) phase, thus matching each development step with a corresponding test phase (Figure 3.2).

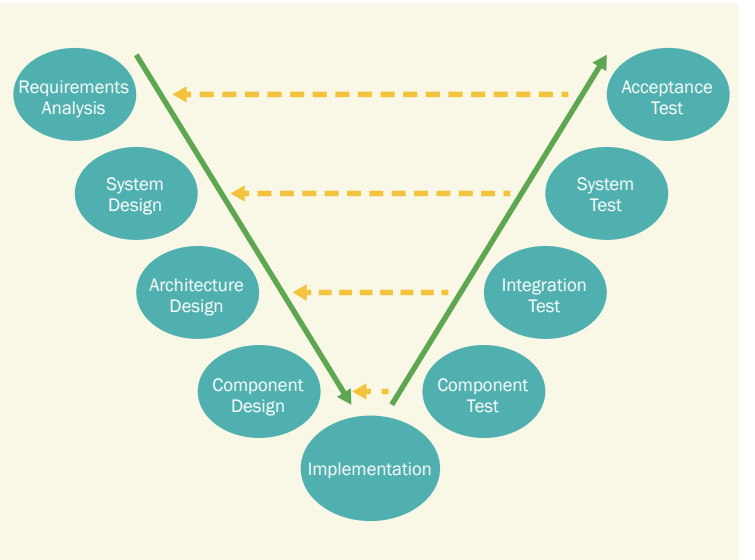

Figure 3.2. The V-model.

The main disadvantage of the V-model is that the entire system design is developed at the beginning. This makes it hard to work with complex systems, where some components, interactions, or other elements might not be known until later in the process.

Model-Based Design Within the V-Model

Model-Based Design enhances the V-model methodology in various ways:

- Automation makes it easier to deal with change—it is easier to redo a step when tasks such as report generation are automated.
- System-level simulation makes it easier to manage complexity by showing the interactions between components.
- What-if analysis fosters innovation by enabling team members to try out new ideas quickly and without risk.
- System-level modeling and simulation make it possible to develop the entire system at the start even though key elements are not known—these elements can be added later without interrupting the development flow.

Iterative and Incremental Development

Iterative and incremental development (IID) proceeds in cycles (releases) that take between one week and six months. The goal of each cycle is to deliver a partially complete system for integration and testing (Figure 3.3). Typically, more time is spent on requirements in the early cycles than in later ones, but all cycles include production coding. This means that there are no prototypes or proof-of-concept releases.

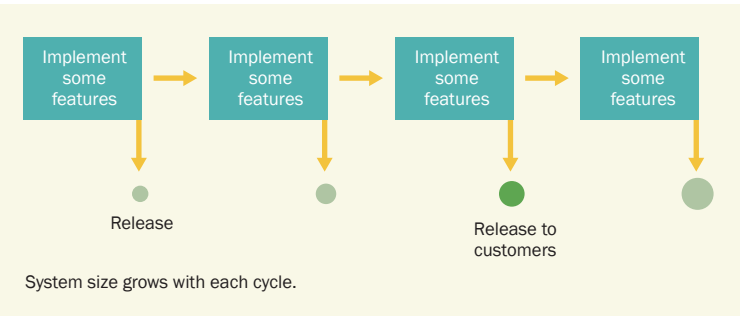

Figure 3.3. Iterative and incremental development.

Teams select the tasks to implement in each cycle in different ways. In risk-based selection, the riskiest tasks are implemented first. If they fail, the project can adapt or be canceled. In client-based selection, the client or customer determines which tasks to implement in each cycle and whether to continue or terminate the project.

IID works well in the following situations:

- Tasks and features are independent.
- Features are being added to an existing, functioning system.
- Features and fixes are being added in a maintenance phase.
- Similar projects have been completed before, and dependencies and complexity are well understood.

IID has these disadvantages:

- Risk evaluations and requirements analysis in each cycle introduce a significant amount of ceremony (see "Managing Ceremony").
- Incremental delivery requires a flexible architecture that allows features to be added continuously. This can be a disadvantage if the architecture is a significant part of all code, or if a fully functioning system requires a large number of features.
- Incremental delivery is delayed when a project contains unknowns. In such cases, extensive research that includes prototyping and experimenting must be done before delivery can start.
- While shorter cycle times reduce the complexity of each cycle, problems may arise with full system dynamics as features are added. These problems will not become evident until all features are in place.
- The overhead of risk assessments, requirements analysis, and planning can make this methodology inefficient for smaller projects.

Model-Based Design Within IID

Model-Based Design supports IID in these ways:

- You can use automation to manage required tasks or ceremony.
- You can increase innovation by using what-if analysis and simulation.
- System-level simulation improves the methodology's ability to handle complexity.
- Automation and simulation can make the work in each cycle more efficient, which helps to reduce cycle time. Automation also reduces the overhead associated with risk assessment, requirements analysis, and planning.

In addition, Model-Based Design enables you to incorporate a top-down approach into the IID workflow (Figure 3.4). In a top-down approach, more details are added to the design at each step in the workflow, and the design is continuously evaluated via simulation for performance, risk, functionality, and so on. This approach brings the following advantages:

- In early iterations, high-level models can be used to evaluate requirements and design concepts. As a result, system and integration problems are caught well before an increment is integrated into the final product.
- With top-down development, prototyping and experimentation can be completed rapidly, reducing the risk of delayed delivery for projects that contain unknowns.
- Simulation outputs and rapid-prototyping results can be presented to the customer at every step of development, making it easier to manage, document, and share complex systems.
- Managing change is easier in the top-down approach. Simulations can be used to understand how the new component will affect the system before the feature or component is deployed.

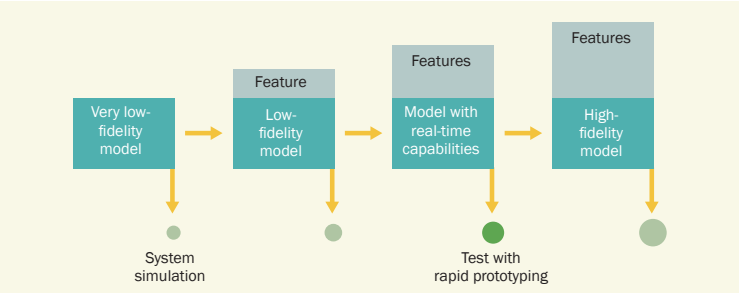

Figure 3.4. Iterative top-down development flow.

Spiral

The spiral methodology uses many of the same steps as the waterfall methodology: requirements, design, implementation, and testing. The difference is that these steps are completed in one-year or two-year cycles that focus on certain features of the whole system. Once a cycle is complete, the team reevaluates the project risks and requirements, and decides whether to continue. If the project continues, another cycle begins. With each cycle, risk is reduced, and more features of the final system are implemented (Figure 3.5).

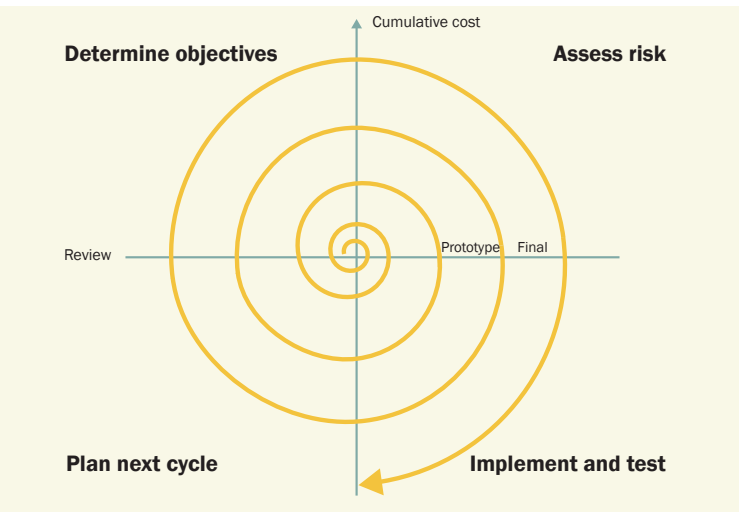

Figure 3.5. The spiral methodology.

The spiral methodology was designed primarily to handle risk in large projects, but it also handles change and innovation well. Its iterative nature enables the team to focus on experimentation and innovation rather than on making the delivery. It scales well from small projects to large, complex systems such as airplanes, military vehicles, or wind turbines.

The main disadvantages of the spiral are that it does not define specific ways to manage complexity in a system under development, and that it has only one major release, making it ill-suited to projects that must be delivered piece by piece.

Model-Based Design Within Spiral Development

- You can support innovation in the spiral methodology by incorporating what-if analysis and simulation into design iterations.
- System-level simulation improves the ability to handle complexity.
- Automation and simulation can make the work within each cycle more efficient, reducing the cycle time.

Scrum

The scrum methodology arose from agile principles and takes a minimalistic approach to development. In scrum, development is performed by small, self-managed teams. Tasks are carried out during a two-week to four-week period called a sprint. Participants report their progress at daily scrum meetings. At the end of each sprint, the customer (who can be internal) is required to accept the work and prioritize tasks remaining in the work queue (Figure 3.6).

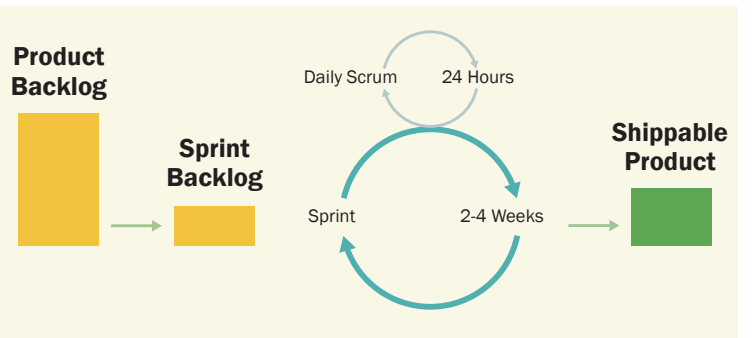

Figure 3.6. A typical scrum workflow.

Scrum demands very little overhead (waste). With its rapid iterations, scrum handles fast-paced changes well.

Scrum defines how to plan and organize projects, but not the way development work is performed. As a result, it provides no procedures for managing complex systems with interrelated parts.

Model-Based Design Within Scrum

With system-level simulation, you can manage larger and more complex systems because the simulations bring all parts together in each cycle, making it easier for teams to handle interactions and interfaces. For individuals on a scrum team, what-if analysis and other concepts of Model-Based Design can prove useful.

Automation can help your team deliver large projects within the short scrum cycles.

Extreme Programming (XP)

Like scrum, the XP methodology is derived from agile principles, but scrum focuses on project management, while XP focuses on improving quality and productivity by defining how to do the actual development work.

XP proceeds in short development cycles known as releases, with built-in checkpoints for introducing new customer requirements. XP emphasizes extensive code reviews and unit testing of all code, but it avoids instituting formal methods and documentation.

XP's minimalist, decentralized approach requires highly experienced programmers who share a workspace. Problems arise when team members are geographically scattered, when less experienced engineers join the project, or when team members leave and take their undocumented or unstored knowledge with them.

Model-Based Design Within XP

XP can benefit from Model-Based Design in a similar way that scrum can:

- System-level simulation supports a decentralized approach by enabling the team to keep the parts of the project together. System-level simulation also helps manage complexity, interactions, and interfaces between parts in this decentralized approach.
- Automation and continuous test and verification support the XP practice of continuous integration (building and retesting).
- Knowledge capture and management using models supports XP's minimalist approach to documentation. Models serve as documentation and store knowledge if people leave or change roles.

Managing Ceremony

Every development methodology involves a certain degree of ceremony—formalized steps and procedures, documentation, review processes, and metrics. Typically, methodologies that use several short cycles require less ceremony than those that use fewer, longer cycles (Figure 3.7):

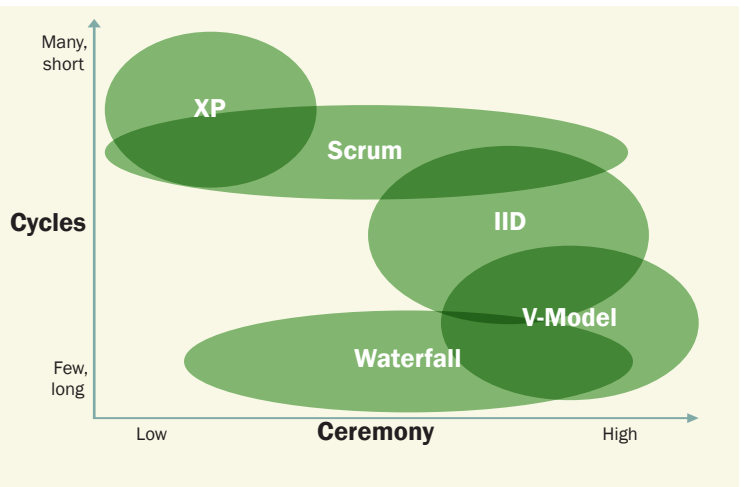

Figure 3.7. Cycles vs. ceremony in common development methodologies.

- Waterfall projects are noniterative and document-based and therefore, high in ceremony.
- V-model scores high on ceremony because it specifies formal steps and documents for test and verification.
- IID uses a few long cycles and depends on continuous risk evaluation and requirements analysis, making it high on ceremony.
- Spiral (not shown in diagram) has a few long iterations with continuous requirements and risk analysis.
- Scrum uses a moderate number of cycles that do not have specific ceremony requirements; as a result, it can be incorporated into a process with any level of ceremony.
- XP has many short cycles and therefore, very little ceremony.

Model-Based Design can help organizations fulfill ceremony requirements without slowing development. Because Model-Based Design automates procedures such as report generation and code reviews, more ceremony or formality can be introduced without affecting cycle time.

At the same time, the automation of ceremony elements such as code reviews means that a methodology with very low ceremony, such as XP, can introduce more ceremony without losing the benefits of that method. This can be useful if you need to incorporate certification requirements into your workflow.

Selecting a Development Methodology

Boehm and Turner (2004) divide the development methodologies discussed in this chapter into two types: plan-driven (waterfall, V-model) and agile (scrum, XP, and IID). Each type has a home ground (area of special strength or competence). Plan-driven methodologies work well for large, complex, high-integrity systems with stable requirements, large development teams, and a culture that demands order. Agile methodologies are better suited to projects with changing requirements and aggressive deadlines (Figure 3.8).

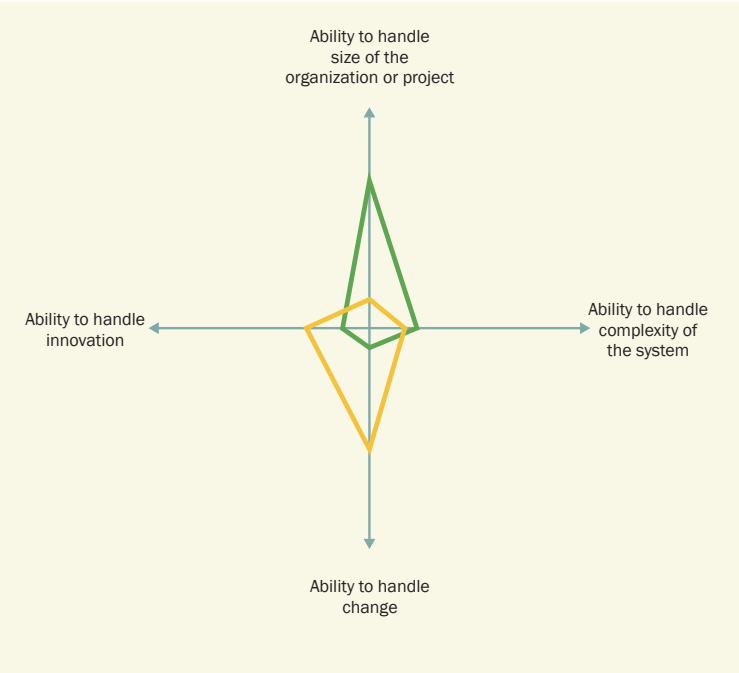

Figure 3.8. The relative strengths of plan-driven approaches (green line) and agile approaches (yellow line).

When selecting a development methodology, consider these questions:

- How stable are your project requirements?
- How aggressive is the deadline?
- What size is the project?
- Are there critical resources involved?
- Is the project business-critical or mission-critical?

Each question is easy to address individually, but considered in combination, they might require a tradeoff. For example, tradeoffs might be required for a small project that is safety-critical and where requirements are expected to change, or for a large project with aggressive deadlines that calls for a critical resource.

As you consider these tradeoffs, however, bear in mind Boehm and Turner's observation that both plan-driven and agile approaches are "critical to future software success." Small projects and teams need to be able to scale up, while large projects and teams need to be more nimble.

Improving Your Development Methodology with Model-Based Design

Model-Based Design can help you adapt any development methodology to your current needs.

Implementing the core concepts of Model-Based Design increases the types and kinds of projects for which each development methodology can be used. Similarly, a smaller project that is not mission-critical but includes new technology and has a high risk can use a methodology that has more ceremony, such as IID, even if the project involves only six engineers.

Model-Based Design is particularly useful when there are tradeoffs to consider. For example, if your project has a critical resource or an expensive test rig, this points towards a plan-based waterfall or V-model. Model-Based Design can help by introducing models you can simulate instead of testing on the prototype. This eases the bottleneck that the prototype causes and allows for a more agile or iterative approach to development.

Another example is a small project with a safety-critical application. The criticality of the project requires a plan-based approach including safety standards. With Model-Based Design, you can automatically generate many of the required documents and artifacts, as well as perform early verification with simulation. Model-Based Design helps with the ceremony related to safety standards and allows for a more agile approach.

Key Takeaways from This Chapter

- Model-Based Design can help you adapt any development methodology to better fit your current needs.
- Most development methodologies lie on a spectrum between a classical, plan-based "construction" approach, such as the waterfall, and a more flexible and iterative "evolution" approach, such as XP.
- Typically, methodologies that use several short cycles require less ceremony than those that use fewer, longer cycles.
- Model-Based Design can help organizations fulfill ceremony requirements without slowing the development process. It can also enable a methodology with very low ceremony to introduce more ceremony without losing the benefits of that method.
- Implementing specific concepts in Model-Based Design increases the types and kinds of projects for which each development methodology can be used.

4

Creating and Managing Knowledge

This chapter presents common theories of knowledge creation and defines knowledge creation in a development context. It then explains how Model-Based Design enables teams and organizations to capture, leverage, preserve, communicate, reuse, and create knowledge.

In the Industrial Age, a company's competitive advantage and profitability were determined by the price of its products. Today, innovation and the ability to adapt to rapidly changing markets and technologies are more important. This shift in emphasis has increased the value of, and need for, knowledge creation.

> ***"When markets shift, technologies proliferate, competitors multiply, and products become obsolete almost overnight, successful companies are those that consistently create new knowledge, disseminate it widely throughout the organization, and quickly embody it in new technologies and products."***
>
> – Ikujiro Nonaka[1]

An organization must "outlearn" the competition by developing strategies for creating and managing knowledge—not only knowledge of markets and technologies, but also intellectual capital, such as team members' expertise, past experience, insights, and design best practices. There is no limit to how much knowledge an organization can create or to how many innovations can arise from that knowledge. Often, however,

[1] *Nonaka, I., (1991). "The Knowledge-Creating Company."* Harvard Business Review, *July 2007.*

knowledge is underused because the organization lacks the infrastructure and means to capture and transfer it.

Knowledge capture and management is a core strength of Model-Based Design. A simulation model is articulated knowledge—a distillation of the model builder's expertise, skills, and experience. Other engineers can experiment with the model to increase their own knowledge. In this way, models can transfer and even create knowledge.

Before considering knowledge management with Model-Based Design, it will be useful to briefly review some well-known theories of knowledge acquisition, transformation, and transfer. An understanding of how knowledge is acquired and transformed will help managers build infrastructures that facilitate and advance the process.

What Is Knowledge?

Ackoff (1989) differentiates knowledge from data, information, and wisdom:

Data is an unconnected fact, statistic, or statement that carries no meaning by itself. Example: 75.

Information is processed data points. Example: 75 km/h is the speed of a car.

Knowledge is a pattern developed from information that makes it possible to predict future trends and behaviors. Knowledge is created by attaching new information to an existing knowledge pattern. Example: using the information above, together with an understanding of how cars work (the pattern), to estimate time of arrival at a destination.

Wisdom is a context-independent understanding of basic principles derived from the knowledge. Example: an understanding of how a car works in comparison to other vehicles, or how speed affects a specific traffic situation.

These four elements form a hierarchy. Progression up the hierarchy is enabled by understanding (Figure 4.1).

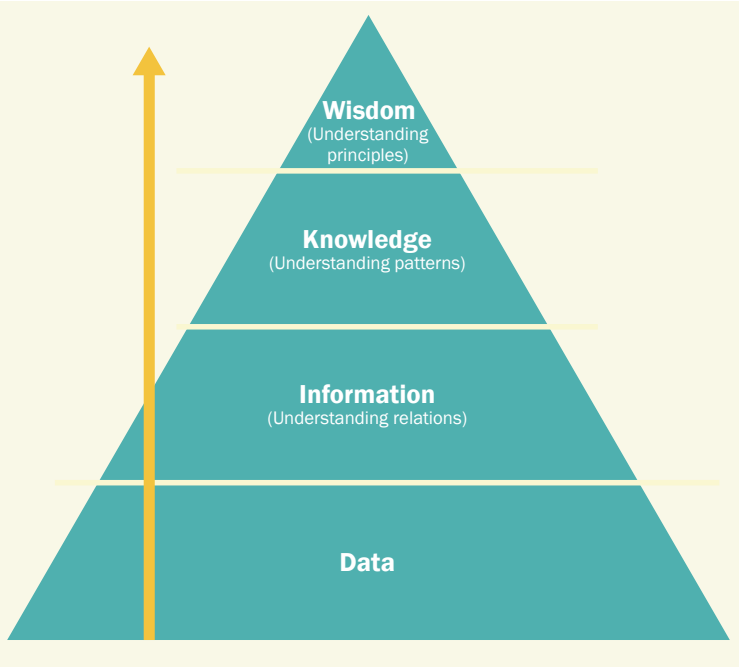

Figure 4.1. The progression from data to information, knowledge, and wisdom.

The two stages in the middle of the hierarchy—information and knowledge—are the primary concern for most managers. The goal is for team members to see patterns in information and apply those patterns to new tasks or situations. To create an environment that optimizes this process, it is first necessary to understand the relationship between tacit and explicit knowledge, and to consider the knowledge-creation spiral.

Tacit and Explicit Knowledge

Nonaka and Takeuchi (1995) define tacit knowledge as subjective, experience-based knowledge that cannot easily be expressed in words, such as cognitive skills, beliefs, mental models, and technical know-how. Explicit knowledge is knowledge that can be codified and captured in manuals, databases, presentations, models, and other media.

New knowledge is created when these two types dynamically interact—when tacit knowledge is converted to explicit, and vice versa. The interaction occurs in one of four conversion processes:

Internalization—converting knowledge from explicit to tacit. Internalization is the process of building or extending a mental model by absorbing explicit knowledge, such as knowledge acquired from a presentation or a prototyping experiment.

Socialization—converting tacit knowledge in one person to tacit knowledge in another. In socialization, individuals acquire new knowledge by interacting with those who already possess it. This process does not require language. For example, a new employee working with someone more experienced will learn by observation, imitation, and practice.

Externalization—converting tacit knowledge to explicit. Externalization is the most important mode of knowledge conversion for organizational knowledge creation. It is often accomplished with the help of metaphors, analogies, concepts, hypotheses, or models. Externalization occurs when tacit knowledge is acquired from others—for example, from customers or technical experts—and translated into a readily understandable form, such as a presentation or model.

Combination—converting knowledge from explicit to explicit. Combination enables knowledge to be transferred among individuals and groups and across organizations through documents, email, databases, meetings, or briefings. It occurs when different types of explicit knowledge are systematically combined to produce new conclusions. For example, you might combine knowledge about an organization's internal conditions with knowledge about external environmental factors to develop a new business strategy.

The Knowledge-Creation Spiral

Nonaka and Takeuchi describe organizational knowledge creation as a spiral (Figure 4.2). The spiral comprises the four knowledge-conversion processes and two additional processes: justification and dissemination.

With **justification**, a new idea, concept, or piece of information is evaluated by means of questions, such as "Is the new concept worth pursuing?" "Do customers like it?" "Is it technically feasible?"

With **dissemination**, newly created knowledge is spread within and outside the organization, where it can be internalized by other individuals who build up their own tacit knowledge. It can then be used for product development, for process improvement, or in other ways.

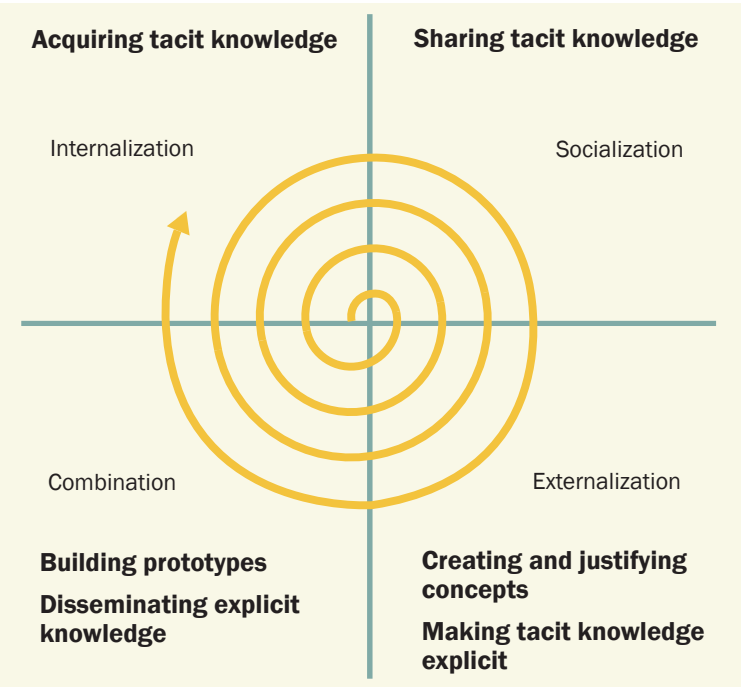

Figure 4.2. The knowledge-creation spiral.

The knowledge-creation spiral starts with an individual's thoughts or understanding (internalization). It moves up through socialization, where individuals interact with colleagues and generate new ideas. Those ideas are then articulated (externalization) and questioned (justification). They become more widespread through the dissemination of new conclusions derived from the combination of pieces of explicit knowledge.

As knowledge becomes more widespread, the spiral expands. At the same time, as individuals access organizational knowledge, apply it, and internalize it, they set the stage for an enhanced piece of knowledge to work its way up the spiral.

Managing Knowledge Creation

Progress through the knowledge-creation spiral depends on efficient management of the four knowledge conversions. Different types of organizations have strengths in different phases of knowledge conversion. Bureaucratic organizations enable knowledge combination and internalization; because a bureaucratic organization primarily handles explicit

knowledge, combination is a natural process. Individuals can build up their own tacit knowledge and internal understanding from the explicit knowledge available in the organization.

Organic organizations, which are built on networks and task forces, are more suited to socialization and externalization. Including individuals from different disciplines on a project creates a foundation for sharing tacit knowledge. When individuals communicate and negotiate in the informal, organic organization, they need to externalize their tacit knowledge, perhaps acquired by prototyping, to explain their views.

The following sections show how Model-Based Design can support or enable internalization, socialization, externalization, and combination within both bureaucratic and organic organizations.

Enabling Internalization

Internalization is a classic form of learning, and common in academia. Students gain further understanding by absorbing (internalizing) explicit information from books, papers, or discussions.

To internalize knowledge efficiently, you need an existing framework of information on which to attach the new knowledge.

Models are much more efficient than books or reports for conveying knowledge. They also provide a built-in framework on which to attach new knowledge. The visual aspect of a model makes it easier to understand the big picture and to extend the framework by attaching the new knowledge. Using simulation makes it possible to experiment with the model and get accurate results without ambiguity.

Enabling Socialization

In the socialization phase of knowledge creation, individuals learn from each other by spending time together. In more bureaucratic organizations, socialization is considered wasteful. That trend is changing, as offices are increasingly being designed around an open plan, with areas designed to facilitate and encourage interaction and informal discussion.

Model-Based Design introduces a language and a way of thinking that improves—and in some cases, actually enables—understanding and communication. For example, when team members are working in a graphical environment, it is much easier for them to see and understand

what is going on than if they were using written documents or code. Socialization is enabled because all team members share a common language. They can use a system model to store their own tacit knowledge and to access the tacit knowledge of their colleagues.

Enabling Externalization

Externalization of knowledge, whether in the form of a book, report, presentation, or even software, is often valued and rewarded in an organization, since it produces a tangible result. Consequently, organizations tend to overemphasize its importance, with the result that they often deplete the store of internal or tacit knowledge.

When experts use Model-Based Design to build up models of a system, they externalize tacit knowledge about the system. A model supports the process of externalization and serves as a carrier for the knowledge. Other team members can experiment with the model and build their own understanding of the system.

Enabling Combination

Combination is fundamental to creative thinking and innovation, yet it is the most underestimated knowledge-conversion activity. A common assumption is that creativity must involve something new. In fact, the most creative solutions often come from new combinations of existing knowledge. Many "new" ideas are not new at all—they are old ideas applied to new areas.

Combination is not only about combining ideas; it is also about combining disciplines to create more complete solutions. For example, when a hydraulic valve is combined with a computer, a computer-controlled hydraulic valve is created.

While combination is important for innovation, it can be challenging to bring about because it introduces problems of communication between engineers from different disciplines, such as between hydraulic engineers and computer engineers.

Modeling the environment (plant) is one way to design and simulate multidomain systems (see the example "Using a Plant Model to Capture and Manage Knowledge"). Not only does the plant model offer early verification capabilities; it also provides a common language for individ-

uals from different disciplines or groups to communicate and share best practices.

Building a Knowledge-Creation Environment

Some environments foster knowledge creation more effectively than others. Nonaka and Takeuchi (1995) identify five factors that are key to a knowledge-creation environment: intention, autonomy, chaos, redundancy, and variety.

Intention—An organization must articulate knowledge creation as an intention and include it in visions and strategies.

Autonomy—Individuals should be permitted to work as independently as possible. When employees take responsibility for their work, they have to think and make decisions. The result adds to the knowledge base.

Chaos—Individuals become more creative when they work in a "chaotic" environment—one where there is a sense of urgency (Sveiby, 1994). When the work environment is too ordered, creativity can suffer. Management can foster a sense of urgency by setting challenging goals.

Redundancy—An organization should provide more staff and resources than the project strictly requires so that team members can spend time on knowledge creation. The redundancy can be introduced by defining overlapping roles or by strategic job rotation.

Variety—When team members come from a variety of backgrounds and bring diverse skills and experience to a project, as they do in cross-functional teams, different inputs and perspectives are introduced, so learning and knowledge creation increase.

Example

Using a Plant Model to Capture and Manage Knowledge

An automotive engineering team is designing a supervisory control system for a hybrid vehicle. They begin by building high-level concept models of the plant and controller, which they use to select the most suitable power-split architecture. After architecture selection, the team adds detail to the control system model, including vehicle speed, battery state of charge, and driver torque request. The plant model, also reused from the concept selection stage, now includes the engine, motors, batteries, brakes, and powertrain. The engineers simulate the plant and controller together as a system, adding details to the plant model as new details are implemented in the controller.

After testing the controller via simulation, the team has sufficient confidence in the design to test it in a test rig. The results from these tests differ in important respects from the simulation. This means that the plant model must contain some inaccuracies. Further on, the test rig gives the team more insight into how the plant works. They refine the plant model based on these insights to make it more accurate. This is the critical step: New knowledge has been externalized and stored in the model, ready for sharing.

The shared modeling environment makes it easy for team members to collaborate. By dynamically interacting with the models, they create new knowledge together and build on each other's experience and findings. They pool their skills and understanding of the system to optimize performance. Because the model includes key system information, they use it to demonstrate the new design to customers and other stakeholders.

They test all the components, assemble the powertrain, and install the system in a vehicle for onroad testing. Each insight acquired during testing is captured in the model.

They check the plant model into a central repository, where everyone in the organization can use it. As others experiment and innovate, they, in turn, add knowledge to the repository, building the company's collective intellectual property.

Key Takeaways from This Chapter

- Development organizations today must outlearn the competition by developing strategies for creating and managing knowledge.
- New knowledge is created when tacit and explicit knowledge dynamically interact in one of four conversion processes: internalization, socialization, externalization, and combination.
- Organizational knowledge creation is a spiral process in which individuals access, apply, and internalize organizational knowledge, setting the stage for an enhanced piece of knowledge to work its way up the spiral.
- Knowledge capture and management is a core strength of Model-Based Design.
- Model-Based Design supports knowledge conversion and the knowledge-creating spiral. It uses a model to capture knowledge about a system, store new information, and provide a common language and frame of reference for team members. Team members can then share their knowledge and learn from each other.

5

Enhancing Work Performance with Model-Based Design

This chapter outlines the key requirements for successful work performance and explains how the core concepts of Model-Based Design can improve both individual and team performance.

It is often assumed that the only requirement for successful work performance is ability. However, an individual with the right skills but no motivation will not perform well, and neither will a skilled individual without the opportunity to exercise those talents. Three elements are critical to successful work performance: motivation, opportunity, and ability. In addition, managers must provide clearly defined performance goals.

The following sections discuss ways to increase your team members' motivation, opportunity, and ability with Model-Based Design.

Performance and Motivation

At some point, most managers struggle with finding ways to keep their teams motivated. Sustaining motivation is easier for small, self-organized teams in an organic organization than for hierarchically organized teams in a bureaucratic organization. However, it is challenging for any organization to motivate staff to perform the essential but repetitive tasks that are part of any development process.

Managers can make even these routine tasks motivating if they provide the following:

Autonomy. Team members can plan their own work and affect its outcome.

Feedback. Team members see the direct results of their task or project—the more immediate the feedback, the higher the motivation.

Skill variety. Completing the task requires several skills. For example, writing and compiling C code requires only C programming skills. When the task also includes testing the code and designing interfaces, however, it requires more skills and, as a result, is more motivating.

Task identity. Each team member takes pride in the work and is invested in its outcome.

Task significance. Several teams or individuals depend on the outcome of the task. The more people who depend on the outcome, the higher the perceived significance of the task.

Improving Motivation with Model-Based Design

Model-Based Design supports, and in many cases enables, each motivational factor outlined above.

Autonomy. With a model of the entire system, a single engineer can test ideas and experiment with different solutions. Automation can increase the number of tasks that a single engineer can perform. For example, code generation for rapid prototyping means that a control developer can test the controller on the target without involving software engineers. Models or parts of models can be implemented first at a high level and then at a more detailed level. As the model still simulates with the rest of the system, the developer has some autonomy in deciding what to change.

Model-Based Design provides the infrastructure to make work more autonomous. For example, with a system model one team member can implement an idea and then test it by simulating it with the entire system. In addition, Model-Based Design supports a decentralized approach to development, with task forces or small teams responsible for specific system components.

Feedback. Simulation provides almost instantaneous feedback on a design idea or modification. For example, suppose that a control engineer is tasked with selecting the best controller gains. With a traditional approach (changing the parameters in C code, compiling the code, downloading it to a target, and running it against hardware), the engineer has little motivation to experiment. How many parameter changes will be required? How much damage will be done to the hardware before the right parameters are found? With a simulation model, a parameter change can be made in seconds. The engineer can try many new ideas and parameters quickly and without risk.

Skill variety. Model-Based Design makes it easy for team members to take on tasks requiring new skills. When all project information is captured in a system model, it is easier for a manager to provide skill variety because individuals can rotate among different positions within the team. For example, with code generation, algorithm developers, such as control engineers, can become responsible for the software as well. Model-Based Design supports both vertical skill variety (the worker takes on increasing responsibility for different parts of the system or development process) and horizontal skill variety (the task calls for a range of technical skills).

Task identity. System simulation gives team members the "big picture." For example, simulation shows how a component that they work on fits into the whole system. A system-level view extends the area that each engineer can address. A control engineer, for example, can use simulation to propose changes in the hardware. In some cases, you might be able to let an engineer follow the development process from concept to production. If specialization is required, you can use the system model to visualize and demonstrate later steps in the process.

With automation, the control engineer can write a script that runs through a sequence of parameters and automatically finds an optimal set. Finding the optimal parameter adds an element of pride to the solution and, therefore, increases task identity.

Task significance. Simulation clearly reveals the effect of a small component on the whole system. Managers can use simulation to demonstrate the importance of a task or component, or its connection to other parts and dependencies. Allowing developers to explore their ideas using what-if analysis and simulation makes them feel that their contributions are meaningful.

Performance and Opportunity

The opportunity to exercise skills, influence the outcome of a project, and suggest or explore new ideas is probably the most overlooked aspect of performance. Lack of opportunity can be a result of organizational structure, the skill sets of other team members, or the constraints of project deadlines. Whatever the cause, there are several steps a manager can take to provide team members with more opportunities to perform well. For example:

- Give them access to key information, resources, and people.
- Actively negotiate for the resources they need.
- Allow them time to work on special ideas or workflow improvements.
- Establish a culture in which knowledge and resources are shared and communication flows.

Increasing Opportunity with Model-Based Design

Ensuring that opportunity exists for employees is primarily a social and management concern. However, the core concepts of Model-Based Design enable team members to create opportunities for themselves by exploring and testing new ideas. With system-level simulation, an idea can be evaluated in a time-efficient and cost-effective way. No expensive hardware or resource coordination is needed. Engineers can even use system-level simulation to test their ideas against the full system. To provide the opportunity for this, managers can set up an open, freely accessible repository that includes all relevant models. This repository can become a mechanism for capturing interesting and feasible new ideas.

Performance and Ability

Ability is, of course, a vital ingredient of performance, and most managers devote considerable time and attention to improving the skill sets of their teams. The traditional approach to skill enhancement is through training courses. In many cases, training focuses on improving weaknesses, which, at best, results in average performance. Less formal methods such as coaching and mentoring, which focus on the individual, can be more effective. A key requirement is that each individual realizes his or

her own potential and has a clear path to success. For many individuals, having clear goals is sufficient motivation for them to start learning by themselves.

Developing Individual Ability with Model-Based Design

The tools, infrastructure, and core concepts of Model-Based Design enhance a team member's abilities. For example, graphical tools with simulation capabilities extend an engineer's ability to understand and develop complex systems. Automation, such as code generation, enables an engineer to work more quickly and efficiently. It also adds to an engineer's ability to learn by reducing the risk of errors that he or she might make while learning. Human errors often occur randomly. Automation such as automatic model reviews enables a systematic approach to error correction. If an error is detected, it is fixed and never seen again.

While some engineers learn well by listening or reading, many need to try things themselves. For those individuals, what-if analysis is an excellent way to learn about a system. An engineer can experiment with a model of the system and learn how it works. In this way, the engineer increases his or her own knowledge and contributes to the knowledge of the entire team.

Improving Team Performance

To improve team performance, the manager must establish long-term goals and motivate team members to work toward those goals. Team performance is optimal when all members are working toward the same goal. Including team members in the development of the goal can increase motivation and commitment.

Managing team performance depends on efficient communication, especially when teams are cross-functional or geographically scattered. The introduction of a simulation environment and models gives the entire team a common language, even if team members have different areas of expertise and come from different countries.

Example

Increasing Motivation and Performance for an Automotive Test Engineer

A test engineer at a large automotive company is responsible for implementing new test cases when a software platform or application changes. This task carries a high degree of task significance—test systems have a clear purpose and are an important part of the development process. There can be no system test executions and no release without them, which means that others depend on this task. At the same time, however, test-case implementation is repetitive and routine.

What can a manager do to make this routine task rewarding? Can the task be turned into opportunity that motivates the engineer to excel?

Using a Traditional Approach

Using a traditional approach, the test engineer receives a document just before the new release. That document details the changes and specifies the number of test cases, the input signals to use, and the correct output. The engineer checks out the test system code from a configuration management system, adds the new test cases, and executes the system to verify the changes. This is merely a dummy test, as the changes to the application or platform have not yet been implemented. The actual test execution is completed much later in the process. If there are no compiler or execution errors, the engineer checks the code in, updates the test system document, and sends it to the project manager.

The traditional approach gives the engineer very limited autonomy—a report that initiates the task is handed over, and the work is predetermined. The engineer receives feedback only if there is an implementation error. Since the job is complete once the dummy test has been run, the engineer sees only a small part of the workflow and will never know the outcome of the real tests. The only skills required are basic programming and the ability to add the test cases and compile the code.

Because test case creation is an established, well-defined part of the development process, changing it will affect other parts. No part of the work can be removed, so the only changes that can be made are to add to the task. The manager is reluctant to do this, fearing that adding to an already unrewarding task will lower, not raise, performance.

Using Model-Based Design

By applying the core concepts of Model-Based Design, the task of implementing test cases can be made more motivating. It can also be designed to increase the engineer's skill set and to present the engineer with opportunities to excel.

For example, skill variety can be greatly improved. As a complement to implementing test cases, the manager can introduce formal property proving and additional tools for verification and validation. The engineer can have a side project to develop an automated test process with nightly builds and automatic execution of test cases.

The manager can create a test design task force that includes the engineers who implement test cases, those who run the tests, and those who write the test case specifications. This task force can meet regularly to identify issues with the test process and discuss ways to improve it. Members can spend some of their time implementing the improvements.

Letting the test engineer implement improvements and participate in task force meetings significantly increases motivation and job satisfaction. Feedback improves because the test engineer is part of a team that sees the overall result of the tests. Task identity increases because the engineer sees more of the process and is able to affect the outcome. Skill variety increases because more skills are needed to develop the process and implement improvements. Task significance also increases because affecting the entire test process affects other parts of the organization.

With the task force up and running, the manager can shift from supervising the team to coaching and advising and, thereby, continue to improve the team's performance.

Key Takeaways from This Chapter

- Work performance depends on three critical factors, all of which must be present for successful performance: motivation, opportunity, and ability.
- Model-Based Design increases individual performance by enabling managers to provide motivation, opportunity, and ability.
- The tools and infrastructure of Model-Based Design enhance motivation, opportunity, and ability by:
  - Providing a graphical view of the complete system to increase understanding
  - Encouraging experimentation
  - Providing a mechanism for developing and storing promising new ideas
- Model-Based Design increases team performance by:
  - Providing a common language for cross-functional or geographically scattered teams to communicate
  - Making it easier for teams to understand and commit to shared project goals

6

Managing the Shift to Model-Based Design

This chapter provides a road map to the adoption of Model-Based Design. It covers the challenges of introducing a major change to a team or an organization, the four stages of creating change, practical and logistical requirements for implementing Model-Based Design, and six steps to a successful implementation.

For the past 10 years, a transmission control engineer has developed control systems and programmed them in C code. This engineer holds a senior position, and he feels comfortable with the way he works. But then he learns that his team plans to adopt Model-Based Design and that he will have to change his development process. In the future, he will be required to develop the control system using a model, implement test cases to work with the model, and prepare the model for automatic code generation. Naturally, he feels anxious about this disruption, and doubts its value. How can a manager help this engineer not only to understand the new approach but to embrace it?

Introducing Change: General Considerations

Shifting to Model-Based Design brings the same challenges as implementing any major and potentially disruptive change: resistance from team members and upper management, skepticism about the value of the new approach, anxiety, and reduced morale. Managers can mitigate these concerns by following some general guidelines and principles:

- When advocating for the shift to Model-Based Design, clearly communicate the benefits to the organization that will result.
- Answer a question most staff members have: How will the change benefit me? Explain exactly how the change will affect individual team members. How will their roles and responsibilities change? Will they still be able use the knowledge and skills they have acquired up to this point? What tangible benefits does Model-Based Design offer them?
- Involve the team in the implementation and let team members affect the process. They will accept the change more readily.
- Avoid responding to resistance by halting the adoption before all the benefits of Model-Based Design have been realized.
- Recognize that full adoption at a large organization can take years, during which time your organizational structure is likely to change. Build flexibility into the process to account for such changes.
- Even when management has agreed to implement Model-Based Design, proceed gradually. Step-by-step adoption is often the best approach. Use short-term wins to drive the change.
- Create a rollout plan that defines the end goal, shows how you will use Model-Based Design to achieve the goal, identifies key implementation milestones, and provides a clear and realistic timeline.

The Adoption Process

Preparation is key to the successful adoption of Model-Based Design. You need to know where your team or organization is today, where you want it to be, and how you plan to get there.

Follow these five steps in your adoption process:

1. Analyze the current situation.
2. Set process improvement objectives.
3. Decide what change options you will use to reach the objectives.
4. Implement the changes.
5. Follow up.

Throughout the process, consider how you will measure the change efforts, and what metrics will be useful once you have adopted Model-Based Design (see the section "Measuring the Outcome").

While it is important to be systematic in your adoption process, avoid overplanning. Even the most prepared and planned change effort is messy—keep your plan flexible enough to deal with uncertainty.

Step 1. Analyze the Current Situation

In the analysis stage, you first determine why change is necessary. What problems will it address? What will the change achieve or improve? For example, teams that have shifted to Model-Based Design commonly cite the following goals:

- Reducing development time
- Finding errors earlier in the process
- Managing increasingly complex systems
- Managing change in the market or in product requirements
- Managing risk
- Improving communication
- Complying with standards or certification requirements

Next, you analyze the internal and external environments within which your team or organization operates.

The **external environment** includes all the social, legal, economic, political, and technological factors that affect an organization and the way work is done. For example, an automotive organization must comply with environmental laws by designing reduced-emission vehicles. For a marine engine manufacturer, engine performance—a technological requirement—could be more important.

What drives the current market for your organization? Is your goal to reach a large number of customers with rapidly changing demands, or is it to establish a few long-term relationships with large companies or organizations?

The **internal environment** focuses on the situation within the organization, and it addresses questions such as these:

- What development methodology do you use?
- What works well and what does not?
- What are the strengths and weaknesses of your team or organization?
- How does your team or organization deal with the requirements of the external environment?
- Is there a formal knowledge layer to support knowledge sharing and creation?
- Does your organizational structure aggravate or minimize the problems you identified?

Step 2. Set Process Improvement Objectives

Once the situation is clear and specific problems have been identified, you can begin setting objectives to address those problems. Select improvements that will produce quick results. These results can be used to drive the change effort.

The most effective objectives are SMART (specific, measurable, achievable, relevant, and time-bound). For example, if your goal is process improvement, you might formulate your objective as follows: "The team will identify two bottlenecks in the current development process and reduce them within nine months by using Model-Based Design."

Step 3. Select Change Options

To ensure a good result and ease the transition to Model-Based Design, managers should involve team members in generating, evaluating, and selecting options. For example, to meet the process improvement objectives defined in step 2, a team might generate the following list of options:

- Create a formal test harness.
- Verify system performance via simulation.
- Introduce rapid prototyping to verify individual components.
- Generate production code from a component model.
- Use models instead of documents for communication between research and application development.

- Implement a library with models of different parts of the environment to enable closed-loop simulations for test, verification, and experimentation with different solutions.

After discussing the options informally and applying formal evaluation criteria, such as scoring systems, managers would select two of the options:

- Create a formal test harness for selected components.
- Implement a library with environment models.

The team might select these options because they will produce the most immediate results. These short-term wins can be used to drive additional changes.

Step 4. Implement the Changes

When course of action is clear, implementation can proceed without undue disruption to the team's day-to-day activities. However, before beginning implementation, make sure that the following are in place:

- Rollout and communication plan
- Team training
- Appropriate tools
- Configuration management system
- Testing platform

Step 5. Follow Up

To ensure that future changes go smoothly, rigorous follow-up analysis is highly recommended. After implementing each change, ask these questions:

- What went well?
- What obstacles did the team encounter?
- What is the next step?
- What was learned?

Use the information gained from this analysis to guide future changes.

Measuring the Outcome

Metrics that provide reliable, objective measures of performance and progress can be used to:

- Set targets
- Communicate progress
- Evaluate the current situation against the plan
- Measure success
- Estimate return on investment (ROI)
- Influence future actions

Both quantitative and qualitative measurements are valuable, but avoid overmeasurement, which is costly and can complicate the issue. Your goal is to measure just enough. Focus on the relevant issues, such as the process bottlenecks identified in your SMART objective.

When developing metrics, follow these best practices:

- Decide what you need to measure. What results or outcomes will be monitored?
- Determine what data will provide the information you need.
- Connect the data you collect to the desired outcome. When context is added, data becomes information.
- Decide how you will present the data. Tailor your presentation to the target audience. Reports and graphs are common, but scorecards, which might include graphs or plots showing different perspectives of the data, provide a more complete picture.
- Decide how often the information will be evaluated.
- Review the data-collection process at regular intervals: weekly, monthly, or quarterly.
- Create a feedback loop in which you monitor results to help fine-tune plans for subsequent phases or follow-on projects. For example, regularly measure target execution time for each component in the system. This action will catch unrealistic implementations and give an early estimate of the processing power you need.

- Gather metrics comparing when and how many failures are found in each development phase before and after adopting Model-Based Design.

Adoption Models for Implementing Model-Based Design

Many organizations planning to adopt Model-Based Design find that adoption models help them identify where they are in the adoption process and where they want to be. An adoption model is also useful when communicating the goal to the organization. You can develop your own custom version, but best practices for implementing Model-Based Design have led to the frequent usage of two adoption models.

The nine-box model shows the extent to which your project team, department, or organization has adopted the core concepts of Model-Based Design (Figure 6.1).

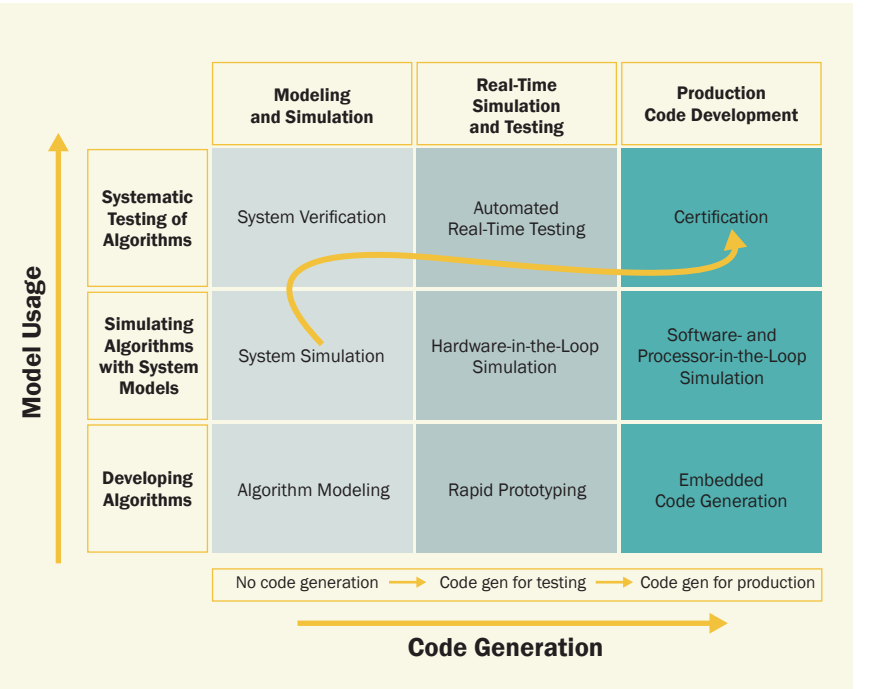

Figure 6.1. The nine-box adoption model. The arrow shows a potential adoption path for an organization that currently uses system simulation.

Figure 6.1 shows models being used only for graphical specifications. The project's goal is to fully implement the concepts of Model-Based Design. Each step on the way to full adoption provides some benefit. If that benefit is highlighted as a short-term win, it can be used to drive further change.

The industry adoption ladder shows the level of collaboration among individuals, departments, and organizations that models have enabled (Figure 6.2).

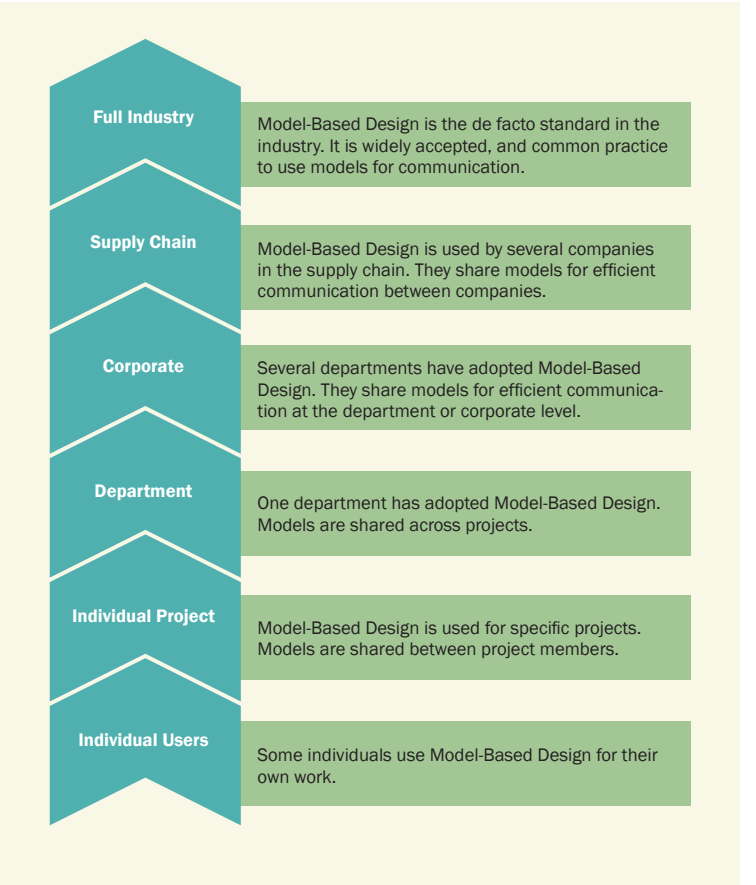

Figure 6.2. Adoption ladder for Model-Based Design, focusing on model usage.

Six Ways to Ensure a Successful Implementation

Most organizations that have successfully adopted Model-Based Design follow the six steps outlined below. These steps might overlap or be

tackled in a different order, but all must be completed to ensure success.

Specify the problem you want to solve and communicate it to your team. Be clear about why you are adopting Model-Based Design, both to ensure that you are solving the right problem and to enable you to communicate the reason effectively. For example, if reduced time to market is the reason, show a graph illustrating the historical year-over-year increase in the time required for system integration and test. Then, present graphs demonstrating the consequences—for example, the number of new features that were developed but not included in the final product.

Create a team to drive the implementation. This team must have enough influence in the organization to coordinate and drive the change. Later in the process, the team members will develop and maintain best practices and customizations and be the go-to people for questions about Model-Based Design. Include the R&D manager and one individual from each group (research, platform, and application development). It is a good idea to include one individual who is resistant to the change.

Create and communicate a vision. Describe a desirable future for the organization. What will the organizational workflow, work style, and culture be like after the change? How will the change benefit individual employees? How will it change their jobs and their status in the organization?

Enable team members to act—remove obstacles. Provide the necessary training on new tools, workflows, and best practices. Run basic training for all employees affected by the change, and specialized training for those with specific responsibilities. Remove obstacles, such as unavailable software licenses.

Plan for and create short-term wins. To maintain motivation and continue to drive the adoption, demonstrate all the improvements that have resulted from the change within a few months. When communicating these improvements to the team, use the terminology and core concepts of Model-Based Design.

Institutionalize the new approaches. Once Model-Based Design has been implemented, set up a system to ensure continuous improvement. Maintain customizations, workflows, and best practices. Provide a mechanism that allows others to submit suggestions and work improvements. Capture and analyze lessons learned from completed projects.

Key Takeaways from This Chapter

- Implementing Model-Based Design is likely to bring the same challenges as any major and potentially disruptive change: resistance from team members and upper management, skepticism, anxiety, and reduced morale.
- A successful adoption process includes five steps:
  1. Analyze the current situation.
  2. Set process improvement objectives.
  3. Select change options.
  4. Implement the changes.
  5. Follow up.
- Demonstrating short-term wins builds momentum and bolsters the change effort.
- The following must be in place before implementation of Model-Based Design begins:
  - Rollout and communication plan
  - Team training
  - Appropriate tools
  - Configuration management system
  - Testing platform
- Use appropriate metrics to evaluate the change process and its results.
- Adoption models, such as the nine-box model or the adoption ladder, can help you evaluate the progress of the implementation and communicate it to the team or organization.

7

Measuring the Value of Model-Based Design

This chapter reviews common approaches to evaluating a new workflow or business strategy. It identifies where Model-Based Design adds value to an organization, project, or workflow, and which criteria and metrics to use in order to measure that value.

The classic way to measure the value of a new approach or business strategy is to calculate return on investment (ROI). Financial returns are important for organizations whose corporate strategies are based on cost savings, but cost savings are not always the primary concern. A company making large mechanical constructions, such as gas turbines, where software is only one part and new features are customer-driven, might focus on cost. On the other hand, a company that produces enterprise software, where production costs are minimal but a constant stream of innovative products is essential to retaining market share, might prioritize innovation over cost.

When measuring the value of Model-Based Design for your project, team, or organization, by all means calculate the cost savings—especially in the early stages of adoption, when short-term financial wins can help drive further adoption efforts. However, ROI calculations support further investment rather than showing the full value of implementing Model-Based Design. Be sure to consider additional factors. What were your goals, both short-term and long-term? What did you set out to improve by implementing Model-Based Design? What objectives did you set? Which core concepts of Model-Based Design did you implement?

Calculating the ROI of Model-Based Design

While most of the core concepts of Model-Based Design yield measurable ROI, automation is the most direct source. It is also the easiest to demonstrate. For example, you might calculate the value of automatic code generation as follows:

If the cost of writing code today is $5 million per year, the best possible savings would be $5 million. From that amount you would subtract the cost of tools, as well as the number of engineer- hours spent on creating models, setting up test harnesses, and simulating the models for verification and validation. You would then factor in the indirect benefits of automatic code generation. For example, you might find that since the adoption of code generation, the number of code failures has dropped significantly. The resulting cost savings would be included in the ROI calculation.

The cost savings from reducing the number of code failures can be high, but they still provide a limited view of the ROI of Model-Based Design. For example, you might find that the adoption of Model-Based Design has improved your team's ability to manage complex systems. This ability can be a significant advantage—even essential for survival in a competitive market—and its ROI is best measured in terms of product sales rather than cost. Similarly, you might find that Model-Based Design has saved time, enabling your team to focus on creating innovative designs and products. You would calculate the ROI of innovation not by time and cost savings, but by the increases in new product development and market share that have resulted.

When presenting this wider view of ROI to colleagues or upper management, be sure to set expectations by pointing out that the actual return sometimes comes a long time after the investment.

In summary, cost saving is an important factor but not the primary means of evaluating the benefits of Model-Based Design. Focus your evaluation on the longer term and on how Model-Based Design enables your organization to stay competitive. The ability to manage more complex systems, adapt to change, and to innovate are the key considerations, not the specific amount saved.

Alternative Approaches to Measuring the Value of Model-Based Design

While ROI is an effective measure of tangible assets, such as real estate, inventory, or stock prices, the primary impact of Model-Based Design is on an organization's intangible assets.

Intangible assets are commonly classified as three types (Sveiby, 1997):

- External structures—brands, relationships with customers, and relationships with suppliers
- Internal structures—management, legal structure, manual systems, employee knowledge and experience, R&D, and IP
- Employee competence—education, experience, and ability

Managers can drive further adoption efforts by demonstrating how Model-Based Design—in particular, the use of models and simulation—improve or enhance these intangible assets.

External Structures

As previous chapters have shown, using models to capture requirements and product specifications improves communication within and among teams. When you share models with your customers and suppliers, you also improve a key component of your organization's external structure: customer relationships.

For example, suppose that a manufacturer of DC motors and controllers sends a controller model to its customer. This model is legally accredited as a proposal. The customer uses the proposal model in a system simulation to verify its performance. Simulation reveals an error in the step disturbance response. The customer sends the model back to the development team and requests refinements. The team quickly implements the new requirements and sends the updated model back to the customer. These iterations continue until the controller behavior is satisfactory.

Both the customer and the manufacturer benefit from this interaction. The customer gains reliable designs and a swift response to requests for changes, while the manufacturer gains system information that can be stored in the knowledge layer and applied to future projects. Communication is unambiguous because a model reduces the chance

of miscommunication or error. Iterating on the model improves the relationship by fostering a sense of collaboration, openness, and trust.

Brands can be strengthened by more innovative products, higher quality, and fewer failures.

Internal Structures

Many organizations find that the value they most realize from Model-Based Design in this area is a significantly enhanced knowledge layer. When used properly, a model repository and processes for its use are important internal structures that can increase a company's value.

A sophisticated model repository that incorporates model accreditation levels and related documentation can form the core of a company's innovation process, as well as its development process. Deviations from requirements and faults in software are continuously fed back to improve the models.

Employee Competence

System modeling and simulation are proven ways to acquire experience and competence. This is true for small simulations to develop control systems, all the way up to training simulators for pilots and astronauts.

To encourage experimentation and foster learning, managers should make the repository as open and accessible as is compatible with protecting IP. New employees can use the model repository to build up their knowledge. Experienced engineers can use it to teach new team members about the system under development and to share their own skills and experience.

Key Takeaways from This Chapter

- Calculating cost savings can be a useful measure of the value of Model-Based Design in the early stages of adoption, when it can be used to demonstrate short-term wins.
- Cost savings motivate further investment rather than showing the full value of implementing Model-Based Design.
- The full value of Model-Based Design should be sought in product sales.
- Cost savings calculations do not capture increases in the ability to manage change, handle complexity, or develop innovative designs and products that have resulted from the adoption of Model-Based Design.
- While most of the core concepts of Model-Based Design yield measurable ROI, automation is the most direct source and the easiest to demonstrate.
- ROI is an effective measure of tangible assets, such as real estate, inventory, or stock prices, but the primary impact of Model-Based Design is on an organization's intangible assets.

Glossary

Automation. The practice of using scripts and tools to perform repetitive or error-prone manual tasks.

Autonomy. Freedom for team members to plan their own work and affect its outcome, thereby increasing their motivation.

Bureaucratic organization. A centralized, hierarchical organization that follows specific, formalized procedures.

Ceremony. Formalized procedures, documentation, review processes, and metrics.

Combination. A means of acquiring knowledge by combining different types of explicit knowledge to produce new conclusions.

Continuous test and verification. The practice of simulating a design at every stage of development using techniques such as rapid prototyping and hardware-in-the-loop.

Data. Unconnected facts, statistics, or statements.

Dissemination. A means of sharing knowledge by spreading newly created knowledge within and outside an organization.

Executable specification. A model that encapsulates design requirements at a specific level of detail.

Explicit knowledge. Knowledge that can be codified and captured in manuals, databases, presentations, models, and other media.

Externalization. The practice of translating tacit knowledge acquired from others into a readily understandable form, such as a presentation or model.

Extreme programming (XP). A development methodology based on agile principles that uses pair programming and short development cycles, or releases, with built-in checkpoints for introducing new requirements.

Information. Processed data.

Intangible assets. Company assets that are not physical in nature, such as brands, customer relationships, employee knowledge and experience, and intellectual property.

Internalization. The practice of building a mental model from explicit knowledge, such as knowledge acquired from a presentation or a prototyping experiment.

Iterative and incremental development (IID). A development methodology that proceeds in cycles (releases), with each cycle resulting in a partially complete system for integration and testing.

Justification. The evaluation of a new idea, concept, or piece of information by means of questions.

Knowledge. A pattern developed from information that can be used to predict future trends and behaviors.

Knowledge capture and management. The practice of using models to store all project information and transfer it to teams, customers, and suppliers.

Knowledge-creation spiral. A model representing the way new knowledge is created in an organization.

Lean development. A development methodology based on specific principles and core values, with a focus on continuous improvement and outlearning the competition.

Model-Based Design. A model-centric approach to the development of control, signal processing, communications, and other dynamic systems.

Model elaboration. The iterative process of turning a low-fidelity system model into a high-fidelity implementation.

Organic organization. A decentralized organization in which all team members participate in decision-making, and projects are coordinated by means of dynamic negotiation.

Plant model. A model of the part of a system that needs to be controlled.

Scrum. A methodology in which development tasks are performed by small, self-managed teams during two-week to four-week periods known as sprints.

Socialization. A process in which an individual acquires new knowledge by interacting with those who already possess it.

Spiral development. A methodology in which four development stages—requirements, design, implementation, and testing—are completed in one-year or two-year cycles that focus on certain features of the whole system.

System-level simulation. The practice of simulating a model of the system to investigate system performance and component interactions.

Tacit knowledge. Subjective, experience-based knowledge, such as cognitive skills, beliefs, mental models, and technical know-how, that cannot easily be expressed in words.

Tangible assets. Assets that have a physical form, such as real estate, inventory, or stock.

Task identity. A motivational technique used to ensure that each team member takes pride in his or her work and is invested in its outcome.

Task significance. A measure of the effect of a task on others.

V-model. A development methodology comprising five steps or phases—requirements, design, implementation, verification, and maintenance—where each development step is matched with a corresponding test phase.

Virtual prototyping. A technique that uses simulation to validate a design before hardware is available.

Waterfall. A development methodology comprising five steps or phases—requirements, design, implementation, verification, and maintenance—where each step must be completed before the next begins.

What-if analysis. A simulation method used to test ideas and build knowledge about a system.

Wisdom. Context-independent understanding of basic principles derived from knowledge.

Bibliography

Ackoff, R. L., "From Data to Wisdom," *Journal of Applied Systems Analysis,* Vol. 16, 1989, pp. 3-9.

Bellinger, G., "Knowledge Management—Emerging Perspectives," *Systems Thinking,* 2004, retrieved 18 February 2015 from systems-thinking.org/kmgmt/kmgmt.htm.

Blundell, R., R. Griffith, and J. van Reenen, "Market Share, Market Value and Innovation in a Panel of British Manufacturing Firms," *Review of Economic Studies,* Vol. 66, No. 3, 1999, pp. 529-554.

Boehm, B., "A Spiral Model of Software Development and Enhancement," *Computer,* Vol. 21, Issue 5, May 1988, pp. 61-72.

Boehm, B., and R. Turner, *Balancing Agility and Discipline: A Guide for the Perplexed,* Addison-Wesley Professional, Boston, 2004.

Cockburn, A., *Agile Software Development,* Addison-Wesley Professional, Boston, 2001.

Dahlbom, B. and L. Mathiassen, *Computers in Context: The Philosophy and Practice of Systems Design,* Blackwell Publishers, Cambridge, 1993.

Hackman, J., and G. Oldham, "Motivation Through the Design of Work: Test of a Theory," *Organizational Behavior and Human Performance,* Vol. 16, 1976, pp. 250-279.

IABG, "V-Model Lifecycle Process Model," IABG Ottobrunn, 1993.

Kotter, J., *Leading Change,* Harvard Business School Press, Boston, 1996.

Larman, C., and B. Vodde, *Scaling Lean & Agile Development,* Addison-Wesley Pearson Education, Inc., 2008.

Nonaka, I., "A Dynamic Theory of Organizational Knowledge Creation," *Organization Science,* Vol. 5, No. 1, 1994, pp. 14-37.

———, "The Knowledge-Creating Company," *Harvard Business Review,* November-December 1991, pp. 96-104.

Nonaka, I., and H. Takeuchi, *The Knowledge-Creating Company,* Oxford University Press, New York, 1995.

Poppendieck, M., and P. Poppendieck, *Lean Software Development,* Addison-Wesley Professional, Boston, 2003.

Royce, W., "Managing the Development of Large Software Systems," Proceedings, IEEE WESCON 26, August 1970, pp. 1-9.

Sveiby, K.-E., "Kreativitet och Makt" [Creativity and Power], RPS-FK Reprocentral, 1994.

Sveiby, K.-E., *The New Organizational Wealth: Managing and Measuring Knowledge-Based Assets,* Berrett-Koehler Publishers, Inc., San Francisco, 1997.

The Eight Core Concepts of Model-Based Design

Executable specification

A model that encapsulates design requirements at a specific level of detail

System-level simulation

The practice of simulating a model of the system to investigate system performance and component interactions

What-if analysis

A simulation method used to test ideas and build knowledge about a system

Model elaboration

The iterative process of turning a low-fidelity system model into a high-fidelity implementation

Virtual prototyping

A technique that uses simulation to validate a design before hardware is available

Continuous test and verification

The practice of simulating a design at every stage of development

Automation

The practice of using scripts and tools to perform repetitive or error-prone manual tasks

Knowledge capture and management

The practice of using models to store all project information and to transfer that information to teams, customers, and suppliers

TECHNOLOGY & ENGINEERING

Managing Model-Based Design

In *Managing Model-Based Design* Roger Aarenstrup draws on a decade of experience helping engineering teams and organizations successfully adopt Model-Based Design. Engineers will learn how to demonstrate the value of Model-Based Design to colleagues and key decision-makers. Engineering managers will gain insight on using Model-Based Design to help teams adapt to change, manage complexity, and foster innovation.

> "This book is the guide that gives a manager courage enough to start the transformation!"
>
> ***–Jenny Elfsberg, Director Emerging Technologies, Volvo Construction Equipment***

What Organizations Are Saying About Model-Based Design

"Model-Based Design . . . enables dramatic reductions in development time and cost, as well as seamless integration between research, development, and production."

–Dr. Seungbum Park, Hyundai Motor Company

"We'll use Model-Based Design from now on because it reduces risk, saves time, lowers costs, and increases our confidence in our designs."

–David Erhart, Stem

"Using Model-Based Design we developed a complex control system in significantly less time than our traditional process would have required. We eliminated months of hand-coding by generating code from our models, and we used simulations to enable early design verification."

–Anthony Totterdell, Alstom Grid

"The fuel system of the A380 is three to four times more complex than that of the A340. Model-Based Design enabled us to handle a substantially more complex project with the same size engineering team."

–Christopher Slack, Airbus
