### 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*" from "*Software Architecture in Practice Second Edition*" Chapter 5.3 Len Bass, Paul Clements, Rick Kazman 2003.  By ensuring that the input type does not affect the functionality of the methods in the different modules it will be possible to add several different kinds of input. |

|  |
| --- |
| **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"* from *"Software Architecture in Practice Second Edition"* Chapter 5.3 Len Bass, Paul Clements, Rick Kazman 2003  By ensuring that each module is strictly separated from the others a change in one hardware emulation will not interfere with the other emulations and it would thus enable the addition of new modules without too much fuss. |

|  |
| --- |
| **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 hardware emulations without impacting the system. |
| **Strategies/Tactics:**  "*Maintain semantic coherence*" and "*Anticipate expected changes*" from "*Software Architecture in Practice Second Edition*" Chapter 5.3 Len Bass, Paul Clements, Rick Kazman 2003  By ensuring that the software emulations are strictly separated it will be possible to add new emulations without being hindered by already existing emulations. |

|  |
| --- |
| **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"* from *"Software Architecture in Practice Second Edition"* Chapter 5.3 Len Bass, Paul Clements, Rick Kazman 2003  By making sure that each testing technique is separated from the others the addition of new techniques would not disturb the already existing techniques. |

|  |
| --- |
| **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*” from *"Software Architecture in Practice Second Edition"* Chapter 5.2 Len Bass, Paul Clements, Rick Kazman 2003  A checkpoint records the state of the system and will be loaded on the event of a crash. |

|  |
| --- |
| **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”  A Broker works as the postal service, sending messages between the different components and the system. This keeps the coupling low. |

|  |
| --- |
| **Name:**  Multiple crash issue |
| **Description:**  When the tested system crashes repeatedly so that the test can’t continue the testing needs to be terminated  **Factors:**  P4.2 Restarting from checkpoint  P5.1 Robust system |
| **Solutions:**  The wrapper around the tested system will keep track of which checkpoint was the last to be used to restart the tested system. If the same checkpoint is used more then a predefined amount this will signal that the testing needs to be terminated. The wrapper then send a message to stop the testing up to the message handler in the data broker. |
| **Strategies/Tactics:** |

|  |
| --- |
| **Name:**  “Quis custodiet?” issue |
| **Description:**  Apart from being run atomatic the system needs to support a monitored modewhere a user can view all the test data and script new input to the MIB  **Factors:**  P2.2 Monitor MIB during runtime  P2.3 Scriptable input |
| **Solutions:**  A Separate scripting component will feed scripted input into the input component for further transfer down to the tested system. Sending testing data up to be viewed during runtime will be handled by the data broker |
| **Strategies/Tactics:** |

|  |
| --- |
| **Name:**  Parallel testing issue |
| **Description:**  The MIB must be able to run separate testing instances of the same system which means keeping check on which data goes where and making sure that only data going to crashed instances are halted, not data that is going to the still running instances.  **Factors:**  P1.7 Run parallel tests |
| **Solutions:**  This will require having several testing subsystem, one for each parellel run of the tested system. They will need a unique identification so that the checkpoint module and the data broker knows which data goes to which subsystem. |
| **Strategies/Tactics:** |