**A Montana Tech Method Software Development Standard**

**MTM Small Module Development Sheet (SMDS) Template**

Version 3.1  
September 16, 2014

A. Frank Ackerman

*Software Engineering  
Montana Tech*

|  |  |  |  |
| --- | --- | --- | --- |
| *Version* | *Date* | *Author* | *Comment* |
| 2.2 | 05/23/10 | Frank Ackerman | Added running tests |
| 2.3 | 06/13/10 | Frank Ackerman | Added estimates and delivery |
| 2.4 | 07/25/10 | Frank Ackerman | Added testing, inspection, and correctness argument requirement |
| 2.5 | 08/31/10 | Frank Ackerman | Expanded Input/Output requirements description |
| 2.6 | 09/16/10 | Frank Ackerman | Revised process description to conform with ESOF322 text. |
| 2.7 | 09/30/10 | Frank Ackerman | Minor corrections |
| 2.8 | 11/10/10 | Frank Ackerman | Added matrix to cover testable requirements |
| 2.9 | 04/22/11 | Frank Ackerman | Complete specification for requirements |
| 2.10 | 12/10/11 | Frank Ackerman | Update to current practice |
| 2.11 | 01/22/12 | Frank Ackerman | Minor corrections |
| 2.12 | 07/27/12 | Frank Ackerman | Added requirements trace (if a correctness argument is not done). Also work product reference to standards version |
| 2.13 | 09/11/12 | Frank Ackerman | Clarify requirements trace. |
| 2.14 | 10/03/12 | Frank Ackerman | Move general requirements comment up |
| 2.15 | 12/16/12 | Frank Ackerman | Add additional size estimates and actuals |
| 2.16 | 05/25/13 | Frank Ackerman | Minor corrections |
| 3.0 | 03/06/14 | Frank Ackerman | Modifications for ASEE&T 2014 |
| 3.1 | 09/16/14 | Frank Ackerman | Strengthened description of Logical Test Conditions |

**Montana Tech Software Engineering Students:**

These Montana Tech Method software engineering standards encapsulate Dr. Ackerman’s decades of development experience in the software industry at best-practice organizations; writing, reviewing, and teaching the IEEE software engineering standards world-wide, and many suggestions from various texts and Montana Tech students. These standards have gone through many revisions and additions over several years. They are part of your software engineering studies so that (1) you may have the experience of developing software to a standard (which you may find you need to do if you take a job that requires that you develop high quality software), and so that (2) you will actually have the experience of developing high quality software. You are also invited to participate in the continuing evolution of these standards by studying them critically and making suggestions for their improvement.

# Introduction

This standard consists of three parts:

1. This brief introduction
2. A description of a process that could be used to develop a small module using this template
3. An annotated template

To use this template a developer would copy this document file, delete the first 8 pages, and delete “Template” from the document header. A developer’s initial page would look like this:



Application:

This standard applies to all small module student software engineering development projects under the sponsorship of the Montana Tech Computer Science Department.

Typographical conventions:

In this standard (and in an SMDS created using this template):

* Courier new font is used to refer to text that will appear in just this way in an actual SMDS or in the code
* In addition to the usual use of italic, a name in *Courier new italic* is a generic name, i.e., in an actual implementation a specific name would be used.

Process

This section describes:

1. The development activities associated with each section of a *Montana Tech Method* *Small Module Development Sheet*
2. The controls associated with these activities

## **Activities**

A *Montana Tech Method* *Small Module Development Sheet* (SMDS) consists of the following parts:

1. a document header
2. a reference to the version of the SMDS standard that this document complies with.
3. a program description header
4. estimates and actuals of size, effort, and delivery date; and comments about these estimates and actuals
5. a requirements section
6. a logical test conditions section
7. a test cases section
8. an inspection report(s) reference
9. a design section
10. a reference to the program code
11. a requirements trace section
12. a correctness argument section
13. a test directories, files, and scripts/scenarios section
14. an optional random test generation section
15. an optional performance test section
16. a test reports section
17. a deliverables section
18. a version history table

The activities associated with each of these parts is described in detail below.

### MTM document header

A version of the standard MTM software engineering document header is used. The data items to be included are: version number, version date, program name, and developer name.

### Reference to standard version

Within any software standards organization standards evolve as conditions change and the needs to be served by each standard are better understood. Hence each work product developed in compliance with a standard should cite the standard version.

### Brief description of the program

The brief description should provide a useful description of what the program does. All of the details of the program's functionality are given in the following **requirements** section.

### Estimates and actuals of program size, effort, and delivery date

Initial versions of this document provide size, effort, and delivery date estimates. The final version in addition provides actual size, effort, and completion dates. As you proceed in your development, and especially at its conclusion briefly discuss your estimates and actuals: What was the basis for your size estimates? How was the estimated delivery date arrived at? If your size estimates were off by 10% or more, what factors caused that? What factors prevented you from meeting your estimated delivery date?

### Specify the requirements

The requirements are succinct, sequentially labeled statements of all of the characteristics and behaviors the module should have when it is completed. Each statement should be brief, unambiguous, and specify a validatable attribute. The set of statements should be consistent and specify all of the characteristics and behaviors of interest to a user or customer.

The priority of each requirement (Imperative, Important, or Desired) should either be stated with each requirement or, as will generally be the case for simple modules, stated immediately following the **Requirements** heading as applicable to all the requirements. The manner in which each requirement will be validated (Not specified, Guaranteed, Observation, Inspection, or Test) should be given on a separate line immediately following the requirement.

Test implies program execution. Inspection refers to an MTM SMDS inspection as define in the MTM SMDS inspection standard. Some requirements may simply be validated by observation that is not necessarily part of an inspection. “Guaranteed” is used when that attribute is guaranteed by an external source. “Not specified” is used when the module’s behavior depends on input from an external source and this input is not fully specified.

See *MTM For Small Module Development Overview* for additional comments.

### Construct the program logical test conditions

See the **Logical Test Conditions** section in the template below.

### Select the module test cases

See the **Test Cases** section in the template below*.*

### Hold an inspection

At least one inspection should be held for every module developed using an SMDS. For really logically simple modules a single final inspection when the SMDS is complete may suffice. For more complex modules intermediate inspections may be held when the requirements and the associated test conditions and test cases are complete. Similarly an inspection might be held when the design is complete, or when the correctness arguments are complete. If intermediate inspections are held the final inspection would include the test process and results. The *MTM Small Module Program Development Sheet Inspection Process* describes, and provides forms for these inspections.

### Describe the design

Design elements are stated using the Unified Modeling Language class diagrams, state charts, and sequence diagrams and/or the *MTM Design Language For Functions/Methods* standard. It should be possible to conceptually execute the inputs described in the requirements to obtain the outputs described in the requirements. In particular it should be possible to mentally “execute” the requirements use cases.

### Reference the code

Code is best kept in compilable files. Reference those files here.

### Provide a requirements to design elements trace

Tracing program behavior requirements to design elements is standard practice for good software engineering. Any program behavior requirements that do not have associated design elements are either unnecessary or the design is not complete

### Construct Correctness Arguments

A correctness argument lists each requirement, cites the relevant design elements that implement that requirement (possibly citing details of the code that implements those elements) and where appropriate explains why the design element and code correctly implements that requirement. A correctness argument may be constructed after just the design is complete or it may be constructed after both the design and code are complete. When it is constructed after the code is complete, code elements can be checked for compliance with design elements. For example:

Correctness Argument for main():

C01 *Brief explanation of why design/implementation satisfy main  
 requirement R01*.

C02 *Brief explanation of why design/implementation satisfy main  
 requirement R02*.

C03 ....

Correctness Argument for Function X:

C01 *Brief explanation of why design/implementation satisfy Function X  
 requirement R01.*

C02 *Brief explanation of why design/implementation satisfy Function X  
 requirement R02.*

C03 ....

### Develop the test files/scenarios

See the **Test Files/Scenarios** section below

### Program to generate random tests (if so required)

Provide an MTM standard program for generating random tests. A reference to a file containing this program is sufficient provided it contains all the information necessary to understand the implemented random test generation process.

### Procedure for performance testing (if so required)

If the module requirements include performance requirements, the validation of these requirements may require separate testing and the collection of performance data demonstrating that these requirements are meet.

### Test the program

Successfully execute, ideally using regression test scripts, the test files/scenarios developed in section 2.1.13 above and display the results in this section or in *Small Module Test Reports.*

### Hold an inspection

If a module inspection was called for in the requirements *\*\*Constraints* section the process described in *MTM Small Module Development Sheet Inspection Process* should be carried out

### List the directory/folder delivery structure

All the work products that are to be delivered should be listed in a directory/folder structure that is appropriate for configuration management and subsequent maintenance.

### Version history table

A version history table similar to the version table on the first page of this standard concludes an *MTM Small Module Development Sheet*.

## **Control**

Process control is provided by the *MTM Small Module Development Sheet Inspection Process*. Where an inspection is not required the developer’s supervisor should review the completed development sheet.

### Work Products

The work products are specified in the template **Deliverables** section.

### Resources

The resources needed to perform this process are:

* This template standard
* A written or verbal problem description or requirements in a higher level document.
* A developer, or at most two or three developers
* Inspectors when inspections are a requirement
* The tools described below

### Tools

The tools needed to perform this process are:

* A word processor
* Where so required, tools for creating and executing random test case generators
* Where so required, tools for creating and executing test scripts
* Where so required, test coverage tools
* Where so required, tools for performance measurement

# Small Module Development Sheet

This document is in compliance with SMDS standard version *3.0*

|  |  |
| --- | --- |
| **Description:** | Brief program description |

**Development Estimates/Actuals**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Item** | **Est Sz** | **Act Sz** |  | **Strt Date** | **Est Cmplt** | **Act Cmplt** |  | **Est Effrt** | **Act Effrt** |
| Reqs |  |  |  |  |  |  |  |  |  |
| Test Cons |  |  |  |  |  |  |  |  |  |
| Test Cases |  |  |  |  |  |  |  |  |  |
| Design |  |  |  |  |  |  |  |  |  |
| # Classes |  |  |  |  |  |  |  |  |  |
| # Methods |  |  |  |  |  |  |  |  |  |
| # Dsng Elms |  |  |  |  |  |  |  |  |  |
| NCLC |  |  |  |  |  |  |  |  |  |
| Correctness |  |  |  |  |  |  |  |  |  |
| Test |  |  |  |  |  |  |  |  |  |
| Inspec |  |  |  |  |  |  |  |  |  |
| Wrap up |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | Total Effrt: |  |  |  |

In this section you practice an important software engineering skill: estimating sizes, effort, and completion dates. Estimates should be made in this order. These need not be especially exact, but if you work through this sheet at different times you will need to keep at least an informal log of your time and what you did. You will probably find that you will not work on these sections strictly sequentially. You should track your effort well enough that you can estimate the total effort on each part.

Estimates/Actuals Comments

* Comment
* ...

**Requirements**

*More than one requirement (Rxx) may be given in each of the categories listed below. Complex requirements may require subparts: (a), (b), ...*

*Each requirement should include a Validation statement that indicates how the requirement will be validated (or not): (*not specified*,* guaranteed, observation, inspection, test).

*If all of the requirements are at the same level of priority (*Imperative*,* Important*, or* Desired*) then that should be stated here.*

*\*\*Constraints*

1. *Constraints are external conditions that need to be satisfied by this program. For example, that the program be coded in a specified language and its documentation adheres to specified standards.*

*\*\*Preconditions*

1. *If there are any system conditions (eg. files, libraries, network connections, etc) that should be true before the program is launched they are given here. If there are none only "None." is entered (no requirement).*

*\*\*Invocation*

1. *This requirement describes how the program will be invoked.*

*\*\*Input/Ouput*

1. *Most of the requirements will be in this section. What is the syntax and semantics of the input items? What is the syntax of the output items? How are the input and output items related. What headers and trailers are required?*

*\*\*Postconditons*

1. *If there are any system conditions (eg. files, libraries, network connections, etc) that should be true after the program closes they are given here. If there are none only "None." is entered (no requirement).*

*\*\*Testing*

1. *Must test files be used, and if so, are test scripts required? Is scenario testing acceptable? Is the use of a test coverage tool required, and if so what are the coverage requirements? Is a random testing required? Are any performance tests required?*

*\*\*Inspection*

1. *Is an inspection necessary?*

*\*\*Correctness Argument*

1. *Is a correctness argument required?*

Example Runs

*If any of the following use cases adds any details to the above requirements this should be noted here.*

*Example runs follow the last requirement. At this level of specification they are sample runs or user scenarios. Ideally these examples should cover all the high level initial conditions, and where possible the handling of all errors. These examples should be included in the Test Cases section below. Usually more than example case should be given. Each use case should be given a name or descriptive phrase. A table should follow the examples that shows how each testable requirement is covered by an example:*

|  |  |
| --- | --- |
| **Testable Req** | **Illustrative Use Case** |
|  |  |

**Logical Test Conditions**

*This is a hierarchical list of test conditions that is relative to the program inputs where:*

* *The partitioning “makes sense” with respect to the module requirements*
* *Each test condition is a complete sentence*
* *Each level of the hierarchy is logically sound and forms a complete logical partitioning*
* *All leaf conditions define at least one verfiable test case, but not more than approximately 10 test cases*

**Test Cases**

*Test cases are explicit listing of test inputs and expected outputs that correspond to the leaves in the test conditions hierarchy. Unless you have reason to believe that the code that might be executed by a test that matches this test condition there should be at least one test case for every leaf logical test condition. To cover boundary conditions and variations in the input format you probably want more than one test case for most leaf conditions.*

**Design**

*The program design may be given here or in the source code. For very small programs it is can be given in the source code as part of the main program header. The design of a complex function is best given in the header for that function.*

*The simplest program designs are for procedural programs. Use the MTM Program Design Language For Function/Methods to describe these designs.*

*Complex or sophisticated functions should have their own REQUIREMENTS and DESIGN subheadings. These functions may have their own Correctness Arguments. For procedural programs the correctness argument for the main function can then just reference these arguments.*

*OO programs generally require a more complex design consisting of a number of UML diagrams with accompanying notes. OO programs that include logically complex or critical methods should document these methods as described above. For OO Programs this section should be divided into Design Overview and Detail Design subsections.*

**Code**

*Reference the source files (and/or) attach copies.*

**Inspectable Requirements Trace**

*Good software engineering practice documents the design elements that implement requirements that are validated by test or inspection. This can be done using a  
Inspectable Requirement→Design Elements Trace* Table*.*

*For an Inspectable Requirement→Design Elements* Trace *use one of the following table:*

|  |  |  |
| --- | --- | --- |
| **Requirement** | **Function(s)** | **Design Element** |
|  |  |  |

or for OO Modules:

|  |  |  |
| --- | --- | --- |
| **Requirement** | **Design Elements Classes and Objects** | **Method and Design Element** |
|  |  |  |

**Correctness Arguments**

Correctness Argument for Function X:

C01 Brief explanation of why design/implementation satisfy Function X  
 requirement R01.

C02 Brief explanation of why design/implementation satisfy Function X  
 requirement R02.

C03 ....

**Test Directories, Files, and Scripts/Scenarios**

*All formal tests should be run using test files and scripts that where possible compare test run output with validated expected test output. Where files & scripts cannot be used complete test scenarios should be described.*

**Random Test Generation**

*This will usually be a program that generates random test cases*

**Performance Test Procedure**

*Performance tests will typically require controlled system conditions and the collection of performance data.*

**Test Report**

*For a test script, display the output. Similarly test scenarios should describe or display the expected output*

**Inspection Report(S)**

*Any Inspection Notice(s), Inspection Defect List(s), and Inspection Report(s) should be included in the deliverables.*

**Deliverables**

*Most deliveries of even small modules will involve several directories. List the files and the directory structure.*

**Version History**

Most real world programs will undergo some modification during their lifetime. The initial program version and any subsequent modifications should be documented here.

|  |  |  |  |
| --- | --- | --- | --- |
| *Version* | *Date* | *Author* | *Comment* |
|  |  |  |  |
|  |  |  |  |