The ITEA Journal of Test and Evaluation 2020; 41: 258-266

~ THE ITEA JOURNAL PEER REVIEWED ARTICLE ~

Test Planning, Documentation, and Impact Analysis with SysML

Joshua Walker
Georgia Tech Research Institute, Atlanta, GA

John M. Borky, Ph.D.
Colorado State University, Fort Collins, CO

With the growing complexity of software-based systems, the complexity of the test procedures used to verify those systems also increases. Without a method of organization for these test artifacts, the collection of test documentation for a system can become a maintenance challenge. This can result in confusing test procedures, duplication of tests, and a general lack of knowledge regarding the purpose of certain test cases. This lack of understanding of the test suite can impact the overall development of the product by potentially causing delays in the process and can also result in the release of an insufficiently tested product. This article provides a distinct method of using the Systems Modeling Language (SysML) to enhance the documentation of test procedures for a project, including the driving objectives for its creation, an overview of the process, potential results from using it, and a discussion of lessons learned from its implementation.

Introduction


Joshua Walker


John M. Borky, Ph.D.

The complexity of software-based systems has been growing at a near exponential rate for the last 40 years.[1] This exponential growth can have dramatic impacts on the documentation maintenance costs of these systems, especially with regard to test artifacts.[2] Ever-increasing test execution times, increases in test procedure size and scope, and rushed timelines because of this expanded system complexity can shift necessary time away from performing test documentation duties like test planning, requirements tracing, and general upkeep. Additionally, inadequate time provided for test documentation maintenance can lead to confusing test procedures, duplication of tests, and a general lack of knowledge regarding the purpose of certain test cases.[3] This lack of understanding of the test suite can impact the overall development of the product by potentially causing delays in the process and could also result in the release of an insufficiently tested product.[4]

A potential solution for this problem is to standardize the approach to test documentation. This can be performed in a variety of methods; however, this article describes a model-based methodology using concepts from Model-Based Systems Engineering (MBSE). The incorporation of MBSE concepts and techniques into the test process provides the potential for a greater understanding of the system, a method of mapping functionality and components of the system, and a strategy for targeted regression testing to increase its efficiency.[5] A model created for test purposes has the potential to be utilized effectively in ways such as:

- Identifying efficient test points based on stressing operational conditions[6]
- Defining the context for test cases in terms of items such as dependencies, interfaces, external interactions, applicable policies and standards[7]

- Analyzing and understanding ambiguous test results
- Providing traceability between system components
- Managing and communicating the scope of regression testing

This article provides a distinct method of using the Systems Modeling Language (SysML) to enhance the documentation of test procedures for a project, including the driving objectives for its creation, an overview of the process, potential results from using it, and a discussion of lessons learned from its implementation.

Project Overview

The targeted project is an airborne Electronic Warfare Management System (EWMS), which interfaces with onboard and offboard aircraft systems to receive and process the threat environment, providing the pilot/crew with a response solution for the protection of the aircraft. It is fielded on multiple aircraft platforms and has been operational for more than 20 years. The methodology described in this article was designed for use with this specific project. However, the developed concepts are applicable to multiple types of projects, especially those with a high level of complexity and scope. This project serves well as a representation of a large, complex aerospace project, as it interfaces with multiple external systems and over a variety of transmission protocols. The project is developed with an Agile process, but a different process does not preclude the use of this methodology.

The system is tested using a keyword-based test automation framework centered around the open source Python module Robot Framework. This allows test cases to consist of natural language phrases that are executable, automating what would otherwise be manual test actions and expected results. While this method does not necessitate the use of an automated testing solution, the examples provided do include their existence.

Methodology

An overview of the developed test documentation methodology and process is described below. No Magic Cameo Systems Modeler™ is the SysML tool of choice for the examples provided in this article, but the methodology can be accomplished with any modeling tool.

Objectives

The main goal of incorporating MBSE concepts into the test process was to promote the organization and structure of test artifacts in a standard, formalized way. Many aspects of the system's test process were in a less than ideal state before the creation of this process, due to some of the factors mentioned previously. Utilizing a model-based approach provided a paradigm that was intended to promote the following objectives:

- Improve the structure and clarity of test case documentation
- Improve rapid comprehension of test procedure contents
- Establish a formal method of requirements traceability
- Provide a method of assessing impact of upstream changes
- Increase understanding of inter-dependencies between test cases
- Facilitate easier review of test planning documentation
- Provide a method to quickly identify test cases for targeted regression testing

The new process that was developed is not a solution for model-based testing. There are many aspects of the targeted system that would make true model-based testing a hard reality to achieve. Instead, this process adopts concepts of MBSE and applies them to a method of test documentation, specifically for the system's test plans and procedures. It is intended to be a lightweight model-based solution to formalize the documentation of those test artifacts to make future analysis easier and to satisfy the goals stated above.

Model Overview

In this approach, the test documentation for a system is a standalone model. All requirements, inputs, messages, data, and any other components necessary to the testing of the system are referenced as model elements. However, to be resilient to changes to any component related to the test cases in the model, the detail on modeled elements external to the test engineer's control is minimized. This is usually accomplished by referencing only the name of the element, but any stable details could be included in the model. If a larger system model already existed or was created later, the test model could easily be linked to the larger model by sharing objects that exist in both.

Model Organization

The model is organized into packages, as can be seen in the high-level view of the Containment Tree in Figure 1.

The Containment Tree is the main point of interaction with the model. While diagrams in a SysML model

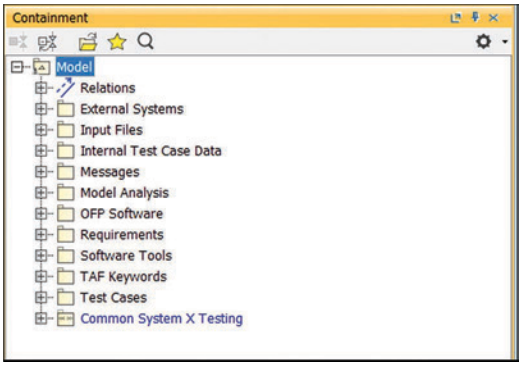

Figure 1: Containment Tree

provide a visual overview of the content of a model, the truth of element characteristics and relationships are accessed there. Each set of elements is contained within its own package. Most packages are used to contain groups of elements that will be used within test cases. The Test Cases package is the primary location where the artifacts of this process reside.

Activity Diagram

The Activity Diagram is used to create a Test Plan. An example Test Plan can be seen in Figure 2.

The intent of this diagram is to describe the flow of the test case. When reading the Actions and Expected

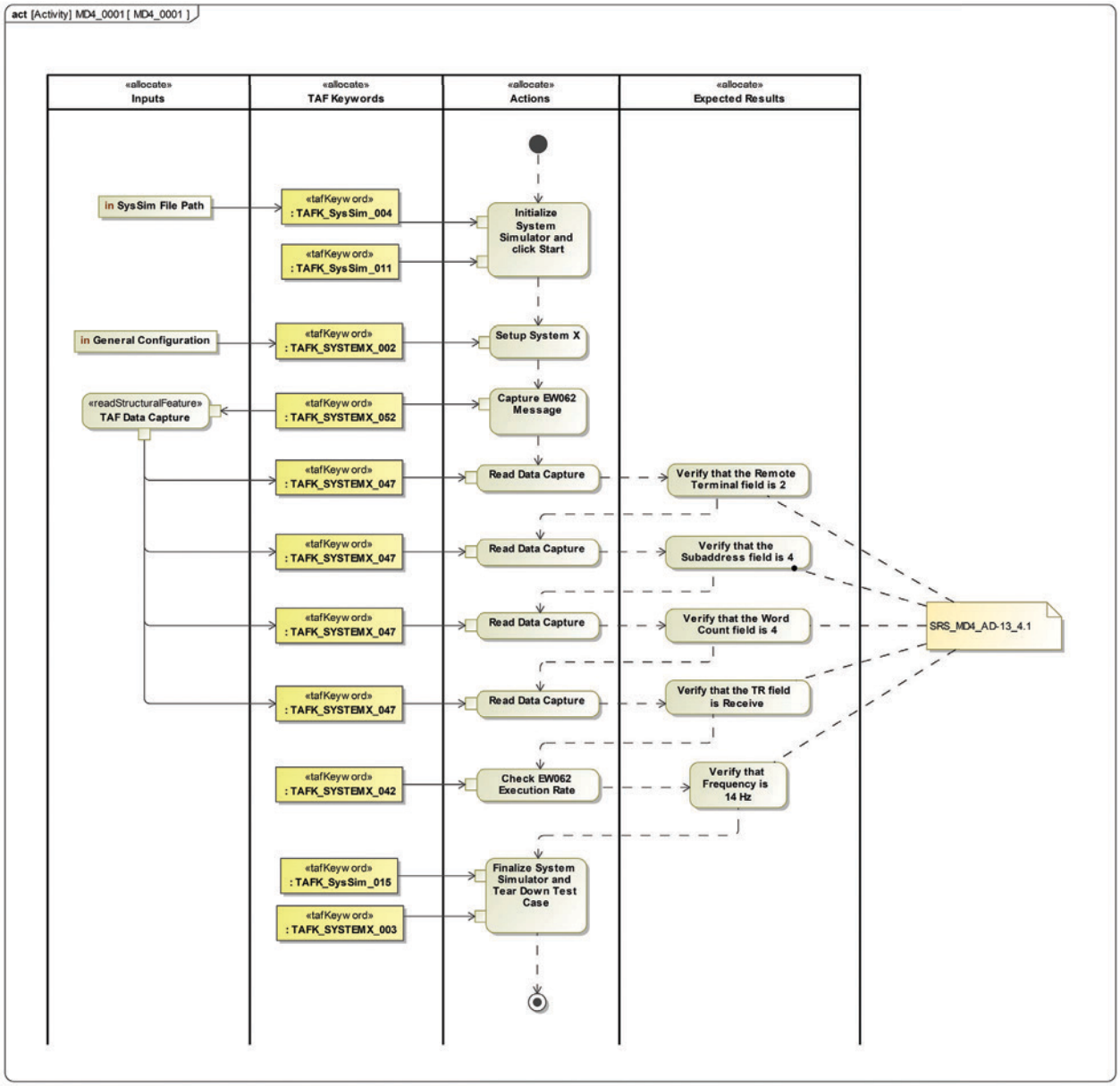

Figure 2: Test Case Activity Diagram

Results swimlanes, this diagram should look similar to a list of pseudo-steps that would normally be created when planning a test case.

Components

The four swimlanes on these diagrams are defined as Part Properties of the Test Case block that will be used in the Block Definition Diagram (BDD). The purpose of these swimlanes is to group the elements of the test case according to their function. Actions tie to Action Keywords, while Expected Results tie to Verify keywords. The Test Automation Framework (TAF) Keyword swimlane is used for identifying the test automation framework keywords that perform the actions, while the Inputs swimlane is used to hold inputs to those keywords (the exact usages of the variables that are contained within keywords).

Items in the Actions swimlane are Action elements. Actions are used to describe the different steps that need to be performed by the test engineer or test automation software in order to perform the test. These actions would typically appear in the left column (Actions) on a typical test case.

Items in the Expected Results swimlane are Action elements. Actions in this swimlane are used to describe the different verification steps that must be performed. These actions would typically appear in the right column (Expected Results) on a typical test case.

Items in the TAF Keywords swimlane are Object Node elements. These items are added to the diagram as a generic element. Then the TAF Keyword block representing the indicated TAF Keyword is dragged onto the Object Node. This creates an instance of that TAF Keyword used to perform the action to which it is linked.

Items in the Inputs swimlane are either Activity Parameter Node elements or Flow Properties of a Test Case block. Activity Parameter Nodes are used to create static inputs to TAF Keywords (e.g., fields, values, file paths). The goal is to create an Activity Parameter Node that lists the actual input planned to be used as a variable to a TAF Keyword. Flow Properties are used when there is data that is created within the test case, as opposed to external inputs to the test case. The case in the example above is the capture of data initiated by a TAF Keyword that is later analyzed further within the test case.

Notes, like the one on the right of the diagram, are being used to provide additional information as well as to indicate where requirements are being satisfied. This helps during review, so the rest of the team can quickly see how and where the test engineer intended to satisfy the requirements. Notes should be used freely to provide additional information wherever necessary.

Relationships

There are two different types of relationships used in this diagram. Control Flow relationships are used to show how the Action elements flow (i.e., how Actions are accomplished and generate Expected Results). Object Flow relationships are used to show how the Object Node elements move to provide input to the Action elements.

Block Definition Diagram

The Block Definition Diagram (BDD) is used to document the important items connected to a specific test case. An example BDD of a test case can be seen in Figure 3.

Components

Each component is represented as an individual element within the model. Custom Stereotypes are applied along with a custom color scheme to easily determine the difference between elements. The test case is represented by the gold-brown block near the middle top of the diagram. Other documented components in the example above include:

- Requirements (Pink)
  - Specific "shall statements" that are satisfied by the test case
- System Configurations (Green)
  - Specific list of software packages loaded on the system used by the test automation framework to validate and load the correct set of software before beginning execution of the test case
- Keywords (Yellow)
  - Action/Verify statements provided by the test automation framework to perform automated testing
- Recorded Data Captures (Purple)
  - Data recorded during the execution of a test case to be processed by the test automation framework to determine pass/fail when necessary in specific instances
- System Messages (Blue)
  - External/Internal message traffic used to verify system behaviors
- Input Files (Orange)
  - Software inputs for external systems that drive or configure those systems
- Protocol Files (Light Blue)
  - Files used by the system monitor to decode message traffic into human-readable form
- External Hardware/Software (Grey)
  - External hardware or software systems used to stimulate or monitor the system

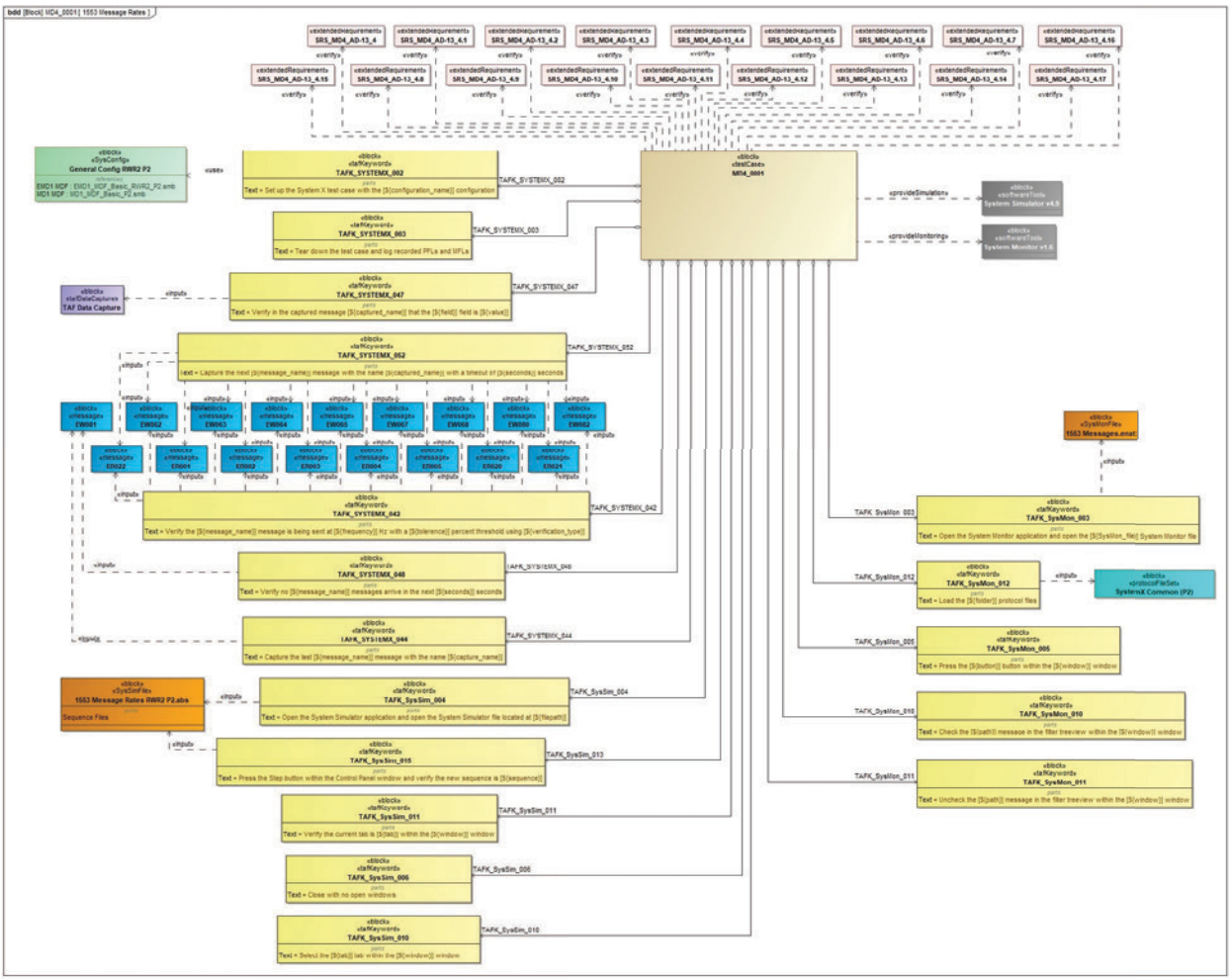

Figure 3: Block Definition Diagram

This is not necessarily a comprehensive list, but it is what was deemed helpful to associate with this test case. Generally, any items that are important to trace to a test case, potentially for later impact analysis, should be modeled and associated with the test case. The elements that need to be traced for a specific project are determined by the test environment of that particular system. Blocks should be created and organized according to a scheme that makes it easy to find and reuse elements, such as the package structure presented earlier.

Relationships

There are three different relationship types used in the BDD. The dashed line from the Test Case block to Requirements is a Dependency relationship with the Verify stereotype. It means that the test case verifies the requirements linked to it. The dashed line to multiple different types of elements is a Dependency relationship with the Usage stereotype. In most, if not all, cases, the Usage stereotype is not displayed on the connector, as a custom stereotype has been applied. The solid line with an open diamond is a Directed Aggregation. This line means that, structurally, the test case is made up of those keywords, but the keywords exist independently, outside of that specific test case.

Process Overview

The following provides a description of how a test engineer creates and interacts with the test model. This process assumes that an Agile methodology (e.g., Scrum) is being used and a keyword-based test automation framework has been implemented.

Test Preplanning

The first step before a test engineer is able to begin working with the model is the generation of requirements for a specific task. These requirements are worked first by the team to provide the foundation for

#	△ Name	Text	Verified By	Verify Method
186	SRS_MD4_AD-13_4		MD4_0001	Test
187	SRS_MD4_AD-13_4.1		MD4_0001	Test
188	SRS_MD4_AD-13_4.2		MD4_0001	Test
189	SRS_MD4_AD-13_4.3		MD4_0001	Test
190	SRS_MD4_AD-13_4.4		MD4_0001	Test
191	SRS_MD4_AD-13_4.5		MD4_0001	Test
192	SRS_MD4_AD-13_4.6		MD4_0001	Test
193	SRS_MD4_AD-13_4.7		MD4_0001	Test
194	SRS_MD4_AD-13_4.8		MD4_0001	Test
195	SRS_MD4_AD-13_4.9		MD4_0001	Test
196	SRS_MD4_AD-13_4.10		MD4_0001	Test
197	SRS_MD4_AD-13_4.11		MD4_0001	Test
198	SRS_MD4_AD-13_4.12		MD4_0001	Test
199	SRS_MD4_AD-13_4.13		MD4_0001	Test
200	SRS_MD4_AD-13_4.14		MD4_0001	Test
201	SRS_MD4_AD-13_4.15		MD4_0001	Test
202	SRS_MD4_AD-13_4.16		MD4_0001	Test
203	SRS_MD4_AD-13_4.17		MD4_0001	Test

Figure 4: Requirements Table

the associated subtasks that must be performed for its completion. Once the requirements are drafted and unique numbers are assigned, even if they are not in their final form, the test engineer adds the requirements to the model. Figure 4 shows a snapshot of a requirements table for the system.

The requirement text was not entered above for two reasons. The first is that the text of the requirement might continually change slightly during a Sprint due to constant refinement. However, the requirement usually has the same basic idea, which is enough to complete a Test Plan. The second reason is that the source for requirements for the system is not actually within the model, but rather in a separate document configuration managed in a different system. In this case, it is not necessary to enter the requirement text, as that would cause the problem of potentially having two different systems not synchronized with each other. It is better in this case to only use the ID of the requirements as a manual reference between the two systems. If the model is the source of requirements, then the actual requirement text would appear in the table.

Test Planning

Test Planning is a fundamental step in the creation of proper testing that both satisfies requirements and fully exercises the system under test. Too often, Test Planning gets skipped or rushed so that test procedures can be pushed through to completion before the test scenarios themselves are fully understood. This can lead to situations such as mediocre test procedures or even important test cases being completely overlooked. Even when it is performed, Test Planning is not always completed in a standard, manageable way. The artifacts of planning, unless the Test Plan is a customer deliverable, are typically informal and only provide short-term utility that dissolves after the accompanying test procedures have been created.

Migrating Test Planning to a SysML model resolves the issue of standardization and, with more formal expectations on the test engineers, provides an artifact that can be referenced in the future as a summary of the developed test cases.

Under this process, Test Planning is documented in the model using Activity Diagrams. After requirements are added to the model under the Requirements package, the test engineer creates the Test Case blocks or identifies existing Test Case blocks that will satisfy the requirements for the Task. Under each new block, an Activity Diagram is created that represents the Test Plan for that specific case. The test engineer creates high-level actions that represent the different steps that will be taken and the expected results that will be verified during the test case.

During Test Planning, the test engineer is expected to identify test dependencies, especially those that have not yet been developed. This includes, but is not limited to, test automation functionality, support tool functionality, and system instrumentation needs. Since the process is being performed in two-week Sprints, it is expected that the teams responsible for developing these additional items are highly responsive to needs

and will provide necessary functionality before the end of the Sprint.

In a test automation keyword-driven test case, the content of the test case is a sequence of keywords with specific input parameters. High-level Actions defined during Test Planning should be able to be tied to the keywords that drive them. After Actions and Expected Results are defined, the test engineer identifies the keywords that will be used during the test. If the keywords do not exist, the test engineer proposes drafted versions and creates a ticket for the automation team to implement. The test engineer also identifies the parameters that will be input to the keyword and adds them to the Activity Diagram.

The final step is to make sure that requirements satisfied by specific Expected Results are tagged appropriately within the Activity Diagram. This is to facilitate easier review of the plan. These tags will be carried over to the written test procedures as in-line requirement citations.

After the test plan is drafted, the test engineer provides the plan to the rest of the team for a review. The intent of the review is to make sure that everyone agrees on the strategy for the test case, that it has been planned correctly, and that the requirements will be fully satisfied by the planned test scenarios. The test plan is revised as necessary with feedback from the reviews. After the test plan is approved, the test engineer transitions to the Test Development phase.

Test Development and Documentation

In the Test Development phase, test engineers translate the approved test plans into test cases with associated test procedures. Test Documentation is not necessarily its own phase, but it is an important part of the process. Ideally, documentation is performed throughout the entire process as each test case is forming. In practice, it becomes a step that is performed towards the end of a Sprint as each developed test case is solidifying. This allows the test case to be more fluid as it is being developed, without having to continually keep the model synchronized with the changing procedure.

Documentation of each test case is performed through the creation of a BDD, as described previously. The purpose of this diagram is to document all items related to a single test case, which could be performed either before or after the test case has been written. Each item is stored as a separate entity, categorized into the proper place in the model, and used as necessary to complete the BDD. Any items not available in the model already are created during this process.

Test Execution and Reporting

During Test Execution for a specific Sprint, the test procedures developed during the Sprint are run against an available build to verify that the requirements have been satisfied. As functionality is developed and published in beta form to the team, the test engineers attempt to run their developed test cases and tweak as needed. Any changes to the test procedure that impact the model are addressed by updating the model when necessary. The results of the Test Execution phase are configuration managed and later summarized for the specific tickets within the Sprint when Test Reporting is accomplished.

Using the Model

While the existence of a test model can help test engineers more easily understand the details of test artifacts, its main purpose is to be used for later analysis. This may take a variety of forms, but the overall use is to identify how an upstream change affects the testing of the system. For example, the modification of a message (e.g., adding data bits for additional information, changing the format to align with a new system) is a common occurrence for systems that integrate with multiple external components. When an aspect of the system is changed, it is important to fully regress testing for impacted components. Without the test model, or a similar level of documentation of test cases in some other manner, identifying how a changed message impacts the system is difficult. Significant time could be wasted trying to find each test case that uses a specific message if not documented properly.

Making use of aspects of the modeling tool allows test engineers to easily identify impacted components. Some of those useful features are described below. The examples are illustrated using No Magic Cameo Systems Modeler™; however, these types of features are common across most modeling tools.

Model Navigation

The most basic type of analysis is to open a diagram, select an element, and use the context menu to navigate to other usages of the element within the entire model. In Cameo Systems Modeler™, this is performed by right clicking and selecting "Go To" and "Usage in Diagrams" in the context menu structure. This action provides a list of all diagrams that contain a specific element. In this way, a test engineer can quickly identify which test cases use a specific message when referring back to the previous example. In the same way, a test engineer can use the tool to trace an element from a diagram to its

actual location in the Containment Tree. This would allow the test engineer to view all of the various relationships from that element to other elements.

Relation Map

A Relation Map shows the connections between elements based on a given starting point. Filters and queries can be applied to only display elements with specific characteristics in order to pare down the data being displayed.

Lookup Tables

Most modeling tools allow the user to create tables to display data in traditional rows and columns to facilitate easier viewing of the data. A specific example of this type of table is used to enter requirements into the model, seen previously in Figure 4.

Dependency Matrices

Dependency Matrices allow a user to display the relationships between two different element sets in table form. Most tools provide a wide variety of filters to help customize this specific type of display. A specific type of Dependency Matrix is the Requirements Traceability Matrix. If the Requirements Table is created appropriately, this table is created automatically by most tools. The purpose of this table is to show how requirements are satisfied within the model.

Data Export

In some cases, it may be helpful to extract the data out of the model and translate it to a different form for easier analysis (e.g., Microsoft Excel). Most modeling tools have mechanisms for this, either through dedicated export capabilities, or through the creation of plug-ins that interface with an available Application Programming Interface (API).

Results

The model created by this process has been used successfully to facilitate countless reviews of test cases and to provide an easy method of test impact analysis for multiple functionality modifications. While these successes do not necessarily result in tangible quantitative results, a number of qualitative improvements can be extrapolated from its use. These are discussed further below.

Test Documentation Quality

The maintenance of documentation on an engineering project is a common problem.[8] Standardizing the documentation and test planning approach facilitates easier creation and review of test artifacts. Test engineers know what is expected of them and how to perform their tasks, leading to a higher quality output.

Training Time Reduction

Because test cases are all documented in the same manner with this process, training new test engineers becomes faster. Once the methodology is learned, these new engineers can use the model to see how the actions of a test case flow to exercise specific functionality and exactly which functionality components are linked to the execution of that test case.

Release Cycle Time Reduction

It is not always possible to perform a full regression test for a release, potentially due to items such as shortened timelines, customer deadlines, or lack of funding.[9] Without a method of understanding how key system components relate to each other, determining which functionality should be exercised in that limited time period is not always an easy process. Because this process provides a systematic way of determining how individual components are linked to test cases, not just through requirements tracing, it provides a method for limiting the regression time available to the most important test cases.

Selected Lessons Learned

Several lessons were learned throughout the creation of this methodology. A selection of key topics is described below.

Reduce Unnecessary Details

Too much detail can profoundly impact prolonged maintainability of documentation. Including small details that are unnecessary to the task at hand can lead to endless rework. For artifacts such as test plans, which are intended to be malleable throughout the creation of the test procedures, it is necessary to find a balance between understandability and thoroughness.

Automate the Process

Generally, automation helps improve the efficiency and speed of processes.[10] This model-based methodology is no different. Most modeling tools have rich APIs or other automation interfaces that can help perform simple, potentially monotonous tasks faster. Automating key aspects, such as generating a test case skeleton based on a test plan or programmatically building a BDD from a created test procedure, can help reduce time spent on documentation while still maintaining the objectives of this process.

Enforce a Standard

Creating and documenting a standard approach to any process has the potential to improve efficiency by providing organization as well as a pattern to follow for future additions. If this methodology is adopted on a project, the details of its expected use should be documented and advertised to the entire team. This documentation of the process includes items such as how to define elements, a plan for Stereotyping, relationships that will be used, and rules for global numbering of items between systems or projects. Having a well-defined process can help clear confusion and disorganization that will harm productivity and maintenance.

Conclusion

A standard method of performing test planning activities, documentation of test cases, and impact analysis is an important aspect of building a highly understandable and sustainable test suite. It is necessary to continue providing a high-quality and adequately tested product, especially for highly complex systems with long lifespans. The process described in this article is one method of organizing the potential chaos and would work well for a project that is already incorporating MBSE concepts into their development process. ❑

JOSHUA WALKER is a Senior Research Engineer at the Georgia Tech Research Institute (GTRI) in Atlanta, Georgia and a Ph.D. candidate in the Systems Engineering doctorate program at Colorado State University. His research interests center around the application of Systems Engineering concepts to the Test Engineering domain, concentrating on the introduction of new technologies and test process improvement. Joshua holds a B.S. in Computer Engineering from the Georgia Institute of Technology, an M.S. in Systems Engineering from Southern Polytechnic State University, and an M.B.A. from Kennesaw State University.

JOHN M. BORKY, Ph.D. (M '67, SM '91, LSM '15) received the B.E.E. degree in electrical engineering from the Catholic University of America in 1967, The S.M. and E.E. degrees in electrical engineering from M.I.T. in 1969, and the Ph.D. degree from the University of Michigan in electrical engineering in 1977.

During a 25 year career in the United States Air Force, he served as a research engineer, program manager, flight test engineer, graduate educator, and laboratory commander. Subsequently, he was a senior engineer, technical fellow, program manager, and consultant with five aerospace companies, one of which he co-founded. He has worked in multiple areas of solid state devices, integrated modular avionics, energy management, high power microwave systems, command and control systems, and others. He was a member of the Air Force Scientific Advisory Board for seven years, three as Vice Chairman.

Dr. Borky's awards include the Air Force Legion of Merit with oak leaf cluster, the AIAA Digital Avionics Award, and the Air Force Decoration for Exceptional Civilian Service. He is an Associate Fellow of AIAA.

Endnotes

[1] M. R. Lyu. 2007. "Software Reliability Engineering: A Roadmap." 2007. *Future of Software Engineering*: 153-170.

[2] Daniel Dvorak. 2009." NASA Study on Flight Software Complexity." *AIAA Infotech@Aerospace Conference 1882*: 1-20.

[3] Abhinaya Kasoju, Kai Petersen, and Mika V. Mäntylä. 2013. "Analyzing an Automotive Testing Process with Evidence-based Software Engineering." *Information and Software Technology* 55(7): 1237-1259.

[4] Gregory Tassey. 2002. *The Economic Impacts of Inadequate Infrastructure for Software Testing*. National Institute of Standards and Technology.

[5] Shin Yoo, and Mark Harman. 2012. "Regression Testing Minimization, Selection and Prioritization: A Survey." *Software Testing, Verification and Reliability* 22 (2): 67-120.

[6] Robert France, and Bernhard Rumpe. 2007. "Model-driven Development of Complex Software: A Research Roadmap." *2007 Future of Software Engineering*: 37-54.

[7] Mark Utting, Alexander Pretschner, and Bruno Legeard. 2012. "A Taxonomy of Model-based Testing Approaches." *Software Testing, Verification and Reliability* 22 (5): 297-312.

[8] Timothy C. Lethbridge, Janice Singer, and Andrew Forward. 2003. "How Software Engineers Use Documentation: The State of the Practice." *IEEE Software* 20 (6): 35-39.

[9] Rick David Craig, and Stefan P. Jaskiel. 2002. *Systematic Software Testing*. Boston: Artech House.

[10] Bosch, Jan. 2014. *Continuous Software Engineering*. New York: Springer.