Report on Evolution and Maintenance of Software – Project

Anusha Mogili
19970626-8167
moan17@student.bth.se

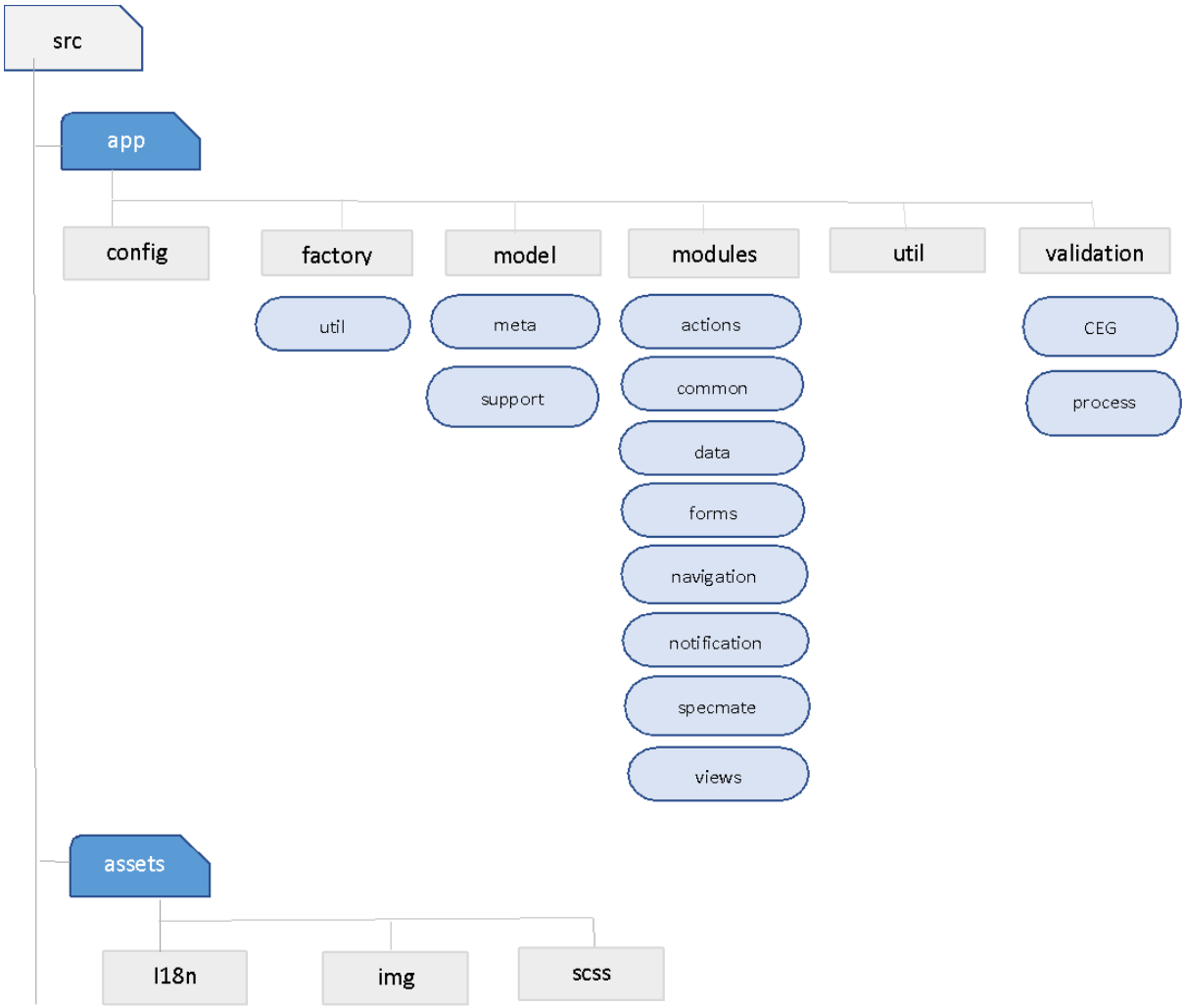
Vadrevu Krithi Sameera
19961031-6086
vakr17@student.bth.se

Kanneganti Joshika
19970429-1781
kajo17@student.bth.se

Manoj Kumar Pallapu
19970919-2018
pama17@student.bth.se

SPRINT 1

Code Structure



In general, Specmate code structure is divided into different modules such as:

config : This folder consists of settings and predefined function required for User Interface.

model : which contains data objects to communicate between front end and back end.

SpecmateRoutingModule : helps to navigate from one component view to another, as users perform required UI tasks. In specmate application following tasks are performed in : *CEGModelEditorModule, PageNotFoundModule, WelcomePageModule, ProcessEditorModule, TestProcedureModule, AuthModule, RequirmentDetailsModule, TestSpecificationEditorModule, LoginModule*.

modules : Main component of the application, which contains different use cases performed by the user in Specmate application are:

actions

- Common-controls like undo, forward, back, save functions are enabled here.
- Export-test procedure and test specification are the important features for Specmate application for taking user input and getting the desired output.
- *view-controller.services* perform tracing links.

common

- The *common* module used for styling the icons, i18n-language chooser, and truncate-pipe.

data

- *data* module consists of service imports such as *HttpClientModule, LogListModule*, and declarations for components present in this module. The exports that are visible in this module are service providers and bootstrapped components with different services such as command, data-cache, e-operation, scheduler, service-interface, specmate-data service.

forms

- Where different type of form operations are performed such as checkbox selection, long-text input, single selections from drop-down. These actions are initiated in a *generic-form module*.

navigation

- Navigation module consists of a navigation bar which is a graphical user interface used by the user to move from one view component to the other, for accessing information.
- Project explorer is initiated to access search queries and search result functions to perform an action for obtaining the desired output.
- *element-tree* component helps in data binding and sorting items by moving elements throughout the tree and prevent from enabling descendent nodes.

notification

- Notification module helps in notifying the unsaved changes, error in opening the required view component, confirmation of requested changes with the following functions: open, confirmDelete, openOk, cancel, confirm, save, show log etc, these actions are controlled in an operation-monitor component with the help of SpecmateDataService, ViewControllerService, ChangeDetectorRef function.

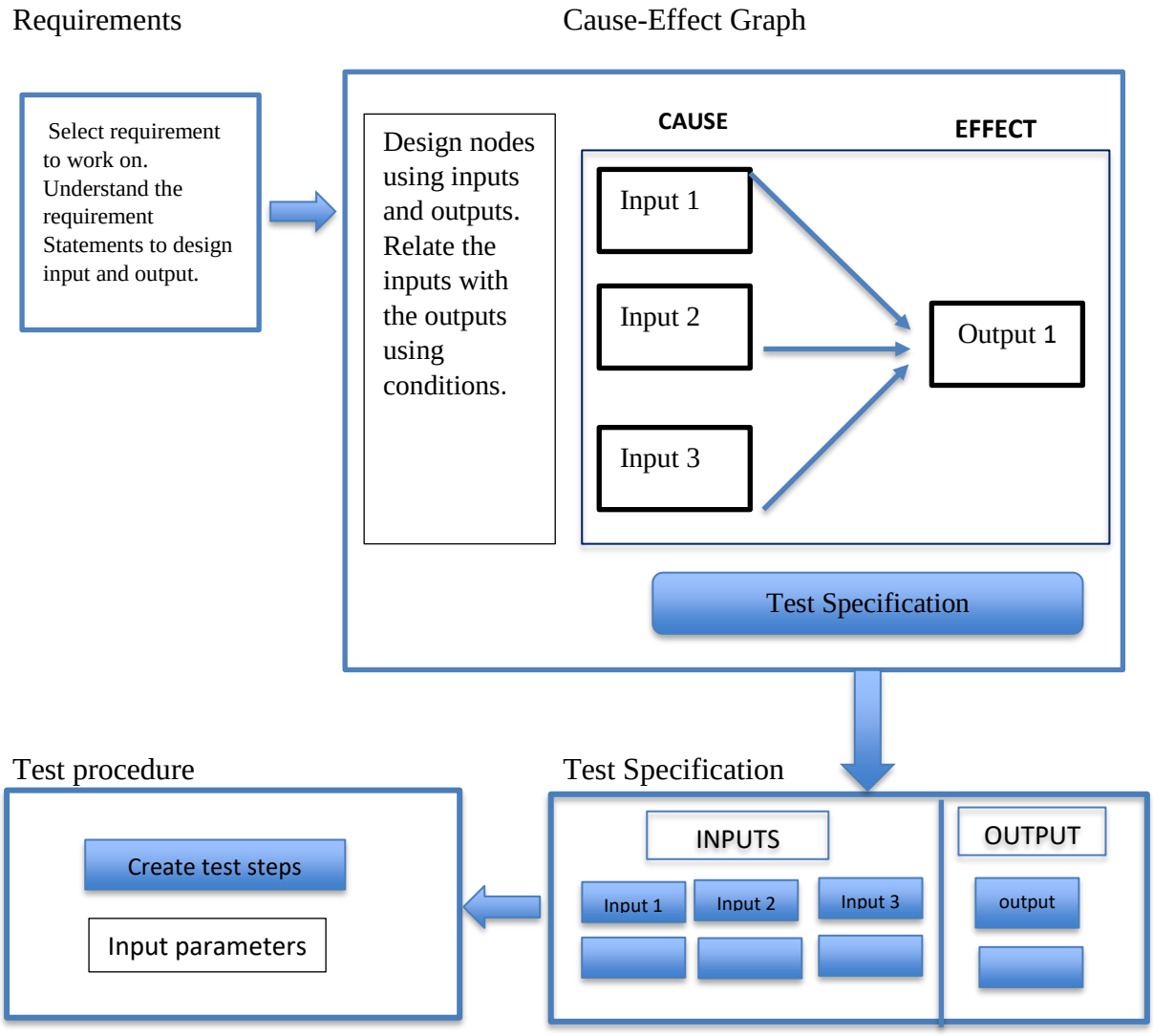
specmate

- Specmate component is the main component of the application where *loggingShown, navigationShown, explorerShown, linkactionsShown, historyShown,* etc. functions are performed to operate the UI actions.

views

- Controller module provides ViewControllerService for injected classes such as EditorMaximized, LoggingOutputShown, etc.
- Main modules help in performing event actions like login, logout, Auth verifications involving user permissions and authentication services. The second function of main modules consists of base editors such as cause-effect graph model editor, graphical editor, maximize button, requirement details, etc.
- Size module consists of log list, history, link actions where the different requirements are linked such as test procedure and test specifications and activated when action is performed, tracing links which trace the add requirement functionality whether it is newly added or already exists, etc.

Architecture of the System



The cause-effect graph is used to formulate test specifications. Here unnecessary cases that do not fit the logic can be removed. In the Test procedure, we can remove unwanted parameters if a reasonable test case cannot be formed.

Problem Domain:

Specmate is requirement based testing and semi-automatic test design tool.

As specmate is used to form test requirements, it is important to classify between rule-based vs process-based requirements. The main difference initiates from, whether they are isolated from other requirements or just the main part of the particular component in a system.

Specmate domain mainly revolves around two aspects:

1. Creating tests from single requirements
2. Creating end-to-end-tests from business process descriptions

Specmate uses the graphical representation for describing the logical implementation of requirements, in a cause-effect model. Generate test specifications for the user given test cases using Cause Effect Graph diagram. From this, it automatically generates test procedures.

Execution Effect:

Execution effect describes how changes done to the tool affects the behavior of the system. As most of the components in specmate tool are interconnected, the modularity affects the maintainability with fluctuating values for every refactoring or new implementation performed. One needs to make changes to the system bearing in mind, the impact of the expected outcome. The overall behavior of the system should not be changed in any case. Generally, by giving different inputs to the system, its behavior changes according to our requirement and also it will affect the overall maintainability of the specmate tool.

Cause-Effect of specmate tool:

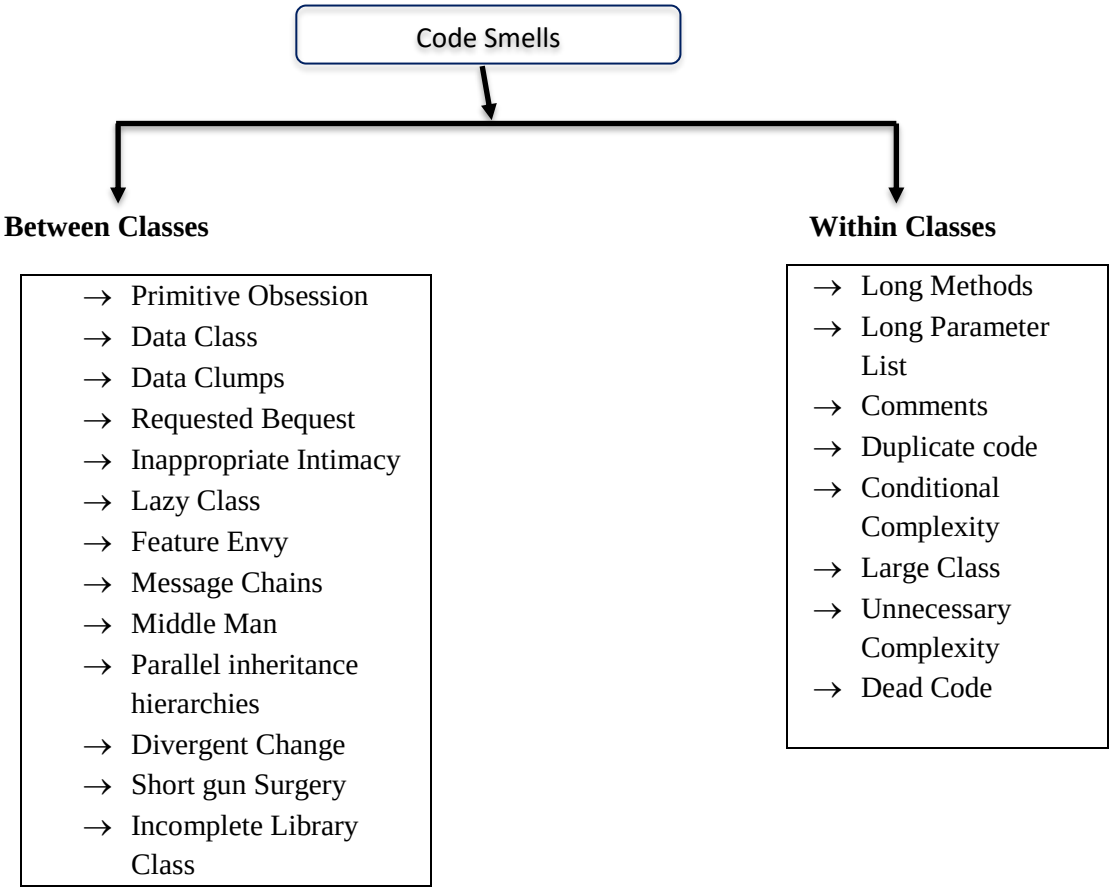
Here, In the specmate tool, the changes in the one component might affect other components as they are interdependent on each other. If we make any change in a particular component, those changes should be compatible with the other components without changing behavior of the system. For example, changes made in *additional-service.information* component will affect *links-actions.component.*

SPRINT 2

Generic Code smells

Code smells can be categorized into two types.

1. Code smells within classes
2. Code Smells between classes



Description of code Smells we encountered while analysing specmate:

1. Long methods: Methods containing too many Lines of code.
Treatment: Break a single method into smaller methods such that the method description must be enough to understand it.
2. Unnecessary complexity: Variables are declared but not used.
Treatment: After proper thorough checking, remove the unnecessary variables.
3. Duplicate Code: Different code fragments look similar.
Treatment: Write the content in a single method and call the method wherever required.
4. Conditional Complexity: extensive use of conditional operators.
Treatment: keeping conditions as simple as possible and break the condition wherever possible.

Description of code smells and their types.

S.no	Issue	Description	Type of code smell
1	Object.ts.equals Complexity	Method has conditional statements more in number which might make it complex.	Long methods
2	ChangeFields Complexity – length	Method has complex code with loops and nested conditions.	Long methods
3	Return type void Used as parameter to another function (promise.resolve)	If a function does not return anything, it will make no sense in passing its output to a variable	-
4	Refactor this function so that its implementation doesn't duplicate	A block of code is repeated in same class in two different places	Duplicate Code
5	Getter – Global	Refactor this getter so that it actually refers to the given property and verify if the correct fields are being accessed.	Unnecessary complexity
6	Arrangement of CSS file - button: focus	Problem with positioning the button.	Conditional complexity
7	Element-tree: CSS	Typo generated while using selectors in CSS file which are defined in HTML page.	-
8	Font-family	The Font-family attribute does not have anything assigned. A font-family must be mentioned to avoid browsers from displaying their default font.	-
9	Not used variables	Variables are declared in a class but not used.	Unnecessary complexity
10	Unnecessary Boolean	Unnecessary use of logical operator. There is no difference in the output even if it is used.	Unnecessary complexity
11	String object - string cast	Changing wrapper class to primitive type.	Unnecessary complexity
12	Undefined parameters	Explicit mentioning of “Undefined” creates unnecessary redundancy.	Unnecessary complexity
13	Unnecessary cast	Typecasting is unnecessarily done for a variable.	Unnecessary complexity
14	Void return	When return type of the function is mentioned as void, then it should not return anything. So, the return statement is unnecessarily written.	-

Table 1: Description of code smells in Specmate.

S.no	Issue	Severity	Real Issue/ False positive	Analysis of issues
1	Object.ts.equals Complexity	Critical	Real issue	
2	ChangeFields Complexity – length	Critical	Real issue	
3	Return type void Used as parameter to another function (promise.resolve)	Critical	False positive	When a void is passed to another variable nothing changes. As creating a new variable and passing nothing to it makes no sense, it is pointless to declare such a variable.
4	Refactor this function so that its implementation doesn't duplicate	Critical	Real issue	Sonar cube identified these issues as code smells and moreover, these are code smells in real. Hence we identified them as True positive/ Real issue
5	Getter – Global	Medium	Real issue	
6	Arrangement of CSS file - button: focus	Medium	Real issue	
7	Element-tree: CSS	Small	Real issue	
8	Font-family	Small	Real issue	
9	Not used variables	Small	Real issue	
10	Unnecessary Boolean	Small	Real issue	
11	String object - string cast	Small	Real issue	
12	Undefined parameters	Small	Real issue	
13	Unnecessary cast	Small	Real issue	
14	Void return	Small	Real issue	

Table 2: Identification of issues (real or false positive) & severity of code smells

Description of the smells/bugs assigned to our team

(i) Refactor this getter so that it actually refers to the property '_model' (M)

Refactor this getter so that it actually refers to the property '_requirement' (M)

Getters and setters are used in order to provide data encapsulation, which means that not all classes can access them. Private variables can be accessed making use of getters and setters. Here getter functions are declared but we need to check if correct references are being made or not.

Refactor code

In *links-actions.component.ts* we have removed the unused public variables '*_requirement'* and '*_model'*. In addition to that we also removed unused variables '*_testSpecification'* and '*descriptionVisible*' as the same case can be applied to it.

\specmate-develop\web\src\app\modules\views\side\modules\links-actions\components\links-actions.component.ts

```
15  15  export class LinksActions {
16  16
17  17      public isCollapsed = false;
18      -
19      -   public _requirement: Requirement;
20      -   public _model: IContainer;
21      -   public _contents: IContainer[];
22      -   public _testSpecifications: TestSpecification[];
23  18      public descriptionVisible: boolean;
24  19
```

In *additional-information.service.ts* we removed the private variables '*_requirement'*, '*_model'* , and *'_contents'*. Moreover, the *"clear"* function was also removed.

\specmate-develop\web\src\app\modules\views\side\modules\links-actions\services\additional-information.service.ts

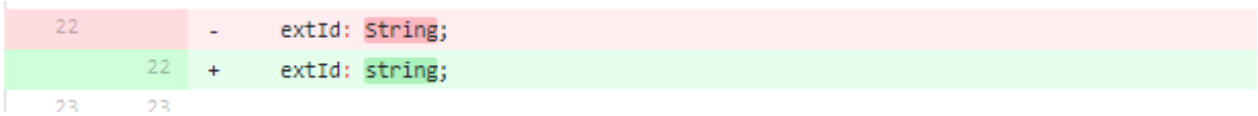
```
18  18  export class AdditionalInformationService {
19  19
20      -   private _requirement: Requirement;
21      -   private _model: IContainer;
22      -   private _contents: IContainer[];
23  20      public element: IContainer;
24  21      private parents: IContainer[];
25  22      private _testSpecifications: TestSpecification[];
    ❄   @@ -29,7 +26,6 @@ export class AdditionalInformationService {
29  26      constructor(private dataService: SpecmateDataService, private navigator: NavigatorService, private auth: AuthenticationService
30  27          this.informationLoaded = new EventEmitter<void>();
31  28          navigator.hasNavigated.subscribe((element: IContainer) => {
32      -           this.clear();
33  29              this.element = element;
34  30              this.loadParents()
35  31                  .then(() => this.loadTestSpecifications())
    ❄   @@ -76,12 +72,6 @@ export class AdditionalInformationService {
76  72          return readParentsTask.then(() => Promise.resolve());
77  73      }
78  74
79      -   private clear(): void {
80      -       this._model = undefined;
81      -       this._requirement = undefined;
82      -       this._contents = undefined;
83      -   }
84      -
```

ii) Replace this 'String' wrapper object with primitive type 'string'(S)

A primitive type is a wrapper class that consists of many data types that can be used in a class or method wherever required. Hence, whenever a variable type needs to be defined, a call is made to this class. But this reduces system performance.

Refactor code

**specmate-develop\web\src\app\modules\views\side\modules\tracing-links\components\tracing-link.component.ts*



During refactoring, we changed the “String” to “string”. By doing that we defined variable type string.

SPRINT 3&4

Id	Feature	Description	Effort Estimate	Time Spent
1	Print view for models	• Focus on model button by hiding all components appearing on the interface. • Show that every component is again visible on rollback. • PDF should be generated at the server and sent to the browser for download with additional functionalities such as setting title to the model.	Changeability = L, Testability = M.	3 working days
2	Export test specifications as CSV	Download test specification in CSV format.	Variant 1: Changeability = S, Testability = S. Variant 2: Changeability = L, Testability = M.	4.5 working days
3	A more intuitive symbol for AND and OR	Come up with different solution to improve representation of model for AND & OR symbols.	Changeability = S Testability = S.	1.5 working days
4	Wordwrap in nodes	• Inside a node, to avoid word-wrap, increase the Text block to an extent where information cannot hide. • Word-wrapping in Name field in process model and condition field in CEG model need to be fixed. Variant1: calculate text length to wrap Variant2: Wrap after fixed number of symbols.	Variant 1: Changeability =M Testability = S. Variant 2: Changeability =M, Testability = S.	1.5 working days

5	Not relevant parameters in test procedure	Unnecessary parameters should not be used in creating test steps. Those parameters should be marked as non-relevant parameters in test procedure	Changeability =M, Testability= M.	2 working days
6	Select field "Name"	Instead of entering name in the field, user must be able to fill data at the node itself.	Changeability =M, Testability= S.	3 working days
7	Background images for the login page	Variant1: Provide a background image for specmate login page. Variant 2: User can select and configure the background. User uploaded backgrounds should randomly display for every reload of the login page.	Variant 1: Changeability = S, Testability = S. Variant 2: Changeability =M, Testability = M.	4 working days
8	Use name of decision nodes	For the input parameters column of test specifications use the name of the decision nodes.	Changeability = L, Testability = S.	2 working days
9	Show the origin of the test specification	When test specifications are inquired, we must be able to figure out if the specifications are derived from a parent Url or manually.	Changeability =M, Testability = S.	1.5 working days
10	Cosmetics	• Icons: At present, the icons are too small that the user cannot recognize its function. • Color: For test procedures and Test cases, the contrast of the grey background to the red and blue buttons is a little too low. • The focus on the logout button is not clear enough. • In the properties of CEGs, there should be no double ":", if there is no external id in the requirement and vise versa.	Changeability =M, Testability = M.	1.5 working days
11	Graphical Editor Tool Palette Line Wrap	When we zoom in or zoom out the display, the bottom buttons are not properly visible and the tool palette of graphical editor is line wrapped. Either remove the word wrapping or adjust the buttons so that they are visible.	Changeability =M, Testability = S.	2 working days
12	Tab-Order of Editor Zoom/Grid/Maximize/Window-Buttons	In the graphical editor, the tab order of buttons is right to left which is reverse in general. So, it should be left to right.	Changeability =S, Testability = S.	3hrs
13	Disabled buttons not	The Buttons in the Toolbar like Save, back, forward etc. when disabled, should be skipped by the Tab-key.	Changeability = S, Testability = S.	1 working days

	reachable by pressing Tab			

** S- 1 working day, M- 2-3 working days, L-3 or more working days.*

Table 3: Description of Features.

Id	Feature	M	A	R	Mo	T	Implementation / parts of code affected/Impact on maintainability
1	Print view for models	Yes	Yes		Yes		● This can be achieved by hiding the tool palette and enabling the required button to focus or by adding hide button to card header. ● When clicking this button all components become invisible except model view and add functionality to the button to print within browser in PDF format. ● By double tapping it UI will be returned to standard view. The parts affected by implementing this feature are: *specmate.component* (call the function and load data in different arrays), *view.controller.service* (add new function to print and hide) **Impact**: Because change in one module effects other modules, modularity is increased. This leads to increase in complexity due to increase in analyzability, which in turn decreases maintainability.
2	Export test specifications as CSV	Yes		Yes	Yes		● Variant 1: create a button to export. Create new module for export test specification button in *action* module like other buttons. ● Create a component in this module and initiate functions to invoke them whenever user clicks the button and these functions invoke other functions and load the data in different arrays to provide download link. ● Parts affected by implementation are *export-testprocedure-button.component* for creating export button *links-action.component* and *additional-information.service* to see that it appear only on test specification page. ● strings translation can be added in *de.json* and *gb.json* files. **Impact:** Modularity and reusability decreases, which reduces maintainability.
3	A more intuitive symbol for AND and OR		Yes				Change the visual representation of AND (&), OR(/) in ceg.graphical.arc component. **Impact:** The change of symbol make it easy to understand and analyze. Hence, maintainability improves.
4	Word-wrap in nodes			Yes		Yes	This feature can be implemented in *truncated-text-component* and by giving the number of lines or maximum height as input parameter.

							Impact: By doing this, reusability increases as it can be used in other components for different text fields. By doing this the visual appeal may increase but performance may decrease in storing and reading the given input, which needs to be tested and evaluated. Overall impact on maintainability is moderate as visual appeal increases, and performance fluctuates.
5	Not relevant parameters in test procedure	Yes	Yes				Create function to sort the unused parameters in test-case-parameters and mark them in separate table and invoke in *test-step-row-component* so that those parameters are avoided in test steps creation. **Impact:** This increases performance but analyzability increases which leads to increase in complexity, and modularity. Hence, maintainability decreases.
6	Select field "Name"	Yes	Yes			Yes	● In these we have to relate the DOM element to the pallete on the right side, when we select the field then it has to highlight the related element.
7	Background images for the login page	Yes	Yes	Yes		Yes	**Variant 1:** We have implemented using afterviewinit hook, which is invoked after angular has initialized components view in *login.component*. By using this method, we have implemented background image for login page and it is not visible after logging into specmate. **Variant 2:** We have used input textbox with type file to upload images in *login.component.html.* ● An array is used to store the URL of uploaded images and background image of specmate is implemented. ● On each reload of the login page, Specmate shows a random image from the user uploaded images. For this, we have saved the array in session at *login.component.* ● When the page is reloaded, the stored images in array index which are present in session are retrieved and display randomly. **Impact:** By implementing this feature usability and customer interaction increases but performance decreases as the large no of bytes are stored in the process of retrieving images. This can be taken care by dynamic array retrieving process. Hence, maintainability decreases.
8	Use name of decision nodes	Yes		Yes			We have to name the input parameters by taking the decision nodes names. To implement these feature, we need to make changes in *test.specification.component.*
9	Show the origin of the		Yes			Yes	We create the test specifications in the different models. If we derive the specification from the other model then it

	test specification						should show the parent Url. So, we use inherit concept in *test.specification.component.* **Impact:** The analyzability becomes difficult which reduces maintainability.
10	Cosmetics		Yes				● The icons in the specmate are small for customers to know its functions. So, we used the grid large class in *language chooser.component* to increase size of the icons. Logout button is not focused, to highlight it we used styling in *logout.component* now the sign out button is focused. ● Due to the grey color in table the words are not clearly visible. To remove the grey background, we removed the table striped in the *test.specification.editor.* ● To clearly understand the external id is present or not in the test data we used the checkbox in *project.explorer. component.* Hence if the external is not present then there will be only single colon. **Impact:** For above all the understandability increases in turn maintainability is improved.
11	Graphical Editor Tool Palette Line Wrap	Yes		Yes		Yes	To overcome this problem, we used bootstrap spacing and padding to make the buttons visible on zoom in and zoom out. To implement the feature the changes made in *tool.pallete.component* **Impact:** it takes less time to analyze the requested changes, modifiability is also moderate so, it improves maintainability.
12	Tab-Order of Editor Zoom/Grid/ Maximize/Window-Buttons	Yes		Yes			By reversing the source code in the *graphical.editor.component* the order changes to left to right. **Impact:** Work is made easy for tester which increases Usability. Changeability is not affected, maintainability improves.
13	Disabled buttons not reachable by pressing Tab		Yes				This feature is implemented by making changes in *common.controls.component.* **Impact:** As we are adding code, analyzability of the source code becomes difficult by which maintainability reduces.

**M – modularity, R – reusability, A – Analyzability, Mo – Modifiability, T- Testability*

Table 4: Impact Analysis of features

The feature assigned to our group is Feature 7.
Effort Estimation:
We performed planning poker for the effort estimation, discussed about the individual decision why we think it is small/medium/large.

Expected effort:
Variant 1: This is small (1 day)
Variant 2: we expected this variant as medium(2 days).
Actual Effort:
Variant 1: To implement this variant we have taken one day and for testing half day.
Variant 2: To implement this variant we have taken 2 days.