UGENE:
User-interface GENeration Engine

REQUIREMENTS DOCUMENT

<u>Authors</u>

Jonathan Benn
Leon McKernan-Milon

TABLE OF CONTENTS

1 READING THIS DOCUMENT 3

1.1 AUDIENCE 3
1.2 DEFINITIONS, ABBREVIATIONS AND ACRONYMS 3

2 PROJECT VISION 4

2.1 WHAT UGENE IS 4
2.2 WHAT UGENE OFFERS 4
2.3 WHAT MAKES UGENE UNIQUE 4

3 ACTORS 5

3.1 DEVELOPER 5
3.2 CLIENT 5
3.3 PEER APPLICATION 5

4 USE-CASES 6

4.1 CREATE APPLICATION 6
4.2 USE APPLICATION 7
4.3 UPDATE STATE 8

5 FUNCTIONAL REQUIREMENTS (FEATURES) 9

5.1 HIGH PRIORITY FEATURES 9
5.2 MEDIUM PRIORITY FEATURES 10
5.3 LOW PRIORITY FEATURES 12

6 NON-FUNCTIONAL REQUIREMENTS (QUALITIES) 14

6.1 ARCHITECTURE 14
6.2 MAINTAINABILITY 14
6.3 USABILITY 14
6.4 CODE 15
6.5 RELIABILITY 15
6.6 PORTABILITY 15
6.7 PERFORMANCE 16

1 Reading This Document

1.1 Audience

We assume that the reader is familiar with common software engineering terminology and concepts.

1.2 Definitions, Abbreviations and Acronyms

- **Actor:** an entity that makes use of UGENE or the Application.
- **Application:** A software program created using UGENE as a framework. For example, a computerized card game. Composed of a back-end, front-end and configuration files.
- **Back-End:** the Application's underlying logic and data; created by the Developer.
- **Client:** one of UGENE's actors; uses the Application.
- **Developer:** one of UGENE's actors; uses UGENE to create an Application.
- **End User:** *synonym for actor*.
- **Front-End:** the Application's human-computer interface. This part of the Application is largely the responsibility of UGENE.
- **Hot Spot:** The single screen pixel where the mouse pointer is really located (normally the hot spot can be logically construed from the mouse pointer's image).
- **Hot Zone:** The region of a Widget where if the hot spot is inside it, then the mouse is considered to be hovering over that Widget.
- **Mean Time To Failure:** the mean average of the amount of time from when the program is first run, to the time when the process and/or computer stops functioning.
- **Panel:** a distinct view of the model; a screen configuration. In part, it is composed of a collection of Widgets.
- **Peer Application:** an identical instance of the Application that communicates with other instances using a network protocol.
- **STL:** Standard Template Library; an indispensable C++ object and function library.
- **UGENE:** User-interface GENeration Engine.
- **Use-Case:** a detailed description of a task that an actor wishes to perform, and which UGENE and/or the Application can support.
- **User:** *synonym for actor*.
- **User-Interface Widget:** *synonym for Widget.*
- **Widget:** a user-interface component that the Client interacts with as part of the user experience. For example, a button.

2 Project Vision

2.1 What UGENE Is

UGENE is a rapid UI development framework for writing combined 2D/3D applications. Similar concept to Swing or XUL, but easier to use, more flexible, and includes 3D! UGENE is programmed in object-oriented C++, uses OpenGL, and currently runs on Windows.

2.2 What UGENE Offers

UGENE offers Windows C++ programmers everything they need to create an interactive application, including: 2D and 3D graphics, sound and input/output functionality. It makes use of OpenGL, the Standard Template Library (STL), the Win32 API, OGLFT, Freetype, glBMP and libpng.

For developers, UGENE's main advantage is that it allows the rapid deployment of an application, while minimizing the need to understand how to create a human-computer interface. Developers will not need to learn a library of functions, instead they will assemble high-level UGENE-provided components called Widgets.

One of UGENE’s key features is that it separates the application’s presentation from its logic. This simplifies the long-term maintainability of the application, since changes to the logic don’t require one to modify the user-interface as well (and vice versa). The separation also makes it easier for large teams to simultaneously work on a UGENE-powered application.

2.3 What Makes UGENE Unique

What sets UGENE's graphics engine apart from similar projects is that it focuses on offering high-quality 2D graphics, with the possibility of including 3D graphics as well. Most graphics engines that we've seen focus on high-performance 3D graphics and neglect 2D. This makes them useless for creating many genres of computer games. With UGENE we hope to rekindle the flame of many of the game genres that have disappeared in the last 10 years.

We also hope that the project will help usability engineers create high-fidelity user-interface prototypes. UGENE, unlike existing prototyping solutions (such as Visual Basic), will allow developers to create interfaces with a unique look and feel.

In general, UGENE is unique because it offers a complete, object-oriented solution to developers. UGENE will not lock designers into creating a certain type of application (such as a 3D shooter, or a console-style RPG). Our engine will accomplish this by using clever object-oriented indirection schemes, such as Object Factories and the Command pattern, in combination with flexible configuration files.

3 Actors

UGENE has two primary actors: the Developer and the Client. Each of these actors has very different and distinct needs. In addition, UGENE has a secondary actor: the Peer Application.

3.1 Developer

For Developers, UGENE's main advantage is that it allows the rapid deployment of an Application, while minimizing the need to understand how to create a human-computer interface. Developers will not need to learn a library of functions, instead they will assemble high-level UGENE-provided components called Widgets.

The Developer is a skilled C++ programmer. Please note that skilled does not mean *experienced*. We assume that the Developer understands advanced C++ constructs such as pointers, function pointers, inheritance and templates (in other words, all of the concepts described in any introductory C++ manual). At the same time, we assume that the developer does *not* know any C++ code libraries except for the Standard Template Library (STL).

The Developer uses UGENE to create an Application. UGENE largely takes care of the Application's front-end, while the Developer is solely responsible for creating the Application's back-end.

3.2 Client

The Client's use of UGENE is incidental—the Developer has created an Application that the Client would like to use, and that Application happens to have been made with UGENE.

We assume that Clients are familiar with the basics of computer use, such as moving and clicking a mouse and typing on a keyboard. We also assume familiarity with typical program interaction constructs such as buttons, input boxes, etc.

3.3 Peer Application

The Peer Application interacts with the Application (and consequently with UGENE) because it needs to send and receive Application data.

The Peer Application is an instance of the Application's program. It is functionally identical to the Application, however, it may contain different data and consequently it may behave in a different way.

4 Use-Cases

Use-cases are the single most important element of any requirements document. They capture the user's needs. Meeting these needs is the essence of any software application. If a program does not meet these basic needs, then it is destined to collect dust on a shelf. Hence, these use-cases are UGENE's most important requirements, above and beyond all others (functional, non-functional, etc.). Making sure that UGENE effectively supports its use-cases means ensuring that UGENE will actually get used.

In writing use-cases, it is always difficult for software developers to think of their product in terms of tasks. For them, it is much easier to decompose their product into functions. It has been especially difficult to analyze and understand the tasks that UGENE must support. However, it is essential that use-cases be task-based and not function-based, otherwise their utility is very limited since they do not represent genuine user requirements.

This section will describe UGENE's use-cases in detail. In fact, UGENE supports very few use-cases. However, the tasks that UGENE does support are very sophisticated and complex, which is why UGENE itself will have to be sophisticated and complex to match.

4.1 Create Application

Use-Case	**Create Application**
Description	Allows the actor to create a new Application, using UGENE as the framework.
ID Number	20
Priority	Highest
Actor(s)	Developer
Pre-Conditions	None
Flow of Events	1. Create a back-end for the Application** 2. Create an API 3. Construct Panels
Post-Conditions	An Application exists

** The sub-case of **Create a Back-End for the Application** is *not* UGENE's responsibility. Creating the back-end is the responsibility of the Developer. Insofar as UGENE is concerned, its requirement is to make the process of creating the back-end and connecting it to UGENE as painless as possible.

Sub-Case	**Create an API**
Description	Allows the actor to create an API for joining together his back-end to UGENE (the front-end), in order to create an Application.
ID Number	2020
Priority	Highest
Actor(s)	Developer
Pre-Conditions	A back-end exists
Flow of Events	1. Create API, which will be used by the front-end to inform the back-end of user events
Post-Conditions	An API exists

Sub-Case	**Construct Panels**
Description	Allows the actor to create new Panels.
ID Number	2040
Priority	Highest
Actor(s)	Developer
Pre-Conditions	A back-end and API exist
Flow of Events	1. Create new Panels, by repeating steps 2-4, as many times as necessary to create a complete user experience 2. Configure a new Panel 3. Create and configure the Panel's Widgets 4. Connect the Panel and Widgets to the back-end via the API
Post-Conditions	A series of new Panels exist

4.2 Use Application

Use-Case	**Use Application**
Description	Allows the actor to interact with the Application.
ID Number	40
Priority	Highest
Actor(s)	Client
Pre-Conditions	An Application exists
Flow of Events	1. Install Application 2. Start Application 3. Interact with Panels and their Widgets 4. End Application 5. Actor repeats steps 2-4 as desired 6. Uninstall Application
Post-Conditions	None

4.3 Update State

Use-Case	**Update State**
Description	Allows the actor to synchronize its state with the Application.
ID Number	60
Priority	Low
Actor(s)	Peer Application
Pre-Conditions	An Application exists, and is accessible across a TCP/IP network
Flow of Events	1. Peer Application connects to the Application 2. Send and/or receive state information 3. It may now do the following in any order: • Send and/or receive state information • Send and/or receive an update (*i.e.* delta) 4. The connection is closed
Post-Conditions	The Peer Application and/or the Application have an updated state

5 Functional Requirements (Features)

UGENE will offer many features. They are sorted by priority. High Priority features must be implemented before a release can be labeled as a Beta release. A Production release must contain all High Priority and Medium Priority features. Low Priority features will be implemented later, as time permits.

5.1 High Priority Features

UGENE has the following High-Priority features:

5.1.1 2D graphics capability

5.1.2 Images can be displayed as texture-mapped rectangles.

5.1.3 Alpha channel transparency support.

5.1.4 Support for loading textures from GIF, JPG, BMP and PNG files.

5.1.5 Limited 3D graphics capability, in the form of 3D models. Lighting is not required.

5.1.6 The 2D graphics can be overlaid on top of 3D graphics.

5.1.7 Moveable camera for viewing 3D graphics from different angles.

5.1.8 Panel configuration files that specify what will be displayed to the screen, and how Panels interact with the Client. These files allow the user interface to be changed without recompiling the program. Panel configuration files must:

   a. Have a version number, to facilitate backward-compatibility of back-ends to new versions of UGENE.
   b. Include the ability to write comments. Comments are indicated by **~!~** (tilde, exclamation mark, tilde), in which case the remainder of the line is ignored by the parser.
   c. Have the capability to run a list of initialization commands.
   d. Permit the Panel's attributes to be set.
   e. Allow the Developer to list the Widgets contained in the Panel, as well as their characteristics.
   f. Permit the Developer to select the default focused Widget.
   g. Include the ability to create Widget groups, which can be simultaneously accessed.

5.1.9 Ability to display 2D text on the screen.

5.1.10 Mouse support for 2D Widgets.

5.1.11 Keyboard support for Widgets.

5.1.12 User-interface Widgets, that are set up by the Developer, and then used by the Client so that he or she can interact with the Application. All user-interface Widgets have the following in common:

   a. They can have the *focus*. Widgets get the focus by being clicked on with the mouse, or by having the Tab key pressed until they are selected. Focused Widgets are eligible to receive keyboard input, and may have a different appearance.
   b. They can be enabled (normal state), or disabled (in which case they cannot receive user input). The disabled state may have a different appearance.

c. They have width, height and depth, as well as an (x,y,z) position.

5.1.13 2D button user-interface Widget. A button:

a. Only has to support rectangular images.
b. Have 4 different states: enabled (ready to be clicked), disabled (the Client cannot interact with the button), mouse over (when the mouse's hot spot is over the button's hot zone), clicked (when the left mouse button is pressed, while the hot spot is inside the hot zone).
c. Each state can be characterized by a different image, or no image.
d. Can contain text.
e. Standard buttons and toggle buttons (that stay in the clicked state after they are let go) are possible.

5.1.14 Label user-interface Widget. A label:

a. Displays text without special formatting. If the text is larger than the label, the text will overflow on the right (optionally, the text can be cut off).
b. Can have a background image.
c. Padding and margin dimensions can be individually specified for all four edges.

5.1.15 Input box user-interface Widget. This Widget allows the user to enter text:

a. Displays text without special formatting. If the text is larger than the input box, the text will disappear on the left side (*i.e.* text becomes right justified and does not overflow).
b. When the input box has the focus, it shows a flashing cursor at the current position.
c. Typing, pressing backspace or using the left/right arrow keys will allow the Client to move the cursor.

5.1.16 Scrollbar user-interface Widget. A scrollbar:

a. Uses 5 buttons: 2 arrows at each end, 1 position indicator, and 2 adjacent bars.
b. Holds state on: current value, minimum value, maximum value, small change increment, and large change increment.

5.1.17 Textbox user-interface Widget:

a. Wraps text and allows line breaks.
b. Excess text is cut at the bottom, with an option for a scrollbar to view missing text.
c. Background image possible.
d. Padding and margin dimensions can be individually specified for all four edges.

5.2 Medium Priority Features

5.2.1 Sound effects available via WAV or MP3 format. Sounds can be played screen-level (music & sound effects) or widget-level (sound effects only).

5.2.2 Capability to change the mouse pointer icon to an arbitrary image, and potentially animate it.

5.2.3 Depth buffer support (for 2D and 3D).

5.2.4 Button extended features:

   a. Buttons can have images & hot zones that are irregularly shaped. Note that the hot zone should correspond to the image, unless there is no image, in which case the hot zone can be anything.
   b. Button can be 3D, and interact with the mouse.

5.2.5 Progress user-interface widget. This widget graphically shows a percentage value. Required features of this widget include:
   a. Optional display of the percentage value as text.
   b. Optional background image.
   c. Progress indication can proceed from the left to the right, or in any of the four cardinal directions.
   d. Progress indication can also proceed two-dimensionally from any of the four corners of the widget.

5.2.6 The ability to map screen-level hotkeys to certain special functions (such as on-screen widgets).

5.2.7 Sprite user-interface Widget. This is an animation Widget that shows a number of pictures in sequence.
   a. The image sequence (where each image is called a frame) has a developer-specified delay in milliseconds between each frame.
   b. It is possible to have multiple frames per image file. An image can be split such that it has 4n frames, where n is a natural number (e.g. 0, 1, 2, etc.). This is an industry standard way to deal with animation, as it gives better performance and is easier to deal with. Some frames within one file may be blank, and the developer then instructs the Widget to ignore them.

5.2.8 Checkbox user-interface Widget. This involves:
   a. Can be true or false, and has an image for each representation. Stays in whatever state it is put in.
   b. Has a label, which should be adjacent to the checkbox.

5.2.9 Flower user-interface Widget. The name refers to the fact that this widget appears as a series of buttons around the mouse pointer, and is reminiscent of a flower blooming (with each button being symbolic of a flower petal). The flower has myriad options:
   a. The radius from the flower's center to each button's center is Developer-specified.
   b. Button size is variable, but must be the same for all buttons.
   c. The Developer can specify the number of buttons the flower will have. The flower then automatically determines the placement for the buttons, based on the radius and button size.
   d. Button images and characteristics are specified starting from the topmost one, and moving clockwise from there.

5.2.10 Option button user-interface Widget. Has distinct behavior from other Widgets, such as:
   a. Has a label (like with checkbox).
   b. Option buttons can be grouped together, so that only zero or one option button in the group is selected at one time (zero option buttons can be in the selected state only in the initial state).
   c. A selected option button that is clicked on remains selected.

5.2.11 List box user-interface Widget. This requires:

a. The capability to select only one entry, or any number. Selection of multiple entries can be done via clicking and dragging, or control-clicking.
b. Excessively long labels in the list box should either be cut off, overflow, or create a horizontal scrollbar.
c. Include a vertical scrollbar if there are more than a Developer-specified number of entries.

5.2.12 Combo box user-interface Widget. It’s a label with a button, and the following requirements:

a. Activating the button creates a list box underneath the combo box, the former of which contains Developer-specified entries.
b. The user can select *one* option from the list box (in which case the entry is automatically placed in the label and the list box disappears), or type in an option manually.

5.2.13 Pop up list user-interface Widget. Appears when called:

a. Must be able to dynamically place itself. Developer specifies where relative to the mouse pointer, or another Widget. For lists relative to the mouse pointer, they are typically drawn so that the mouse pointer is at one of the corners.
b. If the pop up list Widget is told to place itself where it cannot be drawn, it will attempt to draw itself at a different corner, or barring that, will be “shoved aside” enough to fit completely on the screen.
c. If it’s too large to fit on the screen, it will have spinner buttons at either end to allow the user to scroll through the list.
d. Entries in the list can be buttons (including sub-lists), images, or a checkbox.

5.2.14 Screen-level mouse commands are possible.

5.3 Low Priority Features

5.3.1 Data transfer over TCP/IP, using peer-to-peer connectivity.

5.3.2 Initial part of the peer-to-peer handshake can optionally be routed through a central server.

5.3.3 Ability to display 3D text on the screen.

5.3.4 A set of images that are meant to be used with the user-interface Widgets. This is to help speed low-fidelity prototyping.

5.3.5 Spinner user-interface Widget. This is composed of two buttons that have a shared state, containing: current value, minimum value, maximum value, and change increment.

5.3.6 Editable textbox Widget. Identical to a textbox, except that it offers full editing capability:

a. Typing is possible, from any printable ASCII character found on the keyboard, plus backspace and delete. Typing one character moves the cursor one space forward. Typing Enter makes a new line.
b. Mouse clicking to place cursor, or select text.
c. Shift + arrow key selects text.
d. Ctrl + arrow key skips words instead of letters.
e. Shift and control can be combined.

f. Page up and page down skip pages of data (a page can be defined as a specified number of lines, or determined dynamically based on how many lines the widget displays).
g. Delete and backspace remove text.
h. Home and end go to the start and end of the text.
i. Two states: Insert state, and Overwrite state. The Insert key toggles between them. Default is Insert state. Insert state places newly typed letters in between current ones at the cursor position, while Overwrite displays a box instead of a cursor line, and erases the original character at that position.

5.3.7 Combo box extended feature: manual typing should bring up the best match from the pre-selections, just like with URL typing in typical web browsers.

6 Non-Functional Requirements (Qualities)

6.1 Architecture

An architecture must be documented, adopted, used throughout the software's life, and continuously improved as necessary. Further architectural requirements include:

6.1.1 The architecture must be object-oriented.

6.1.2 The architecture must create a distinct separation between the program's presentation and the Application logic. This means that any changes to the underlying components of UGENE should not negatively affect a program that uses it.

6.1.3 The architecture must be understandable, and maintainable. In addition, an architecture document must be written and continuously updated to reflect the current state of the system.

6.1.4 When creating the architecture, the most important software quality is maintainability, while the least important is performance. Performance can be improved later, but only if the architecture is maintainable.

6.2 Maintainability

In order to ensure UGENE's long-term stability, the following principles must be pursued:

6.2.1 Information hiding: objects must not access each other's private data directly. This lowers inter-object coupling and improves the code's long-term maintainability.

6.2.2 An architecture (see **Architecture,** above) is necessary, and should be adhered to by UGENE's developers and the maintainers. The architecture documentation should be updated whenever the architecture is modified.

6.2.3 Comments and a coding standard (described under **Code**, below) are required. In order for the comments to be useful to maintainers, they must give additional information that cannot be read directly from the code itself (*e.g.* useful comments could describe why a particular function exists, or what it does on a large scale, rather than what it does on a local scale).

6.3 Usability

For the most part, UGENE's usability requirements will be met by ensuring that its functionality supports all of the use-cases specified in Section 4. However, it is also necessary for UGENE to respect additional usability requirements. UGENE's two distinct human actors each have different needs that aren't necessarily expressed by the use-cases.

The additional usability requirements that support the Developer are:

6.3.1 The Developer is expected to follow the framework provided by UGENE's architecture. However, this framework must offer the Developer as much flexibility as possible.

6.3.2 The Developer is assumed to be a C++ programmer of intermediate to advanced skill. He must understand pointers, function pointers and inheritance.

6.3.3 The Developer needs no other skills aside from those just mentioned. For example, no knowledge of OpenGL or the STL is required to create an application. However, knowledge of these is still helpful (especially in the case of the STL).

The additional usability requirements that support the Client are:

6.3.4 The user interface Widgets must be easy to use. This requirement hinges on the use of transfer effects—in other words, Widget behavior must be very similar to what Windows users are familiar with.

6.3.5 All other Client usability requirements are the responsibility of the Developer.

6.4 Code

UGENE's code must adhere to the following:

6.4.1 The program is to be written in C++.

6.4.2 Programming must be object-oriented, in order to ensure the modularity, reusability and understandability of the code.

6.4.3 A complete coding standard must be developed and adhered to, in order to ensure understandability of the code, and to reduce the number of errors, and reduce the cost to find and fix errors.

6.4.4 High-level comments must be included for every non-trivial class and function in order to describe *what* they do and *how* they do it (if important). Wherever required inside the code, comments will focus on describing *why* that code is necessary, with comments describing *what* or *how* only when necessary (if, for example, an obscure system function is being invoked).

6.4.5 Each header file needs to have a section devoted to a change log. This log should contain information on *who* made a change, *where* inside the code and *when*.

6.5 Reliability

6.5.1 UGENE, using a small test application as a driver, must have a mean time to failure of 6 hours or more (see **Definitions, Abbreviations and Acronyms**).

6.6 Portability

UGENE has the following high-priority portability requirements:

6.6.1 **Platform:** UGENE must be able to run in the Win32 environment.

6.6.2 **Operating System:** UGENE must function on Windows 98 SE, Windows 2000 and Windows XP.

6.6.3 **Language:** UGENE Widgets and configuration files can only accept the Latin character set.

The following portability requirements are low-priority:

6.6.4 **Compiler:** UGENE must be able to compile under one of the following: Microsoft Visual C++ 6.0 or gcc.

6.6.5 **Platform:** UGENE must be able to run on X11, as well as Win32.

6.6.6 **Operating System:** UGENE runs on RedHat Linux, Mac OS X (through X11) and Windows 98/2000/XP.

6.6.7 **Language:** UGENE offers Unicode support.

6.7 Performance

6.7.1 UGENE must run at 30 frames per second or more under the following conditions:

- The computer is a Pentium 500 MHz with 128 MB of RAM, and an NVIDIA GeForce II 32 MB graphics card.
- The operating system is Windows 98 SE.
- The application is running full screen, and drawing 100 three-dimensional images on screen simultaneously.
- The images are loaded from 100 kB BMP files.