Increasing test efficiency through coverage-based test selection

/ Department of
Mathematics and
Computer Science
/ PDEng Software
Technology

Increasing Test Efficiency through coverage-based test selection

Trupti Manik

Where innovation starts

Increasing Test Efficiency through coverage-based test selection

Trupti Manik
2019

Eindhoven University of Technology
Stan Ackermans Institute / Software Technology

Increasing Test Efficiency through coverage-based test selection

Eindhoven University of Technology
Stan Ackermans Institute / Software Technology

Partners

ASML Netherlands — Eindhoven University of Technology

Steering Group Ing. Roger Lahaye (ASML, Project Owner and Supervisor)
Ir. Paul van Gorp (ASML, Project Manager)
Dr. Rick Smetsers (ASML, Project Mentor)
Dr. Ir. Tim Willemse (TU/e, Supervisor)

Date 4 October 2019

Document Status Public

SAI report no. 2019/095

The design described in this report has been carried out in accordance with the TU/e Code of Scientific Conduct.

Contact Address
Eindhoven University of Technology
Department of Mathematics and Computer Science
MF 5.080A, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
+31402474334

Published by
Eindhoven University of Technology
Stan Ackermans Institute

Printed by
Eindhoven University of Technology
UniversiteitsDrukkerij

SAI report no.
2019/095

Preferred reference
Increasing Test efficiency through coverage-based test selection. , SAI Technical Report, September 2019. (SAI REPORT NUMBER: 2019/095)

Partnership
This project was supported by Eindhoven University of Technology and ASML.

Acknowledgments

The opportunity I had to carry out my PDEng graduation project at ASML was an excellent chance for learning and professional development. During this time period of 10 months at ASML, I got an opportunity to meet and work with highly motivated professionals.

Taking this opportunity, I would first like to express my gratitude to Mr. Paul van Gorp, the Manager of VirtualFab group at ASML who, in spite of being so busy, provided me with valuable inputs and resources necessary for this project. I appreciate his detailed and timely feedback on documentation. I would like to thank ST Management, who made it possible for me to pursue this project. I would like to express my sincere gratitude to my supervisors Mr. Roger Lahaye and Dr. Tim Willemse, for their guidance and constant encouragement. I appreciate Mr. Lahaye's inputs on brainstorming sessions throughout the project. I highly value that Dr. Willemse always provided essential feedback and suggestions to ensure that the project remains on track. I would like to express my great appreciation to my mentor Dr. Rick Smetsers, for valuable and constructive suggestions and encouragement through various phases of this project. I choose this moment to acknowledge all the time and effort Dr. Smetsers has put in mentoring me during this project. His in-depth feedback on documentation is highly appreciated. I would also like to appreciate the VirtualFab group, who made me feel very welcomed in the team and provided inputs as needed. I cherish all the coffee and lunch breaks with the VirtualFab group that helped me adapt to the ASML work culture very smoothly.

Like it is said, distance is just a number when it comes to family and friends. In spite of me being so far away from home, my family and friends were persistently there to ensure my mental and physical well-being, which was very essential for the completion of this project. I want to take a moment to express my deepest gratitude to my family and friends for their selfless contribution.

Executive Summary

The semiconductor fabrication process is getting more and more complex. The industry is moving towards a so-called 'holistic' manufacturing approach, in which traditionally different manufacturing steps (e.g., IC design, lithography, metrology) are incorporated in one complex, tightly integrated process [1]. ASML makes multiple systems that are needed in this process. Hence, systems at ASML are getting more and more complex.

One of the consequences of the complex system is that the number of tests is growing exponentially. However, the resources to execute and maintain these tests are not growing exponentially. Moreover, the tests might contain a lot of duplication. Hence, it is not feasible and beneficial to execute all the tests at all times. Currently, at ASML, the tests to execute are selected manually. Manual test selection becomes unscalable and inaccurate when the number of tests grows. Hence, this requires a solution that can enable efficient test selection.

In this project, we develop a framework that can be used to automatically select the tests to run within a given amount of test time. Tests are selected based on their coverage. The framework allows the user to define the notion of coverage that is used to select the tests.

It is beneficial for ASML to do test selection using the proposed framework (as opposed to doing it manually) because it based on a uniform, formalized definition of coverage (as opposed to a tester's experience and domain knowledge). The benefit of having a uniform and formalized definition of coverage is that it can be reviewed and agreed upon a priori by internal and external stakeholders. Previously this was not the case. Therefore, the overhead of the proposed solution does not outweigh its benefits.

Moreover, the solution is scalable: the test selection currently can work with 100,000 tests, and this limit can be increased further. It is also generic enough to be used for other domains.

Table of Contents

Contents

List of Figures

List of Tables

1.Introduction

The semiconductor fabrication process is getting more and more complex. The industry is moving towards a so-called 'holistic' manufacturing approach, in which traditionally different manufacturing steps (e.g., IC design, lithography, metrology) are combined in one complex, tightly integrated process [1]. ASML makes multiple systems that are needed in this process.

With an increased number of systems and inter-systems interactions, the number of tests is also increasing. The rate at which the number of tests is increasing is causing a test explosion. Here, we refer to the test explosion as a phenomenon where the number of tests is increasing exponentially, but the resources to handle the tests are not increasing equally. For any industry dealing with complex systems, there can be three primary reasons for an exponential increase in tests [2], all of which apply to ASML:

1. **The pace at which the technology is changing:** The technology is changing exponentially. This can easily be inferred from Moore's Law[3], which states that: "The number of transistors in a dense integrated circuit doubles every two years."
2. **The devices and services are ever more inter-connected:** With technological advancements, the new devices and the services need an interface with each other so that the systems can work together. These interconnections need to be tested well to ensure the systems are working correctly with each other.
3. **The need to release software updates more often:** For a certain domain, the pace of updating or releasing the software is quite high. This can be due to new changes coming up rapidly, or to stay ahead in the market, one needs to add new features and updates. With new features and updates, the software needs to be well tested for all the changes.

There is no direct relation between the (increasing) number of tests and test efficiency. We hypothesize that exponential growth in the number of tests has a *negative* impact on test efficiency because the resources to execute these tests cannot grow exponentially. As a result, we have to make a test selection.

Currently, the test selection is carried out manually. However, manual test selection is neither scalable nor accurate, because it is based on the experience and domain knowledge of the tester. Moreover, there is no uniform definition of coverage to base the test selection. As a result, the outcome of the test selection process is ambiguous.

The goal of this project is to develop a framework that can be used to define a notion of coverage and automatically select tests based on their coverage, such that their combined coverage is as high as possible. We can not yet quantify 'high' here, but we expect that our solution leads to a coverage that is at least as high as the current one (manual test selection).

By defining the notion of coverage a priori, it can be reviewed and agreed upon by internal and external stakeholders (e.g., those who request the tests to be executed). This also allows uniformity of the definition of coverage over time.

1.1 Outline

The remainder of this report is structured as follows. In Chapter 2, we briefly discuss the domain of ASML and the VirtualFab group for which this project is carried out. In Chapter 3, we discuss the problem in more detail and define the scope of the project. Chapter 4 gives an overview of the different stakeholders and their system requirements and design criteria. This is followed by Chapter 5, which gives an insight into the high-level design and architecture of the proposed system. Chapter 6 discusses the system design in detail. Design decisions and technological choices are explained in this chapter. Chapter 7 consists of the implementation and validation of the proposed system based on the system design mentioned in previous chapter. This chapter also consists of an analysis of the results of this project. Chapter 8 provides conclusion of the project. Chapter 9 discusses the future work, extensions, and recommendations that can be carried out in future. Chapter 10 briefly describes project management plan and project risks. Chapter 11 is retrospective of the project, reflecting upon the entire project experience and revisiting the design criteria.

2.Domain Analysis

To understand the problem further, it is essential to understand the primary work domain of ASML. This section gives a brief introduction to the lifecycle of creating semiconductor wafers using ASML systems. Furthermore, we explain the role of the Virtual Fab group, in which this project is carried out.

ASML is the world-leading manufacturer of *lithography systems* for the semiconductor industry. Lithography is an essential step in the process of creating integrated circuits or microchips. Lithography systems or *scanners* are complex machines that print fine structures on silicon *wafers*. These wafers are eventually diced and packaged as microchips. ASML's scanners are called *TWINSCAN*.

The error margins for lithography are getting smaller and smaller, to a point where data is needed from other steps of the process to be able to improve the performance of the scanner. Most importantly, it is required to *measure* the quality wafer at different steps in the process. For this purpose, ASML has developed a system called *YieldStar*.

The measurements of the wafer that are performed on YieldStar are used to calibrate or *correct* the scanner for the next batch of wafers. ASML has developed several different solutions for calculating such corrections. These solutions are called *applications* that are deployed on a system called the *Litho Computing Platform* (LCP).

This cycle of measuring wafers and calculating corrections for the next lithography step is called the *holistic control and monitoring loops* within ASML. An example of a holistic control loop can be seen in Figure 1.

Figure 1 Holistic control loop

In the past, these holistic control and monitoring loops were integrated for the first time at the customer's fabrication plant or *fab*. As a consequence, the *interoperability* between ASML systems was left untested until deployment at the customer site. As a result, some issues with interoperability were found too late.

As a response, the *Virtual Fab* group was introduced to tests the holistic control and monitoring loops in-house. Virtual Fab does both the test development and test execution for the holistic test cases. Test execution is done before the ASML systems are integrated at customer fab to ensure the integrated systems work as intended at customer fab. The testing is also carried out in case of any change or update in the ASML systems.

Formal data & control interfaces are extremely important in testing interoperability between systems because ASML systems and the customer fab interact via formal data interfaces & control interfaces.

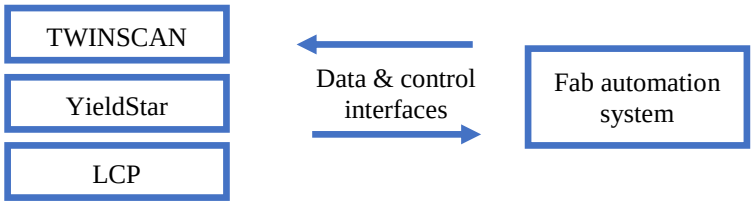

Figure 2 Data & control interfaces

3.Problem Analysis

Testing holistic control and monitoring loops involve testing a combination of systems. The number of possible combinations of systems grows exponentially because each system and product has different versions and configurations. As a result, the number of tests is also growing exponentially.

It is not feasible to run all the tests because it is not possible to exponentially increase the resources to execute all the tests. Currently, as a solution to this problem , the Virtual Fab group manually selects the tests to be executed instead of executing all the tests. However, manual test selection is neither a scalable nor an accurate solution to the problem, because it relies on the experience and domain knowledge of the tester (which may vary).

The underlying problem is that ASML has no quantitative way of establishing the quality of the tests that are executed. This is the first problem that we will address. Once such a notion of quality is established, we want to use it to select tests. However, currently, there's no systematic way of doing this. This is the second problem that we will address.

Problem 1 There is no way to quantify the quality of the test that are executed.

3.1 Quantify the quality of tests that are executed

Problem 2 There is no systematic way of selecting tests based on their quality.

One way of quantifying the quality of a test is by measuring it is coverage. Coverage can refer to a number of things, but in our case, high coverage means the test was able to check features that the test was intended to. This section presents our definition of coverage.

In order to define the coverage, the application domain to be taken into consideration, i.e., it is essential to base the notion of coverage on the purpose of testing. For example, if the goal of testing is to check the code, then the notion of coverage, in this case, can be code coverage. The notion of coverage should be such that it is expressible by the user and quantifiable in nature. In order to define coverage for this project, it is essential to base the notion of coverage on domain knowledge available within ASML For the Virtual Fab group, for example, it is important to test formal data interfaces and control interfaces, and therefore their notion of coverage should revolve around that.

There are different sources that capture the domain of tests performed by the VirtualFab group.

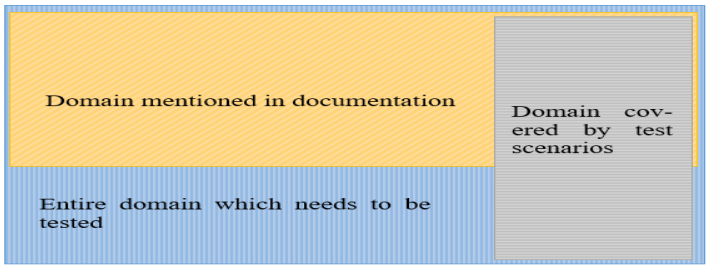

Figure 3 Test domain

Figure 3 shows that the various sources capture the domain in a different capacity. The blue area refers to the actual domain that needs to be tested. The yellow area shows the domain as captured in the documentation. It can be seen that the domain mentioned in the documentation[3] does not capture the entire domain. The grey area represents the domain that is actually covered by existing tests. The existing tests are not able to cover the whole domain. There does not exist any documentation or formal knowledge repository that captures the entire domain shown in blue. This leaves two choices to base the notion of coverage on. The first is to rely on the test case to formulate the notion of coverage or, second, rely on the documentation to formulate the notion of coverage. The notion of coverage is based on the available documentation [4] as it is a more reliable source as compared to the actual tests. The coverage of each test is calculated based on this notion of coverage. However, in order to check if a test has covered particular features mentioned in the notion of coverage, the test logs are checked. Looking into test logs confirm if a test has covered a feature or not because test logs reflect the actual behavior of a test. This leads to define coverage as follows:

Coverage: If a feature is tested and hence observed from the test log, that marks the coverage of that feature.

3.2 Selecting tests based on their quality

Currently, there is no uniform quality measure to base the test selection on. As a result, the outcome of the test selection process is ambiguous. If test selection is made based on the quality of tests, higher test efficiency can be achieved. Section 3.1 mentions that the quality of tests is quantified by the coverage of the tests. Hence, to increase the test efficiency, test selection should be made based on the coverage of tests. This leads to the definition of efficient testing as follows:

Efficiency: Given the available time for test execution, select a test set that achieves high coverage.

3.3 Scope

The following is in scope for this project:

1. A solution to represent the notion of coverage.
2. Calculate the coverage and store it for future use.
3. A solution to parse the test logs.
4. An algorithm that for a given amount of available time, selects tests that achieve high coverage.

4.Stakeholder Analysis and System Requirements

4.1 Stakeholder Analysis

This project will be used by the VirtualFab team at ASML. The stakeholders' group can be primarily divided into two groups, organizational and technical.

Table 1 Organizational stakeholders

Name	Role	Task
Paul Van Gorp	Manager, Virtual Fab Group, ASML	Ensure that the project brings the added value to the Virtual Fab group and thereby to ASML. • Reviewing the final project report and presentation.
Yanja Dajsuren	Program Director, PDEng - Software Technology	Ensure that the project meets the quality requirements for PDEng-Software Technology
Tim Willemse	Project supervisor – TU/e	• Supervising the project • Monitoring the progress of the project and checking if it meets the PDEng standards. • Evaluating the progress and providing regular advice and feedback to the trainee. • Reviewing the final project report and presentation.

Table 2 Technical stakeholders

Name	Role	Task
Roger Lahaye	Project supervisor from the Virtual Fab group, ASML.	• Providing relevant information to make the problem statement clear. • Providing direction for the project. • Supervising the project. • Reviewing the final project report and presentation.
Rick Smetsers	Project mentor from the Virtual Fab group, ASML.	• Involving in the project as a mentor • Imparting domain knowledge. • Providing direction for the project. • Reviewing the final project report and presentation.
Trupti Manik	PDEng- Software Technology trainee, TU/e.	• Coordinate, design, and develop the project. • Ensure that the project meets TU/e and ASML standards.

4.2 System Requirement

System requirements were gathered from the ASML stakeholders. In order to gather the requirements, meetings, and brainstorming sessions were conducted. Based on these meetings and brainstorming, the two main business requirements(BR) for this project are as follows:

1. **BR1 :** There should be a solution for measuring the test coverage.
2. **BR2 :** There should be a solution for the test selection.

4.2.1. Functional and non-functional requirements for BR1

BR1 : There should be a solution for measuring the test coverage.

Table 3 Functional and non-functional requirements for BR1

Requirement ID	Stakeholder Role	Functional/Non-Functional	Requirement
R1	Model Creator	Functional	The user should be able to create their own notion of coverage.
R2	Model Creator	Functional	The user should be able to change/edit the existing notion of coverage.
R3	Tester	Functional	The user should be able to see the coverage decomposition of the test, i.e., the user should be able to see what coverage is composed of.
R4	Tester	Functional	The user should be able to see the coverage value of the test.

NFR1	-	Non-Functional	The notion of coverage can be made using domain-specific keywords. (Ease of use)
NFR2	-	Non-Functional	The solution for defining and measuring the coverage should be adaptable to changes in the definition of coverage. (Adaptability)
NFR3	-	Non-Functional	The solution for measuring coverage should be scalable to an increasing number of tests. (Scalability)
NFR4	-	Non-Functional	The solution for calculating coverage should be reusable to find coverage for different types of tests. (Reusability)
NFR5	-	Non-Functional	The solution for defining and measuring coverage should be modular in nature. (Modularity)
NFR6		Non-Functional	The test coverage module should be independent of the Test selection module.(Loose coupling)

The functional and non-functional requirements are further divided into tasks as follows:

4.2.1.1 Tasks for achieving the requirements

Table 4 Tasks for functional and non-functional requirements

Requirement ID	Task ID	Task
R1, R2	T1	Create a Domain Specific Language (DSL), which can be used to describe and modify any notion of coverage.
NFR1,NFR2 ,NFR3	T2	Create coverage models using DSL.
R3	T3	Generate code to display the decomposition of the coverage
R4, NFR6	T4	Parse the test logs
	T5	Provide coverage values to the generated code of coverage models
	T6	Enable coverage models to calculate coverage based on data from test log parse.
	T7	Store the coverage values in the database
	T8	Retrieve the coverage values from the database to display to the user.
NFR4, NFR5	T9	Enable coverage models to reference other coverage models.

	T10	Create coverage models such that each model is one independent model based on documentation[4]

4.2.2 Functional and non-functional requirements for BR2

BR2 : There should be a solution for the test selection.

Table 5 Functional and non-functional requirements for BR2

Requirement ID	Stakeholder Role	Functional/Non-Functional	Requirement
R5	Tester, manager	Functional	The user should be able to input the maximum time available to conduct tests
R6	Tester, manager	Functional	The user should be able to see the set of selected tests.
R7	Tester, Manager	Functional	The user should be able to see the time taken to select the test set.
R8	Tester, Manager	Functional	The time required to select tests should be non-exponential in nature.
NFR7	-	Non-Functional	The test selection should be in a way so as to achieve coverage that is at least as high as that for manual test selection.
NFR8	-	Non-Functional	The solution for the test selection should be able to handle the increasing number of tests. (Scalability)
NFR9	-	Non-Functional	The solution for calculating coverage should be adaptable to the change in the notion(definition) of the coverage (Adaptability)
NFR10	-	Non-Functional	The test selection module should be independent of the test coverage module.(Loose coupling)

The functional and non-functional requirements are further divided into tasks as follows:

4.2.2.1 Tasks for achieving the requirements

Table 6 Tasks for functional and non-functional requirements

Requirement ID	Task ID	Task
R5	T11	GUI to allow inputting the maximum time available to execute tests.
R6,NFR7, NFR10	T12	Implement the test selection algorithm.
	T13	Feed the coverage values from the coverage database to the test selection algorithm.
	T14	GUI to display the results of the test selection algorithm.
R7, NFR8,NFR9 ,NFR10	T15	GUI to display the time taken to execute the test selection algorithm.
	T16	Enable test selection algorithm to be independent of the notion of coverage by making algorithms take input as any kind of coverage expressed in bitstream from the coverage database.
R8	T17	Implement a test selection algorithm using dynamic programming to ensure that execution time is non-exponential.

4.2.3 Design criteria

This section describes the design criteria for the proposed system. These criteria are revisited in chapter 11 to verify how well the proposed solution satisfies the intended design criteria.

Loose coupling: The proposed solution is mainly divided into two modules, defining and calculating coverage and test selection. Loose coupling here means that both the modules should be independent of changes in another module. Both of these modules should be such that these modules can be used independently of each other.

Ease of use: The proposed solution should be such that the domain experts(model creators) can use the solution with their domain knowledge, and the testers should be able to use the solution with their current skill set with Python. For this to work, it is essential that the solution exhibits both abstract levels for the domain experts and also allows testers to work on lower abstraction without getting involved in higher abstraction. So by separation of concerns, the solution can have ease of use for all the users.

Scalability: The solution is proposed so as to deal with the problem of exponentially increasing tests which cannot be manually selected to be executed. This makes it essential to have a solution such that it is easily scalable to the increasing number of tests.

5.System Architecture

The software architecture of a system gives direction to the design, the realization, and the evolution of the software in its environment. The proposed system has two main use cases:

The proposed solution is mainly intended to give a solution for:

1. Defining and calculating coverage.
2. Selecting tests.

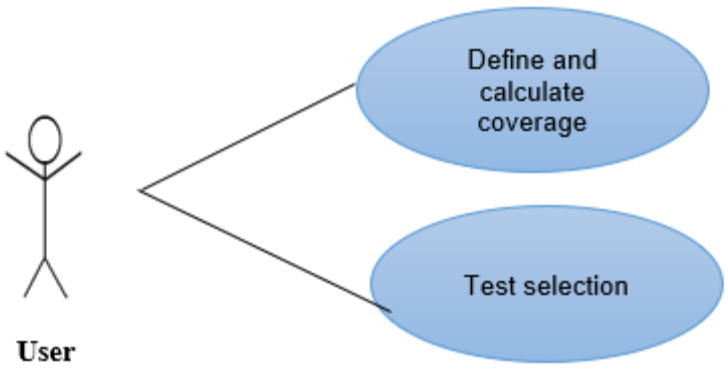

Figure 4 Primary use case of the proposed solution

The high-level use case in figure 4 is according to business requirements BR1 and BR2, as mentioned in chapter 4. So the proposed system has primarily two modules, module for defining and calculating coverage and module for test selection.

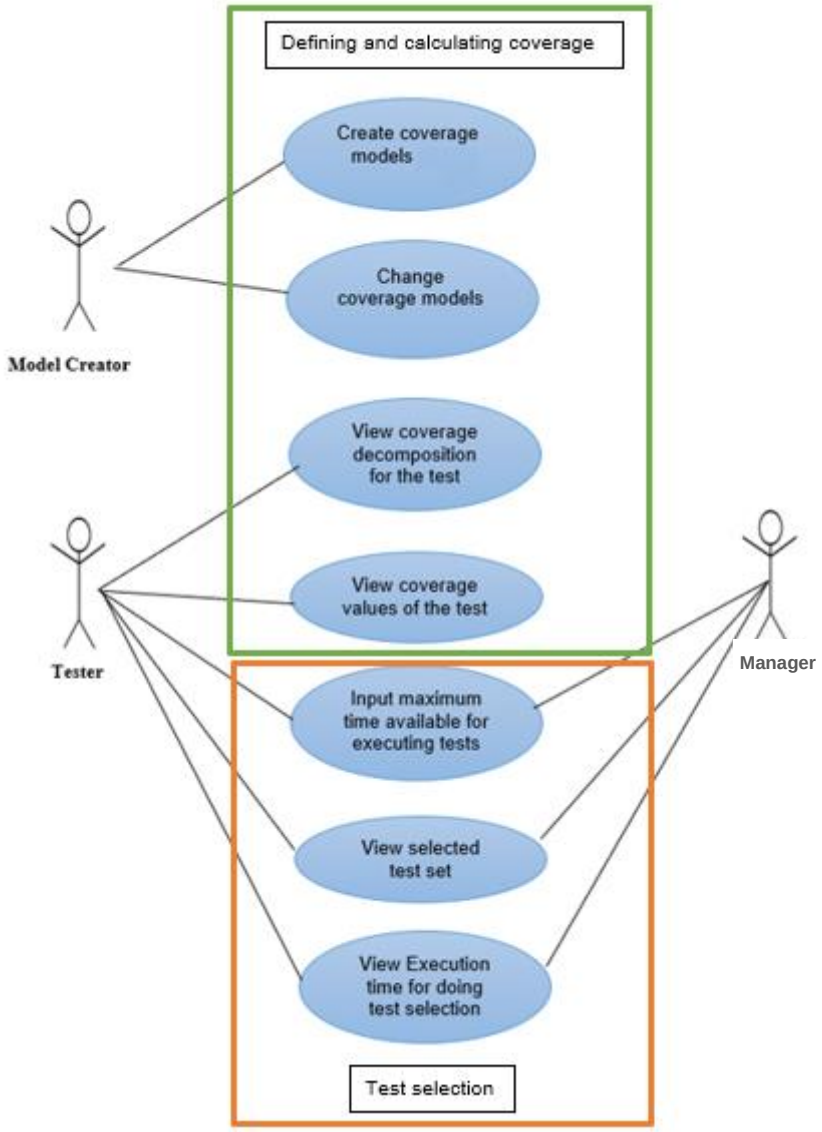

Figure 5 A use case of the proposed solution

Figure 5 shows the typical use case of the proposed system. The user of the proposed solution can have the role of a model creator, tester, and manager. Each of the user roles can perform different tasks as follows:

Model creator:

The model creator is a user with the domain knowledge who intends to create the notion of coverage, which can be later on used by testers to calculate coverage.

- **Defining and calculating coverage.**
  1. Create a coverage model. This intends to satisfy requirements R1, NFR1, and NFR2 as mentioned in table 3
  2. Change coverage model. This intends to satisfy requirements R2 and NFR3, as mentioned in table 3.

Tester :
Tester is a user who intends to use the coverage models created by the domain expert or model creator and calculate the coverage of the test.

- **Defining and calculating coverage.**
    1. View coverage decomposition for the test. This intends to satisfy the requirements R3 as mentioned in table 3
    2. The view coverage value of the test. This intends to satisfy the requirement R4 as mentioned in table 3

- **Test selection:**
    1. Input maximum time available for executing tests. This intends to satisfy the requirement R5, as mentioned in table 5.
    2. View selected test set. This intends to satisfy the requirement R6 as mentioned in table 5
    3. View execution time for making test selection. This intends to satisfy the requirement R7 as mentioned in table 5

Manager:

The manager should be able to see the time taken to select the tests and the selected test set. The manager is a user who wants to look if the performance of the test selection algorithm, in terms of execution time, is acceptable or not. This intends to satisfy the requirement R7 and R8.

- **Test selection:**
    1. Input maximum time available for executing tests. This intends to satisfy the requirement R5, as mentioned in table 5.
    2. View selected test set. This intends to satisfy the requirement R6 as mentioned in table 5
    3. View execution time for making test selection. This intends to satisfy the requirement R7 as mentioned in table 5

The system architecture is created for two modules, namely, measuring coverage and test selection. The tasks associated with each module is represented in the use case diagram in figure 4.
To accomplish the tasks mentioned in the use case, the following architecture is designed, keeping in mind the non-functional requirements. The decisions pertaining to the system architecture are as follows:

Architecture decision 1:

"The two modules of the proposed solution, namely, defining and calculating coverage, and test selection, are loosely coupled. "

Rationales:
1. The modules should be loosely coupled so that these modules can operate independently as a system on their own.
2.The change in one module will not affect the other module. This makes it easy to be able to use the test selection algorithm with a different notion of coverage.
3. This independence of modules is achieved by connecting two modules using a database. Hence, as long as coverage data is provided in the bitstream, the test selection algorithm can operate independently of the notion of coverage.

Requirement: This architecture decision is taken to satisfy non-functional requirements NFR6, as mentioned in table 3 and NFR10, as mentioned in table 5.

Architecture decision 2:

"The design of the proposed solution should not be dependent on the domain. However, the design should enable the user to capture the domain."

Rationales:
1. The proposed solution should be applicable to the different testing environments where coverage and test selection are intended. This makes it important to design a solution that is independent of the domain of testing.
2. The proposed solution, however, should enable users capturing their domain of testing in order to define coverage.

Requirement: This architecture decision is taken to satisfy non-functional requirements NFR1, NFR2, NFR4, as mentioned in table 3 and NFR 9, as mentioned in table 5.

Based on the design decision 1 and 2, the system architecture for the proposed system is as follows:

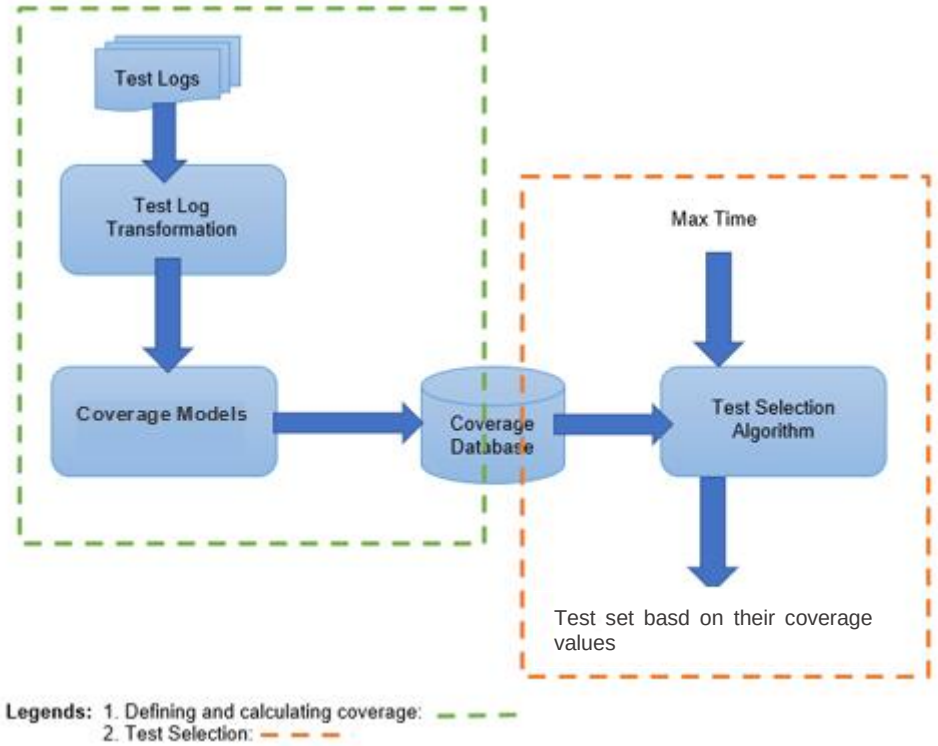

Figure 6 System architecture

The following are the steps that need to be followed in order to use the system with the above-mentioned architecture.

1. Parse the data of test logs.
2. Create coverage models
3. Provide parsed test log data to the coverage models. The coverage models have a data filter that filters only required data from the parse test logs and propagates the filtered data to its child node.
4. Store the value of coverage calculated by models in the coverage Database.
5. Provide the coverage Database to the test selection algorithm.

6.System Design

System design focuses on giving details about the lower-level design, which is based on the architecture mentioned in chapter 5. This chapter briefly explains the system design for two main business requirements mentioned in chapter 4. In order to propose the system design for the proposed solution, certain design decisions were taken. Section 6.1 and section 6.2 describe the system design, design decisions, rationale behind the design decisions, and requirements, which lead to taking a particular design decision for modules: 1. Defining and calculating coverage 2. Test selection, respectively.

6.1 Defining and calculating coverage.

For defining and calculating coverage following design, decisions were taken:

Design decision 1:

"Follow the model-driven approach to define coverage."

Rationales:

- **i.** Model-driven is a methodology that focuses on creating domain models. The domain models are conceptual models pertaining to specific problems. The domain models intend to represent domain knowledge through abstract representations instead of using computing concepts. This means one can use the model-driven approach to represent the domain using domain-specific keywords, which are high abstraction as compared to using a general-purpose programming language, which is a lower abstraction.
- **ii.** Models allow expressing a solution in the level of abstraction that adheres to the problem domain. This makes it easy for domain experts to understand, create, validate, or modify the models.
- **iii.** Models make the solution concise, uniform, and easily adaptable to changes since one does not need to write the code in traditional programming languages; one can generate code for the models.
- **iv.** Models allow the validation of concepts at the domain level itself.
- **v.** Models capture domain knowledge. Hence, this makes it possible to preserve the domain knowledge and reuse it or even extend it.

Requirement: This design decision is taken to satisfy the NFR1 and NFR2 as mentioned in table3

Discarded Alternative:

- i. **Programming Language:** By using a programming language, users can describe the notion of coverage. The coverage tools mentioned in [5] are implemented using a general-purpose programming language instead of the model-driven approach. However, the problem with this approach is users need to be very well acquainted with the programming languages to create, use, and update this solution. Also, it would be a bit difficult to make the solution abstract enough to be adapted to changing or different notions of coverage. Hence a layer of abstraction is needed on top of any programming language to make it more understandable for the domain experts. Therefore this approach was discarded, and instead, a more abstract approach of creating models is chosen.

Design decision 2:

"*Coverage shall be represented in a decomposable tree-like structure.*"
Rationales:

i. The notion of coverage should be decomposable from a higher abstraction level to more concrete observable parameters. By doing so, it can be confirmed that a certain feature has been covered or tested by the test, and also this can help to understand what is coverage composed of.

Requirement: This design decision is taken to satisfy the requirement R3, as mentioned in table 3
Discarded alternative:

i. **Flat representation of coverage:** In this approach, the hierarchical nature of the coverage model would not be preserved. This would make it difficult to understand the decomposition of coverage. It would be difficult for users to understand what the mentioned coverage is composed of.

Based on design decisions 1 and 2, the notion of coverage is described using a domain-specific language to create the coverage models in a decomposable tree-like structure. These models, in turn, can reference other models which finally creates a directed acyclic graph-like structure. The tree-like structure has different types of nodes in a tree. Since it is a tree and eventually a directed acyclic graph, there should not be any cycles in the graph. The nodes in meta-model of DSL are described below:

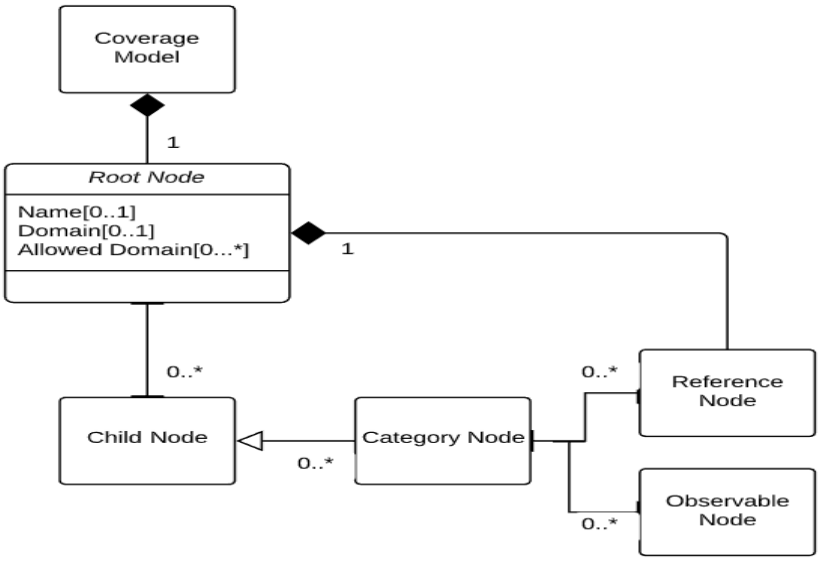

Figure 7 Class diagram of DSL meta model

Figure 7 shows the meta-model of the DSL. The coverage model is composed of exactly one root node. The root node can have many child nodes. The child nodes are of type category node. A category node can further have one or more observable nodes or reference node. The observable nodes are the leaf nodes; hence, they do not have any further child nodes. The reference nodes are reference to another tree that has root node at its top. The detailed relevance of each type of node is explained below:

1. **Root node:** The root node represents the most abstract level of the coverage. This node shall have the child nodes.
    a. **Constraint:** There can be only one root node per coverage tree.
2. **Child node:** The child node describes coverage in finer granularity. There can be different types of child nodes. The child node has a category which is called a category node.
3. **Category node:** A category node is used to describe the category of the coverage a particular model is representing.
    a. **Constraint:** None**.**
4. **Observable node:** The observable nodes are the leaf nodes of the tree. These nodes hold the actual observable event for the coverage tree.
    a. **Constraint:** An observable node cannot have a child node. Also, an observable node cannot be a root node. An observable node will always have a category node as its parent.
5. **Reference node:** Reference nodes are used to reference other coverage models. The reference nodes are root nodes of other cover models that mean the reference node refers to another tree with its own root node, and this tree would, in turn, have category nodes, reference nodes, observable node, and reference nodes. Reference nodes are always intermediate nodes in the whole directed acyclic graph.
    a. **Constraint:** The reference node cannot reference itself.

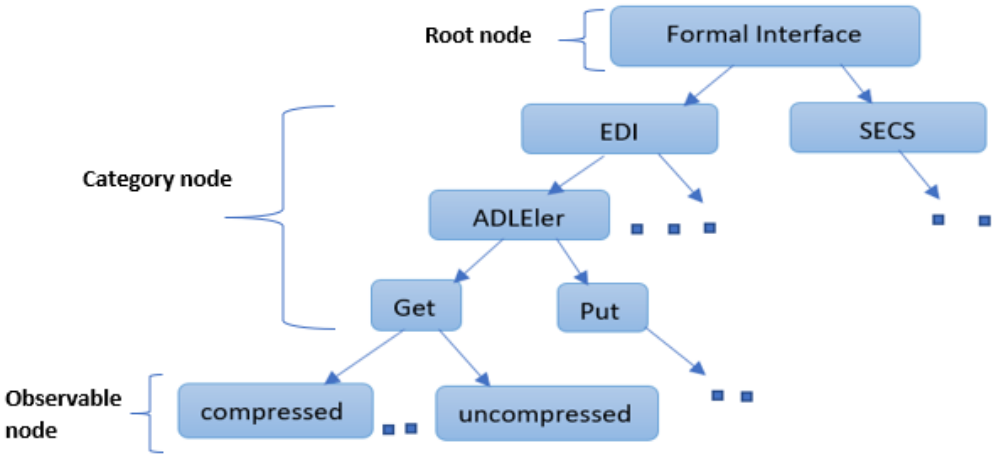

Figure 8 Tree structure of the coverage model

The tree structure in figure 8 is an example of the coverage tree with the root node, category nodes, and observable nodes. One tree can reference multiple other trees. The example of the same is shown below in figure 9.

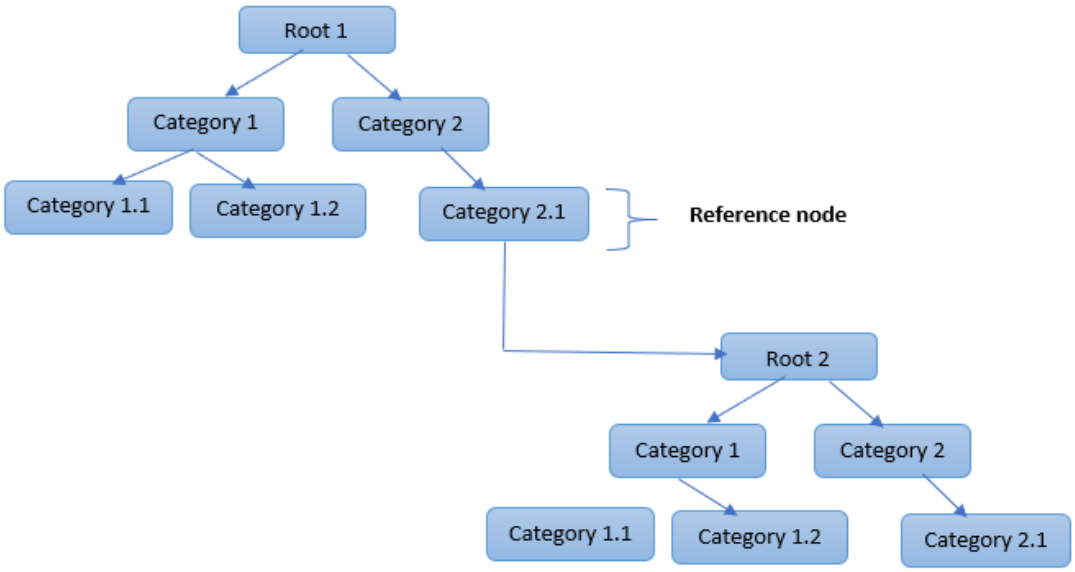

Figure 9 The tree structure of the coverage model referencing another coverage model.

The coverage models shown in Figures 8 and 9 are like placeholders for the concept of coverage. The coverage models are used to calculate the coverage of each test. These placeholders need to be instantiated for the coverage models to be able to calculate the coverage. Where instantiating the coverage models for the test means the values for each test needs to be filled in the coverage models to that the final coverage value for the tests can be computed. In order to do so, the test logs are looked into to check if a feature is covered by the test or not. Accordingly, the value is filled in the coverage model to finally compute the coverage values of the test. Following design decisions were taken for the same:

Design decision 3:
"The coverage models shall have a data filtering capability."
Rationale:

i. The root node gets the data from the parsed test logs. Each node should be able to filter out the required data from the preprocessed data and pass on the data that is required by its child node. This results in a faster solution, as unnecessary data is not propagated further for data processing.

Requirement: This design decision is taken to satisfy the requirement R4, as mentioned in table 3

Design decision 4:
"The value of coverage is calculated bottom up to the root of the coverage tree."
Rationale:

i. Each observable node populated with appropriate data should confirm the observance of a feature. With the observance of a feature, a value shall be assigned to the observable node. The presence of the feature is marked by bit 1, and absence is marked by bit 0.Each observable node should pass this value to its parent node. This passing of value should be done up until the root node. The final bitstream value at the root node is the coverage of the coverage model.

Requirement: This design decision is taken to satisfy the requirement R4, as mentioned in table 3

Based on design decision 3 and 4, the data is provided to the coverage model as follows:
The test logs contain the logs of every event that has been observed by the intended test case. Hence, the test logs are checked to see if there is any data corresponding to the node in the coverage tree. If the data is present in the test log, it is provided to the model. Each node of the model has a data filter. This data filter enables filtering out the data for the child node. Only the data needed for the child node is passed to the child node. The rest of the data is discarded. This passing of data occurs until the leaf node of the tree. If the data is left with the leaf node, it is concluded that the particular feature has been observed from test logs. If the feature has been observed from the test log, a binary 1 is assigned to the observable node in the coverage model. If the feature is not observed, then a binary 0 is assigned to the node in the coverage model. Finally, the coverage tree has all its leaf nodes with either value 1 or 0, depending on whether a feature has been covered or not. The coverage of the root node is represented as the bitstream of 0's and 1's. Figure 8 shows the process of instantiating coverage model with values from the test log.

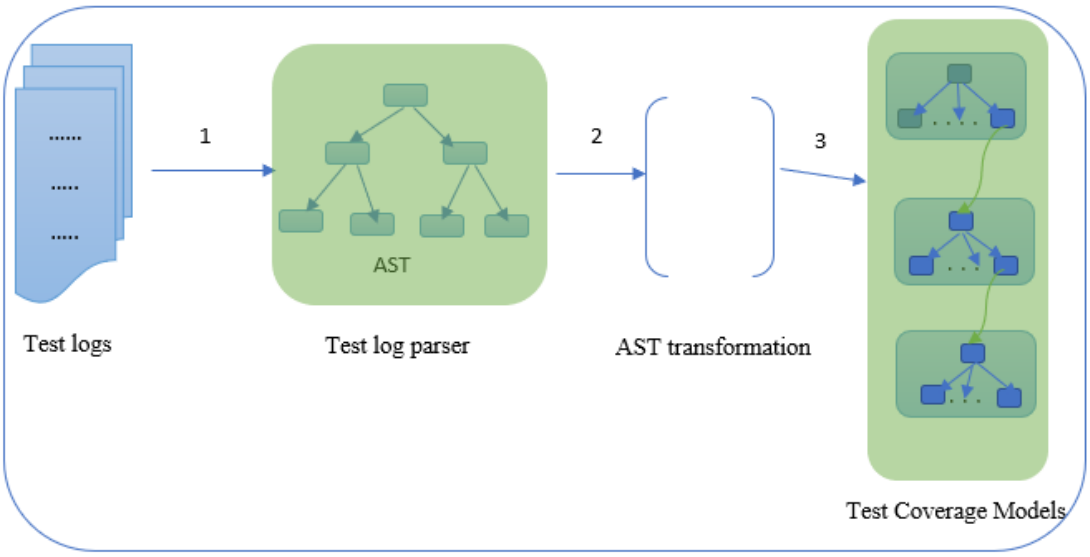

Figure 10 Process of instantiating the test coverage models with the data from the test logs

Figure 10 shows the process of instantiating the test coverage models with the values from the test logs. Instantiating the coverage models with values from the test log mainly involves 3 steps, which are mentioned below.

1. The test log parser reads the test logs and tokenizes the plain text statements from the log. The log parser generates the abstract syntax tree (AST) from the tokenized test logs. This AST can be crawled to read the data.

2. The transformation should be performed on the abstract syntax tree to group the nodes of the AST. Each group consisting of the nodes of AST corresponds to one observable in the test coverage model. This grouping of the nodes of AST into logical chunks requires intensive domain knowledge. Due to the lack of intensive domain knowledge, AST transformation is out of scope for this project. However, it can be developed as part of future work. As for now, the AST transformation does not exist, and hence, manual data is provided to the test coverage models.
3. After the AST transformation, the data is provided to the coverage model. Each node of the coverage model has the filtering capability. Hence, each node takes data that is needed by its child nodes and discards the rest of the data. This filtering and passing of data are done until the leaf node that is an observable node of the coverage tree. If the data for the observable node exists, then the observable node in the coverage model is assigned the value binary 1. If the data is missing, then the value binary 0 is assigned to the node in the coverage model. These bit strings for the observable nodes helps to determine the coverage of the root node of each coverage tree. A test can be represented by the coverage models. The coverage bit streams of each individual coverage model, when propagated to all the way up to the root node, gives the coverage of the test in terms of the bitstreams. These coverage values are stored in the database in the form of bit strings.

Figure 10 shows an example of the above-mentioned process of assigning values from the test log to the coverage model. Let us say we have a scenario that we want to observe. This observable scenario is an observable node in the coverage tree SECS communication.

Scenario: "The host Attempts to establish communication via SECS."
Goal: To parse the test logs and check if a given scenario has been covered.

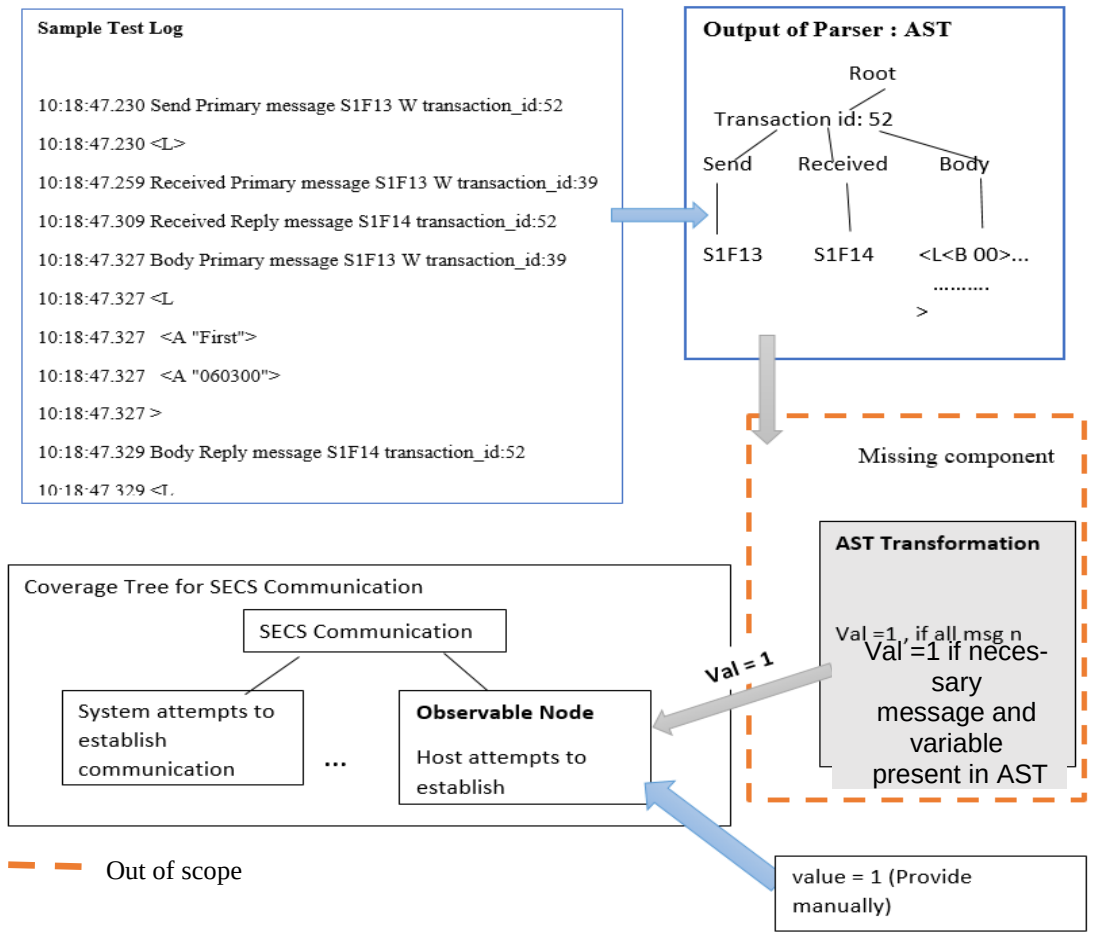

Figure 11 Process for assigning values to the coverage model.

In figure 11, the test log is taken as an input for the test log parser. The test log parser is converted to an AST by the parser. This AST is create based on grammar rules in the parser. The AST should be fed to the AST transformation module. However, the implementation of the AST transformation module is out of scope for this project. Hence, currently, the user has to manually enter the coverage values to the observable node. If the feature is covered, then the observable node has to be assigned value 1 otherwise value 0.

6.2 Test Selection Algorithm

The test selection algorithm takes three inputs: maximum time available to run the tests, coverage values of the tests, and test time. With this input, the test selection algorithm does computations to give the selected test set as an output. The test selection is made to achieve high coverage in the given time. The high-level design of the test selection process can be seen in figure 12.

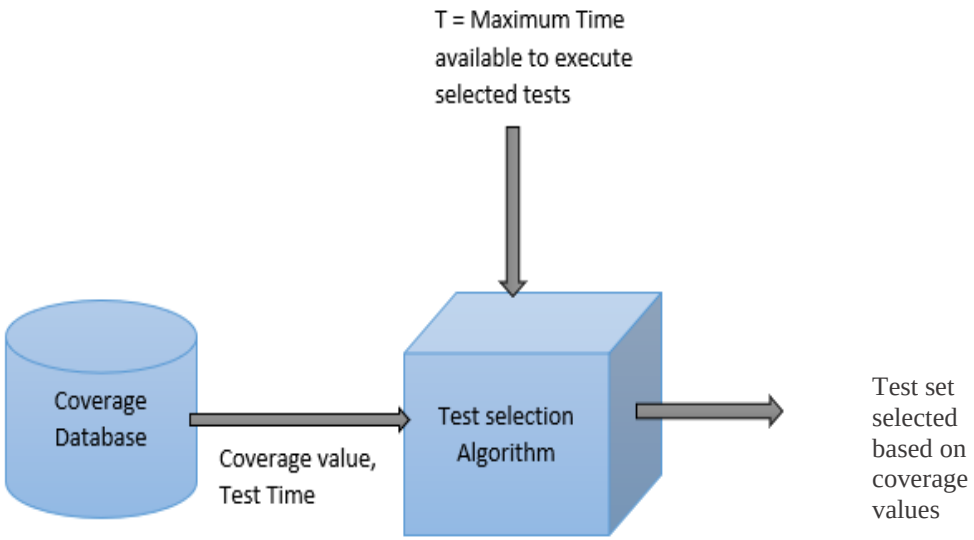

Figure 12 High-level design of Test selection

The coverage values are calculated by the coverage models and are stored in the database. The test selection algorithm reads the coverage values of tests from the coverage database. The algorithm selected here is a modified 0-1 Knapsack with dynamic programming. The following are the design decisions for the test selection algorithm.

Design decision 5:

"Use a modified 0-1 Knapsack algorithm with dynamic programming for test selection."

Rationale:

- **i.** The time and resource consumption of executing an actual test is quite high. Hence, it is essential to select the tests as accurate as possible. A random or manual selection of tests might not guarantee high coverage. The chosen algorithm is such that the time and resource required to execute the algorithm for test selection is less than the time and resources required to execute the actual tests.
- **ii.** 0-1 Knapsack with dynamic programming is an exact algorithm. Exact algorithms are the ones that give optimal results. However, since the algorithm is modified to fit the current domain, it does not guarantee optimal results, but it gives good enough results for test selection.

Discarded alternative:

- i. **Machine learning:** The machine learning algorithm learns over a period of time from the inference and training data. However, machine learning requires to train the algorithm, and in this case, having a consistent training data is a non-trivial problem. Hence this option was ignored.
- **ii.** **Evolutionary algorithm:** The evolutionary algorithm uses the concepts from biological evolution like reproduction, mutation, recombination, and selection, to find the solution. An evolutionary algorithm requires multiple runs in order to reach the solution. This was not required in this case. Therefore this option was discarded.
- **iii.** **N-SAT:** n-SAT algorithm, also known as Boolean satisfiability, is used to determine if there exists a solution that satisfies a given Boolean formula. The 'n' -SAT refers to the number of literals in the Boolean formula. For solving the test selection problem, n-SAT would have to be adapted to the domain. In this case, choosing an encoding for the n-SAT to be adapted to the problem domain is a non-trivial task. This makes it challenging to use n-SAT for test selection. Also, the time complexity of Boolean satisfiability is O(that makes it difficult to scale up for an increasing number of tests.
- **iv.** **Variants of 0-1 Knapsack:** The 0-1 Knapsack algorithm has different variants that are explained below:
  1. **Bounded Knapsack:** This algorithm allows selecting the same item multiple times, but has an upper bound on how many times the same item can be selected. However, this conflicts with the purpose of test selection. The purpose of the test selection algorithm is to select a test only if it adds to the coverage, so selecting the same test again does not add anything to the coverage. Hence, this alternative was discarded.
  2. **Unbounded Knapsack:** Unlike bounded Knapsack, there is no upper limit on how many times the same test can be selected. This renders it useless to be used for test selection as it allows selecting the same test as many times without any addition to the coverage

Based on decision 5, a modified 0-1 Knapsack with dynamic programming is selected to solve the combinatorial problem here. The section below describes how to solve the 0-1 knapsack problem with dynamic programming, which is later modified to fit the domain of this project. In this context, we call the modified algorithm as the modified 0-1 Knapsack algorithm.

6.2.1. Knapsack Algorithm

The *Knapsack algorithm* [6] is used to find the solution for a combinatorial problem given a constraint. For a given a set of items, each item having a weight and a value, Knapsack determines which items need to be selected such that the total weight of the selected items is not larger than the total capacity of the knapsack and the total value of the selected items is maximum. 0-1 knapsack allows either selecting the item fully or not selecting it at all. Given the values below:

> n = number of items
> w = weight of an item
> v = value of an item expressed as an integer
> W = the capacity of the knapsack.

The objective of Knapsack is to select the items to put inside knapsack such that the total combined value of selected items in knapsack is maximized, and the combined weights of the selected items do not exceed *W*.

The Knapsack algorithm can be solved using *dynamic programming*. There are two main attributes that a problem must have for dynamic programming to be applicable: *optimal substructure* and *overlapping sub-problems*. We cite the definition of these attributes from [7]:

> "*Optimal substructure* means that the solution to a given optimization problem can be acquired by the combination of optimal solutions to its sub-problems.
> (…)
> *Overlapping sub-problems* means that the space of sub-problems must be small; that is, any recursive algorithm solving the problem should solve the same sub-problems over and over, rather than generating new sub-problems."

Intuitively, the overlapping sub-problem is the decision problem of picking up an item or not. This is based on the following aspects:

1. Does the item fit in the knapsack?
2. Is it beneficial to pick up the item?

The overlapping sub-problem has been proven to be an optimal substructure for the Knapsack algorithm [8].

Dynamic programming allows not only finding the optimal solution, but it also enables finding an optimal solution faster as compared to checking all possible solutions brute force. Also, as dynamic programming solves the subproblem only once and stores it for later use, this makes it faster. However, the memory consumption in case of dynamic programming can be higher as compared to the greedy approach as it stores the solution of the subproblem to reuse later while that does not happen in case of the greedy approach.

On the next page, we explain the 0-1 Knapsack algorithm with dynamic programming. This explanation and the pseudo code is based on [9].

Input:

- Values (stored in array v)
- Weights (stored in array w)
- Number of distinct items (n)
- Knapsack capacity (W)

Output: the combination of items with the maximum combined value that fits in knapsack capacity.

Arrays v and w are assumed to store all relevant values starting at index 1.

Matrix m[i,j] stores the maximum value we can get while selecting among first i items such that the weight does not exceed W.

The weight of each item is a positive number (for all i, w[i] > 0).

```
1.  for j from 0 to W do:
2.      m[0, j] := 0
3.      for i from 1 to n do:
4.          for j from 0 to W do:
```

If the weight of the current item is larger than the available capacity of the knapsack, then don't pick the item. Hence, the value in knapsack would be same as before:

```
5.              if w[i] > j then:
6.                  m[i, j] := m[i-1, j]
```

If the weight of the item is less than or equal to the available capacity of the knapsack, check using the max function if selecting or rejecting an item gives higher value in the knapsack. If selecting an item gives higher value, then add the value of the current item to previous values in knapsack. Else, do not pick the item, and hence the value in knapsack remains same as before.

```
7.              else:
8.                  m[i, j] := max(m[i-1, j], m[i-1, j-w[i]] + v[i])
```

After the matrix m is filled, it is traversed to determine which items are selected for the given capacity. For this, we traverse the matrix from the bottom right to the top left. Array s holds the indices for the items that are selected.

```
9.  x := W
10. s := []
11. for i from n to 1 do:
```

If the current item has contributed to the value of the knapsack, we pick it up, and we subtract its weight from the remaining available capacity of the knapsack:

```
12.     if m[i, x] != m[i-1, x] then:
13.         s.append(i)
14.         x := x – w[i]
```

When we have evaluated all items, we return those that were selected:

```
15. return s
```

6.2.2. Modified algorithm

The 0-1 Knapsack is modified to fit in the domain of the project. In the context of this project, the items are tests. The weight for an item is the time is taken to execute the test. The value of the item is its coverage. The capacity of the knapsack is the time available for testing.

Coverage values are represented as a sequence of bits. Each bit represents a feature that is defined in the coverage model. The bit is set to 1 if the feature is covered (observed), and 0 otherwise.

This means that we have modified the original algorithm as follows:

a. The values in the cells of the matrix m are sequences of bits instead of integers (as discussed above).
b. As a pre-processing step, items are sorted in descending order the number of 1-bits of their value. The original idea behind this was that it would make the algorithm give optimal results. However, later, a counterexample was found that shows that this is not the case. This counterexample is presented in Section 6.2.4.
c. The + operator that is used to determine if selecting an item gives a higher value is replaced with the bitwise-OR operator |
d. The max(x, y) function that is used for selecting or rejecting an item is modified to count the number of 1-bits in x and y. The max function, therefore, returns the argument that has the highest number of 1 bit.
e. We terminate the selection loop when the final coverage has been achieved. This is the case when the bitwise-OR over the values of the selected tests (v[i] for i in s) is equal to the value in the bottom-right cell of the matrix (m[n, W]).

As a result this, the algorithm now looks like this:

```
1.  for j from 0 to W do:
2.      m[0, j] := 0
3.      for i from 1 to n do:
4.          for j from 0 to W do:
5.              if w[i] > j then:
6.                  m[i, j] := m[i-1, j]
7.              else:
8.                  m[i, j] := max(countSetBit(m[i-1, j]), countSetBit(m[i-1, j-w[i]] | v[i]))
9.   x := W
10.  s := []
11.  C :=[]
12.  for i from n to 1 do:
13.      if m[i, x] != m[i-1, x] then:
14.          s.append(i)
15.          x := x – w[i]
16.          C := C | C[i]
17.          if C == m[n,W]
18.              break
19.  return
```

where function countSetBit(x) returns the number of bits in x that are 1, and function bitwiseOR(x) returns the sequence of bits that is the bitwise-OR of values in array x.

6.2.3. Example

Input:

- v := [011, 100, 101, 110] (values/ test coverage bit sequences)
- w := [2, 3, 4, 5] (weights/ test durations)
- n := 4 (number of items/ tests)
- W := 5 (knapsack capacity/ available test time in hours)

Output: a selection of tests that can be run in 5 hours.

When we follow the algorithm, the values of the first row and first column of m are guaranteed to be 0:

i↓ j→	**0**	**1**	**2**	**3**	**4**	**5**
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

First of all, we sort the inputs based on coverage values in descending order.

v := [110, 011, 101, 100]
w :=[5, 2, 4, 3]

The first item i and capacity j for which the item can be selected, is i=1 and j=5, because the weight of item 1 is 5. The value for item 1 is 110. Here, 2 bits are set to 1, so selecting item 1 gives higher value:

	0	**1**	**2**	**3**	**4**	**5**
0	0	0	0	0	0	0
1	0	0	0	0	0	110
2	0					
3	0					
4	0					

The next interesting cell is m[2, 5]. The item (2) we are considering has value 011 and weight 2. When the current capacity (j) is 5, then this item fits in the knapsack. The equation for deciding to pick up this item is therefore:

m[i, j] := max(m[i-1, j], m[i-1, j-w[i]] | v[i])

- ➔ m[i-1, j] = m[1, 5] = 110
- ➔ m[i-1, j-w[i]] = m[1, 2] = 0
- ➔ v[i] = v[2] = 011
- ➔ m[2, 5] := max(110, 0 | 011) = 110 (here both choices have same number of 1's , this is an ambigious choice to make, in this case we pick up the previous item.)

	0	**1**	**2**	**3**	**4**	**5**
0	0	0	0	0	0	0
1	0	0	0	0	0	110
2	0	0	011	011	011	110
3	0					
4	0					

The completed matrix looks as follows, and therefore the final coverage is m[4,5] = 111.

	0	**1**	**2**	**3**	**4**	**5**
0	0	0	0	0	0	0
1	0	0	0	0	0	110
2	0	0	011	011	011	110
3	0	0	011	011	011	110
4	0	0	011	011	011	111

Once the matrix is filled in with all values, the selection of tests needs to be done. In order to do so, following steps needs to be followed:

	0	**1**	**2**	**3**	**4**	**5**
1	0	0	0	0	0	110
2	0	0	011	011	011	110
3	0	0	011	011	011	110
4	0	0	011	011	011	111

Now we determine which tests to select. We start from m[n, W] = m[i, x] = m[4, 5] and go up until the following condition holds: m[i, x] != m[i-1, x]. This is true for i=4 and x=5. Therefore, we pick up test 4 and subtract its weight from the current capacity: x = 5 – 3 = 2. The next item (2) is also picked up for the same reason. Hence the selected tests are 2 and 4. The coverage for these tests is 011 and 100, respectively. Therefore, the final coverage is 011 | 100 = 111.

6.2.4. Counterexample

At first, it seemed that sorting the inputs gives optimal results. However, the following counterexample suggests that it is not the case. For our modified algorithm, the optimal substructure property cannot be guaranteed, and hence the outcome of the algorithm might not be optimal.

In this section, we present a counterexample that shows that the algorithm is not optimal.

Input:

- v := [011110, 111000, 000111] (values/ test coverage bit sequences)
- w := [1, 2, 3] (weights/ test durations)
- n := 3 (number of items/ tests)
- W := 5 (knapsack capacity/ available test time in hours)

When running the algorithm as described for this data, we get the following matrix, and items 1 and 2 are selected. The combined value (coverage) for these items is 111110. However, if the algorithm were to give optimal results, then items 2 and 3 should have been selected, giving combined coverage as 111111.

i↓ j→	**0**	**1**	**2**	**3**	**4**	**5**
0	0	0	0	0	0	0
1	0	011110	011110	011110	011110	011110
2	0	011110	011110	111110	111110	111110
3	0	011110	011110	111110	111110	111110

The reason that the algorithm does not give the optimal solution is that first, the optimal result (111111 in this case) does not appear as a solution to any of the subproblem. Secondly, doing bitwise OR instead of the addition operation and counting number of set bits to determine maximum out of two numbers in line 8 of the algorithm alters the algorithm in a way that it no longer posses optimal substructure property and hence does not gives the optimal result.

6.3 *Example of an integrated solution*

Below is an example showing the working of the proposed system having two modules: Defining and calculating coverage, and test selection.

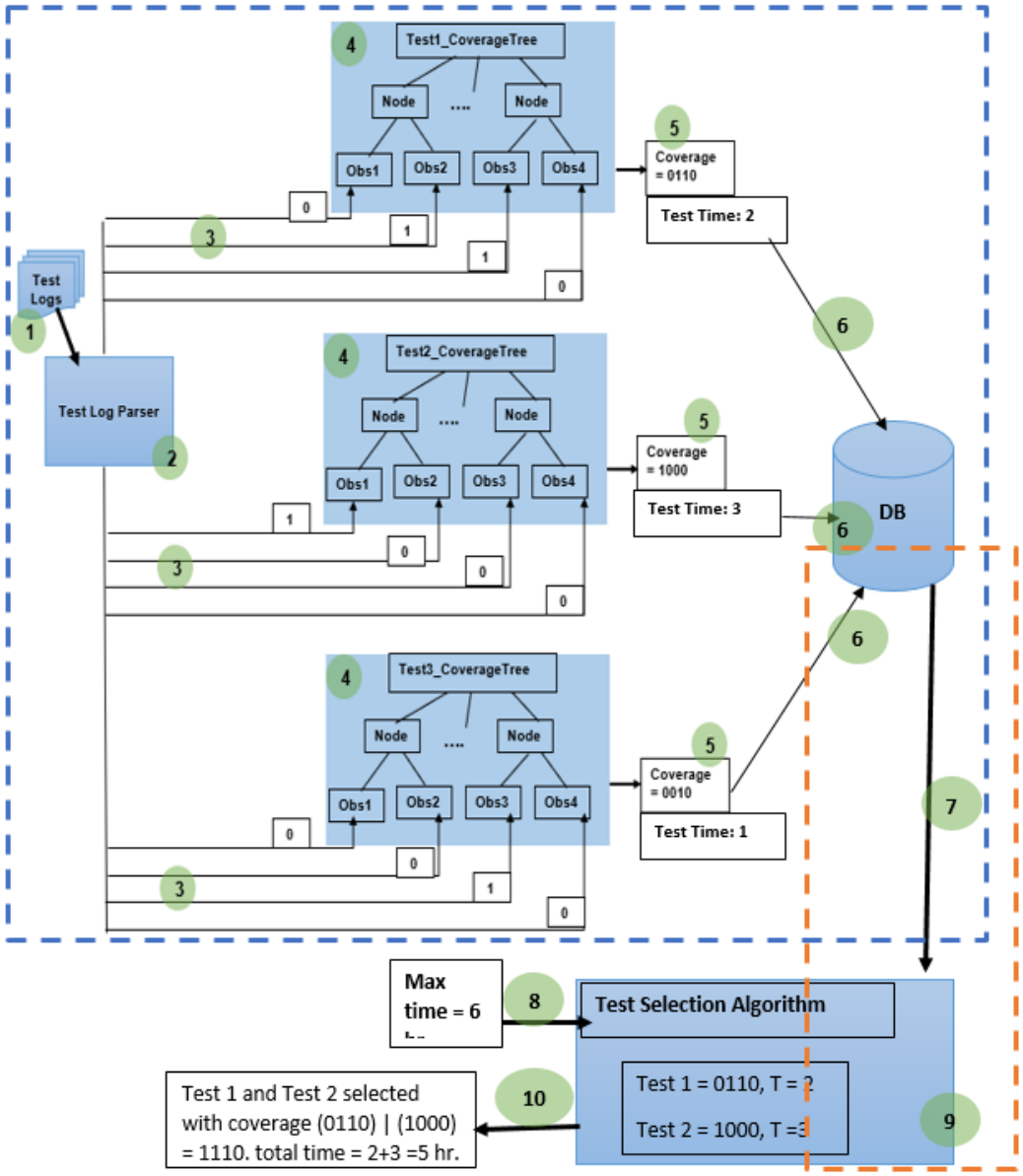

Figure 13 Example of working of the proposed solution

The example mentioned in Figure 13 has the following data:

Scenario: Find test set, given : max time to execute tests is 6 hours. The dataset consists of the test logs for Test 1, Test 2, and Test3.

Procedure:
Step 1: Input test logs to the test log parser.
Step 2: Test log parser generates data for the coverage models.
Step 3: Binary 1 or 0 is assigned to the observable nodes of the coverage trees depending on if a feature is observed or not in the test log.

Step 4: The coverage trees for tests assign the value to the observable nodes the test.
Step 5: Based on the assigned coverage values, the coverage model calculates the coverage for the test as follows:

Test 1: Coverage = 0110 Time: 2 hr.
Test 2: Coverage = 1010 Time: 3 hr.
Test 3: Coverage = 0010 Time: 1 hr.

Step 6: The coverage values with the time taken by the tests are stored in the coverage database.
Step 7: The test selection algorithm reads the coverage values from the coverage database.
Step 8: The test selection algorithm takes max time to execute tests from the user; in this case, 6 hr.
Step 9: The test selection algorithm operates on the coverage data and time taken by the tests. Here for the given data, Test 1 is selected as it covers maximum features (two out of four). Out of Test 2 and Test 3, Test 2 is selected as Test 2 covers a feature that Test 1 and Test 3 do not cover. The given time limit is 6 hours. After the selection of Test 1 and Test 2, we still have time left to choose Test 3. However, choosing Test 3 does not add to the coverage, and hence Test 3 is discarded.
Step 10: The result of test selection is displayed to the user.

6.4 Technological choices

To develop the project according to the architecture and the design choices, the tools and technologies were selected. This selection was made during the architecture phase. This section explains the choices for selecting tools and technologies to develop this project.

6.4.1 Defining and calculating coverage

For defining and calculating coverage , a model-based approach is used. The technology choice pertaining to the model-based approach is as follows:

Technological choice 1:

"JetBrains MPS 2018.3 is used for modeling DSL."

Rationale:

i. JetBrains MPS had a projection-based .
ii. MPS can be used to generate code that is not the only base language, which is Java-like but other programming languages as well. For this project, the code is generated in Python.

Discarded alternative:

i. **Xtext:** The first reason to discard Xtext was the VirtualFab group wanted to use MPS explicitly over Xtext in order to explore MPS. Xtext is a traditional parser-based approach that works with ordinary textual files. The main difference is that in MPS, you can edit the model directly, while in Xtext, one can edit the syntax, and the model is generated/parsed. Working directly on the model in the case of MPS makes it more powerful.

Technological choice 2:

"Python2.7 code is generated from the coverage models."

Rationale:

i. The generated code needs to be modified or further developed by the testers for the purpose of coverage calculation. Python 2.7 is the most preferred programming language for the VirtualFab group to work on. In order to use Python 2.7 as code generation language in MPS, the plaintextgen plugin needs to be installed in MPS.

Discarded alternative:

ii. **Code generation in Java:** MPS JetBrains, by default, generates code in Java-like language. However, the VirtualFab group is more acquainted with Python 2.7, and hence this alternative was discarded.

Technological choice 3:
"Use ANTLR 4.0 to parse test logs."
Rationale:

i. The test logs are used to check if a feature is tested by a test or not. However, the test logs are in plain text. This needs to extract and group the information for the test log, which can confirm the observance of the feature.
ii. ANTLR is a powerful parser generator. From a grammar, which is a formal language description, ANTLR generates a parser. This parser can automatically build the parse trees. ANTLR also enables creating the tree walkers, which can traverse the parse tree and access the nodes of the tree for further processing. Using ANTLR makes it easy to write a parser that can be adapted to changes merely by changing grammar rules.

Discarded alternative:

i. **Writing a parser from scratch using programming language:** Writing a parser from scratch using a programming language not only takes a lot of effort but is also less flexible to changes. For making new changes, the amount of effort required would be more as compared to using a framework like ANTLR

Technological choice 4:
"Use MongoDB to store coverage values of the tests."
Rationale:

iii. MongoDB is a NoSQL Database. This enables having a dynamic schema for unstructured data. This is particularly beneficial for this project because the schema of coverage data can change over a period.
iv. MongoDB is horizontally scalable, which helps reduce the workload and scale with ease. Horizontal scaling means scaling is done by adding more machines into the pool of resources. Dynamically scaling up is easier with horizontal scaling.

Discarded alternative:

i. **Using relational Database:** The scaling in a relational database is vertical. This means to scale up one needs to add more power in terms of CPU and RAM to an existing machine. This limits the scaling based on the capacity of a single machine. Also, there might be issues with downtime if one machine is down. Apart from this, relational databases need to define the schema early on, and making changes to the existing schema is not easier as compared to a non-relational database like MongoDB. Hence this choice was discarded.

6.4.2 Test selection Algorithm

The technological choice for developing the test selection algorithm is as follows:

Technological choice 5:
"The test selection algorithm is implemented using Python 2.7."

Rationale:

i. The stakeholders are more acquainted with Python 2.7. Hence, the algorithm is implemented using Python. So later, it becomes more convenient for stakeholders to modify and maintain the test selection algorithm with their current skill set.

7.Implementation and Validation

7.1 Introduction

The implementation of the project is based on system architecture and system design. There are two main parts in implementation: 1. The DSL and model, and Test Log Parser 2. Test Selection Algorithm. The below section describes the implementation details of these two parts.

7.2 Measuring coverage

7.2.1 Domain-Specific Language and Models

The purpose of domain-specific language (DSL) is to allow the users to describe their own notion of coverage using domain-specific words. This DSL is created using MPS JetBrains 2018.3. Dialects and domain-specific lingo help people to communicate specifically and effectively. MPS brings the same flexibility into the world of programming languages. Unlike traditional programming languages with strict syntax and semantics, MPS allows a language to be created, altered, or extended by its users[10]. The below figure shows the sample concept from DSL.

```
concept Tree extends     BaseConcept
             implements  INamedConcept
                         IMainClass

  instance can be root: true
  alias: tree
  short description: A tree describes a decomposition of a coverage concept.

  properties:
  << ... >>

  children:
  domain                : Domain[0..1]
  allowed_child_domains : Domain[0..n]
  root                  : AbstractNode[1]

  references:
  << ... >>
```

Figure 14 Tree concept from DSL

Above Figure. 14 shows the concept Tree from the DSL. The DSL has various concepts like Tree, which is the root node, Category Node, which allows users to allocate the category of coverage tree, Observable Node, which allows defining what needs to be observed in order to confirm coverage, and Reference Node which allows referring other Trees. Apart from these, two other concepts called domain and allowed_child_domain are also part of DSL. The domain allows the user to define the domain of a particular coverage tree, where examples of the domain can be TWINSCAN, YieldStar, etc. The field allowed_child_domain describes the domains of the other Trees that a Tree is allowed to reference.
Using the DSL, users can make Models. Models are actual instances created using the DSL. Below a sample model for SECS communication can be seen. This Tree can be referenced by Trees for TWINSCAN or YieldStar, which involves SECS Communication. The root node of the tree is Communication, which belongs to domain SECS and allowed domain is SECS. Allowed domain SECS determines that the Communication tree can have reference to other trees that also belong to the SECS domain.

```
tree Communication {

  domain :
    domain SECS
  allowed_child_domains :
    domain SECS
  root :
    category node Communication {

      children :
        category node system_attempts_comm {

          children :
            observable node obs_sys_attempts
        }
        category node host_attemps_comm {

          children :
            observable node obs_host_attempts
        }
        category node sys_receives_commack {

          children :
            observable node obs_sys_receives_commack
        }
        category node sys_sends_commack {

          children :
            observable node obs_sys_sends_commack
        }
        category node sys_sends_heartbeat {

          children :
            observable node obs_sys_sends_heartbeat
        }
    }
```

Figure 15 SECS Communication

In Figure 15, a user first needs to create the model using the DSL. Python code is generated for each model. This python code then reads the data from the Test Log via the Parser. But since the Abstract Syntax Tree(AST)transformation of the Parse Tree is missing, the generated code is not able to read the observables directly from the parser. Implementation of AST transformation is part of future work. In the absence of this transformation, testers currently need to manually enter the values of observable in the generated code. Once the user enters the values of observable in the generated code, the coverage values are calculated and stored in the MongoDB database.

7.2.2 Parser for Test Logs

The parser for the test logs is created using ANTLR 4. ANTLR (ANother Tool for Language Recognition) is a powerful parser generator for reading, processing, executing, or translating structured text or binary files. It's widely used to build languages, tools, and frameworks. From a grammar, ANTLR generates a parser that can build and walk parse trees[11] . Currently, the parser has some grammar rules which enable us to parse the test log to an AST. Below figure 16 shows the sample of AST generated for a sample test log.

```
10:18:47.230 Send Primary message S1F13 W transaction_id:52
10:18:47.230 <L>
10:18:47.259 Received Primary message S1F13 W transaction_id:39
10:18:47.309 Received Reply message S1F14 transaction_id:52
10:18:47.327 Body Primary message S1F13 W transaction_id:39
10:18:47.327 <L
10:18:47.327 . <A "First">
10:18:47.327 . <A "060300">
10:18:47.327 >
10:18:47.329 Body Reply message S1F14 transaction_id:52
10:18:47.329 <L
10:18:47.329 . <B 00>
10:18:47.329 . <L
10:18:47.329 . . <A "Second">
10:18:47.329 . . <A "060300">
10:18:47.329 . >
10:18:47.327 . <L
10:18:47.327 . . <A "Third">
10:18:47.327 . . <A "060300">
10:18:47.327 . >
10:18:47.329 >
```

Sample test log in Figure 16 is converted to AST in Figure 17 by the Test Log Parser

Figure 16 Sample test log

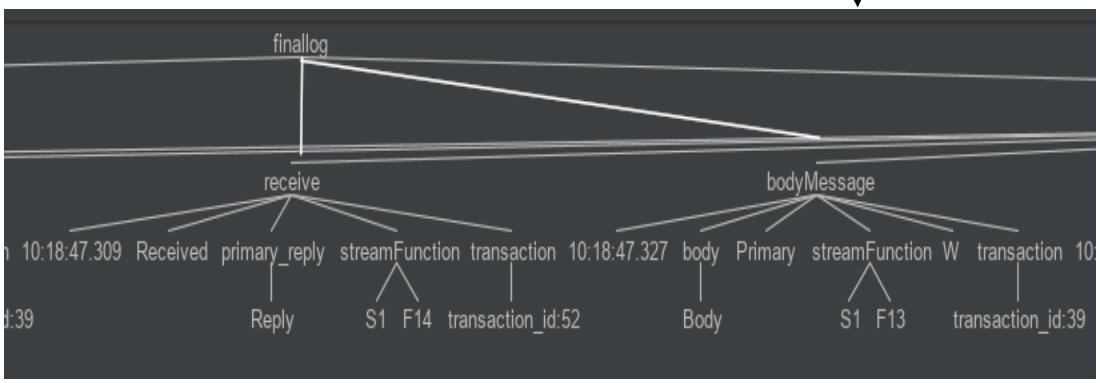

Figure 17 Sample Abstract Syntax Tree generated

The AST shown in Figure 17 can be traversed through the TreeVisitor created in ANTLR. The TreeVisitor allows us to traverse and extract information from the AST. This extracted information is useful to provide values for observable in the coverage model.

7.2.3 Test Coverage Database

A test coverage database is created using MongoDB 4.0.10. MongoDB had a collection which can be seen as a table in SQL. In place of records or rows in SQL, MongoDB has documents.

Test Coverage Database currently has 100,000 documents. Each document has the fields as shown in figure 18

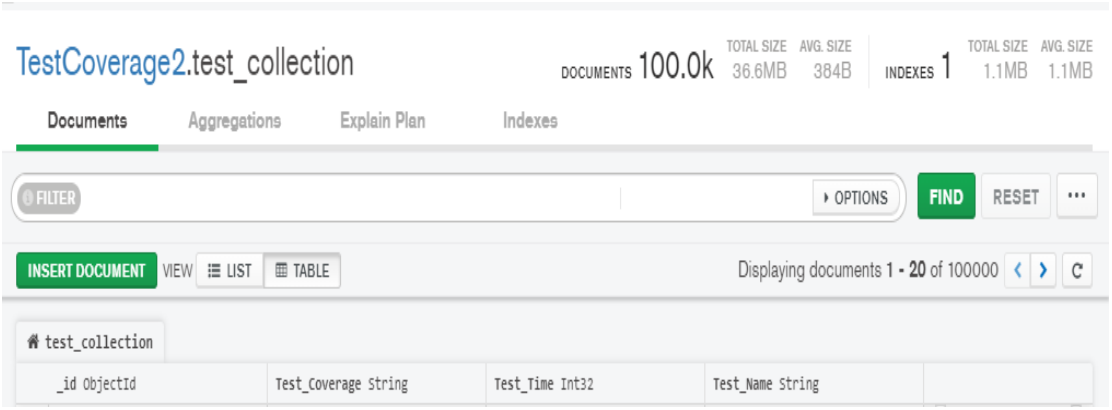

Figure 18 Structure of document in Test Coverage Database.

Figure 18 shows that each document has object_id, which is a default field in MongoDB. Test coverage has a datatype string. Test coverage is in a binary stream, and hence it should have Int64 as its datatype. However, Int64 allows having the number of observables of length, only 8 bytes. While storing the coverage data as a string makes it possible to store much larger bit-streams than 8 bytes. Currently, the bitstream with 100,000 bits is converted into a string and stored in a database which otherwise with Int64 could only have been 64 bits.

7.3 Test Selection Algorithm

For the test selection algorithm, a modified 0-1 Knapsack algorithm is implemented using dynamic programming. This algorithm is implemented using Python 2.7.The algorithm reads the coverage values in bitstream from the coverage database in MongoDB. Apart from reading coverage values, the algorithm needs one user input, and that is the maximum time available to run the test set.

Below experiments are conducted with 100,000 tests, each having 64 observables. The data for the same is generated randomly since there is no real data available. However, on the availability of real data, there shall be no changes in solution as the fabricated data is created to have a similar format as the real data.

Below Figure 19 shows the database created in MongoDB. Dataset for each test is stored in one document; hence, for 100,000 test data, 100,000 documents are created in the database. The total size of the database is 8.6mb, which indicates that storing the coverage data of tests in MongoDB is not heavy on memory.

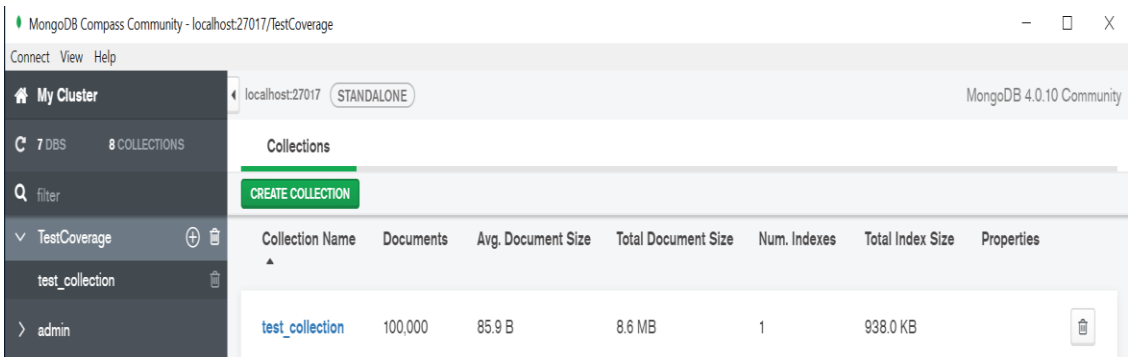

Figure 19 Test Coverage Database having 100,000 test coverage entries.

A detailed view of the sample data inside a coverage database can be seen in Figure 20. In MongoDB, each document is given a unique id which can be seen in the _id field in Figure 20.

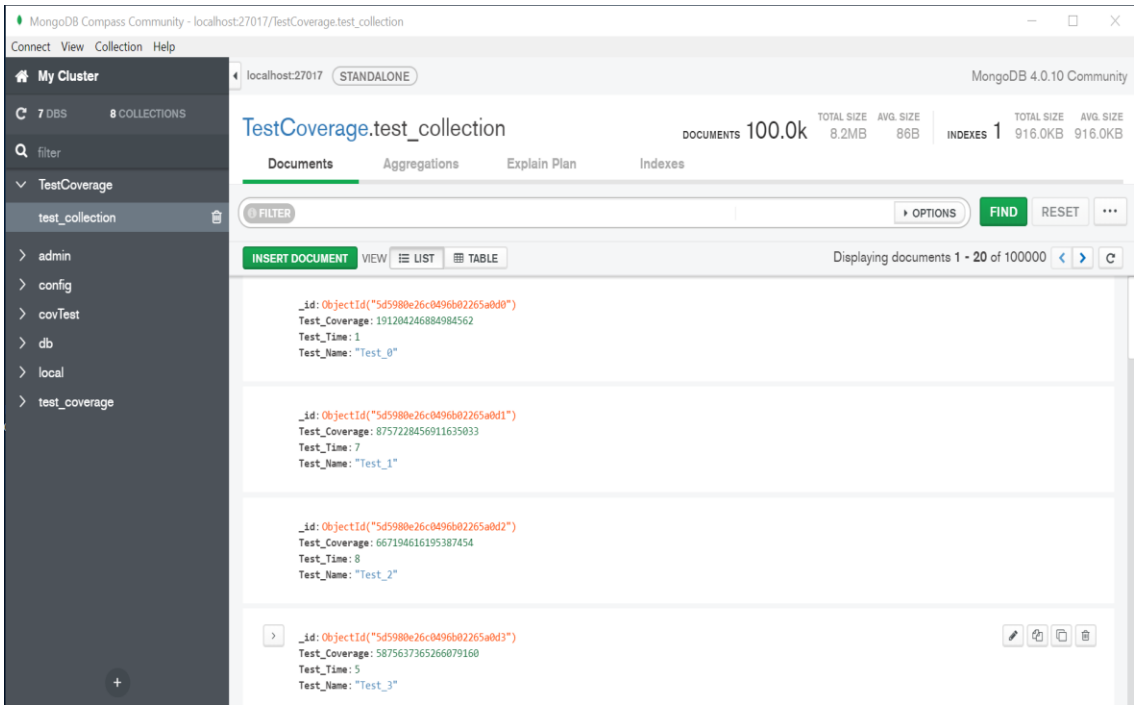

Figure 20 Detailed view of data in Coverage Database.

The Test Selection Algorithm reads data from the above-mentioned Coverage Database. Figure 21 shows the screen of the Test Selection Algorithm. The GUI is made in python using the TKinter package.

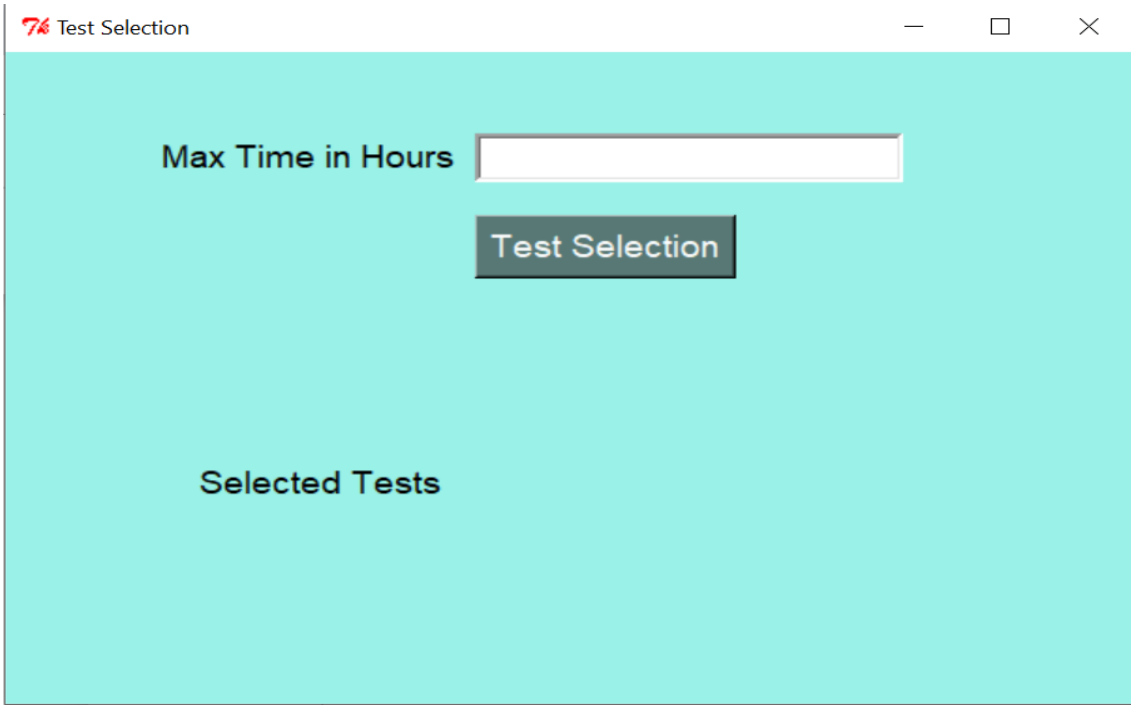

Figure 21 Test Selection Algorithm

Figure. 22, Figure. 23 , and Figure. 24 shows the output of the Test Selection Algorithm for 8 hours and 16 hours, respectively. The time taken to execute the algorithm with database operations can also be along with a selection of the tests.

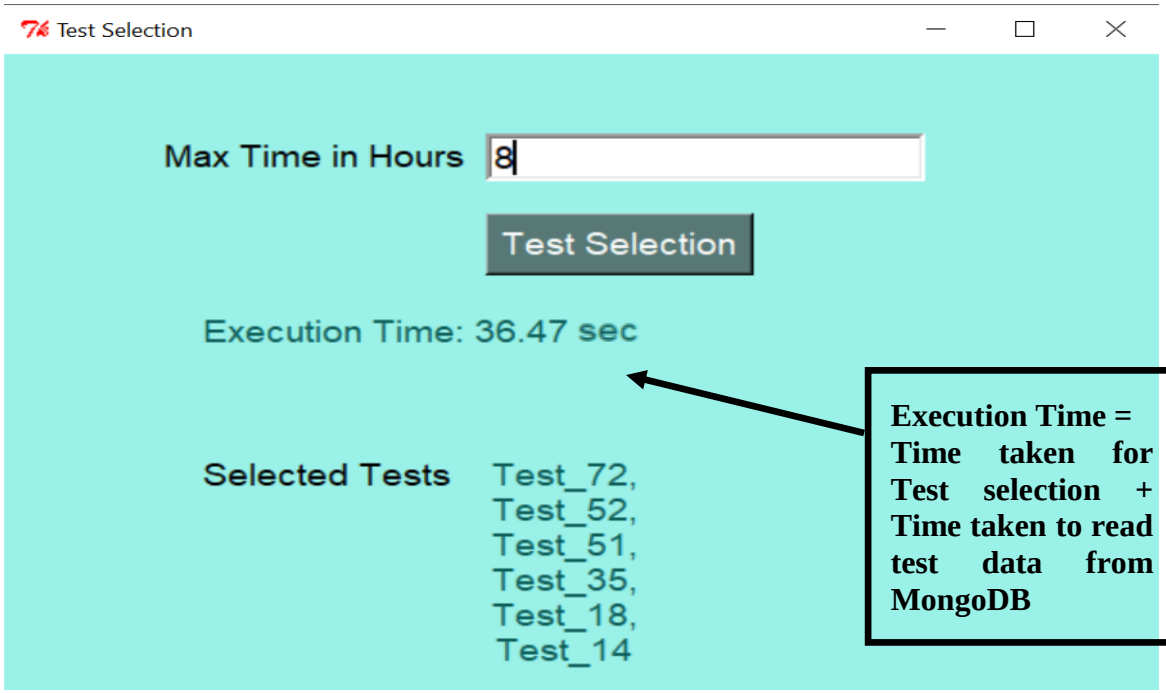

Figure 22 Test Selection Algorithm for 8 hours of total time to execute the tests.

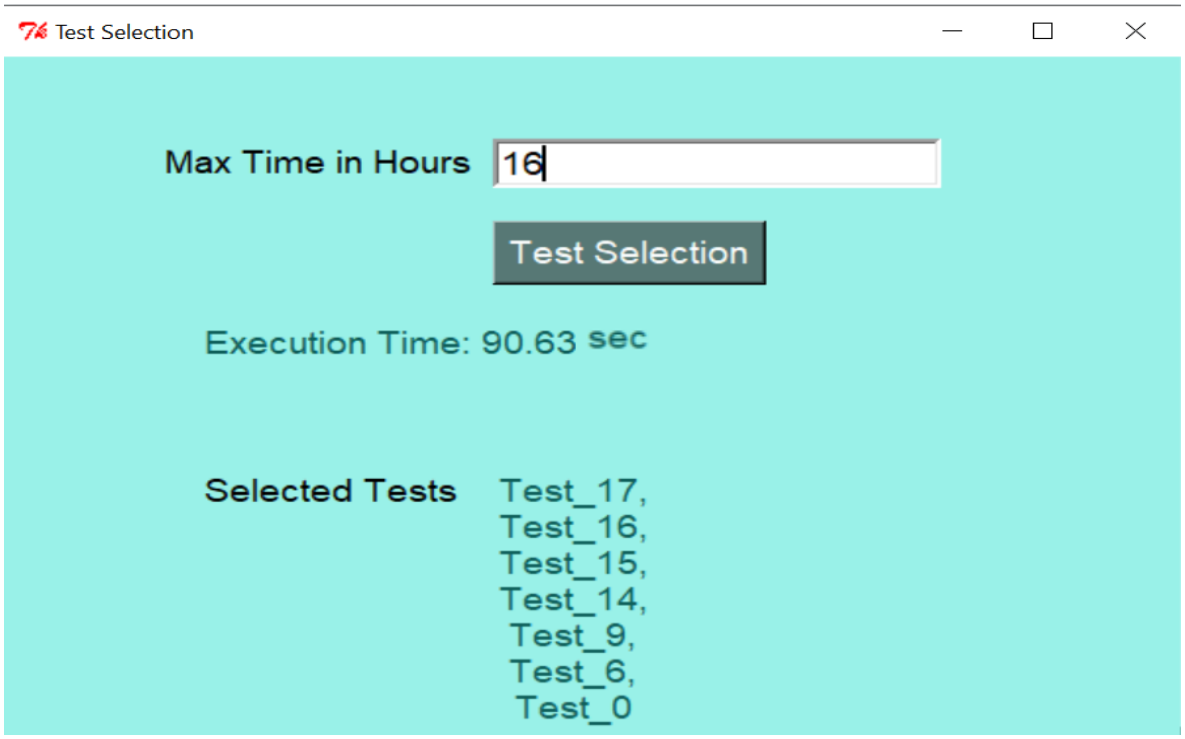

Figure 23 Test Selection Algorithm for 16 hours of total time to execute the tests

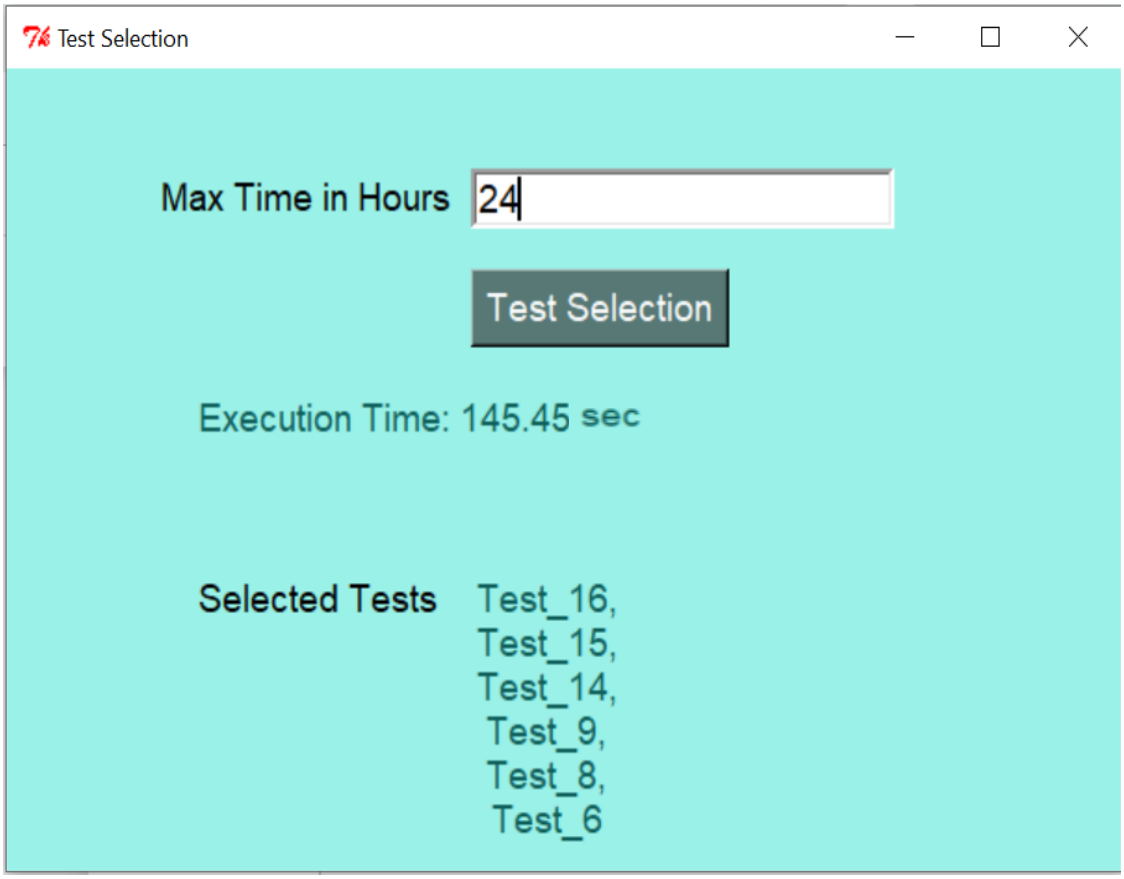

Figure 24 Test Selection Algorithm for 24 hours of total time to execute the tests.

Above Figure. 24 is a screenshot captured for the Test Selection Algorithm when a total of 24 hours are available to execute the tests. The above results show that out of 100,000 tests, 6 tests are selected based on their coverage values when available time is 24 hours. The algorithm took 145.45 seconds to display the result of test selection.

The following results show the execution time of the Test Selection Algorithm. For the sake of this experiment, the number of observables each test has is 1000.

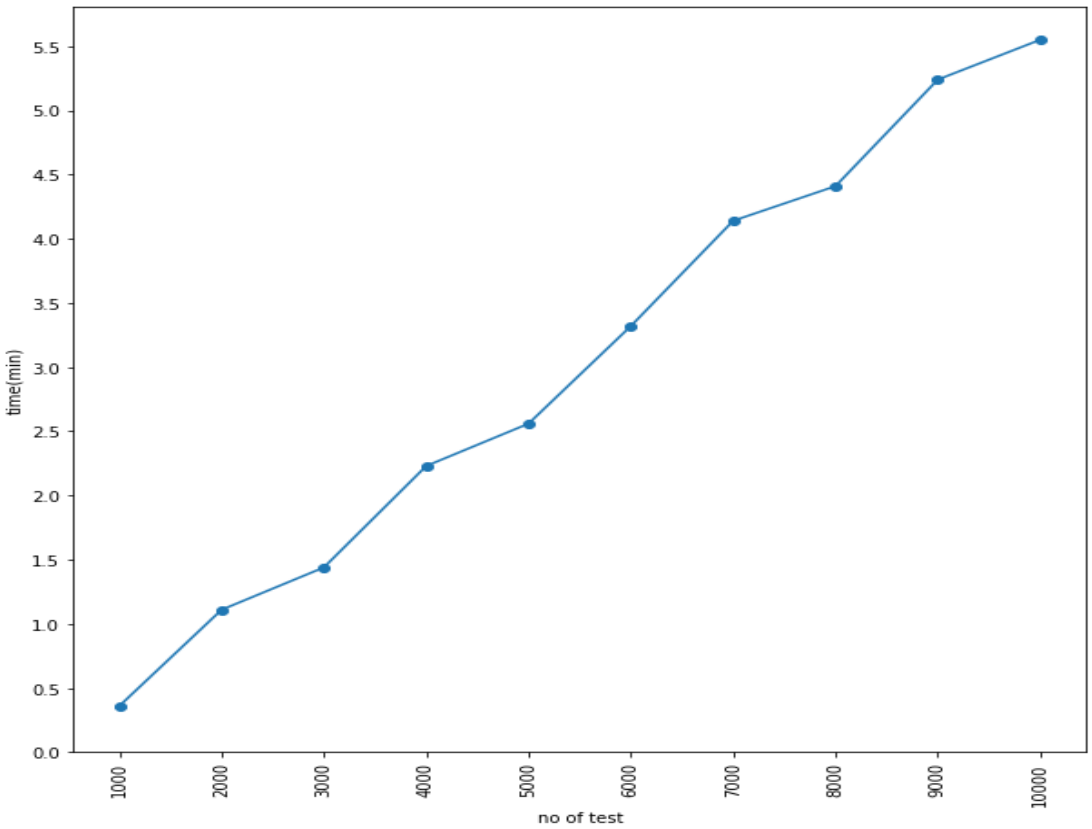

Figure 45 Graph of no of tests vs. execution time for Test Selection Algorithm for 10,000 tests and 1000 observables

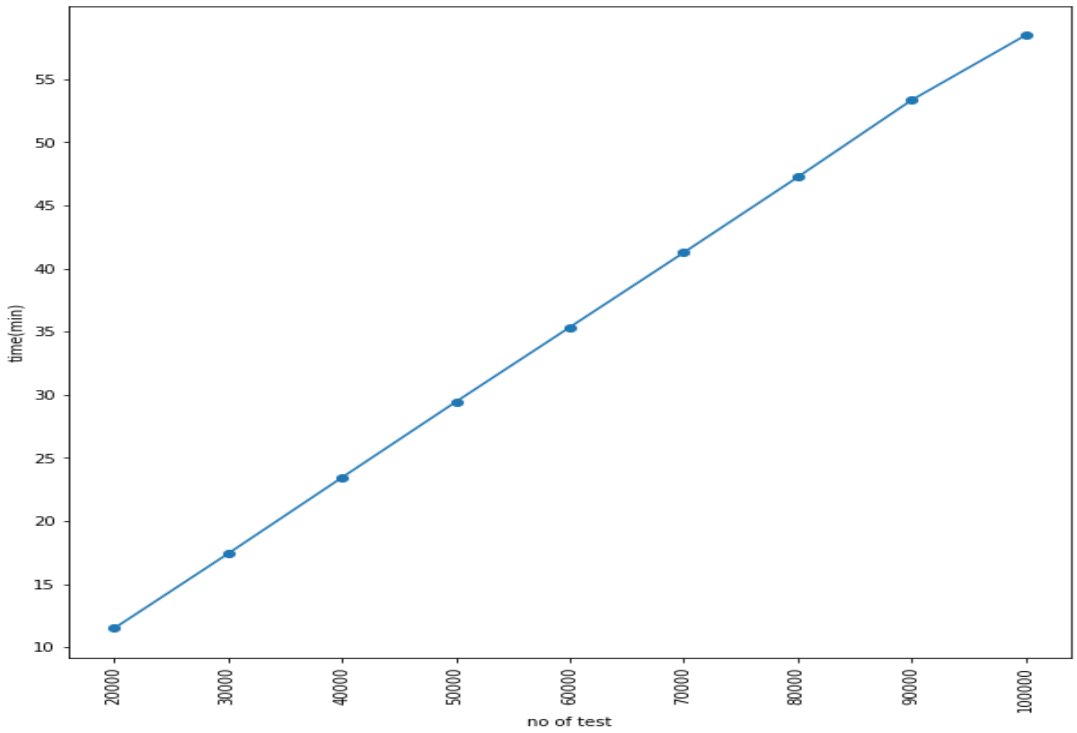

Figure 26 Graph of no of tests vs. execution time for Test Selection Algorithm for 100,000 tests and 1000 observables

Figure 25 and 26 shows the execution time of the Test Selection Algorithm. From the graphs, it can be seen that for100,000 tests, each test having 1000 observable and max time to execute selected time are 24 hours, then the Test Selection Algorithm selects the tests in 58.48 mins. It is not feasible for a human to go through 100,000 tests and determine which tests to select based on their coverage values. The algorithm is scalable in terms of handling the number of tests and the number of observables.

7.3.1. Analyzing the results of Test Selection Algorithm

The test selection algorithm proposed as a solution is a modified 0-1 Knapsack algorithm with dynamic programming. The time complexity of the proposed Test Selection Algorithm is O(NTM) where:
N = Number of tests
T = Total time available to execute the tests
M = Total number of observables each test has.

We cite *pseudo-polynomial time* from the source[12] :

> "In computational complexity theory, a numeric algorithm runs in *pseudo-polynomial* time if its running time is a polynomial in the length of the input (the number of bits required to represent it) and the numeric value of the input (the largest integer present in the input).In general, the numeric value of the input is exponential in the input length, which is why a pseudo-polynomial time algorithm does not necessarily run in polynomial time with respect to the input length".

The proposed solution runs in the pseudo-polynomial time since the number "T" that is Total time available to execute the tests, requires log T bits to represent T.

7.4 Pros and cons of the proposed solution

The pros and cons of the proposed solution are as follows:

Pros:

1. The proposed solution allows users to define their own notion of coverage. This makes the solution usable for a different notion of coverages.
2. The domain experts can use domain specific keywords, which they are more acquainted with, to describe the notion of coverage instead of using a general programming language.
3. A uniformity about the notion of coverage can be established across the team using the proposed solution.
4. The coverage module and the test selection module can be used independently as long as the coverage is represented in bitstream format.
5. The proposed test selection algorithm adds benefit over manual test selection by making the test selection process automatic and scalable.
6. The test selection algorithm has an execution time that is fairly acceptable by stakeholders.
7. The resource consumption by the algorithm is low as compared to the resources consumed to execute actual tests.

Cons:

1. The users must be acquainted with the model-driven approach to use the proposed coverage module.
2. The proposed test selection algorithm does not guarantee an optimal solution.
3. The execution time and hence, the time complexity of the proposed test selection algorithm depends on the value of the maximum available time and the number of tests. So in case of higher values for maximum available time and the number of tests, the execution time would be slower.

8. Conclusion

Due to technological advancements, the systems at ASML are becoming more complex. It is crucial for ASML to test the systems such as TWINSCAN, YieldStar, and LCP to ensure the quality of its customers. It is essential to test that these three systems are interoperable. The VirtualFab group at ASML carries out the interoperability tests. With new features and changes being made to these systems, the number of tests is exponentially increasing. However, the resources to handle these tests are not increasing exponentially. This leads to a test explosion. In spite of the test explosion, the quality of the testing is not guaranteed because an increased number of tests does not necessarily mean that the test coverage is increasing. This leads to inefficient testing. The proposed solution, "Increasing Test Efficiency," deals with the problem of inefficient testing.

The proposed solution enables the users to describe their own notion of coverage and calculate coverage using a Domain Specific Language(DSL). The generic nature of the DSL allows it to be used for other domains and environments where expressing and calculating coverage is essential. Based on the calculated coverage, the algorithm "Modified 0-1 Knapsack with Dynamic Programming" does the test selection. The test selection is made so as to avoid the execution of those tests which do not increase the test coverage. This algorithm selects the test set in such a way that the tests are selected based on their coverage in a given amount of time. The proposed test selection algorithm is independent of the notion of coverage; in this case, the algorithm works for any notion of coverage expressed in bitstreams. This also allows the algorithm to adapt easily to changes in the notion of coverage.

The proposed solution decreases the number of tests to be executed by selecting the tests based on their coverage values. This makes the proposed solution cost-effective as it selects tests and hence decreases the unnecessary usage of resources. Also, the proposed solution is fast as it can be executed in pseudo-polynomial time with time complexity of O(NTM) where N = number of tests, T = maximum time available to conduct tests, and M = number of observables in each test. The modular nature of the solution enables the solution to be easily adaptable to the changes. This project shows that it is beneficial for ASML to use and develop further on the proposed solution.

9. Future Work and Recommendation

This project is a proof of concept that in order to increase test efficiency, it is essential to know the coverage of the tests and make test selection. The results show that the proposed solution can be used to define and calculate coverage and carry out test selection even in case of a large number of tests. However, the proposed solution has not been validated yet. We suggest to first compare the outcome of the proposed solution to the current way of working (manual test selection), to determine what the value is.

In addition, we give some suggestions to extend the current solution:

Integration with test execution framework: The current system can be interfaced with the test execution framework. So that the selected tests can be automatically executed by the test execution framework.

Report generation: The current solution can be extended to be able to generate reports for coverage and test selection based on which view on data that is required by the stakeholders. Such scenarios can include but not limited to:

1. The report can be generated to give suggestions on how much coverage can be increased if there would be more time to execute the tests. With such a report, the user can decide if they want to increase the maximum time to have better coverage.
2. The report that can show the changes in coverage of tests over a period of time can be generated. This can help to get more insight into parameters affecting the change in coverage of a test.

Predictive coverage: The coverage database can also be used as a training dataset for a predictive coverage algorithm. The predictive algorithm can look at the coverage of the new test and can predict as how much coverage can be increased for a given time if the new test is selected.

Reverse selection: The algorithm can be modified to take coverage as input, and the output of the algorithm should be how much time would be required to achieve that coverage and using which tests.

Feature priority: The algorithm can be modified for tests to be selected based on coverage and (variable) feature priority by pre-processing the input data.

Assume the coverage model describes six features, of which the first three are not important for the upcoming test execution. Then the feature priority can be represented by the following sequence of bits:

$$p = 000111$$

Assume we have three test cases with the following coverage; then we can pre-process these values by doing a bitwise-AND with the priority bits p.

Input	Priority (p)	Pre-processed input
011100	000111	000100
001110		000110
000111		000111

The algorithm does not have to be modified to use the pre-processed input.

10. Project Management

This section elaborates on the project management activities carried out during the lifetime of the project.

10.1 Work-Breakdown Structure(WBS)

In this section, the Work-Breakdown Structure of the project is discussed. The project is divided into four main categories: Planning and Management, Analysis, Design and Implementation, and Documentation. Figure 27 shows the detailed activities conducted in each category.

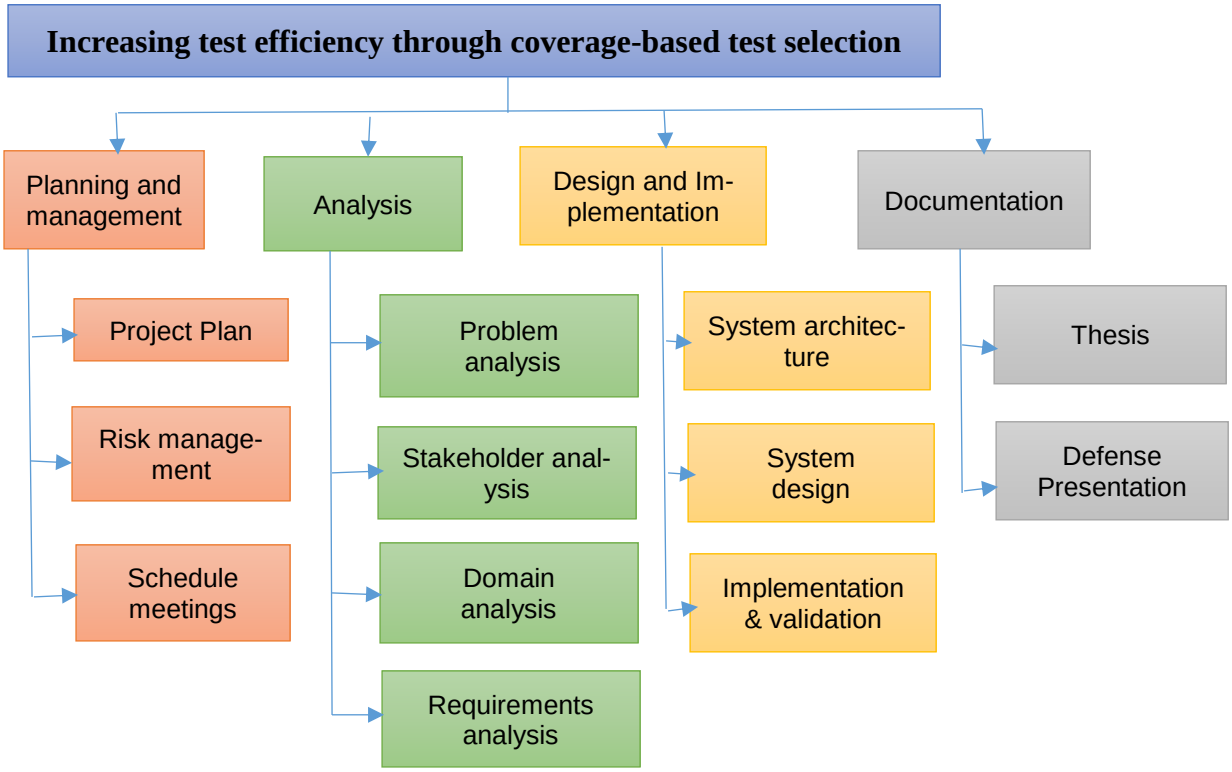

Figure 27 Work-Breakdown Structure of the Project

10.2 Project plan

The project is planned according to the work breakdown structure. The planning was updated during the lifetime of the project in case of early or late completion of the task. The project planning was done using Gantt-chart created using excel.

Every month, a Progress Steering group (PSG) meeting was held to update the stakeholders about the status of the project. The meetings allowed the supervisors to give feedback on the work and the planning of the project. Necessary changes were made to incorporate the feedbacks of the supervisors. Figure 28 shows only high-level project planning. The major activities described in high-level planning was further divided into concrete tasks. For planning the tasks, the Jira board was used. Jira is a proprietary issue tracking software, in which sprints can be planned, and for each task, Jira tickets can be made and kept track of. For this project, Jira tickets were made to keep track of the tasks over a period of sprints where each sprint was of 2 weeks duration.

The two weeks sprints during the first half of the project were essential, as working in sprints helped to define clear short-term goals. This was necessary during the early stages of the project

when the uncertainties and vagueness of the tasks were higher as compared to the mid and later part of the project. The uncertainties and vagueness were higher. Once an initial proof of concept was ready, the uncertainty about the chosen approach was reduced.

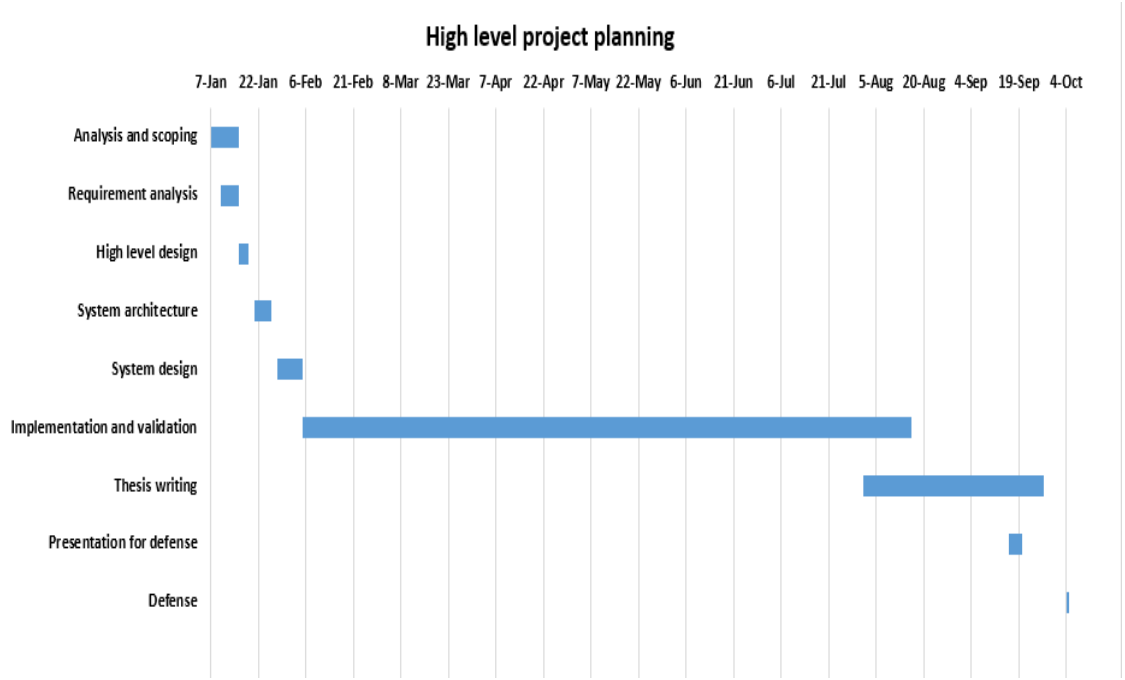

Figure 28 High-level project planning

The snippet of detailed planning for documentation can be seen in Figure 29.

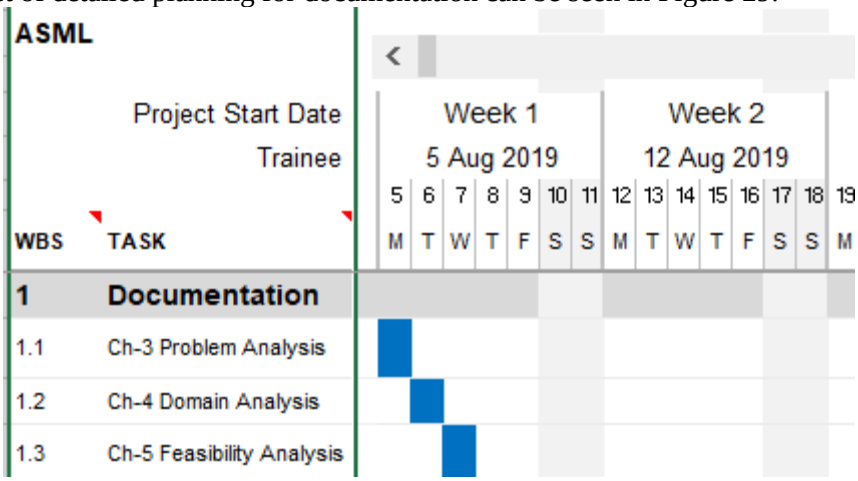

Figure 29 Snippet of detailed planning for documentation

10.3 Risk Management

Identifying risks and managing risks is essential for a project to succeed . The identified risks and actions to be taken for avoiding the risks are shown in Table 7

Table 7 Risk Management

Risks	**Who might be harmed, and how?**	**What are you already doing?**	**Do you need anything else to control risk?**	**Action by Who?**	**Action by when?**
A different idea of the "expected" result of the project.	Stakeholders and Me. I might deliver something which was not expected.	Regular meetings to be on the same page	No.	Me	After regular meetings
The end product might not be usable for the team	Team and Me	Verification by demonstrating to supervisors at regular intervals.	To demo the first version and get feedback from the team	Me	Regularly
Enough data might not be available for processing	The result of the project might not be verifiable.	Talked to stakeholders about arranging the data.	Yes, to have a regular follow up on this.	Me, Roger Lahaye, Rick Smetsers	Changed the scope of the project after the first two months.
The learning curve for technologies might be high	The project deliverables might get delayed.	Learning technologies	No.	Me	Regular during the entire duration
Leaking confidential data	ASML, Me,TU/e	Following the NDA	No	Me	Regular action of correctly labeling the data confidential, not sharing confidential data. Getting approval from ASML technical Board before publishing or sharing content.

Not being able to finish the project on time	ASML,Me, TU/e	Plan and execute. Prioritize the tasks and de-scope if necessary	Regularly monitoring progress to check if deadlines are met	Me	Regularly during the project lifetime.

11. Project Retrospective

This chapter reflects upon the overall experience of working on this project. Moreover, this chapter also revisits the design criteria to show the convergence of decided design criteria and actual design.
The project initially started as a vague statement, “Increasing test efficiency.” Being completely new to the domain of the company and the team, it was first essential for me to understand the context of the problem statement. The first few weeks were spent I understanding the domain, the problem, and what stakeholders want. This process went on smoothly for me due to cooperation from stakeholders. The challenge was to come up with the solution, which is generic enough to be adaptable for a different domain of testing. The learning curve of the technologies used for the project was steep. However, investing more time in technologies made things work better over a period of time.

The uncertainty about the chosen approach for the project was higher during the first half of the project. This was because there existed no proof of concept for the proposed ideas. However, a proof of concept of the partial solution was created during the early stages of the project. This gave more confidence in the chosen approach for the proposed solution. The planning had to be revised during the creation of the proof of concept as the exact estimations of the efforts required to construct the proof of concept were non-trivial. This was due to the fact that understanding the domain and using new technologies had a steep learning curve. The most challenging part of the project was to construct the Domain-specific language such that it is generic enough to be used to define coverage for other domains that need to be tested. This was because a DSL is supposed to be adhering to the domain. However, here, the goal was to give a generic DSL, which in itself would not be dependent on the domain and still unable to capture the domain.

This project allowed me to learn things on higher as well as lower abstraction. This project, in particular, made me aware of the work culture and team dynamics in ASML. This experience will help me with my carrier henceforth.

Below section revisits the design criteria mentioned in section 4.2.3

11.1 Revisiting design criteria

The design criteria mentioned in section 4.2.3 is the criteria that this project should be addressing. This section reflects upon how well the design criteria are reflected in the actual design.

Loose coupling: The coverage definition and calculation module and test selection module are designed in a way that allows these modules to be used independently. These modules are connected through a coverage database. The coverage module calculates the coverage and stores in the coverage database. The test selection module reads from the coverage database. Hence, the change in one module does not affect another module.

Ease of use: The proposed solution involves domain specific language. This allows the domain experts to define coverage using domain-specific words, which makes it easier for them to describe the coverage as compared to using traditional programming language. Also, the code is generated based on the coverage models, which makes it easier for tests to use the coverage models as they don’t have to code from scratch.

Scalability: The proposed solution currently works for 100,000 tests. However, the solution is designed such that it can work for an increasing number of tests. To achieve this, the test selection algorithm is optimized such that it takes the test coverage as bitstream and then operates

on the bitstream. The operations on bitstream are faster as compared to other complex data. This allows executing the algorithm for an increasing number of tests.

Bibliography

1. Martin van den Brink-Lithography goes holistic, https://staticwww.asml.com/doclib/productandservices/images/asml_holistic_images_spr08.pdf, ASML Images, Spring Edition , 2008

2.The combinatorial explosion of software testing-and why you need to change your test strategy, https://www.eurofins-digitaltesting.com/our-company/opinion/the-combinatorial-explosion-of-software-testing-and-why-you-need-to-change-your-test-strategy/, last accessed on 17 Oct 2019.

3. Gordon E. Moore, Cramming more components into integrated circuits, Electronics, Vol. 86, Number 1, Proceedings of the IEEE, 1998.

4. SECS & EDI interface manual for TWINSCAN & YieldStar, ASML tech wiki, last accessed on 17 Oct 2019.

5. Muhammad Shahid , Suhaimi Ibrahim -An Evaluation of Test Coverage Tools in Software Testing ,International Conference on Telecommunication Technology and Applications, 2011.

6. Maya Hristakeva, Dipti Shrestha – Different approaches to solve the 0-1 Knapsack problem, Midwest Instruction and Computing Symposium, 2005.

7. Wikipedia contributors. "Knapsack problem." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 5 Oct. 2019. Web. 17 Oct. 2019.

8. Thomas H Cormen, Charles E Leiserson, and Ronald L Rivest Introduction to Algorithms, TMH,1990.

9. Wikipedia contributors. "Dynamic programming." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 12 Oct. 2019. Web. 17 Oct. 2019.

10. https://www.jetbrains.com/mps/concepts/, last accessed on 17th Oct 2019

11. https://www.antlr.org/, last accessed on 17 Oct 2019.

12. Wikipedia contributors. "Pseudo-polynomial time." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 13 Aug. 2019. Web. 17 Oct. 2019.