Reflection Report

PA2558 – Software evolution and maintenance

Koyyada Sai Pranav
990429-9311
saky19@student.bth.se

***Abstract*—Maintainability is an external attribute and associated in cohesion with the quality of the software. It governs the scale and progression of the software through its project span. This report captures reflection on Specmate, a requirements modeling and test generation tool over 3 Sprints each inspecting and analyzing its code structure, quality, and response to change.**

***Index Terms*—Software Maintainability**

I. Introduction

Maintenance accounts for the majority of the project cost over its span. Maintenance activities are varied and classified into corrective, adaptive, perfective, and predictive types. The architecture and design of the software should facilitate change to engineer any type of modification as part of maintenance.[1,2,3]

Maintainability is an external attribute of software defining its potential to adapt and retain changes. Maintainability governs the scale and progression of the software. The constant demand for improvements and changes in a software requires it to demonstrate zero or no resistance to modifications. A software with appalling maintainability is not expected to ship value or endure in a dynamic ecosystem. According to IEEE 25000, the maintainability of a software can be reflected from its modularity, reusability, analyzability, modifiability, and testability.[1,2,3]

This report reflects on the maintainability of Specmate, a requirements modeling and test generation tool over 3 Sprints each inspecting and analyzing its code structure, quality, and response to change. Section 2,3 and 4 address the Sprint activities of Sprint 1, 2, 3 respectively. Section 5 concludes the 3 Sprints and acknowledges their observations.

II. Sprint 1

The objective of Sprint 1 was to get accustomed with the Specmate software by understanding the technologies used and inspecting its code structure.

A. *Program Comprehension*

Program comprehension is a sequence of activities to generate a mental model of the software to address comprehension focus. These mental models establish a high and low-level representation of the software critical to implement or operate the comprehension focus on the software. The mental models are generated using static and dynamic analysis of the software.[4]

Sprint 1 introduced program comprehension on Specmate with a comprehension focus on fixing bugs and implementing changes as improvements and or features.

B. *Specmate overview*

Specmate is a requirements modeling and test generation tool supporting integration with JIRA, and other requirements and project management tools. Specmate supports requirement modeling as Cause-Effect-Graphs and Activity Diagrams. It also allows importing existing requirements to generate optimal test-suite and export them. The software is realized using OSGi specifications for generating Angular and RESTAPI builds as OSGi bundles.[5]

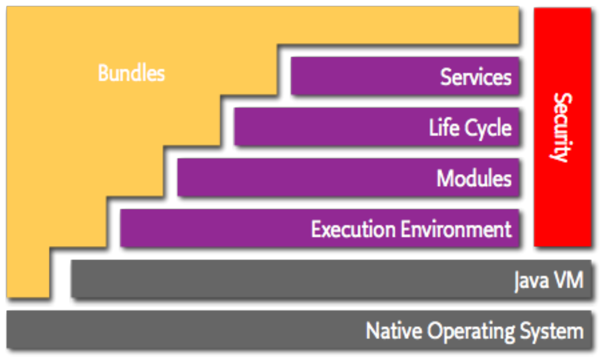


Fig 1: Layering OSGi Architecture

The OSGi is a modularity centric java framework promoting code as components. The layers in the OSGi framework are plugins supporting to build reusable blocks of standalone programs. Fig 1 models the layering OSGi Architecture. Components are individual pieces of source codes shipping specific functionality. Bundles wrap each component under an isolated environment housing all the required dependencies to support the execution of the component. Each bundle exposes services for other bundles to interact and utilize. This communication is channelized via the service layer of the OSGi framework. Services facilitate a service registry holding available services registered by each bundle to provide and listen to various services. Bundles connect to the required service of another bundle to utilize the functionality provided by it. A pictorial representation of the same is demonstrated in Fig 2.[6]

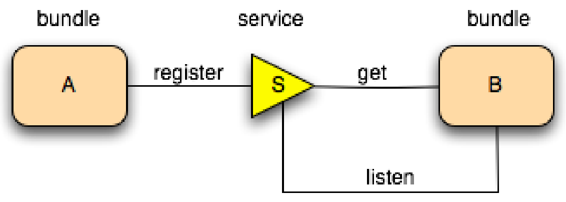


Fig 2: Design of the service layer in OSGi architecture

A specific snippet or block of code is exported using the Modules layer of the OSGi framework when requested over a service. The exported code of each bundle is optimized to occupy resources only when demanded through the Life Cycle API which manages the scope of code snippet running on the Java VM shared between all the bundles.[6]

The containerized nature of the OSGi architecture minimizes conflicts between bundles on code modification and refactoring. Therefore, the coupling between components is optimized to accommodate independent scaling and change implementation without affecting other parts of the software.[6]

Specmate takes advantage of the OSGi architecture using Angular and RESTAPI builds as bundles. Thus, from an architectural standpoint, Specmate reflects favorable characteristics for high maintainability given the low coupling and high cohesion between individual bundles offered by the OSGi architecture.[6, 7, 8, 9, 10]

C. Code structure of Specmate

Specmate is primarily decomposed into two segments of self-contained programs. The ***"web/"*** folder in the root directory of the Specmate project houses an Angular application providing user interaction interface. Individual user interaction components are classified and segregated under subdirectories within the ***"web/src/app/"*** folder. On execution, the root application folder located in ***"web/src/app/modules/specmate/"*** maps requested route with route patterns provided in ***"web/src/app/modules/specmate/routing/"*** to call other components required for initialization and rendering on index.html.

Each component delivers a specific DOM object and its function triggers on request by the root app. They are largely categorized and arranged in chained folder hierarchy ranging from common objects viz forms, icons, buttons, navigation-bar to complex tool-specific interaction utilities discussion of which is beyond the scope of this report.

The build of the angular application outputs compiled files to the ***"bundles/specmate-ui-core"*** bundle later used to communicate with other bundles over the service layer.

The ***"web/src/app/modules/data/modules/data-service/services/"*** establishes a connection with the RESTAPI endpoints to serve data and functionality to the user via the angular application. The RESTAPI is shipped as OSGi bundle in the ***"bundles/specmate-emfrest/"*** folder.

The core functionality of Specmate is shipped from 48 different bundles in the ***"bundles/"*** directory. Specific combinations of bundles work together to deliver a unique service to the software.

The ***specmate-std-env*** bundle hosts a shared environment for other bundles to reside and communicate. The primary services provided by the bundles are:

1) **Authentication and Access rights management:**
   Session creation and restricted access for information protection and security are shipped by ***specmate-administration, specmate-auth-api, specmate-auth***. These are also the primary bundles of specmate, since all the other bundles rely on the information and service provided my them given that all project requires authentication and authorization in Specmate.

2) **Connectors service:**
   Connectors are communication channels between two parties interfacing for information exchange. These services include third-party api integration for Jira and Trello. **specmate-connectors-api, specmate-connectors, specmate-file-connector, specmate-hp-connector, specmate-jira-connector, specmate-trello-connector** are different connector bundles offering these services. The connector service is critical for the system information exchange essential for every bundle. Without the connectors service no other bundle can access information from and outside the system.

3) **Database Providers and migration tracking:**
   Information storage and retrieval are realised by database driver integration to query and manage them. These include **specmate-dbprovider-api, specmate-dbprovider-h2, specmate-dbprovider-oracle**. Different bundles in Specmate use these drivers to access the data crucial for proper functioning and persistence. Migrations are information about the state of database at a given instance. Updating the schema of the database will be tracked and managed by the migrations service bundles viz **specmate-migration-api and specmate-migration**. Tracking changes in schema enables the software to preserve the integrity and consistency of the database.

4) **REST request managers:**
All the HTTP requests flowing in from the Angular application have be manged and mapped to their respective services. **specmate-emfrest-api, specmate-emfrest, specmate-rest** manage all the incoming requests and call the relevant function for each request. The return information from each function is delivered to the angular application for user consumption using the **org.json** bundle which parses and clean json data to be sent via **specmate-emfjson**.

5) **Logging services:**
Information and activity logs are managed by standard java library slf4j. The log message creation and management are handled by **specmate-logging-slf4j-julbridge, specmate-logging-slf4j, specmate-logging** bundles.

6) **Featured services:**
Featured services are the face of Specmate. These include creation of cause-effect-graphs, process models, generating test cases and procedures and automating all of these with product backlog or requirements information from Trello or Jira. These services are realised by **specmate-cause-effect-patterns** for cause-effect-graphs, **specmate-model-ecore , specmate-model-gen, specmate-model-generation, specmate-model-generators-typescript and specmate-model-support** for process models, **specmate-testspecification** for generating test specifications and **specmate-nlp and specmate-objectif** for automated generation of all the above. The automated generation requires connectors services to fetch cards for understanding and mapping the requirements.

7) **Common and global settings services:**
The configuration of Specmate bundles is globally managed by the **specmate-config and specmate-config-api** bundles. The common assets and variables accessible globally are contained in **specmate-common** bundle.

8) **Testing Services:**
Testing the software is vital. A few important bundles have their own test bundles. For instance, **specmate-auth-test** is the test bundle for authentication service. Similarly, Migration testings are managed by **specmate-migration-test**. Although the complete software has an integration test bundle which evaluates the compatibility of each bundle with another. This is done by **specmate-integration-test** bundle. All the dummy data required for testing is provided by **specmate-dummy-data** bundle.

9) **UI Service:**
UI service bundle is a rather unique bundle. It is an Angular build as an OSGi Bundle. Therefore on running the application, the user interface is streamed via this bundle which then communicates with other services to behave a whole system

There are other bundles as well to manage software setup and resource acquisition. These include **specmate-jetty-starter, specmate-cdo-server, specmate-persistency-api, specmate-persistency-cdo** bundles.

D. *Tools and techniques used to understand Specmate*

The following are the tools used to understand Specmate:

1) **Visual Studio Code:** Visual Studio Code is a code editor optimized for building and debugging modern web and cloud applications. Its user experience and directory tree visualization allowed easy exploration of Specmate source code. The syntax highlighting and Intellisense available in Visual studio code contributed to better code readability and navigation. Chaining up the declaration hierarchy and locating critical function calls from one component to another was seamlessly managed and supported by Angular and Java Language extension packs available as NuGet package in the marketplace of Visual Studio code. Also, the search functionality simplified the process to find a particular code block with a unique id or line of code from the complete code base.[11]

2) **FireFox Developer Tools:** Inspecting and identifying different components rendered on Specmate user interaction interface was realized with FireFox Developer Tools. It was extensively used to tinker and test changes on the page before applying the modifications to the angular components itself.[12]

The following are the techniques used to understand Specmate:

1) **Understanding the technology:** Before inspecting Specmate it was essential to understand the architecture of technologies used. Therefore, the first stop was to identify the technologies used to build Specmate and gather sufficient information and knowledge on them to progress further.

2) **Understanding the Directory Structure:** The knowledge of technologies used to build Specmate simplified the process of consuming the directory structure of Specmate. The code arrangement and categorization appeared meaningful and assisted in easy exploration and navigation through a complex hierarchy of folders.

3) **Running through snippets of code:** It was critical to get comfortable with the language and complexity of code in Specmate. Reading through chunks of code enabled understanding the purpose and execution of the

component.

4) **Establishing a relationship between different components:** The naming convention of Specmate greatly contributed to establishing a relationship with different components of the application. Knowing the purpose of a component aided in connecting and pooling all associated components together. This was further verified by the directory structure housing inter-related components together in a common parent folder. With a good understanding of the technology, everything now came together and generated a mental model about how different components were interacting in Specmate to run as a whole.

III. Sprint 2

The objective of Sprint 2 was to evaluate the code quality of Specmate and refactor code smells and bugs in the software.

A. *Code Quality Evaluation and Refactoring*

Code quality is subjective and coupled with the context of the software. Analyzing the code quality of Specmate using static code analysis exposed fragile points in the software capable of sabotaging it. Refactoring these fragile points defined the potential of the software to accept and retain change thus, commenting on its overall maintainability.[13]

B. *Static code analysis on Specmate*

Static code analysis is a method of evaluating the correctness and quality of the source code for debugging before execution against a standard set of language and coding rules.[14]

Sonarqube is an open source static code analysis tool for monitoring code quality through discovery of code smells, bugs, vulnerabilities and security hotspots.[15] It was used to analyse the source code of Specmate. Fig 3 shows the measures generated by Sonarqube on running an inspection on Specmate.

Sonarqube identified and estimated the quality of code against the discovered bugs, code smells, vulnerabilities, and security hotspots in Specmate. The initial impressions on Specmate from the analysis by Sonarqube were negative and demonstrated poor code quality. On manual exploration of the reported bugs and code smells, it was interesting to realize that majority of the reported issues were false positive. This was a result of incompatible language rules adopted by Sonarqube. The language rules were not in compliance with Java Language Specifications(JLS). The rules appeared to be biased by personal choices and practices. Also, Sonarqube cannot understand the Specmate as a system and consume its implementation of different architectures. Additionally, The java language syntax used by Sonarqube was not particular to OpenJDK 11 which resulted in false reporting of lazy finals declarations initialized during the compile time as bugs which contributed to 169/230 total bugs in the Specmate software. There are multiple occasions when developers resort to unconventional methods of code implementation to realize a feature. This is primarily because of the complexity of the system and the unavailability of no possible alternative. Sonarqube penalizes them without understanding or viewing the file as a whole and suggests changes that would certainly break the feature of the software.

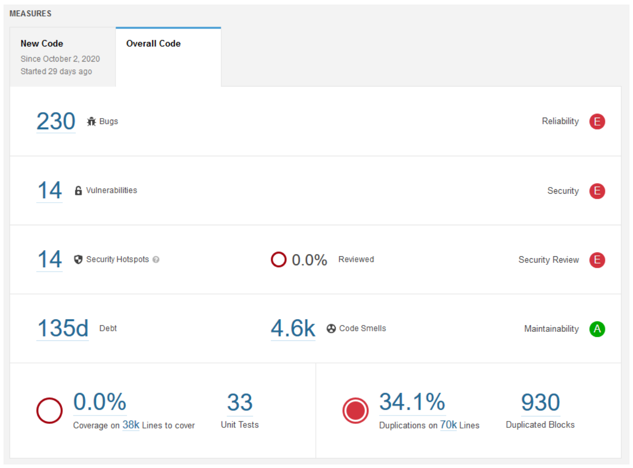


Fig 3: Sonarqube measures on Specmate

Isolating the shortcomings of Sonarqube, there were many positives about it as well. Sonarqube exposed critical security hotspots and vulnerabilities capable of comprising the software. These issues were real and demanded immediate attention due to their severity. Also, Sonarqube did find many instances of improvements that could be made to comply with better coding standards and practices.

The observations made by the manual exploration of issues reported by Sonarqube drives to an important conclusion that static code analysis tools always play to developer advantage by reporting a variety of issues from a large codebase through a comprehensive compilation of coding standards. Although, there has to be a human essence for confirming these issues before initiating a patch for them.

C. *Issues*

The majority of issues reported were code smells and bugs. A bug is an unintended and unwanted occurrence of a sequence of events triggered by a certain block of code. Bugs generate an irrelevant or wrong instance of information. Although, occasionally, they may be capable of collapsing the software as well.

A code smell in contrast to a bug is considerably less capable to create chaos but indicates deeper problems in the software. The classification of code smell is subjective although, in commonality, they are an ill-structured implementation of features and functions which may not be the ideal way of executing them. Despite not creating a problem, they may be responsible for the deteriorated standard of non-functional requirements in the software. These may range from memory leakages, unwanted resource utilization, and extensive computational time to simple code readability concerns.

Since majority of the issues reported throughout Specmate were minor and false positives, minimum effort and changes were required to fix real problems. Also, restriction on addressing the issues specific to ***"specmate-model-gen"*** bundle further narrowed down the choice of bugs and code smells that can be attended to. The ***"specmate-model-gen"*** bundle was clean for the most part and only possessed false positive bugs in conjunction with minor code smells.

The following are issues in Specmate assigned to me for refactoring:

- **Bug #1: Change this condition so that it does not always evaluate to "true".**

  Nature: False Positive
  Quantity: 8
  Location: Multiple Files in ***"specmate-model-gen"*** bundle

  Justification: Lazy finals are default type declarations of final variables initialized at runtime of the program. A default type final variable is either null or none type. Although, when initialized by the program, these are rather interesting values depending on different factors occurring and affecting the program at runtime. Therefore there is a need to check if this final variable is assigned or null before returning data relevant to this variable. There also may be a need to reinitialize a certain set of commands if the final is null. Since having a final variable null and not utilizing it is certainly not normal in a bundle not part of the testing group, there has to be a process that assigns a value to this variable later to be used for governing the flow and output of the program. Sonarqube is a static code analyzer and hence does not take dynamic code flow into consideration. Therefore this issue was reported as a bug despite not being one. This bug exists all across Specmate and contributes to a total of 169/230 reported bug in the software by Sonarqube.[16]

  Refactoring: Adding comments to document the use of lazy final and indicate the purpose of null check for better understandability and avoid unintended changes.
  Refactoring Assertion: Since comments do not influence the execution of code, this change will keep the build stable.

  Issue Reference: https://github.com/EriksKlotins/PA2558-specmate-2020/issues/21 and https://github.com/EriksKlotins/PA2558-specmate-2020/issues/22

  Pull Request: https://github.com/EriksKlotins/PA2558-specmate-2020/pull/24

- **Code Smell #1: Replace this "switch" statement by "if" statements to increase readability.**

  Nature: Real
  Location: Multiple Files in ***"specmate-model-gen"*** bundle
  Quantity: 4
  Severity: Minor
  Effort Estimation: 20 Minutes each

  Description: Switch cases are more readable in comparison to if-else. Although, using the switch in nesting is unconventional and not readable. There are instances in the program which demand nested switch because of no other alternative to ship the functionality. But, the cases refactored were supporting nested ifs without changing the logic of the code. If these nested switches were not replaced then there may be occasions where developers add wrong cases into different switch cases due to its complex readability resulting in the collapse of the software. Also, this would be malfunction would be hard to debug as well given it is not readable enough.

  Refactoring: The switch cases were thoroughly analysed for understanding the code flow. A simulation taking predefined variable for running only the reported code snippet was designed and replaced with if-else case to confirm if the patch is possible in a playground. This code flow was than replicated using if-else statements and replaced with the nested switch case section causing the code smell in source code post patch confirmation from the playground simulation.
  Refactoring Assertion: Post replacing the code block, the software was run to check if there were any errors. Also, static code analysis was conducted post implementing the changes to inspect for new bugs and code smells that might be arising due to the change. Both the software and static code analysis did not return any signs of errors or bugs/code smells. On triggering the workflow using the replaced code snippet, return expected output further confirming the change. Finally, with Travis-CI, the build was checked through intensive tests which the software passed thus confirming that the changes made were valid and did not break the software.

  Issue Reference: https://github.com/EriksKlotins/PA2558-specmate-2020/issues/20
  Pull Request: https://github.com/EriksKlotins/PA2558-specmate-2020/pull/24

IV. Sprint 3

The objective of Sprint 3 was to implement set of features in Specmate.

A. *Impact Analysis*

Impact Analysis is the identification of impact set or parts of components and blocks of code affected by the incoming change request for improvement, migration, or feature implementation. It visualizes the effort and context switch on introducing the change to the system. Also, it estimates the cost and schedule to deliver the change.[17]

To implement the requested set of features, impact analysis was conducted on Specmate to identify the impact sets for each individual feature. This was realized predominantly by the call-graph based analysis on Specmate. For each requested feature, DOM objects involved in or part of change were inspected during execution using Firefox Developer tools. Using their Element ID all the folders were searched for any occurrence of their instance in codebase using Visual Studio Code. When an element was found its call hierarchy and onclick functions were analyzed to understand the code flow. Progressively following the call hierarchy and examining different function declarations and their parent classes involved in the process lead us to speculate on probable directory or component that had to be changed to implement a feature.

Specmate, with its naming convention, made exploring and navigating through the code predictable. The directory structure and component groupings of Specmate contributed to understanding the code and purpose of each component. Therefore, the impact set became rather obvious when analyzed in cohesion with its grouped components. The source code in Specmate also demonstrates good naming conventional with meaningful class, functions, and variable name thus, requiring the minimum need for adding comments between blocks of code. The functions, when read, sounded like regular sentences directly translating their purpose and process.

The impact set of all requested features were residing in the angular application of Specmate software. The granular description of each requested feature clearly defined the components to investigate. Since the angular application of Specmate facilitates all the components within the ***"web/src/app/modules/"*** directory, speculating the impact set for each feature was streamlined. For most cases, only the component had to be modified to implement the feature, although there were instances where multiple files apart from the components themselves had to be tinkered to realize a feature. The feature #11 to have default names for creating new cause-effect-graphs, process models and test specifications instead of disabled create button if no name input is provided by the user was a unique case where the change had to be supplied in the ***"web/src/app/modules/views/main/editors/modules/contents-container/"*** rather than the component itself. Despite this, the code structure and readability of Specmate source code lead us to the actual impact set with the help of call-graph analysis executed using the language extensions available in Visual studio code. Thus, Specmate has a good analyzability as a result of its naming convention and code structure.

B. *Feature Implementation*

The following were the features implemented by our group:

1) **Feature #2: Dark Mode GUI**
   Dark mode GUI was realized by adding bootstwatch Cyborg theme stylesheet CDN to the root index.html of the angular application. This CDN value was toggled between the Cyborg theme and Yeti theme used as the default or light theme for the application with a button on the navigation bar of the Specmate software. Themes are subject to the overall user experience of the application. Therefore its scope lasts throughout the usage of the application. Hence, we had to identify the entry point of the software first. Initially, our plan was to avail the theme switch on the Login screen as well. We implemented the toggle button at the entry point of the software. But post its implementation, the User interface felt cluttered, and decided to push it in the post the login screens. Thus Navigation bar was the perfect host for having a theme switch. We implemented the button on the navigation bar component in ***"web/src/app/modules/navigation/modules/navigation-bar/components/"*** and tested it. We made the default theme available on the login screen to keep the user experience conserved by setting the default CDN value to the Yeti theme. Fig 4. shows the execution post feature #2 implementation.

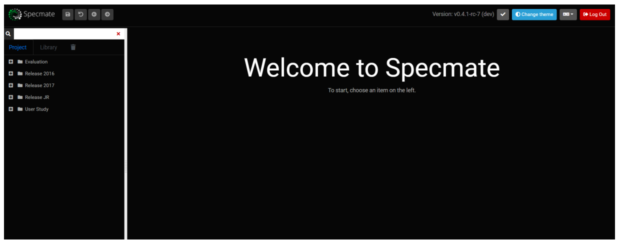
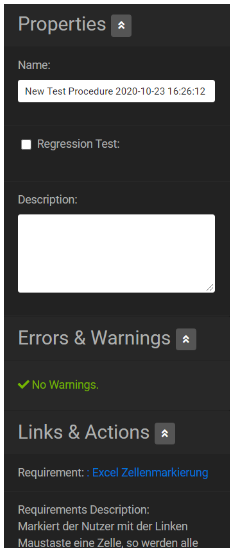


Fig 4: Dark Mode theme in Specmate

2) **Feature #4: Misaligned Checkbox**
   The input checkbox in the properties tab of Specmate was sticking to the left corner of the tab split despite being aligned left by default. Forms are individual components of their own in Specmate. Thus, we navigated to find the checkbox component. Before implementing the change we tested it on the running webpage of Specmate using the Firefox Developer tools by changing the style attributes. Upon verifying that the change was

functional and as expected, we made the modification in ***"web/src/app/modules/forms/modules/generic-form/components/form-checkbox-input.component.html"*** file. We added a 3-pixel margin on the left of the checkbox and further 3 more for its label to align with it perfectly.Fig 5. shows the execution post feature #4 implementation.


Properties

Name: New Test Procedure 2020-10-23 16:26:12

Regression Test:

Description:

Errors & Warnings

✓ No Warnings.

Links & Actions

Requirement: : Excel Zellenmarkierung

Requirements Description:
Markiert der Nutzer mit der Linken
Maustaste eine Zelle, so werden alle

Fig 5: Aligned Checkbox in properties tab of Specmate

3) **Feature #7: Icon for cause-effect-graphs**
The previous icon for cause-effect-graphs was fa-share-alt of the font-awesome pack 4 and looked inconsistent for representing the same. Therefore we first looked for relevant icons to represent cause-effect-graphs on the newer font awesome 5 pack. We believed that the fa-project-diagram will be apt to showcase the cause-effect-graphs. Now we started looking for the component to change. Since icons are common components used by different views, we started looking in the ***"web/src/app/modules/common/modules/"***. We were able to find icons component in this directory and successfully update the icon for cause-effect-diagram. Although we needed to migrate from the existing font awesome 4 to 5 for the icon to work. This was done by completing feature #12. Fig 6. shows the execution post feature #7 implementation.

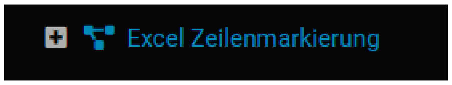


Fig 6: Icon for cause-effect-graphs

4) **Feature #11: Default Names**
The create button for cause-effect-graphs, process models, and test specifications were disabled if the user didn't supply a name for them. Thus it was meaningful to have default names for creating new cause-effect-graphs, process models and test specifications instead of disabled create a button if no name input is provided by the user. This feature was realised by making changes in the ***"web/src/app/modules/views/main/editors/modules/contents-container/components/"***. This was an interesting case where the generate button components themselves had little or no required change. This was because all the elements required different components placed in a container to render as a whole. Also, all the generate buttons were calling a single function with signature **create(name: string)**. This function was responsible to validate the passed name pattern and transfer control to abstract functions **createElement(name: string)** having its own respective implementation in cause-effect-graphs, process models and test specifications component. Therefore we had to include an alternative case if the name parameter was null. On null we assigned a default name and passed the control to the **createElement(name: string)** function. Fig 7. shows the execution post feature #11 implementation.

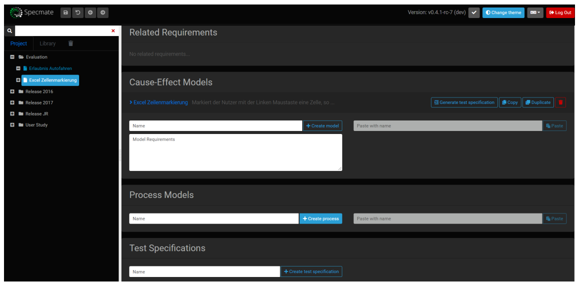


Fig 7: Default names for element creation

5) **Feature #12: Prominent Icons**
Specmate was previously using font awesome icons pack version 4. Our objective was to migrate it to version 5. This was a rather simple feature to complete but required knowledge of angular applications more than the overall understanding of Specmate. The analyzability of Specmate had nothing to do with this feature. All the angular applications import their styles in the ***"angular.json"*** configuration file residing in the root of the project directory. Since ***"web/"*** is the root directory of the angular application of Specmate, the ***"angular.json"*** was located in this directory. We replaced the font awesome version 4 import with version 5 to implement this feature. Every font awesome version adds new icons but also preserves previously used icons with a newer design. Therefore all the existing icons just upgraded to their new design without us having to do anything. These icons were significantly more prominent and represented their purpose. Fig 8. shows the execution post feature #12 implementation.

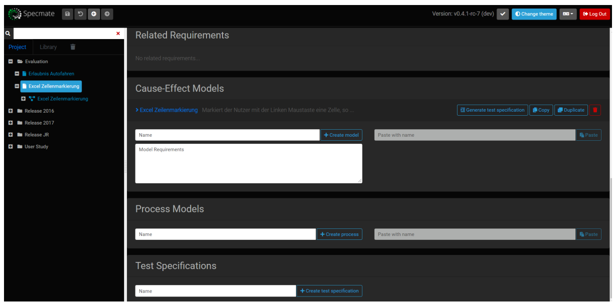


Fig 8: Updated fonts pack in Specmate

C. *Estimating Effort and Asserting Impact Set*

The expected time estimation for each feature implementation varied from its actual time. All the expected time estimation accounted for testing and validating the feature post its implementation. For majority of the features, the expected time was overestimated taking the benefit of doubt into consideration. Although, there was also an instance where we underestimated expected time. Table 1 compares the estimated and actual time for each feature implementation.

Sno	Feature	Estimated time	Actual Time
1	#2	240 mins	240 mins
2	#4	30 mins	15 mins
3	#7	120 mins	20 mins
4	#11	60 mins	120 mins
5	#12	60 mins	10 mins

Table1: Comparing estimated and actual effort for feature implementation

Feature #2 matched the expected time estimation as a result of the identification of the correct impact set. We speculated to have changes in ***"web/src/index.html"*** since it was the root rendering component that proved to be the exact location for including theme CDNs. The theme switch button was initially planned to be a part of both the navigation bar and login screen. Later we pushed it to the navigation bar alone. Thus the function to switch the CDN value was included in the ***"web/src/app/modules/navigation/modules/navigation-bar/components/"*** which was part of our impact set for implementation of dark mode GUI feature. The analyzability of the Specmate also contributes to accurate time estimation. Specmate implements navigation bar as a component in the ***"web/src/app/modules/navigation/modules/"***. Thus, it was predictable to push changes in the navigation bar component if we wanted to implement a new DOM object in the navigation bar. Taking these aspects into consideration alongside buffer time for testing and mitigating blockages enabled us to estimate accurate estimated time.

Feature #4 was overestimated to take 15 minutes more than what was required. This was hugely influenced by the speculation of disturbing other properly aligned input checkboxes with the change. Since the input box was already left aligned but was not replicating the same on the properties tab, it was obvious to add padding from the left of the container. This would take mere minutes but be overestimated to verify if other checkboxes in different views were affected with this fix. Our verification could only range to exploratory testings since the UI-test for Specamte was still in development and disabled in the Travis-CI check. Therefore it is not completely right to say we overestimated the required time. There may be a possibility that we might have missed some view given that Specmate Software has many views. In conclusion, we believe the fix to deliver well on resolving the issue from what has been tested across manually. Although the impact set was correct and again can be credited for the analyzability of Specamte implementing forms as a component in ***"web/src/app/modules/forms/modules/"***.

Feature #7 time estimation was heavily influenced by peer group members. Personally, a couple of minutes should have been sufficing but respecting the opinion of fellow developers, we decided to resort to an unrealistic estimation. Although, since it was an overestimation, there was little to be worried about since we did not choke our feature implementation with an underestimated schedule and work distribution. The personal choice of a couple of minutes for this feature was part of Specmate having a good code structure. All the icons are generated from the common icons components in ***"web/src/app/modules/common/modules/"***. Thus changing the icon with the desired one would suffice the feature. Therefore ***"web/src/app/modules/common/modules/"*** the exact file included in the impact set for this feature.

Feature #11 was criminally underestimated as a result of the wrong impact set. In our impact analysis we speculated that if we change the generate button component of each of the three elements viz cause-effect-graph, process model, and test specification, then we can implement the feature. But later during the actual implementation, we discovered that all three elements used a common function call to create a new element from a complied container. Although this container using the generate button component, its onclick function was calling the common create method before passing the control to specific implementations of element creation. This was a personal error not to be associated with the analyzability of Specmate. The code structure of Specmate helped us realize this mistake rather quickly and exposed the real impact set in a couple of minutes with help of other tools like Visual Studio Code and Firefox Developer Tools.

Feature #12 was opinionated on the approach that will be followed to replace the current icons with better ones. There were two alternatives. Either to upgrade the current font-awesome pack of Specmate to a newer version and keep the icons intact with better designs or change the icons library altogether. We choose the former keeping maintainability and evolution in mind. Migrating from an older version to a newer version was relatively more effective and maintainable than changing the complete library. This feature has no association with the analyzability of Specmate as it requires the knowledge of Angular applications instead to implement this feature. Therefore, it cannot be concluded that we underestimated or overestimated the time to implement this feature. It was solely based on personal choice. The impact set was ***"web/angular.json"*** since all the style imports and angular application configurations reside there.

D. Maintainability of implemented features

Maintainability was the central character around which all the changes for requested features were devised. The feature implementation accounted for the ability to extend or restore existing modifications to preserve the maintainability of Specmate.

Feature #2 primarily utilized the navigation bar component for hosting the theme switch button. This button can be easily replaced with a toggle switch or any other mode of clickable interface for later incoming change requests. Also, the DOM Object is part of mr-auto aligned tag and can be easily removed by deleting the HTML element for the theme switch button. The **changeTheme()** function binding with this button is also flexible to move to other components or part of Specmate view since all the data members required for the functioning of the feature are independent and do not couple with any component objects.

Feature #4 will invite a minimal change provided there is a change request to edit it which is unlikely given that it is a generic form input field. Although making changes to it would simply require the developer to remove the ml-3 margin class from its class attribute to revert to as it originally was. Also, the alignment can be further adjusted according the developer requirement by changing the pixel value in ml class.

Feature #7 can be reverted to its original state by replacing the fa-project-diagram class in the i tag class attribute with any other desirable icon. The change will demand only a few seconds to complete since the impact set is already asserted.

Feature #11 was carefully evaluated for maintainability before implementing it. Optimal changes were introduced to maintain the integrity of the system. Initially, before implementing the change, we examined if duplication of names was permitted for each of the three affected elements viz cause-effect-graphs, process models, and test specifications. Post confirmation, we included and if case to check for nullability of the name. This check includes a default name which can be changed to more complex and unique cases on each new element creation. Also if wanted to removed from the system, the developer will have to comment or remove the nullability check only and enable button click disabled. The check for button disabled has not been disturbed in the button component code. Hence, the default names feature can be easily extended or removed from the program.

Feature #12 uses font-awesome 5 icons pack. Font-awesome is a standard icon library used widely for different web applications. Also, Specmate previously used font awesome 4. Migrating to newer version ensures that the legacy of the code is preserved thus inflicting minimum change to realize the feature. Also migrating to a newer version will as simple as the current one. Additionally, there will not be any requirement to rename the icons or find new ones since font awesome generally preserves all the icons from previous versions and adds new designs to them. Hence updating the icons will be piece of cake provided the developers choose to stick with font awesome.

V. Conclusion

This report captures reflection on Specmate, a requirements modeling and test generation tool over 3 Sprints each inspecting and analyzing its code structure, quality, and response to change.

Specmate takes advantage of the OSGi architecture using Angular and RESTAPI builds as bundles. Thus, from an architectural standpoint, Specmate reflects favorable characteristics for high maintainability.

The high analyzability of Specmate supports seamless navigation and code exploration thus, inviting and retaining changes without breaking the software.

New features implemented on Specmate were adapted and integrated without any impediments hence commenting on the modular code structure of Specmate.

Therefore, with all the observations taken into consideration, it is safe to say that Specamte has good maintainability.

References

[1] Riaz, Mehwish, Emilia Mendes, and Ewan Tempero. "A systematic review of software maintainability prediction and metrics." In Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement, pp. 367-377. IEEE Computer Society, 2009.

[2] Heitlager, Ilja, Tobias Kuipers, and Joost Visser. "A practical model for measuring maintainability." In Quality of Information and Communications Technology,2007.QUATIC2007.6th International Conference on the, pp. 30-39. IEEE, 2007.

[3] de AG Saraiva, Juliana, Micael S. De França, Sérgio CB Soares, J. C. L. Fernando Filho, and Renata MCR de Souza. "Classifying metrics for assessing object-oriented software maintainability: A family of metrics' catalogs." Journal of Systems and Software 103 (2015): 85-101.

[4] Von Mayrhauser, Anneliese, and A. Marie Vans. "Program comprehension during software maintenance and evolution." Computer 28, no. 8 (1995): 44-55.

[5] Specmate.Qualicen GmbH. Dehttps://specmate.io/. Accessed on 01/11/20.

[6] OSGi Alliance. OSGi Architecture. https://www.osgi.org/developer/architecture/. Accessed on 01/11/20

[7] Almugrin, Saleh & Albattah, Waleed & Melton, Austin. (2016). Using Indirect Coupling Metrics to Predict Package Maintainability and Testability. Journal of Systems and Software. 121. 10.1016/j.jss.2016.02.024.

[8] Dubey, S.K., Rana, A., 2011. Assessment of maintainability metrics for object-oriented software system. ACM SIGSOFT Softw. Eng. Notes 36, pp. 1–7. doi:10.1145/2,020,976.2020983.

[9] Frappier, Marc, Stan Matwin, and Ali Mili. "Software metrics for predicting maintainability." Software Metrics Study: Tech. Memo 2 (1994).

[10] Hegedűs P., Bakota T., Illés L., Ladányi G., Ferenc R., Gyimóthy T. (2011) Source Code Metrics and Maintainability: A Case Study. In: Kim T. et al. (eds) Software Engineering, Business Continuity, and Education. ASEA 2011. Communications in Computer and Information Science, vol 257. Springer, Berlin, Heidelberg

[11] Visual Studio Code. Microsoft. https://code.visualstudio.com/. Accessed on 01/11/20

[12] Mozilla Developer Tools. Mozilla Foundation. https://developer.mozilla.org/en-US/docs/Tools. Accessed on 01/11/20

[13] Stamelos, Ioannis, Lefteris Angelis, Apostolos Oikonomou, and Georgios L. Bleris. "Code quality analysis in open source software development." Information systems journal 12, no. 1 (2002): 43-60.

[14] Louridas, Panagiotis. "Static code analysis." Ieee Software 23, no. 4 (2006): 58-61.

[15] Sonarqube. SonarSource. https://www.sonarqube.org/. Accessed on 01/11/20.

[16] Lazy finals. OpenJDK.https://cr.openjdk.java.net/ jrose/draft/lazy-final.html. Accessed on 01/11/20.

[17] Mohr, Lawrence B. Impact analysis for program evaluation. Sage, 1995.