Development of a Graphical User Interface for the TSE Test Framework

Gustav Lundin
Abstract
This thesis describes the technical process of developing the graphical user interface for the TSE testing framework that is being developed at the University of Trollhättan/Uddevalla. The primary objectives of the paper are to make the reader understand the purpose and context of where the software was developed, give a thorough and detailed description of the developed software, and describe the process of the development. The test framework of which the GUI will be a part of will also be described. This will help the reader understand the development from a wider perspective. Furthermore will the importance of automated testing be explained. We conclude the paper by discussing whether the requirements of the product where met or not. Concurrently the prototype is deemed to fulfill the requested requirements. However as the test framework is not fully developed yet there are still aspects of the prototype design that are not determined at this point. These parts are left to future work.
# DEGREE PROJECT

## Table of contents

Abstract .............................................................................................................................. ii
Abbreviations ....................................................................................................................... 1

1. Introduction ...................................................................................................................... 1

2. Project context and goals .............................................................................................. 2
   2.1. Hardware ........................................................................................................... 3
   2.2. Goals .............................................................................................................. 4
   2.3. Limitations ....................................................................................................... 4

3. Background ................................................................................................................... 4
   3.1. Black box testing ............................................................................................ 4
   3.2. Boundary value analysis ................................................................................. 5
   3.3. Random testing ............................................................................................... 5
   3.4. Equivalence partitioning ................................................................................ 5

4. Methodology ................................................................................................................. 5
   4.1. Requirements .................................................................................................... 6

5. Development threats ..................................................................................................... 6

6. Technical design ........................................................................................................... 7

7. Implementation ............................................................................................................. 7
   7.1. Structure ........................................................................................................... 8
   7.2. FUT management ............................................................................................ 8
   7.3. Project management ......................................................................................... 9
   7.4. System states and paths ................................................................................... 9
   7.5. Output management ......................................................................................... 9
   7.6. Profile handling ............................................................................................... 10
   7.7. Method colors .................................................................................................. 10

8. Graphical design ......................................................................................................... 10

9. Acceptance testing ....................................................................................................... 10

10. Discussion ..................................................................................................................... 11

11. Conclusion .................................................................................................................... 11

12. Future work .................................................................................................................. 11

13. References ................................................................................................................... 11

### Appendix

A Graphical Layout .......................................................................................................... 1
B List of requirements ....................................................................................................... 2
C List of development threats .......................................................................................... 3
**Abbreviations**

- **GUI**  Graphical User Interface
- **TSE**  Testing Software Environment
- **SD**  Software Development
- **BVA**  Boundary Value Analysis
- **RT**  Random Testing
- **EP**  Equivalence Partition
- **DD-Group**  Design Development Group
- **FUT**  Function Under Test
- **IID**  Iterative and Incremental Development
- **USD**  United States Dollar
- **XML**  Extensive Mark-up Language.
- **GIO**  GUI Information Object
DEGREE PROJECT

Development of a Graphical User Interface for the TSE Test Framework

Gustav Lundin
Dept. of Technology, mathematics and computer science
University of Trollhättan/Uddevalla
P.O. Box 957, SE-46129
Trollhättan, Sweden
gustav.lundin@student.htu.se

Abstract

This thesis describes the technical process of developing the graphical user interface for the TSE testing framework that is being developed at the University of Trollhättan/Uddevalla. The primary objectives of the paper are to make the reader understand the purpose and context of where the software was developed, give a thorough and detailed description of the developed software, and describe the process of the development. The test framework of which the GUI will be a part of will also be described. This will help the reader understand the development from a wider perspective. Furthermore the importance of automated testing will be explained. We conclude the paper by discussing whether the requirements of the product were met or not. Concurrently the prototype is deemed to fulfill the requested requirements. However as the test framework is not fully developed yet there are still aspects of the prototype design that are not determined at this point. These parts are left to future work.

1. Introduction

Software testing is a process that is aimed to determine and improve the correctness, completeness and quality of a piece of code. “It involves executing an implementation of the software with test data and examining the outputs of the operational behavior to check that it is performing as required” [1]. When software testing is applied, a number of faults are usually detected. The discovery of these faults builds up confidence in the tested code. The confidence is based on that the number of faults in the tested code gradually decreases as more faults are found and corrected. It is important to understand that software testing can never completely establish the absence of faults, unless exhaustive testing is used. This can only be done by formal verification. Instead the aim of software testing is to find software faults. It can plainly be said that the purpose of testing a program is to find defects in it. “A test that reveals a problem is a success. A test that did not reveal a problem was a waste of time” [2].

Unfortunately the process of testing software is both costly and time consuming. The costs of finding and fixing faults could rise to as much as 40 - 80% of the total project costs [2]. It is therefore important for a project manager to be able to motivate and understand why this should be done. As said before the benefits of the testing software is that a higher level of confidence is built into the code. The importance of this should not be underestimated.

Consider a hypothetical scenario where a project is shipped to a customer without being properly tested. When the customer is the one that will discover the problems it will be even more time consuming and costly to fix them as they are reported sporadically. More importantly though is that the customer will most likely be disappointed with the product and defer from ordering any other software from the supplier. Inadequately tested software will result in technical and economical problems. These problems affect not only the developers, but the entire society. Recent research indicates that the American society alone is burdened by an annual cost of more than 50 billion USD, solely related to faulty and erroneous software [3]. This confirms the importance of software testing and its relevance to software development.

The reasons for not testing the software may vary but the time issue is always there. Needless to say if the testing of software or at least a part of it could be automated, the potential to save time and thereby money would be substantial. Apart from decreasing the time of testing there are other advantages as well. An automated test is consistent in a way that manual testing is not. The automated test can be repeated exactly the same way any number of times. This makes it easy to determine if a fix of a software fault was successful or not. Another positive aspect of repeating the test is that if the same piece of code would be altered further on in the project, and the fault fix were to be broken, this would be detected. Automated testing also
simplifies analysis of test results. This can be very
time consuming in manual testing.

There is however areas which are not suitable for
automated testing. In smaller projects it could prove
to be more time consuming to set up or configure
an automated test suite [4]. In these cases manual
testing could prove to be more suitable. There are
also alternatives to software testing. Code
inspection is one which has proven to be able to
supply nearly fault free software [5]. However this
method cannot be automated, it requires human
resources and it scales no more than linearly with
its resources.

Generally, automated testing has the potential to
help developers perform testing faster and more
efficient in the areas where there is little need of
creative thought, and the work is mainly
standardized repetition. This could lower the cost of
projects significantly. Examples of where
automated testing could be suitable are Random
Testing (RT), Boundary Value Analysis (BVA),
Equivalence Partition (EP) regression testing,
compatibility testing, portability testing,
performance testing and configuration testing.

The Testing Software Environment (TSE) test
framework is being developed by a software
engineering research team at the University of
Trollhättan/Uddevalla. The TSE will initially be
designed to conduct BVA, EP and RT, but is
possible to extend with additional techniques.

The GUI that has been developed for this thesis is
a part of an undergoing development of the TSE
test framework. Therefore the results of this project
should be viewed, not as a ready product but as a
prototype. A consequence of this is that more
considerations need to be taken in certain issues,
whereas other matters may be passed on to future
developers. Aspects which require more focus is the
scalability of the program and also the ease of
future developers to familiarize themselves with the
code. Other areas such as more thorough and
extensive testing of the GUI are therefore left to
future developers.

The results of the project were satisfying. All of
the main objectives were met along with each of the
requirements with a priority of 4 or 5. Some of the
additional project work that needs to be done is
stated in section 2.3 and 12.

2. Project context and goals

A great deal of research has been conducted on
the area of BVA, EP and RT [6-10]. However little
of this research focus on the aspect of the resources
that are utilized to conduct the tests i.e. construction
of test cases, preparation of the tests, post
evaluation of the tests and so forth. These types of
related test costs as well as resource utilization and
test effectiveness have often been overlooked in
contemporarily studies.

Another shortage of prior research is that it
merely presented a general model of resource
utilization [11]. Stakeholders in the academic and
private sector are in need of a more detailed and
coherent model for resource utilization.

The ambition of the research conducted for the
TSE framework is to evaluate the resource
utilization vs. software reliability. In other words,
the test effectiveness shall be measured in respect to
the amount of resources that was used to achieve it.

To accomplish this a parameterized quantitative
model is needed to describe the connection between
the utilized resources and the test results from the
automated or semi automated tests. At a more
detailed level this involves dealing with the
following issues taken from the research proposal
paper [12].

1. Compile a qualitative descriptive model of
test resources (including techniques) vs.
established reliability within the limits of
semi-automatic EP, BVA and RT.
2. Formulate a quantitative model of test
resources (including techniques) vs.
established reliability, open for empirical
evaluation and extension to other testing
techniques.
3. Using dedicated computational resources
and higher student courses at the
university to assemble extensive empirical
data consisting of several hundred
different software, information on set-
up and test initialization efforts, test result
evaluations etc.
4. Iteratively evaluate the underlying
statistical hypotheses in the quantitative
model, making use of the collected
empirical data.

The TSE framework has been designed into a
number of independent components. The layout of
the TSE framework is illustrated in figure 1.1
Information regarding the specific components can
be found in the TSE specification [13]. It should be
noted that this is according to the 0.2 revision of the
TSE specification. The design may be altered as the
work progresses.

---

1 The following text is an approved extract from
[12]
2 The figure was taken from [13] with consent from
the author.
**Figure 1: The TSE System overview.**

The focus of this paper lies in the development of the User Interface (UI) component. This component is accountable for tasks such as allowing the user to choose which code should be tested. Presenting the information to the user and giving him/her the choice of test technique as well as configuration settings.

It is important to be clear on which tasks the UI component performs. In figure 2 a simplified and general overview of a test framework is illustrated together with an indication of where the development of the UI takes place. This is demonstrated so that it is clear which part the UI component plays in the TSE test framework. It is used to assign which tests that should be performed i.e. it does not create any tests itself. The figure is a modification of a general overview of software testing, taken from [14].

**Figure 2: General software test framework.**

As seen from the illustration of the test framework in figure 1, the UI component is separated from the rest of the framework in a way that it only communicates with the configuration component and the output engine. Due to that the test framework is not fully defined yet and the protocols for communicating with the test framework have not been specified this has not been done in this thesis. However the basic idea is that the configuration file the GUI generates should be sent to the configuration component. Thereafter this component initiates the tests specified in the configuration file. The PrePost component of the UI will then be responsible for the communication with the output engine, receiving the test result and passing them to the result analyzer component. This component will thereafter be displaying the relevant information in the GUI.

2.1. Hardware

To enable the gathering and processing of extensive empirical data in the amount that is needed for this type of research, it is desirable to use a distributed system. It has been shown in [15] that a distributed computer setup can be made both time and cost efficient.

The Midnight cluster [16] was created at the University of Trollhättan/Uddevalla in 2003, to be used as a test bed for studies on Metamorphic Computer Resources (MCR). The cluster is constructed out of 20 desktop nodes and 1 management node, using the Gentoo Linux distribution as software. The hardware characteristics of the Midnight cluster are further illustrated in table 1.

<table>
<thead>
<tr>
<th>Node</th>
<th>CPU</th>
<th>Speed (GHz)</th>
<th>Chip</th>
<th>Memory (GB)</th>
<th>Disk (GB)</th>
<th>Network (Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main</td>
<td>Xeon 2.4G</td>
<td>2.4</td>
<td>Intel</td>
<td>2 GB DDR2</td>
<td>80</td>
<td>1000</td>
</tr>
<tr>
<td>Work</td>
<td>Pentium 4.8</td>
<td>2.8</td>
<td>Intel</td>
<td>4 GB DDR2</td>
<td>100</td>
<td>1000</td>
</tr>
</tbody>
</table>

Optimizing the utilization of computer processing is a problem [17], the idea behind the Midnight cluster is that it will function as an ordinary part of the University network during the daytime, and be allocated as other resources during night time. In this case as computational power for the TSE test framework.

To describe the background purpose and context of the Software Development (SD) this paper has pointed out the importance of software testing. The paper will in the next sections start with an elaboration of the goals and some of the project limitations. The following part of the paper will continue to provide the reader with background information regarding important aspects of testing relevant to the TSE. This involves topics such as, black box testing, RT, BVA and EP.

The methods that have been used to determine the project requirements will be described, and the resulting list of requirements will be presented. The means of how these requirements have been determined will also be described. Thereafter an explanation of the possible threats towards the development will be presented and how certain threats were dealt with. These issues will be followed by a detailed description of the prototype and how the requirements were implemented. In this section, key design issues of the development will be described as well as various problems that were encountered and how these were dealt with.

---

3 Table was extracted from [5] with the consent of the authors.
2.2. Goals

This thesis focuses on the technical development of the Graphical User Interface (GUI) of the TSE test framework [13]. The development of the GUI has been conducted jointly with Andreas Bergqvist, Joan Ljungh and Roger Hammar [18]. The thesis is based on the technical/functional part of the GUI design whereas the other group has focused on the graphical layout and design. This group will throughout this paper be referred to as the Design Development Group (DD-Group). Even though the work has been carried out in close collaboration with this group, these boundaries have been well kept with a few exceptions. The exception is described in section 7.6.

The overall development goal is to develop a user interface component that enables an operator to configure and utilize the TSE framework for automated testing. The GUI should not require any in-depth computer knowledge to manage the system i.e. the GUI should be user friendly. The concept of having a user friendly layout is addressed in [19]. However the issues of a user friendly GUI will not be attended to in this paper. Instead this is addressed by the DD-Group group [18].

The main goal of the development naturally includes fulfilling the project requirements stated in the appendix. The development can be considered a success, if at least all requirements with priority 5 and 4 have been fulfilled. In a more general view the overall goals can be divided into the following.

1. The GUI shall have the ability to retrieve the FUT information from a compiled .NET file.
2. The system should be scaleable and easy to make additions to.
3. The GUI should be able to handle FUT method parameters of all built in data types of C#, and also multidimensional matrices of these data types.
4. Display this information, and allow the user to configure the test settings for the TSE test framework.
5. The information of a GUI session should be able to be saved or loaded at any point of the program state.
6. The GUI shall be able to generate an XML structured configuration file which contains all relevant TSE test information.
7. The GUI should adhere to the graphical design developed by the DD-Group.

2.3. Limitations

Some limitations have been made by this development. An important issue is the limitation to allow for only one type of FUT files that can be loaded into the GUI. In future versions the program should be able to handle both plain source code files as well as a variety of different compiled files. In the first version the GUI will only be able to handle .NET compiled files. However as one of the goals involves scalability, an effort has been made to allow for such additions to be made without much exertion. This is further elaborated on in section 7.2.

Another important issue is which type of method arguments/parameters the GUI will support. In this version we have limited ourselves support only simple data types and symmetrical, multi-dimensional matrices of these. That is arguments such as complex objects, pointers or dynamic vectors are not supported. The implementation of such functionality lies in separate components and additions can therefore be considered to be easily scalable. More details of this can be seen in section 7.

To certify that the GUI fulfils the requested requirements acceptance testing will be made together with the research group. A more thorough and extensive testing of the GUI will be left to the future developers to do on the final version.

3. Background

As the TSE test framework initially will be using test techniques such as RT, BVA and EP. These methodologies will be described in this section. The test framework will also be dealing with combinations of these methodologies. As described in [20], such combinations can be very efficient and is of vital interest in any automated scheme.

3.1. Black box testing

The concept of black box testing (also known as functional testing) is based entirely on the input and expected output of the test. The inner functions of the software in these types of tests are irrelevant i.e. it is only the actual output that is relevant. The test engineer simply provides a given input, collects the resulting output and decides if this should be considered successful or not.

An advantage of this type of testing is that the test is unbiased and can be done separately from the design. This also means that the tester does not require skills in the specific programming language. However the type of testing involves that it is difficult to design test cases that are not redundant, and exhaustive testing is usually not an option as it typically requires a vast amount of time.

BVA and RT are two typical black box testing techniques. There are however other types as well such as State Transitioning Testing, Cause-Effect Graphing or Syntax Testing, these will however not be addressed.
3.2. Boundary value analysis

Testing all possible inputs are as previously mentioned not an attractive option. Even though the processing power of computers have increased rapidly, it is still impossible to test everything due to the exponential growth in available software [20]. And as stated earlier every test that does not detect a software fault can be considered a waste of time. One could say that the key to a good test case is to know where the faults are located. In a BVA test case, the black box tests are configured with the boundary values as input. For example if a program should receive a two digit positive number, boundary value for such a test case would be 00 and 99. The reason for using boundary values is that it is a difficult part of programming and it is recognized to be where most bugs are located [2]. It is usually good practice to test for boundary values which lies outside of the accepted boundary. This type of testing is referred to as Illegal boundary testing.

3.3. Random testing

This form of testing is perhaps the most trivial of the black-box test methods. In RT the input values are chosen at random. Usually though, the values are bound within a certain range. Factors for the test engineer needs to consider in this type of testing are which range of values to choose as well as the amount of randomly generated inputs. The advantage of RT is that it is very easy to generate random test cases and therefore very suitable for automated testing. The drawback of this technique is that the test cases are not efficient and this needs to be countered with a larger amount of tests.

3.4. Equivalence partitioning

Equivalence Partition Testing is a test case design technique for a component where test cases are designed to execute representatives from equivalence classes. In other words, the test is divided in such a way that the system is expected to act in the same way for all tests of each equivalence partition.

When using EP the input domain is usually too big for exhaustive testing, by partitioning it into sub-domains (equivalence classes) testing can become more time efficient.

4. Methodology

As the TSE framework were developed concurrently with the GUI prototype there were no absolute specification list at the start of the development and many of the requirements were unknown. An iterative and incremental development (IID) model was therefore used as a guideline through the development [21]. The IID approach is designed to deal with just this type of problems. The IID process is also a mature and reliable methodology. Prominent developers have during a long time held it as recommended practice [22]. The IID method was further also suitable to work with as the DD-Group development utilized evolutionary prototyping [18].

In the IID approach, the development is divided into a series of projects. These smaller projects are called iterations. Each iteration has phases that are similar to those of a normal project i.e. a requirements phase, design phase, implementation and test phase. At the end of each iteration the total system is incremented.

Some of the benefits of IID development include

- Early mitigation of high risks.
- Early visible progress
- Early feedback and user engagement.

These benefits are well needed in the context that this development took place. It should be noted that the IID methodology were not strictly followed, it was more of a guideline during the development. Some additional best practices and ideas were also used from the Unified Process [21] (which uses IID), these include building the core architecture at an early stage of the project, identifying high-risk aspects and tackle them early. The project development model (or to be precise, two iterations of the model) is illustrated in the following figure.

Figure 3: Project development model.

To enable an early implementation of the core architecture of the system, an attempt was initially made to identify the different areas of the project that were necessary for this. The conclusion was that the following issues needed to be resolved.

1. Information needed to be gathered from the FUT. This could be either a compiled file or in the form of source code.
2. This information needed to be displayed in a way that enabled the user to modify and configure the FUT.
3. The resulting information needed to be sent to an XML generated output file.

When this issue were solved and implemented, we were able to present it to the research group at the University of Trollhättan/Uddevalla, and receive feedback. The following project
DEGREE PROJECT

development was based on this core architecture and the system grew incrementally from it. It should be noted that the early solutions were not in any way complete. The idea was to establish if each of the solutions were possible by implementing a small example.

4.1. Requirements

The process of finding out, analyzing, documenting and checking services and constraints (requirements) is known as requirements engineering [1].

The requirements engineering conducted during this project was conducted together with the DD-Group. The requirements were deducted through a modification of the user stories methodology [23]. The idea was to use the core idea behind user stories without being overambitious with each aspect of the method i.e. to use short descriptions of high-level requirements as basis for conversations with the research group.

The requirements were deducted through a number of meetings. These were more frequent in the initial stages of the development. Most of the requirements were also discovered in the early stages of the project. However the meetings were carried out throughout the entire development and as the development progressed the requirements were refined and a few new ones were determined.

The requirements have been categorized according the FURPS+ model [24]. To enable better understanding, each of the requirements has also been prioritized. The levels of priority have been divided into a scale from 1 to 5 whereas 5 being the top priority. The priorities are stated below.

5. **Critical** – Project would fail if these are not met.
4. **High** – These are needed to provide required functionality.
3. **Moderate** – These requirements are important but the project will not fail without them.
2. **Low** – These are not important, but they will good additions to the project if they are met.
1. **Nice** – These requirements will make very little impact on the project and may be ignored.

From the FURPS+ model the categorization of the requirements defined for this project are as follows.

**Functional requirements** – These types of requirements generally represents the main product features and functionality. An example of such a requirement taken from the appendix is F1: “The GUI shall generate an output configuration file.”

**Performance requirements** – These requirements are concern characteristics such as throughput, response time, start-up and shutdown time and recovery time. An example of such a requirement is P1: “The system delays should be within acceptable limits.”

**Supportability requirements** – These types are involve characteristics such as adaptability, maintainability and testability. S3 can be seen as an example of these requirements: “The GUI should in this first prototype be able to handle .NET compiled files.”

**Interface requirements** – These types deals with characteristics such as format specification and interaction with other items. An example of such a requirement is I1: “The UI component should together with the Pre/Post and Result analyzer be a separate component of the TSE framework.”

5. Development threats

Taking time to perform a risk analysis is often a valuable experience [25]. The risks of the development were therefore considered at an early stage of the project. As the research group was still working on the TSE framework, not all aspects of the GUI requirements were known to them. This was one of the primary reasons of why the IID methodology was adopted. The advantages of IID are stated in chapter 3. It enables the research group to observe the development at an early stage, give early feedback and allows for mitigation of high risk areas early in the project. These advantages were important in dealing with the different risks of the development.

The development risks have been divided into three categories of probability, low, average and high. These are stated in the appendix. The development risks with high categories are in this section further elaborated on.

- **R1**: “It will become very difficult and time consuming to retract information from a compiled file.”
- **R6**: “The classes, methods and parameters cannot be displayed in a satisfactory way.”
- **R9**: “Some of the requirements are inaccurate, and will be needed to be changed.”
- **R10**: “Some of the requirements are misinterpreted by the development team.”

The seriousness of the R9 and R10 are threats that should not be underestimated. The in agreement with the DD-Group it was decided to tackle these threats by having frequent meetings with the research group and thereby updating the requirements alongside the project development.

When dealing with threats R1 and R6, it was decided to attack these at an early stage of the
DEGREE PROJECT

development. In R1 the reflection functionality of .NET was found as a straightforward solution. And for R6 a hierarchical node system used with certain icons and varying colors was decided to be adequate for the task.

In short it can be said that requirement engineering model (described in section 4.1) together with the IID development model (described in section 4) were the solution to many of the threats for this project.

6. Technical design

The goals of the technical development are described through its expected requirements. These are stated in chapter 3.1. To enable an understanding of how these are fulfilled a more general description of the technical design is presented below. This is followed by a more detailed elaboration of how it was implemented.

The system is built up around a simple concept with a central object, the Config object. The user interface enables the user to manage this object in a number of ways.

1. Load selected FUT information into it.
2. Browse through its parameters.
3. Make configurations.
4. Save it to file.
5. Generate output from it.

To enable FUT information to be loaded into the Config object there needs to be different solutions created depending on if the FUT resides in a compiled file or a source code file. If it resides in a compiled file, it also depends upon the type of compilation. In concurrence with the research group, this first prototype of the GUI was focused on the support of .NET compiled files. However as the system should support other type of files in the future versions, the system needed to be created in a way which made such additions possible. This section of the system were therefore constructed so that it checks what type of file has been selected and also if there is currently support for it.

To give the user the ability to browse through the parameters a node tree structure was chosen to display the different parameters. This choice was made in together with the DD-Group. The feature might seem trivial at a first glance. However there were factors which made this task more complicated. If a parameter is of the type array for example, the length of the array that should be subject for testing needs to be specified. However this must be done by the user i.e. after the FUT has been loaded into the GUI. This together with the fact that a FUT parameter may have more than one dimension made it more difficult.

The node tree will consist of three different levels of objects. The method classes, the actual methods and the method parameters. Each of these is objects which require different types of methods and settings. An example of this is that the colors of the methods in the node tree needs to be altered depending on the method state. I should also be noted that the parameters need to be of different types, string, char, integer etc.

Allowing for changes to settings to be done to the Config object is simply that all configurations made by the user are reflected in changes in the subsequent part of the Config object. This task does not require further explanation.

A larger file of test code may require a number of different configuration settings to be made. If there is a problem with the test or if the test needs to be repeated it is a good idea to be able to save the work that has been done. To allow for this the entire Config object needs to be able to be saved to file. Needless to say, this also means that an entire project with all of its settings should be allowed to be loaded. This task involves two things, loading the object, but also correlating the GUI with the current state of the Config project i.e. setting the activation of different GUI components etc.

The final technical task is to allow the Config object and all of its settings to be written to the configuration file. This file should be in the form of XML and in the format as specified by [13].

7. Implementation

In this section of the paper, the implementation of the GUI is described. This involves an elaboration of how the overall aims of the technical design were solved i.e. the FUT management, GUI management, project management and output management. This also involves describing a few of the problems that occurred and how these were solved and a few other topics that were relevant for the development.

The programming language C# was used for the development of this project. The choice of C# was motivated by its ease of constructing graphical interfaces, an advantage primarily used by the DD-Group. The object oriented nature of C# was also a desirable aspect which influenced the decision.

To avoid crashes of the system, some exception handling has been used in this project. This has been used at certain selected areas. In accordance with common guidelines, it has been used in areas where an exception (or such an event) would occur infrequently. In areas where frequent such events frequently take place, normal code handles the actions i.e. (if/else statements). A reason for this is that exceptions are usually slower executed.

One of the goals was the readability and understanding of the source code. To enable future developers to quickly be accustomed with the code, effort has been made to make project methods more
understandable i.e. the method names are long and descriptive to the nature of the method.

7.1. Structure

When designing the structure of the system there are a number of aspects to consider. A well designed Object Oriented (OO) system should preferably adhere to established design patterns. A few of these patterns which are of importance are the concept of cohesive classes, controller classes and also low coupling between classes [21].

Low coupling is simply the concept of avoiding unnecessary couplings between objects. This is achieved by assigning correct responsibilities to the different classes.

The concept of cohesive classes is that classes should fulfill well defined and not perform tasks which lie outside of its definition. This pattern helps to keep the complexity manageable.

The idea of the controller class is defined by the class which acts as the event handler. This should according to [21] be assigned to either the class that represents the overall system, device, subsystem or a use case scenario within which the system event occurs.

Due to the drag and drop nature of GUI creation in C# the concept of the controller pattern has been ignored. There are many different event handlers, all included in the GUI class. The patterns of cohesion and low coupling have been followed to best effort. The central MainForm class could be considered to be breaking the cohesion pattern, but has for other obvious reasons been left as it is.

An important structural issue is the placement of certain classes in the myLib component. The reason for this separation is to allow for the system to be easily scalable. As these classes are needed in any additional FUT component, (a component added to enable support for additional file types) this solution enables the myLib classes to be reused in any additions.

To clarify the structure of the system, the InfoObject that is returned from a component is a part of the overall Config object. The InfoObject contains all FUT information such as classes, methods, parameters also their settings. The InfoObject also contains the map of the different partitions (in the form of a hash table).

The structure of the InfoObject is best illustrated from the class diagram in figure 4. It contains one or more Domains objects. Each of these contains one or more Methods objects which contain one or more parameter objects i.e. a subclass object of the Var class. The subclass objects represents the different data types found in C# such as Integer, String etc.

The sought result of this structure was that the arraylists of the Methods would contain different objects which represented the different data types of C#.

In later stages of the development it became apparent that the hash table that represented the partition map could not be apart of an object which should be serialized. This resulted in that certain changes needed to be made. The Config object was equipped with a InfoSerial class which is a copy of the InfoObject class except that it does not have an hash table representing the partition map. This is instead replaced by a class representation of a hash table. The idea is that it will give the required structure to the configuration file, while still remaining possible to serialize. The InfoObjToSerialObj() method is therefore needed to fill the different values from the InfoObject to the InfoSerial object.

Figure 4: Simplified class diagram

7.2. FUT management

Retrieving information from the function that needs to be tested plays an essential role in this project. The idea is that it should be possible to with ease extend the program to support any type of language in compiled or source code form. For this to be achieved it was decided to separate this implementation into separate binary components. As stated earlier this first version of the GUI will only implement support for .NET compiled files, hence only one such component have been implemented.
The task of analyzing a source code file and providing structured information of all its classes, methods and parameters is a task which is far from trivial. However for .NET compiled files the .NET framework provides reflection. The possibilities of reflection for C# are described in [26]. Reflection offers a solution to this very problem. The fact that code from other than C# also can be compiled into .NET files, made this very good to use as a first component to the system.

The implementation idea is in a way plug-in related. There is an object DLLHandler which handles all of the requests to open new FUT files. It checks which type of file it is about to open and assigns the relevant component to perform the task.

To achieve this functionality all of the components need to implement a common interface with certain functions. The getInfoObject() function in this interface is the key function for the components. This function goes through the all the classes, methods and parameters of the FUT file and transfers them into a standardized information object (the infoObject). The InfoObject is thereafter returned to the GUI via the DLLHandler.

This type of solution allows for components to retrieve the relevant FUT information in different ways, while presenting it to the GUI in a defined and structured way (the information object). This is desirable as solutions for a compiled file would be very different than from a source code file (where it would involve parsing of the actual source code).

7.3. Project management

To enable the user to save a test configuration project all of the data that was relevant to the test needed to be saved. With the design choices made this task was basically a matter of using a Serializable attribute on the relevant classes and then serializing the entire Config object. To load a project the procedure an object was the procedure was reversed. However factors that make this a bit more complex are that correct setup for the current state needed to be set.

7.4. System states and paths

In a larger development it is important to understand the flow of the program that is being developed. This is perhaps especially important when dealing with a graphical interface. This because it is difficult to oversee all possible paths the system may take. More states and paths lead to increased complexity, which in turn allows for more software failures to occur. To reduce this complexity, an effort was made to reduce the number of input paths and limit the amount of states for the GUI.

This was in practice achieved by making the components (such as buttons, lists, frames) within the interface to become active or inactive depending on the current state of the system. A flowchart diagram of the system is illustrated in figure 5. It should be noted that the load process is not present in this diagram as this is considered to be excessive information. At any state the system could load a project. It would then jump to the state of the saved project.

![Flowchart](image)

**Figure 5: Flowchart**

7.5. Output management

The resulting output of the GUI is an important part of the design and is one of the few parts of this project where a clear specification of structure was given. The fact that configuration file should be generated in XML affected the entire project in certain aspects. As the C# is able to utilize libraries that offer to generate an equivalent XML structure of an object. It was decide that the best way to achieve the output requirements was to simply construct an object (the Config object) that fulfilled the structural demands, and use the System.Xml.Serialization class to achieve the result.

This feature was fairly straightforward to implement and involved setting certain attributes to the variable arrays such as XmlArrayItem. As the arrays that should be generated needed to be dynamic (ArrayLists), the xml.serialization class needed to know what type of objects the dynamic arrays would contain.
DEGREE PROJECT

The XML serialization feature functioned so that it printed public objects and their public attributes. A problem of that was that no good way was found to avoid serialization of a public attribute. Private attributes encapsulated with get and set methods were also considered public. The result of this was that care needed to be taken when using attributes in certain classes, so not to generate excessive information to the configuration file.

Small sections of a generated configuration file are illustrated in figures 6 and 7. In figure 6 a partition in XML format is displayed and in 7 a method containing two float parameters is shown.

Figure 6: Section of configuration file.

```xml
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="config.xsd">
  <instance>
    <name>Mymodule</name>
    <attribute>
      <name>MyAttribute</name>
      <value>MyValue</value>
    </attribute>
  </instance>
</config>
```

Figure 7: Section of configuration file.

7.6. Profile handling

The GUI will in future versions be able to load and use existing test profiles. The test profiles that are loaded should be displayed and it should also be possible to modify them. This work has been initiated and conducted by the DD-Group. However additional work needs to be done before this part is operational.

7.7. Method colors

To make the GUI more understandable and user friendly the DD-Group required the methods to change colors according to the state they were in. The methods were to have three different states. Each of these was represented by a color. A green method represents a method which is ready for testing, i.e. both oracle and type of testing has been selected for all of the method parameters. Orange represents a method which is not ready for testing but has either all parameters configured or an oracle support assigned. If the method color is red the method does have neither all parameters configured nor oracle support assigned.

To implement this functionality, the method colors needs to be updated at every parameter change. This is due to that if a single parameter would be altered it could change the state of a method. This is done in the `updateMethodColours` method. In practice this means that after every parameter or oracle change, all parameters of the test suite needs to be explored.

8. Graphical design

The graphical structure and design of the prototype is based around three main features. The first of these is a navigation window where the user is able to navigate. This is organized in a tree hierarchy and displays the different classes, methods and variables of the code that is to be tested. The second is a window that displays all the selected variables and their characteristics, and finally a series of tabs. Each tab enables the user to edit different project test information or variable test characteristics.

An interesting aspect of the graphical design is that the colors of the different methods changes from red to orange to green during the configuration process. This was described in the previous section. All work concerning the graphical layout and design of the prototype has been conducted by the DD-Group and they are addressed in more detail in [18]. In appendix the graphical design of the GUI is demonstrated.

9. Acceptance testing

To determine that the system fulfils the required functionality, it has been put through acceptance testing together with the research group. The purpose of using acceptance testing is not to prove that no faults exist in the source code. It is rather used to assure the customer that the system is able to perform the required functionality. This is done by actually performing the different operations together with the customer.
10. Discussion

Issues such as response time and system delays were originally not considered to be a threat to the development. This was due to that it was believed that the system was not presumed to do any heavy calculations. However further into the development it was realized that if larger matrices were used as input parameters the number of parameters could quickly become very large. At this point the system can be considered to handle up to approximately 3000 method parameters (tested on a Pentium 4 desktop PC) before the system delays becomes intolerable. The requirement P1 from the research group states that the delays should be within acceptable limits. The current system delays were considered acceptable by the research group for a first version of the GUI. It is however recommended to do additional work here.

The ability to save a project has an additional advantage that should be taken into consideration. In future versions the Pre/Post component of the test framework will communicate with the output engine component. It will need to acquire the results from tests. However, to do so it will need a database id to specify which test it is interested in. This will most likely be generated when the test is initiated and at this point be sent to the GUI. The ability to save a project offers the database id to be saved within the project. This means that the GUI program can be shutdown and restarted while the test framework is executing, as long as the project is saved.

There were a few requirements of lower priority which were not fulfilled these are stated below.

The implementation of the edit profile and method equivalence requirements (F11, F20) was postponed due to lack of time. The requirement I2 was not possible to implement as the specification had not been completed.

11. Conclusion

This paper has described the development of the GUI component of TSE test framework. It has stated the overall objectives and functions for the GUI as well as described the process of how these were implemented. Some of the implementation problems that were encountered during the development have also been addressed.

The paper furthermore described the context in which the development took place. This involved an elaboration of the TSE test framework of which the GUI will be a part of. Background information has also been given to support this explanation.

The development can be considered a success as all requirements with a priority of 4 or 5 have been met.

12. Future work

The most obvious extension that could be made to the system is the construction of additional FUT components. This should be done to enable the system to handle compiled files of other than .NET type or to allow for source code files to be parsed in a certain programming language. A natural addition would here be to add support for Java compiled files as the java language also supports reflection.

Another primary aspect of the GUI that needs to be achieved is the integration with the TSE test framework. The components that are needed for this integration are the Pre/Post component and the Result analyzer component. The result analyzer component will take up a large space of the GUI. However due to the tab design of the GUI (chosen by the DD-Group) the integration of these additions can easily be made by adding another tab.

The profile section of the GUI is still incomplete and needs to be further developed before it can be used.

It is also recommended that further actions be taken to speed up the system. Primarily this involves a faster method of creating new partition assignments.

As declared in this report, this version of the GUI has limited itself to support basic data types and symmetrical matrices/arrays of these. This is a vital part of the system. In future versions it is recommended that additional support is added for types such as complex objects, asymmetrical matrices, pointers and so forth.

13. References


[12] Christiernin, S, “Model for resources and software reliability in (semi-) automatic software testing”, Department of Computer Science, University of Trollhättan/Uddevalla, Sweden.


DEGREE PROJECT

Appendix A: Graphical Layout
## Appendix B: List of requirements

### Functional requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>The GUI shall generate an output configuration file.</td>
<td>5</td>
</tr>
<tr>
<td>F2</td>
<td>The configuration file shall be structured in XML format.</td>
<td>4</td>
</tr>
<tr>
<td>F3</td>
<td>The XML structure should be as specified in [13]</td>
<td>3</td>
</tr>
<tr>
<td>F4</td>
<td>The configuration file shall contain all data described in [13]</td>
<td>4</td>
</tr>
<tr>
<td>F5</td>
<td>The GUI shall be able to view the name and parameters of the FUT.</td>
<td>5</td>
</tr>
<tr>
<td>F6</td>
<td>Each of the FUT parameters shall be possible to configure.</td>
<td>5</td>
</tr>
<tr>
<td>F7</td>
<td>The possible configurations for each partition are BVA, RT.</td>
<td>5</td>
</tr>
<tr>
<td>F8</td>
<td>In RT there should be two choices, manual settings or profile.</td>
<td>5</td>
</tr>
<tr>
<td>F9</td>
<td>If BVA is selected, illegal BVA should be possible to select.</td>
<td>5</td>
</tr>
<tr>
<td>F10</td>
<td>If profile in BVA is selected there shall be two choices, using an existing or creating one.</td>
<td>4</td>
</tr>
<tr>
<td>F11</td>
<td>An existing profile should be possible to edit</td>
<td>2</td>
</tr>
<tr>
<td>F12</td>
<td>If profile is selected, standard deviation, quality demand, ranks and remove optimization shall be configurable.</td>
<td>4</td>
</tr>
<tr>
<td>F13</td>
<td>Each variable except Boolean should be able to be locked to a certain value.</td>
<td>5</td>
</tr>
<tr>
<td>F14</td>
<td>Each variable (where it's possible) should be assigned a unique partition when a new test file is loaded into the GUI.</td>
<td>3</td>
</tr>
<tr>
<td>F15</td>
<td>The id of the partitions group shall be capital letters A, B, BA etc.</td>
<td>2</td>
</tr>
<tr>
<td>F16</td>
<td>New unique partitions should be able to be assigned for unassigned variables.</td>
<td>4</td>
</tr>
<tr>
<td>F17</td>
<td>An already existing partition shall be assignable to any variable.</td>
<td>4</td>
</tr>
<tr>
<td>F18</td>
<td>If a method parameter has more than one dimension, the number of dimensions shall be configurable.</td>
<td>5</td>
</tr>
</tbody>
</table>

### Performance requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>The system delays should be within acceptable limits.</td>
<td>3</td>
</tr>
</tbody>
</table>

### Supportability requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>The source code should be easy to maintain and understand.</td>
<td>3</td>
</tr>
<tr>
<td>S2</td>
<td>The source code should be easy to expand and make additions to.</td>
<td>4</td>
</tr>
<tr>
<td>S3</td>
<td>The GUI should in this first prototype be able to handle .NET compiled files.</td>
<td>5</td>
</tr>
<tr>
<td>S4</td>
<td>It should be easy to expand the prototype to handle other types of compilations.</td>
<td>4</td>
</tr>
</tbody>
</table>

### Interface requirements+

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>The UI component should together with the Pre/Post and Result analyzer be a separate component of the TSE framework.</td>
<td>5</td>
</tr>
<tr>
<td>I2</td>
<td>GUI shall communicate with the TSE and a database according to spec.</td>
<td>4</td>
</tr>
</tbody>
</table>

### Oracle requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>F19</td>
<td>One or more oracles shall be able to add to an existing project.</td>
<td>5</td>
</tr>
<tr>
<td>F20</td>
<td>A method within an oracle should be able to be given a different, equivalent name.</td>
<td>3</td>
</tr>
</tbody>
</table>
## Appendix C: List of development threats

### Development risks

<table>
<thead>
<tr>
<th>ID</th>
<th>Risk</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>It will become very difficult and time consuming to retract information from a compiled file.</td>
<td>High</td>
</tr>
<tr>
<td>R2</td>
<td>Coordinating the work assignment may result in inefficiency.</td>
<td>Average</td>
</tr>
<tr>
<td>R3</td>
<td>Project development time will be insufficient</td>
<td>Average</td>
</tr>
<tr>
<td>R4</td>
<td>Saving and loading an entire project may be a difficult and time consuming task.</td>
<td>Low</td>
</tr>
<tr>
<td>R5</td>
<td>Not enough meetings with the research group may lead to insufficient feedback.</td>
<td>Average</td>
</tr>
<tr>
<td>R6</td>
<td>The classes, methods and parameters cannot be displayed in a satisfactory way.</td>
<td>High</td>
</tr>
<tr>
<td>R7</td>
<td>Is it not possible to achieve the user friendly appearance requested by the DD-Group.</td>
<td>Average</td>
</tr>
<tr>
<td>R8</td>
<td>Some of the requirements will not be completed in time.</td>
<td>Average</td>
</tr>
<tr>
<td>R9</td>
<td>Some of the requirements are inaccurate, and will be needed to be changed.</td>
<td>High</td>
</tr>
<tr>
<td>R10</td>
<td>Some of the requirements are misinterpreted by the development team.</td>
<td>High</td>
</tr>
</tbody>
</table>