SYSTEM AND SOFTWARE DESIGN DESCRIPTION (SSDD): Incorporating Architectural Views and Detailed Design Criteria
FOR

Project: UML

Version 1.1
[12/16/2011]

Prepared for:
Professor Bruce Bolden, Dr. Clinton Jeffery

Prepared by:
Coleman Beasley, Alex Dean, Austin Enfield, Jason Fletcher, Theora Rice, Cable Johnson, Adrian Norris, David Summers

Emeritus Contributors:
Noel Klein, Max McKinnon

University of Idaho
Moscow, ID 83844-1010

CS383 SSDD
RECORD OF CHANGES (Change History) (N/A)

Change Number	Date completed	Location of change (e.g., page or figure #)	A M D	Brief description of change	Approved by (initials)	Date approved
01	2012.02.02	line 1107	A	Added class diagram based on current code.	DS	

A - ADDED M - MODIFIED D – DELETED

PUML
TABLE OF CONTENTS

Section Page

1 INTRODUCTION (Alex)

1.1 IDENTIFICATION

This document describes the first version of pUML. pUML is a UML diagram drawing tool that will operate under any system able to compile it with QT Creator 4.7.

1.2 DOCUMENT PURPOSE, SCOPE, AND INTENDED AUDIENCE

1.2.1 Document Purpose

This Software Design Document provides the design details of Project Unified Modeling Language (pUML).

1.2.2 Document Scope and/or Context

This document contains a complete description of the design of pUML. The project is written with C++ and the Qt framework. The project leader designated is Coleman Beasley. He will have full access to make changes as he deems necessary.

1.2.3 Intended Audience for Document

The intended audience is Bruce Bolden, the pUML developers, and the people who maintain pUML.

1.3 SYSTEM AND SOFTWARE PURPOSE, SCOPE, AND INTENDED USERS

1.3.1 System and Software Purpose

This software provides the user with a quick and efficient means to construct high quality UML diagrams.

1.3.2 System and Software Scope/or Context

The scope of this application is provide the user with a means to construct high quality UML diagrams. This includes implementing the following features at a minimum: spacing

- All shapes needed for standard UML diagrams (including an actor shape)
- Line objects for creating connections
- Text areas to write comments for the diagrams
- Ability to save and load projects
- Exportability to PDF format

1.3.3 Intended Users for the System and Software

The intended users for this application are students and professionals seeking an advanced tool to construct a UML diagram.

1.4 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS

Term or Acronym	Definition
Acquirer	The person, team, or organization that pursues a system or software product or service from a supplier. The acquirer may be a buyer, customer, owner, purchaser, or user. ISO/IEC 42010:2007 (§3.1).
AD	Architectural Description: "A collection of products to document an architecture" ISO/IEC 42010:2007 (§3.4).
Alpha test	Limited release(s) to selected, outside testers
Architect	"The person, team, or organization responsible for systems architecture" ISO/IEC 42010:2007 (§3.2).
Architectural Description	(AD) "A collection of products to document an architecture" ISO/IEC 42010:2007 (§3.4).
Architectural View	"A representation of a whole system from the perspective of a related set of concerns" ISO/IEC 42010:2007 (§3.9).
Architecture	"The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution" ISO/IEC 42010:2007 (§3.5).
Beta test	Limited release(s) to cooperating customers wanting early access to developing systems
Design Entity	"An element (component) of a design that is structurally and functionally distinct from other elements and that is separately named and referenced" IEEE STD 1016-1998 (§3.1).
Design View	"A subset of design entity attribute information that is specifically suited to the needs of a software project activity" IEEE STD 1016-1998 (§3.2).
Final test	aka, Acceptance test, release of full functionality to customer for approval
DFD	Data Flow Diagram
SDD	Software Design Document, aka SDS, Software Design Specification
Software Design Description	"A representation of a software system created to facilitate analysis, planning, implementation, and decision making, A blueprint or model of a software system. The SDD is used as the primary medium for communicating software design information" IEEE STD 1016-1998 (§3.4).
SRS	Software Requirements Specification
SSDD	System and Software Design Document
SSRS	System and Software Requirements Specification
System	"A collection of components organized to accomplish a specific function or set of functions" ISO/IEC 42010:2007 (§3.7).
System and Software Architecture and Design Description	An architectural and detailed design description that includes a software system within the context of its enclosing system and describes the enclosing system, the enclosed software, and their relationship and interfaces.

Term or Acronym	Definition
System Stakeholder	"An individual, team, or organization (or classes thereof) with interests in, or concerns, relative to, a system" ISO/IEC 42010:2007 (§3.8).

1.5 DOCUMENT REFERENCES

1) CSDS, *System and Software Requirements Specification Template*, Version 1.0, July 31, 2008, Center for Secure and Dependable Systems, University of Idaho, Moscow, ID, 83844.

2) ISO/IEC/IEEE, *IEEE Std 1471-2000 Systems and software engineering – Recommended practice for architectural description of software intensive systems,* First edition 2007-07-15, International Organization for Standardization and International Electrotechnical Commission, (ISO/IEC), Case postale 56, CH-1211 Genève 20, Switzerland, and The Institute of Electrical and Electronics Engineers, Inc., (IEEE), 445 Hoes Lane, Piscataway, NJ 08854, USA.

3) IEEE, *IEEE Std 1016-1998 Recommended Practice for Software Design Descriptions*, 1998-09-23, The Institute of Electrical and Electronics Engineers, Inc., (IEEE) 445 Hoes Lane, Piscataway, NJ 08854, USA.

4) 3) ISO/IEC/IEEE, *IEEE Std. 15288-2008 Systems and Software Engineering – System life cycle processes,* Second edition 2008-02-01, International Organization for Standardization and International Electrotechnical Commission, (ISO/IEC), Case postale 56, CH-1211 Genève 20, Switzerland, and The Institute of Electrical and Electronics Engineers, Inc., (IEEE), 445 Hoes Lane, Piscataway, NJ 08854, USA.

5) ISO/IEC/IEEE, IEEE Std. 12207-2008, *Systems and software engineering – Software life cycle processes,* Second edition 2008-02-01, International Organization for Standardization and International Electrotechnical Commission, (ISO/IEC), Case postale 56, CH-1211 Genève 20, Switzerland, and The Institute of Electrical and Electronics Engineers, Inc., (IEEE), 445 Hoes Lane, Piscataway, NJ 08854, USA.

6) Qt Development Frameworks Documentation. [Online] Available: http://doc.qt.nokia.com/ 6) Qt Development Frameworks Documentation. [Online] Available: http://doc.qt.nokia.com/

1.6 DOCUMENT OVERVIEW

Section 2 of this document describes the system and software constraints imposed by the operational environment, system requirements and user characteristics, and then identifies the system stakeholders and lists describes their concerns and mitigations to those concerns.

Section 3 of this document describes the system and software architecture from several viewpoints, including, but not limited to, the developer's view and the user's view.

Section 4 provides detailed design descriptions for every component defined in the architectural view(s). Sections 5 provides traceability information connecting the original specifications (referenced above) to the architectural components and design entities identified in this document.

Section 6 and beyond are appendices including original information and communications used to create this document.

1.7 DOCUMENT RESTRICTIONS

This document is for LIMITED RELEASE ONLY to UI CS personnel working on the project, Bruce Bolden, and his associates.

2 CONSTRAINTS (Noel)

This section identifies and describes in detail the architectural and usability constraints that are imposed by the physical environment and system requirements and the user characteristics.

2.1 Environmental constraints.

The tool can be used in every environment that satisfies the system requirements defined in the next subsection. Since it is a software design tool, it will most likely be in a software design environment.

2.2 System requirement constraints.

The tool requires normal x86 hardware and runs on Windows, Mac OSX, and Linux.

2.3 User characteristic constraints.

It is a tool that supports the creation of UML diagrams and therefore requires the knowledge of how UML diagrams work.

3 SYSTEM AND SOFTWARE ARCHITECTURE (Theora)

3.1 DEVELOPER'S ARCHITECTURAL VIEW

From the developer's point of view, the architecture is modular, based around separate tool bars and windows gathered around a single menu. It has been written in C++. The Drawing Area module is where the events of the program are centred and displayed. It uses different methods for mouse-movement and dragging events. It also contains a paint method to illustrate the final output shape. The objects it displays come from the objects class, which holds the various subclasses that contain each shape. Each of these subclasses contains only their own creation method.

The Toolbar is the next most important module, as it controls the drawing of objects and their properties afterwards. It triggers inserting an object with the insertShape() and insertLine() methods, as well as contains the method that calls the OptionsDialog.

The Options Dialog updates each shape object with the new user-chosen colours, text, and line weight. It is a user interface where they will be presented with a selection of options to choose from to modify their diagram.

Objects will be represented within one common class, that will contain links to each shape's individual subclass. These subclasses will contain the shape's properties such as position, colour, size, etc., as well as their own creation method.

The File Method contains any information about the file itself, and will default save the file to a database as an XML file. It also has open(), savePDF(), and BMP() for opening files and saving them to different formats. This will also contain the list of recent user actions for the undo and redo functionality. These actions will be kept in a linked list for reference.

3.2 USER'S ARCHITECTURAL VIEW

From the user's viewpoint, there are three separate modules: The Main Window, the Drawing Area, and the Toolbar. The Main Window will act as background, which the other components will use to orient themselves. This window will contain the file management buttons, such as save, open, undo, redo, and help.

The Drawing Area will be where users get to manipulate the individual objects and the overall diagram they create. The background will be a plain white drawing space, across which the users will be able to drag their objects. By left clicking each object, users will be able to access the Options Dialogue box, where they will be able to change the line width, fill colour (for non-lines,) font (for text boxes,) and line colour. Thus, the Drawing Area controls where one can put the shapes, the Options dialogue changes the shape's properties and the shapes keep track of their own values.

The Toolbar allows the user to create shapes on the Drawing Area, and access the Options Dialogue box to change the default value of a shape before it is created. It will also be used to toggle the Grid on and off the Drawing Area.

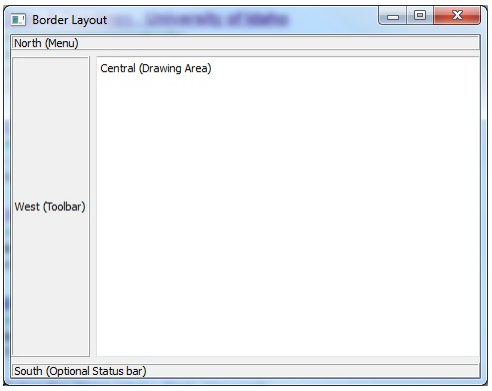
3.3 User's View Identification

From the User's point of view, things are mostly GUI based for easy utilization. Within the top level of the architecture, the Main Window, there are file operation processes. The save button will summon a drop down menu that contains buttons to (default) save the file as an XML document, export to PDF, export to BMP, and exit without saving. The open button will automatically bring up a directory dialogue so that the user can choose files. Undo will reverse the previous user action, and redo will reinstate it. The help button will contain a drop down menu with buttons that summon the "About"dialogue window and the User Manual Dialogue. The About dialogue will display general information about the program and who created it. The User Manual Dialogue will contain the usual Help window, in which users can manipulate different tabs to get the information they need in order to use the program.

The Drawing Area will contain the processes that allow for manipulation of the shapes and diagram. By left clicking on a shape and holding down on the mouse button, the user can drag shapes across the background and change their coordinates. By dragging down on the bottom right hand corner of these objects, users will be able to resize them. Multiple shapes can be selected by clicking on each in turn while holding the ctrl key. All above processes can be performed upon these group selected shapes as well.

The Toolbar contains the processes that will be used to signal the creation of a shape, line, and text box. It also contains the ability to open the Options Dialogue, and the button to toggle the grid on and off.

3.3.1 User's View Representation and Description



The Toolbar will reside to the left of the screen, the file manipulation will be on top, and the drawing area will be the bulk of the screen.

3.4 Developer's View Identification

From the point of view of the Developer, the architecture makes it easier to separate processes into separate categories.

File operation resides in the main window, and takes care of converting file formats to XML, PDF, and BMP. The Open function accesses the home directory so that users may use previously stored files. Undo and Redo take from two stacks to retrieve their commands. The Help button summons a widget to explain the program.

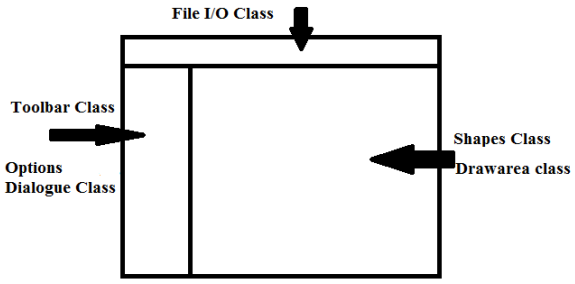
The Drawing Area keeps track of the shapes. It contains a list of each shape object and it's properties (size, shape, color, etc). The QGraphicScene() it is composed of allows the user to drag QGraphicsItems around the background. It will also allow users to right click on objects to summon the options widget. The linked list of objects will be referred if multiple objects need to be selected. Therefore, Drawing Area is higher in the hierarchy than the objects subclass.

Toolbar can affect the Drawing Area, in that it summons new instances of shapes to appear and displays the grid. It can also trigger the Options widget which allows the user to change shape properties.

By using this architecture, programming can be easily broken up into different sections and responsibilities. That way, multiple people can work on different areas without having to constantly rely on each other for input.

3.4.1 Developer's View Representation and Description

Developer's View Representation andDescription



3.4.2 Developer's Architectural Rationale

There are many reasons behind the selection of this structure. One is that it has been implemented in similar forms in other popular graphics tools. It is graphically similar to the layout of Paint, and other popular UML diagramming programs. It is also more modular to code, as different areas contain different responsibilities, and thus can be coded by individuals in their own time.

3.5 CONSISTENCY OF ARCHITECTURAL VIEWS

For compliance with ISO/IEC 42010:2007 (§5.5) an Architectural Description (AD) shall include a list of all known inconsistencies between the architectural views and an analysis of consistency across all the architectural views.

3.5.1 Detail of Inconsistencies between Architectural Views

There is consistency between the Architectural views, though the two focus on separate qualities. One inconsistency, however, is that the Developer's architecture stresses the modularity of the code, without focusing much on the connections between the modules. The user's code, however, focuses on how each section can be used in order to create a final product.

3.5.2 Consistency Analysis and Inconsistency Mitigations

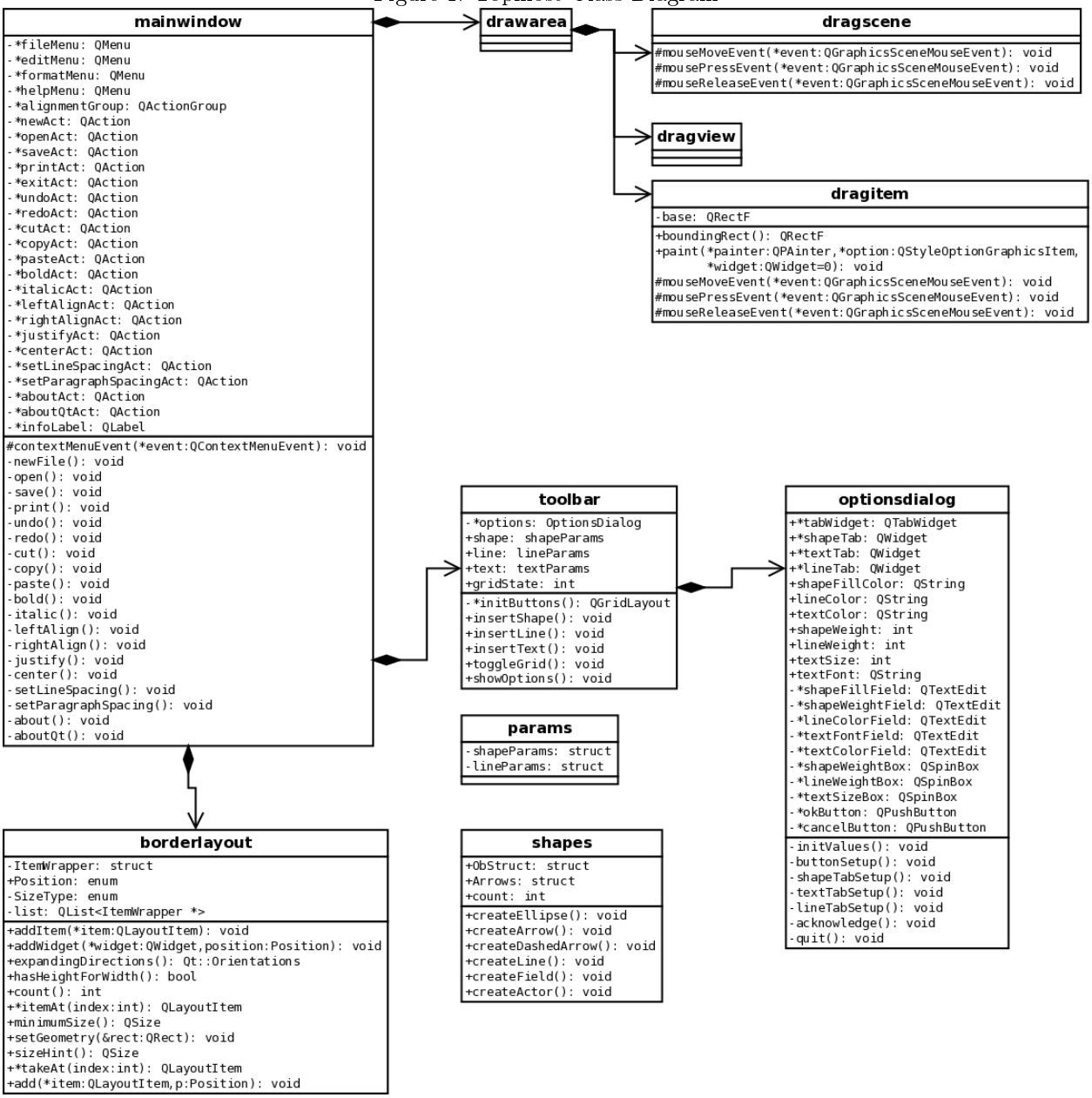
A possible solution to the inconsistency listed above is to use group meetings to focus on creating a smooth connection between parts. If each member of the team works as efficiently as possible on their individual tasks, then the hours spent together can be to work out bugs between modules as they are linked up to form the final program.

4 SOFTWARE DETAILED DESIGN

4.1 DEVELOPER'S VIEWPOINT DETAILED SOFTWARE DESIGN

A modular approach was preferred, separating the application into different distinct components that could operate independently. This allowed different developers to work on their own aspects of the program independently, with an emphasis on the Drawing Area, Toolbar/Options, and the Main Window, allowing the team to maximize work versus time.

Figure 1: Topmost Class Diagram



4.2 COMPONENT/ENTITY DICTIONARY

.

Component/Entity Dictionary				
Name	***Type/Range***	***Purpose/Function***	***Dependencies***	***Subordinates***
MainWindow	*widget*	*Manage overall layout*	*none*	*none*
Drawing Area	*widget*	*Create graphical view*	*none*	*none*
Toolbar	*widget*	*Manage buttons and change options*	*Options Dialog*	*none*

4.3 COMPONENT/ENTITY DETAILED DESIGN

4.3.1 Detailed Design for Component/Entity: MainWindow

4.3.1.1 Introduction/Purpose of this Component/Entity *Manage the other components' positions and sizes in relation to each other. Will also terminate the other components when closed.*

4.3.1.2 Input for this Component/Entity *User-generated resize/move/exit requests.*

4.3.1.3 Output for this Component/Entity *Positions and sizes for the other components.*

4.3.1.4 Component/Entity Process to Convert Input to Output *Calculate the appropriate sizes and positions based on a fixed ratio/layout.*

4.3.1.5 Design constraints and performance requirements of this Component/Entity *None.*

4.3.2 Detailed Design for Component/Entity: Drawing Area

4.3.2.1 Introduction/Purpose of this Component/Entity *Create and show the appropriate representation of all the objects for the current file in a graphical view.*

4.3.2.2 Input for this Component/Entity *Position and size of new objects from the user, various aesthetic features, such as font and color, from the toolbar options.*

4.3.2.3 Output for this Component/Entity *Graphical representation of all currently existing shapes.*

4.3.2.4 Component/Entity Process to Convert Input to Output *Call object creation functions from shapes with the appropriate parameters, which will then handle object creation and return an appropriate reference, to be used in drawing the object.*

4.3.2.5 Design constraints and performance requirements of this Component/Entity *None.*

4.3.3 Detailed Design for Component/Entity: Toolbar

4.3.3.1 Introduction/Purpose of this Component/Entity *Manage buttons for creating the various shapes, making sure that the specified shape is appropriate for the current diagram type. It will also manage the grid state, as well as default options, such as font, weight, or color.*

4.3.3.2 Input for this Component/Entity *User pushing a specific button, selecting new option values, or toggling the grid.*

4.3.3.3 Output for this Component/Entity *Button presses will signal the appropriate function, such as create new shape, save, or load. Changing the default options will result in all new shapes being created with those parameters. Toggling the grid will result in the grid becoming visible or invisible.*

4.3.3.4 Component/Entity Process to Convert Input to Output *For creating new shapes, the Toolbar will send a request to make a new object of the specified type. For options, the Toolbar will open an Option Dialog and allow the user to specify new changes, saving those changes when confirmation is received. For other buttons, appropriate functions will be called, signalling MainWindow or DrawArea if needed.*

4.3.3.5 Design constraints and performance requirements of this Component/Entity *None.*

4.4 DATA DICTIONARY

Data Dictionary				
Name	***Type/Range***	***Defined by...***	***Referenced by...***	***Modified by...***
ObStruct	*Data Structure*	*shapes.h*	*drawArea.cpp*	*none*
Arrows	*Data Structure*	*shapes.h*	*drawArea.cpp*	*none*
OptionsDialog	*QDialog*	*optionsDialog.h*	*toolbar.cpp*	*Toolbar*

5 REQUIREMENTS TRACEABILITY

This section shall contain traceability information from each system requirement in this specification to the system (or subsystem, if applicable) requirements it addresses. A tabular form is preferred, but not mandatory. A detailed mapping between requirements and constraints in the SSRS and architectural components and detailed entities in this SSDD is required. For compliance with ISO/IEC 15288:2008 (§6.4.3.3.c) an Architectural Description (AD) shall provide roundtrip traceability between the system and software requirements and the architectural design entities. All requirements and constraints within the SSRS shall map to a set of architectural entities. All entities in all the architectural views shall be associated with either a requirement or constraint in the SSRS or an architectural constraint within this SSDD.

Req No.	*SDD*	*Alpha Test Results*
UC1		*F*
UC2		*F*
UC3		*F*
UC4		*F*
UC5		*F*
UC6		*50% - grid displays, but button does not toggle*
UC7		*30% - several shapes can be placed*

Req No.	*SDD*	*Alpha Test Results*
UC8		*F*
UC9		*F*
UC10		*F*
Select shapes	*UC11*	*The user selects the shapes for further commands.*
Move shapes	*UC12*	*The user moves the shapes to a different location on the drawing area.*
Copy shapes	*UC13*	*The user copies the selected shapes to the clipboard.*
Cut shapes	*UC14*	*The user cuts the selected shapes to the clipboard. The shapes are removed from the drawing area.*
Paste shapes	*UC15*	*The user pastes the shapes from the clipboard into the drawing area.*
Edit shape properties	*UC16*	*The user edits a shape on the drawing area.*
Edit default properties	*UC17*	*The user edits the default properties of the UML tool.*

Priorities are: ***M****andatory,* ***L****ow,* ***H****igh*
SDD link is version and page number or function name.
Test cases and results are file names and ***P****ass/****F****ail or % passing.*