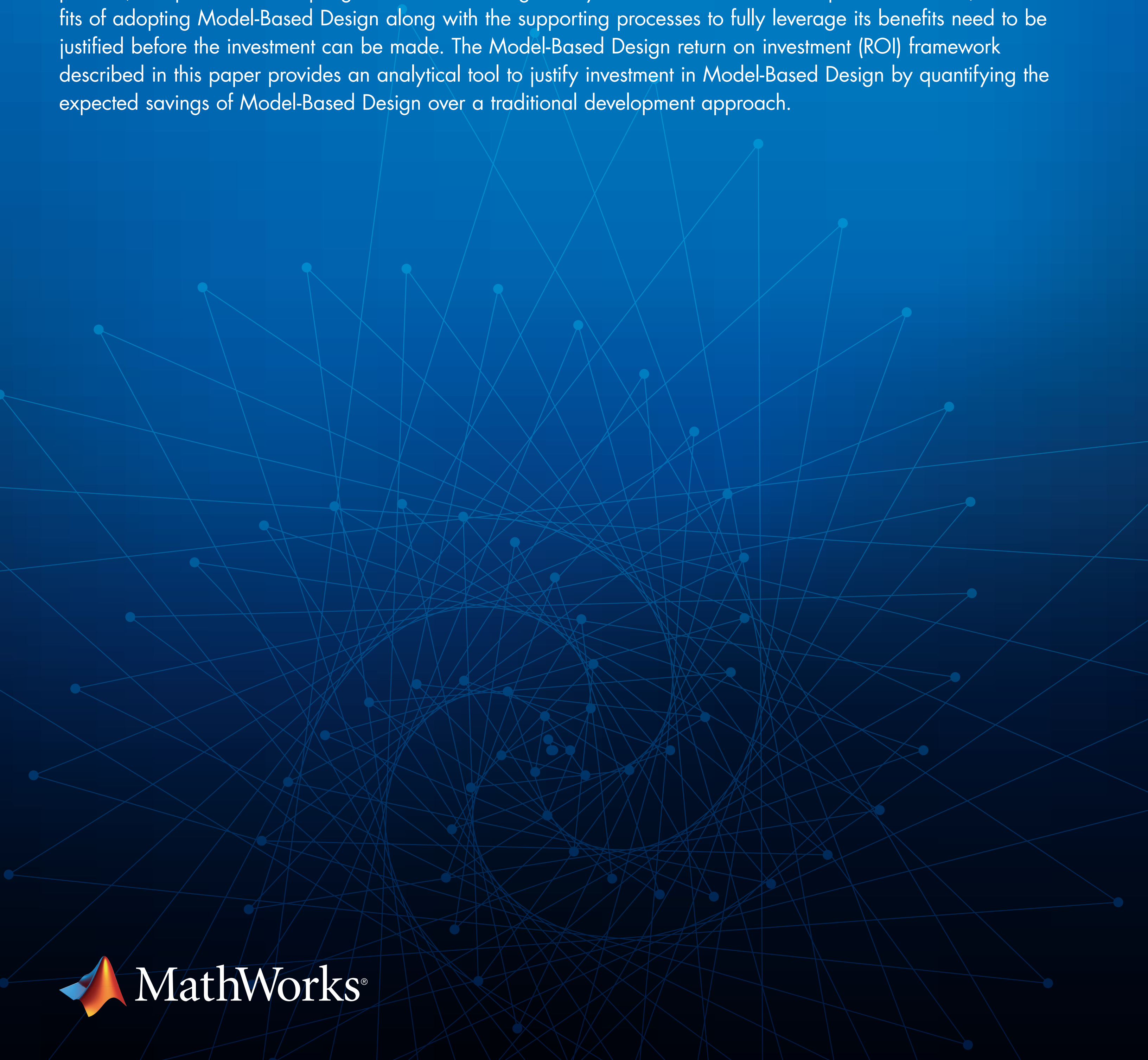
WHITE PAPER

Measuring the Return on Investment of Model-Based Design

As cyber-physical systems become more complex, it is becoming more difficult to maintain quality and control costs with traditional approaches to system development. To meet this challenge and improve their competitive position, companies are adopting Model-Based Design for systems and software development. However, the benefits of adopting Model-Based Design along with the supporting processes to fully leverage its benefits need to be justified before the investment can be made. The Model-Based Design return on investment (ROI) framework described in this paper provides an analytical tool to justify investment in Model-Based Design by quantifying the expected savings of Model-Based Design over a traditional development approach.



MathWorks®

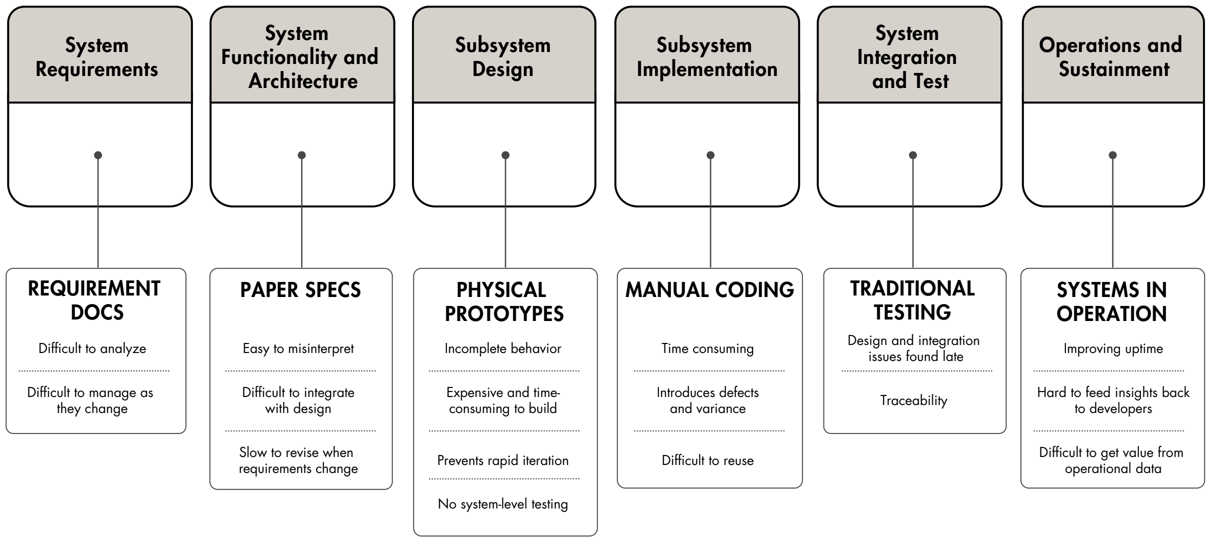
Introduction

As customer requirements increase in scope and complexity, the logic and control software for systems has also grown in scope and complexity. As organizations develop the millions of lines of code required for airplanes and automobiles under ever tightening schedules, they find that traditional development processes are not sufficient to meet quality and schedule targets. Model-Based Design for system development lowers costs by identifying defects early in the development process and reducing the total number of latent defects. By helping companies deliver higher-quality industrial machinery, robotics, wireless systems, and other complex systems at lower cost and in less time, Model-Based Design provides a competitive advantage.

Traditional Development Process vs. Model-Based Design

In a traditional development process, tasks at each stage are performed sequentially in different tool environments, with many manual steps that introduce pain points (Figure 1). In each stage of development, from defining system requirements to the operations of a system, inefficiencies are introduced when a Model-Based Design approach is not adopted.

System requirements are captured textually, using tools such as Microsoft® Word® or IBM® Engineering Requirements Management Doors®, while system architectures are specified in drawing tools, making them difficult to analyze, interpret, and manage as changes are made. Subsystem designs are created using domain-specific tools, which precludes system-level testing until after implementation in software or hardware. The designs are then manually translated into code, which is a time-consuming and defect-prone process. At each phase, some defects are introduced, leaving the test phase to be the catch-all for all the defects that have accumulated throughout the previous phases. Lack of a common tool environment, multiple manual steps, and late-stage defect discovery all drive up development time and cost. Operational data and the resulting insights are not effectively utilized either, forgoing valuable improvements to system efficiency and up-time while the system is in operation.



System Requirements
System Functionality and Architecture
Subsystem Design
Subsystem Implementation
System Integration and Test
Operations and Sustainment
REQUIREMENT DOCS
Difficult to analyze
Difficult to manage as they change
PAPER SPECS
Easy to misinterpret
Difficult to integrate with design
Slow to revise when requirements change
PHYSICAL PROTOTYPES
Incomplete behavior
Expensive and time-consuming to build
Prevents rapid iteration
No system-level testing
MANUAL CODING
Time consuming
Introduces defects and variance
Difficult to reuse
TRADITIONAL TESTING
Design and integration issues found late
Traceability
SYSTEMS IN OPERATION
Improving uptime
Hard to feed insights back to developers
Difficult to get value from operational data

Figure 1. The traditional software development process. This approach requires many unnecessary manual steps that can introduce defects.

Model-Based Design (Figure 2) makes systematic use of models throughout the development process. It starts with the same set of system requirements as a traditional process. Rather than serving as a basis for textual specifications, however, the requirements are used to develop a system architecture that is an executable specification in the form of behavior and architecture models. Engineers use these architecture models to clarify requirements and specifications. The models are then used as a basis to develop a detailed subsystem design.

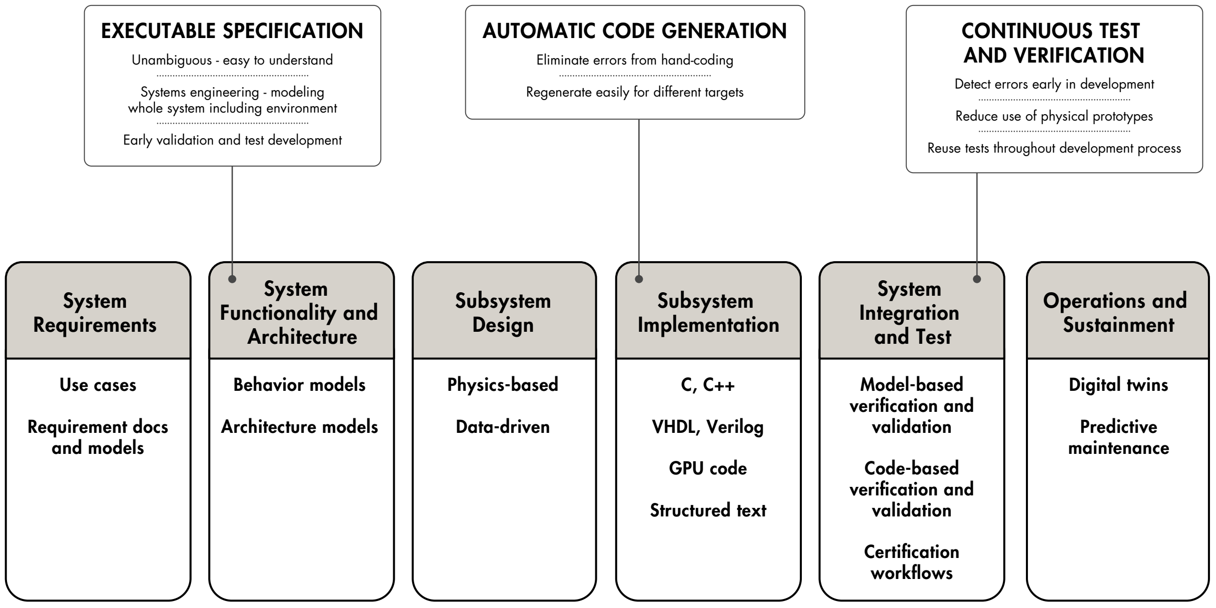

EXECUTABLE SPECIFICATION
Unambiguous - easy to understand
Systems engineering - modeling whole system including environment
Early validation and test development
AUTOMATIC CODE GENERATION
Eliminate errors from hand-coding
Regenerate easily for different targets
CONTINUOUS TEST AND VERIFICATION
Detect errors early in development
Reduce use of physical prototypes
Reuse tests throughout development process
System Requirements
Use cases
Requirement docs and models
System Functionality and Architecture
Behavior models
Architecture models
Subsystem Design
Physics-based
Data-driven
Subsystem Implementation
C, C++
VHDL, Verilog
GPU code
Structured text
System Integration and Test
Model-based verification and validation
Code-based verification and validation
Certification workflows
Operations and Sustainment
Digital twins
Predictive maintenance

Figure 2. Software development process using Model-Based Design. This approach uses a system-level model as an executable specification throughout development and supports system- and component-level design and simulation, automatic code generation, and continuous test and verification.

Using MATLAB® and Simulink® for Model-Based Design, engineers can simulate the design at the system level, uncovering interface defects before implementation. Once the design is finalized, the engineers automatically generate production-quality code and test cases from the models, eliminating errors caused by hand-writing code. This workflow enables engineers to stay in the same environment from system requirements definition to system testing, minimizing the amount of manual work. In addition, testing can begin at the requirement phase, when engineers simulate their executable specifications in models to verify that the requirements are met. As a result, defects are caught and removed earlier, lowering the total cost of development.

Savings from a Systems Engineering Perspective

The systematic use of models in model-based systems engineering can produce up to 55% overall savings after two years, according to research conducted by Jerry Krasner for Embedded Market Forecasters [1]. The use of models reduces dependency on text requirements that are often over-specified due to past miscommunications. With Model-Based Design, models serve as a common language throughout the development process. This approach reduces ambiguity in product specifications and enables the use of simulations to validate requirements. The use of models to simulate and refine product specifications before building the product is a key reason for savings.

Model-Based Design also enables the exploration of multiple solutions when developing system requirements and architectures, enabling faster design iterations and consensus building early in the design process. With a model-based approach, a single tool chain is used from the systems engineering phase through the development and operation phases of a project, enabling system architectures to be connected to system designs seamlessly and efficiently.

Savings from a Development Perspective

Organizations that adopt Model-Based Design realize savings ranging from 20 to 60%, when compared with traditional methods [2, 3]. The bulk of these savings come from better requirements analysis combined with early and continuous testing and verification. As requirements and designs are simulated using models, defects are uncovered much earlier in the development process, when they are orders of magnitude less costly to fix (Figure 3).

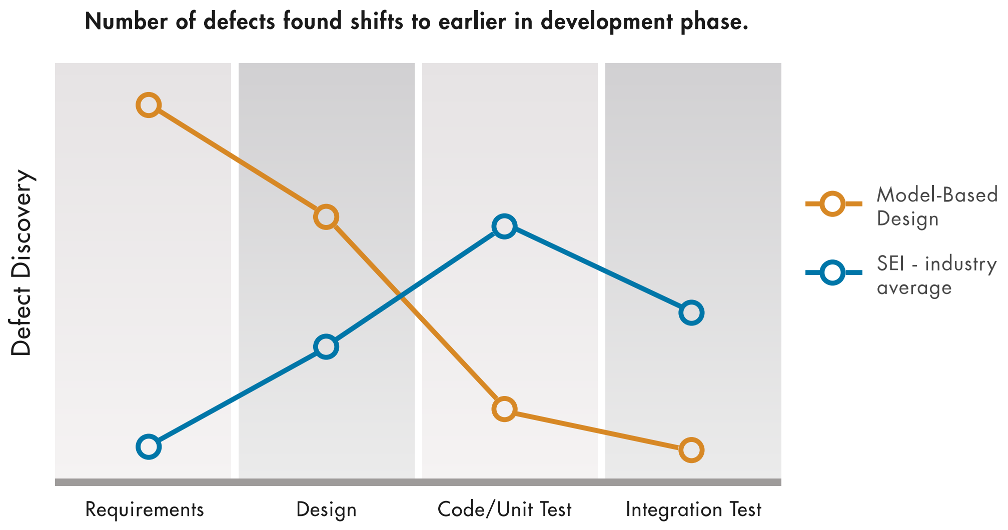


Figure 3. Defects found by development phase. Model-Based Design shifts defect discovery earlier in the process .

Quantifying Savings Using the Model-Based Design ROI Framework

The Model-Based Design ROI framework is designed to estimate the ROI from adopting Model-Based Design on specific projects. Based on project size, team size, and other factors, the framework calculates a cost for traditional development using the basic Constructive Cost Model (COCOMO) model, and then subtracts the savings from Model-Based Design to obtain the Model-Based Design cost of development. (The basic COCOMO model was chosen for the framework because it is a general parametric cost estimation tool widely used in the aerospace and defense industry, in which procurement cost accountability demands rigorous models for software cost estimation.)

ROI is then calculated by accounting for the savings that can result from using Model-Based Design for development. This calculation uses metrics from the Software Engineering Institute (SEI), Institute of Electrical and Electronics Engineers (IEEE), and industry studies [1, 4, 5, 6]. Because project scope, existing processes, and team expertise using Model-Based Design vary across industries and companies, the Model-Based Design ROI framework can be customized for specific projects and teams.

Consider a baseline case of a software project with 50,000 lines of code. Using the basic COCOMO model, the cost of development using traditional methods would be approximately $6 million (USD). To calculate the savings of Model-Based Design over traditional methods, each development phase—requirements, design, implementation, and test—is analyzed based on industry metrics. The savings are then summarized and subtracted from the traditional cost of development. In this case, the Model-Based Design cost is $3 million (USD), a 50% savings compared with the traditional method.

To arrive at the 50% savings, the framework examines inefficiencies in the traditional development process that Model-Based Design eliminates, and calculates the savings based on empirical data from customers, customer interviews, industry metrics, and averages. Savings for each development phase are calculated separately, so that the framework is adaptable for step-by-step adoption of Model-Based Design.

Savings During Requirements Development

To give a sense of how the framework works, let's look at an inefficiency in the requirements phase. Using models to uncover vague, inconsistent, or un-testable requirements enables engineers to uncover a higher percentage of defects. The baseline case assumes 15% of requirements contain defects or need reworking, which is an estimate obtained from industry interviews. Uncovering these defects at the requirements phase means avoiding costly rework later in the development phase. Part of the requirement savings is arrived at by multiplying the number of defective requirements by the average length of time to resolve defective requirements that are detected after the requirements phase. In the baseline case, the average number of hours per requirement defect is 4.5 hours [6]. By this calculation, Model-Based Design saves 3,375 engineering hours. Table 1 demonstrates a section of the framework that deals with the requirement analysis pain point.

Requirements Development and Traceability	
Percentage of requirements needing rework	15%
Number of incorrect requirements	750
Hours to rework each requirement	4.5
Hours saved to fix incorrect requirements post–requirement phase	**3,375**

Table 1. ROI framework calculation of the number of engineering hours saved by fixing incorrect requirements early.

Savings During Testing

Inefficiencies in the testing phase are also captured in this framework and result in the bulk of the savings. With Model Based-Design, tests can be generated automatically to verify and validate models, and obtain full test coverage. This approach saves valuable time that would be otherwise spent addressing missing test coverage late in the development process. These test cases can be used through several stages of testing, from desktop simulation to hardware-in-the-loop testing. Reports that summarize test results and comply with industry standards can also be automatically generated. Assuming a project with 5,000 requirements, using Model-Based Design can result in up to 12,000 hours being saved due to comprehensive test case generation during the modeling phase, and the ability to reuse test cases in multiple testing environments. Figure 5 demonstrates the section of the framework that deals with testing pain points.

Testing and Reporting	
Hours spent per requirement to address missing test coverage	2
Automatic test generation savings factor	70%
Hours saved addressing missing coverage	**7,000**
Hours spent validating/debugging tests in lab per requirement	2
Test reuse factor (from desktop testing to lab)	50%
Hours saved in lab testing (test reuse)	**5,000**

Table 2. ROI Framework calculation of the number of engineering hours saved in the testing and reporting workflow by using Model-Based Design.

Overall Development Savings

The framework contains several additional sections that deal with different inefficiencies at each stage of the development process. Summarizing the savings from the entire development process, almost half of the savings in this example came from the test phase (Figure 4). This is due to more thorough requirements analysis early in the development process, which results in fewer defects being carried through to the testing phase, when requirements and tests become costly to revise and improve. Put simply, better requirements lead to better designs. Early and continuous testing results in more defects being identified and addressed in the same phase that they are introduced, which leaves fewer latent defects in the software and lowers overall development costs.

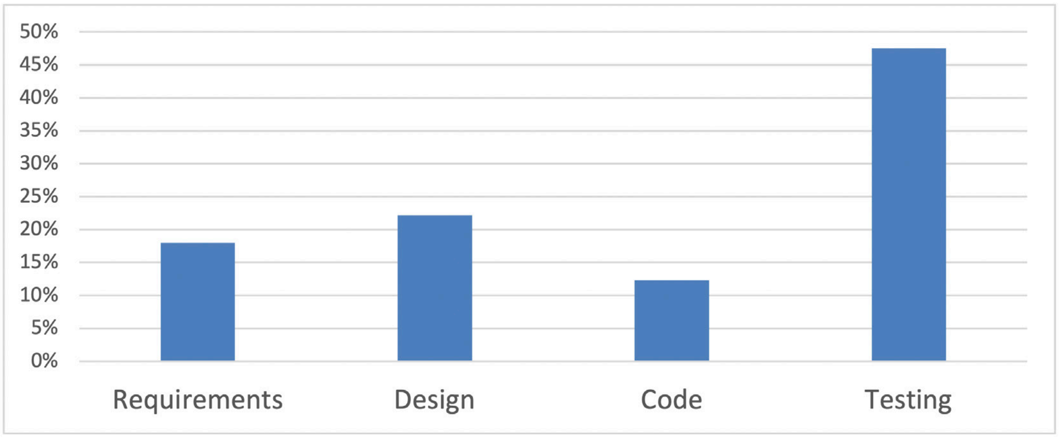


Figure 4. Percentage of total savings by development phase.
Savings in the testing phase accounted for most of the total savings.

When companies who are adopting Model-Based Design collaborate with MathWorks, the ROI framework helps guide the adoption process, enabling them to identify areas that will benefit immediately and significantly from transitioning to Model-Based Design.

Savings Beyond the Development Lifecycle

While the ROI framework quantifies the cost savings through the development lifecycle, these savings are not the only financial benefit of adopting Model-Based Design. The faster development pace accelerates the time to market, giving customers who use Model-Based Design a competitive edge. In addition, the reduced overhead and miscommunication enabled by Model-Based Design frees up resources for core business activities and innovation.

Summary

For most companies, investing in new technology and processes is a risky endeavor. The return on investment calculation described in this paper aims to provide analytical evidence to support an investment in Model-Based Design. In addition to justifying the investment, the ROI framework enables teams to identify areas in which Model-Based Design provides the most savings, as well as areas in which further investigation could lead to substantial additional cost reductions.

To get a customized ROI calculation for your team or organization, *contact Sales*.

Learn More

- *Why Adopt Model-Based Design?*
- *How Engineering Teams Adopt Model-Based Design*
- *Simulink for Simulation and Model-Based Design*
- *Model-Based Design Process Assessment and Maturity Framework*

References

1. Krasner, Jerry. How Product Development Organizations Can Achieve Long-Term Cost Savings Using Model-Based Systems Engineering (MBSE). October 2015. ***https://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:how_product_development_organizations_can_achieve_long-term_savings_1_.pdf***
2. MathWorks, OHB Develops Satellite Guidance, Navigation, and Control Software for Autonomous Formation Flying. ***https://www.mathworks.com/company/user_stories/ohb-develops-satellite-guidance-navigation-and-control-software-for-autonomous-formation-flying.html***
3. MathWorks, BAE Systems Achieves 80% Reduction in Software-Defined Radio Development Time. ***https://www.mathworks.com/company/user_stories/bae-systems-achieves-80-reduction-in-software-defined-radio-development-time.html***
4. Over, James W., *Managing Software Quality with the Team Software Process*, Software Engineering Institute. ***http://c-spin.net/2010/c-spin201004Managing%20Software%20Quality%20with%20the%20Team%20Software%20Process.pdf***
5. Vallespir, Diego, and William Nichols. *Analysis of Code (and Design) Defect Injection and Removal in PSP*. CARNEGIE-MELLON UNIV PITTSBURGH PA, 2012. ***https://resources.sei.cmu.edu/asset_files/Presentation/2012_017_001_298088.pdf***
6. Tom King, Joe Marasco, *What Is the Cost of a Requirement Error?* StickyMinds. ***http://www.stickyminds.com/sitewide.asp?ObjectId=12529&Function=edetail&ObjectType=ARTCOL***