A01 Deliverable

M.U.P

massively underdeveloped project

Rasmus Tilljander - rati10@student.bth.se  
Nils Forsman - nifo08@student.bth.se  
Calle Ketola - cake10@student.bth.se  
Kim Hansson - kiha10@student.bth.se

## Introduction

The assignment consists of creating and documenting an architecture for a predefined system and in doing so learn how to transform quality goals into a practical solution.

The system is an automated test bench for use on various kinds of software. It will feed predefined input into the tested system and verifies the output, reporting any deviations. The system need to be easy to maintain and adding new features such as input types, emulation mechanics and testing techniques with minimal effort and cost.

It also needs to log all the testing data and generate this as a report containing test statistics, how to recreate the errors and the type of errors encountered.

The idea is that the system will be used in maintenance departments of software organizations with a demand for advanced and automated testing.

## Assumptions

We have assumed that:

* all standard computers have an operative system that is either Windows, Linux or MacOS.
* a standard computer has at least a Giga bit network card, DDR2 memory, 2 Ghz single core.
* we do not emulate any software within the system, instead we channel the information through the system between the tested system and the exterior software.
* the MIB will be maintained for 20-30 years.
* the MIB will be developed by a competent team with the required skills to implement it.
* the Development does not have a tight schedule nor any specific budget constraints.

## System analysis

### Factors

|  |  |  |  |
| --- | --- | --- | --- |
| **Category:**  Development platform  **Name:**  O3.1.1  **Components:**  All components(if changed)  **Stage:**  Design(unless changed),else Architecture | **Description:**  Run on standard computers | **Flexibility:**  None  **Changeability:**  A change of environment from standard computers to different computer types(not likely) | **Impact:**  If a change of environment from standard computers to a different type occurs then the whole architecture would likely have to be rewritten. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Category:**  Functional Features  **Name:**  P1.1.1  **Components:**  One component  **Stage:**  Architecture | **Description:**  Simulating various inputs types | **Flexibility:**  None  **Changeability:**  None | **Impact:**  None |

|  |  |  |  |
| --- | --- | --- | --- |
| **Category:**  Functional Features  **Name:**  P1.1.2  **Components:**  One component  **Stage:**  Architecture | **Description:**  identify output data and compare it to expected output | **Flexibility:**  None  **Changeability:**  None | **Impact:**  None |

|  |  |  |  |
| --- | --- | --- | --- |
| **Category:**  Functional Features  **Name:**  P1.1.3  **Components:**  One component  **Stage:**  Architecture | **Description:**  Emulate various different hardware applications | **Flexibility:**  None  **Changeability:**  Hardware emulation could be removed(not likely) | **Impact:**  Components handling the emulation of hardware would have to be removed. Limited change to the rest of the system(if any) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Category:**  Functional Features  **Name:**  P1.1.4  **Components:**  One component  **Stage:**  Architecture | **Description:**  Emulate various different software applications, platforms and protocols. | **Flexibility:**  None  **Changeability:**  Software emulation could be removed(not likely) | **Impact:**  Components handling the emulation of software would have to be removed. Limited change to the rest of the system(if any) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Category:**  Reliability  **Name:**  P4.2.1  **Components:**  Two components  **Stage:**  Implementation | **Description:**  Input and output components should gracefully recover if the tested system crashed | **Flexibility:**  None  **Changeability:**  None | **Impact:**  None |

|  |  |  |  |
| --- | --- | --- | --- |
| **Category:**  Reliability  **Name:**  P4.2.2  **Components:**  One component  **Stage:**  Architecture | **Description:**  Restarting the tested system from a certain point after a system crash | **Flexibility:**  Yes, latency for restarting the test could be increased to allow different kinds of recovery  **Changeability:**  No need for restarting from a certain point(not likely), No need for restart at all(not likely) | **Impact:**  Changes to this factor would only affect the component that is responsible for restarting the tested system |

|  |  |  |  |
| --- | --- | --- | --- |
| **Category:**  Availability  **Name:**  P4.1.1  **Components:**  Several components  **Stage:**  Architecture | **Description:**  Adding new input mechanisms | **Flexibility:**  None  **Changeability:**  None | **Impact:**  None |

|  |  |  |  |
| --- | --- | --- | --- |
| **Category:**  Availability  **Name:**  P4.1.2  **Components:**  One component  **Stage:**  Architecture | **Description:**  Adding new hardware emulation mechanisms | **Flexibility:**  None  **Changeability:**  None | **Impact:**  None |

|  |  |  |  |
| --- | --- | --- | --- |
| **Category:**  Availability  **Name:**  P4.1.3  **Components:**  One component  **Stage:**  Architecture | **Description:**  Adding new software emulation mechanisms | **Flexibility:**  None  **Changeability:**  None | **Impact:**  None |

|  |  |  |  |
| --- | --- | --- | --- |
| **Category:**  Availability  **Name:**  P4.1.4  **Components:**  One component  **Stage:**  Architecture | **Description:**  Adding new testing techniques | **Flexibility:**  None  **Changeability:**  None | **Impact:**  None |

|  |  |  |  |
| --- | --- | --- | --- |
| **Category:**  Schedule vs Functionality  **Name:**  O1.2.1  **Components:**  All components  **Stage:**  Architecture | **Description:**  Keeping cost and time to implement as low as possible | **Flexibility:**  The balance between cost and time to implement could be altered to better fit the team, making one more important than the other  **Changeability:**  Increase in budget or schedule | **Impact:**  A higher budget or a longer schedule would enable the implementation of better software/hardware |

|  |  |  |  |
| --- | --- | --- | --- |
| **Category:**  Acquisition performance  **Name:**  P3.1.1  **Components:**  One component  **Stage:**  Architecture | **Description:**  Handling large throughput of data | **Flexibility:**  None  **Changeability:**  The required throughput could be made even higher than at this moment | **Impact:**  Worst case the hardware would have to be updated. Software changes would be kept within a minimal amount of components |

|  |  |  |  |
| --- | --- | --- | --- |
| **Category:**  Functional Features  **Name:**  P1.1.5  **Components:**  One component  **Stage:**  Architecture | **Description:**  Logging all the test data | **Flexibility:**  None  **Changeability:**  None | **Impact:**  None |

### Issues

|  |
| --- |
| **Name:**  Multiple input issue |
| **Description:**  The system must support several types of input simulations as well as being able to add new types if the costumer requires it.  **Factors:**  P1.1.1 Simulate various input types  P4.1.1 Adding new input mechanisms |
| **Solution:**  Making the interface between the module handling input and the sub modules for different input types work the same no matter what input type it is. |
| **Strategies/Tactics: "***Generalize the module" "Software Architecture in Practice Second Edition"* Chapter 5.3 Len Bass, Paul Clements, Rick Kazman 2003 |

|  |
| --- |
| **Name:**  Multiple hardware emulations issue |
| **Description:**  The system must support several types of hardware emulations as well as being able to add new types if the costumer requires it.  **Factors:**  P1.1.3 Emulate various hardware devices  P4.1.2 Adding new hardware emulations |
| **Solution:**  Explore standards for hardware communications currently used or in development to support most hardware emulations without impacting the system. |
| **Strategies/Tactics:**  "Maintain semantic coherence" and **"***Anticipate expected changes" "Software Architecture in Practice Second Edition"* Chapter 5.3 Len Bass, Paul Clements, Rick Kazman 2003 |

|  |
| --- |
| **Name:**  Multiple software emulations issue |
| **Description:**  The system must support several types of software emulations as well as being able to add new types if the costumer requires it.  **Factors:**  P1.1.4 Emulate various software applications  P4.1.3 Adding new software emulations |
| **Solution:**  Explore standards for software communications currently used or in development to support most software emulations without impacting the system. |
| **Strategies/Tactics:**  "Maintain semantic coherence" and **"***Anticipate expected changes" "Software Architecture in Practice Second Edition"* Chapter 5.3 Len Bass, Paul Clements, Rick Kazman 2003 |

|  |
| --- |
| **Name:**  Multiple testing techniques issue |
| **Description:**  The system must support several types of testing techniques as well as being able to add new types if the costumer requires it.  **Factors:**  P4.1.4 Adding new testing techniques |
| **Solution:**  Keeping the semantics of the testing modules coherent so that further testing techniques can be added with minimal changes to the current structure |
| **Strategies/Tactics:**  "Maintain semantic coherence" and **"***Anticipate expected changes" "Software Architecture in Practice Second Edition"* Chapter 5.3 Len Bass, Paul Clements, Rick Kazman 2003 |

|  |
| --- |
| **Name:**  Not crashing with tested system issue |
| **Description:**  The MIB must not crash just because the tested system crashes. This means that the input must be stalled until the tested system is running again.  **Factors:**  P4.2.1 Reliable input and output components |
| **Solution:**  If a crash occurs the wrapper around the tested system will send a message to the data broker stating that further testing must halt until the system is running again. This will stop further data transfers from crashing the rest of the system. |
| **Strategies/Tactics:** |

|  |
| --- |
| **Name:**  Creating report issue |
| **Description:**  The MIB needs to be able to create a report once the testing is done. This report must contain data from both the output component as well as the log for all the test data.  **Factors:**  P1.1.2 Identify and compare output  P1.1.5 Logging all test data |
| **Solution:**  There must be a connection between the component handling the output verification and the component that logs all the test data so that data can be sent between them to be combined into the final report at the test end. This will be handled by the data broker |
| **Strategies/Tactics:** |

|  |
| --- |
| **Name:**  Keeping the system running through a test crash |
| **Description:**  The MIB needs to be able to standby further testing if the tested system crashes until it has been restarted from a earlier point.  **Factors:**  P4.2.2 Restart tested system on crash |
| **Solution:**  A component wrapped around the tested system will record the state of the tested system at regular intervals. If a crash occurs the component will restart the system using the latest checkpoint as reference. |
| **Strategies/Tactics:**  “Checkpoint/Rollback” *"Software Architecture in Practice Second Edition"* Chapter 5.2 Len Bass, Paul Clements, Rick Kazman 2003 |

|  |
| --- |
| **Name:**  Running system on all standard computers with required performance |
| **Description:**  The MIB needs to be able to run on all standard computers on the market which have enough performance to run both the MIB itself and the tested system.  **Factors:**  O3.1.1 Development platform |
| **Solution:**  Making the MIB cross-platform compliant |
| **Strategies/Tactics:** |

|  |
| --- |
| **Name:**  Data transfer |
| **Description:**  Data and messages need to be sent between several components that do not have knowledge of each other  **Factors:**  P3.1.1 large throughput of data  P1.1.5 Logging test data |
| **Solutions:**  We will implement a central data broker that will handle transferring data between components. |
| **Strategies/Tactics:**  “Broker pattern” |

## Conceptual view description

Here follows short descriptions of all the components of our conceptual view and their link to the strategies we decided on.

### User Interface

The user interface that the tester will use to setup the test bench before the testing can begin. It will also be responsible for showing the final report of once the test is done.

### I/O

This module handles the different types of predefined input and output.

#### User input

This part of the module that handles the predefined input. It uses different components for each input-type that in turn have a common interface such as was decided by the strategy mentioned in the issue-card for *"Multiple input issue"* , strategy: **"***Generalize the module".*

#### Output validation

This part of the module takes care of validating the output from the tested system against the predefined output. Like the Use input module it will house separate components for each type of output validation and in turn have a common interface. It will also create the final report once the data log has been received, which is mentioned on the issue-card *"Creating report issue".*

### Data Broker

This module is responsible for transferring data between the other modules. It houses two components, the data distributor and the message handler.

#### Data distributor

This component handles sending data between the different modules of the system. Having a central data distributor takes care of the issue mentioned in the issue-card *" Data transfer".*  It will also handle the issue with connecting the data log and the output validation for creating the final report that is mentioned in the issue-card *"Creating report issue".*

#### Message handler

Messages about a crash in the tested system will be handled in this component. It will then send out a message to the rest of the system telling them to halt the testing until it receives an "ok"-message from the test wrapper to initialize the testing again. This takes care of the issue mentioned in the issue-card *"Not crashing with tested system issue".*

### Emulation

This module holds the two separate modules for handling software and hardware emulation.

#### Software emulation

Holds the different interface-components for channeling different types of software data. The issues mentioned in the issue-card *" Multiple software emulations issue"* will be handled here by maintaining semantic coherence between the interfaces and by exploring standards in software communications.

#### Hardware emulation

Holds the components for emulation the different hardware devices, one component for each device. Like the Software module mentioned above it will solve the issues of the issue-card *" Multiple hardware emulations issue"*  by maintaining semantic coherence between components and by exploring standards for hardware communication.

### Datalog component

This component receives all the data from the testing and stores it so that it can later be made into the final report by the output-component which is mentioned by the issue-card *"Creating report issue".*

### Test wrapper

This modules holds two components, one for keeping a checkpoint updated in case of a crash in the tested system, and one component is just to represent the tested system itself.

#### Checkpoint component

This component keeps a checkpoint for the tested system. If the tested system crashes it will be restored by the checkpoint held in this component. This solves the issue mentioned in the issue-card *"Keeping the system running through a test crash"* by using the strategy mentioned in the same card,

Strategy: *“Checkpoint/Rollback”* .

#### Tested system

Representation of the tested system.

## Conceptual View

