table of contents

## testing and evaluation

Test report 2

Comparison with original specification 4

Efficiency and elegancy of code 5

Alpha and beta testing 7

Final user interface 8

## maintenance

CASE tools 11

Catering for changing requirements 12

Handling code changes 13

testing and evaluation

# test report

Module to be tested

The module b\_addToHistory takes in the messages being sent and received by r.IRC, and, after some analysis and classification, appends these messages to the array holding previous messages, thus enabling messages to be displayed in the history pane in the GUI.

def b\_addToHistory(

messageArray,

newLine,

id,

direction

)

if direction == 'i'

# incoming message, process it !

message = b\_processIncoming(newLine)

if message[:command] == "PING"

# autorespond to pings

# putToConsole("PING") # DEBUGGING

b\_pingResponse(id,message)

else

if messageArray.count >= MAX\_HISTORY

messageArray.shift

end

messageArray << message

# putToConsole(newLine)

end

elsif direction == 'o'

# outgoing message, process it differently !

if messageArray.count >= MAX\_HISTORY

messageArray.shift

end

messageArray << newLine

# putToConsole(newLine)

end

end

Note:

* direction can only be of values ‘i’ or ‘o’

Pathway testing

|  |  |  |  |
| --- | --- | --- | --- |
| **direction** | **message[:command]** | **messageArray.count** | **expected end result** |
| ‘i’ | “PING” | [n/a] | b\_pingResponse() |
| ‘i’ | “PRIVMSG” | < MAX\_HISTORY | messageArray << message |
| ‘i’ | “PONG” | >= MAX\_HISTORY | messageArray.shift  messageArray << message |
| ‘o’ | [n/a] | < MAX\_HISTORY | messageArray << newline |
| ‘o’ | [n/a] | >= MAX\_HISTORY | messageArray.shift  messageArray << newLine |

Syntax testing

Below is the output from the JRuby syntax checker, with warning and verbose flags enabled.

$ jruby -cvw common/methods\_rirc.rb common/methods\_irc.rb common/methods\_io.rb common/classes.rb rircgui.rb

jruby 1.7.13 (1.9.3p392) 2014-06-24 43f133c on OpenJDK 64-Bit Server VM 1.7.0\_65-b32 [linux-amd64]

Syntax OK

Syntax OK

Syntax OK

Syntax OK

Syntax OK

# comparison with original specification

The original design specification for r.IRC is as follows:

r.IRC aims to:

* provide an interface for group text communication via the IRC protocol
* do so through the provision of an attractive and featured user interface

r.IRC DOES NOT aim to:

* serve as a complete replacement of large scale official communications for businesses, organisations, and governments
* provide a guaranteed secure medium through which users can communicate in secrecy
* facilitate the transfer of data wherein the data type is of a nature other than text e.g. image, video, audio

r.IRC will be developed using:

* Ruby version 1.9.3
* Shoes version 3.1.0

r.IRC will aim to support the following platforms (subject to change):

* GNU/Linux
* Microsoft Windows (Vista and above)

As thus far, these changes to the original specification have been made:

r.IRC will be developed using:

* JRuby version 1.7.13 (Ruby 1.9.3p392 equivalent)
* Shoes version 4.0.0.pre1

r.IRC has software dependencies including:

* Java Runtime Environment supporting Java SE 7

# efficiency and elegancy of code

A modular design to the algorithm, where each individual component is isolated from unrelated functions, will allow for separate testing of any changes to any modules during the maintenance stage of the software development cycle. For example, the ‘Help’ window is called by the button on the main menu; however, in the context of the main menu, the ‘Help’ window is abstracted into a single module that is executed when the button is pressed. This would allow for the use of stubs when making changes to higher level modules. Furthermore, having modular code would allow for the use of drivers to test code changes to lower level modules, not to mention reduce the effective complexity of the code by separating it into different sections to simplify the holistic overview of the program, as demonstrated below.

## window modules ##

def g\_menu

button do

g\_newTab

end

button do

g\_settings

end

button do

g\_help

end

end

def g\_newTab; end

def g\_settings; end

def g\_help; end

## main function ##

b\_startup

g\_menu

The source code of r.IRC follows an aesthetic style guide in regards to bracing, indenting, and variable naming. Aspects such as the use of camelCase and preference for do/end over braces in containing code blocks are consistent throughout the entire code, as well as the classification of functions and methods into separate files and categorised with the use of prefixes. Overall, the elegance of the source code of r.IRC stems from the consistency of code layout and style, which provides for ease of readability and accessibility, as demonstrated in the following code snippet.

def b\_addToHistory(

messageArray,

newLine

)

message = b\_processIncoming(newLine)

if message[:command] == "PING"

b\_pingResponse(message)

else

if messageArray.count >= MAX\_HISTORY

messageArray.shift

end

messageArray << message

puts newLine

end

end

In addition, separate code modules can be utilised throughout the source code many times, thus simplifying and reducing the length of the source code. This allows for ease of future maintenance as individual modules can be modified without great negative effect on the program’s operation as a whole. An example of this is demonstrated below.

def f\_createCopy(

originalFileName,

copyName

)

arr = f\_loadFileIntoArray(originalFileName)

f\_writeFileFromArray(copyName,arr)

end

def f\_loadFileIntoArray(

fileName

)

arr = Array.new

f = File.new(fileName,"r")

while line = f.gets

arr << line.chomp

end

f.close

return arr

end

def f\_writeFileFromArray(

fileName,

arr

)

f = File.new(fileName,"w")

arr.each do |l|

f.print "#{l}\n"

end

f.close

end

# alpha and beta testing

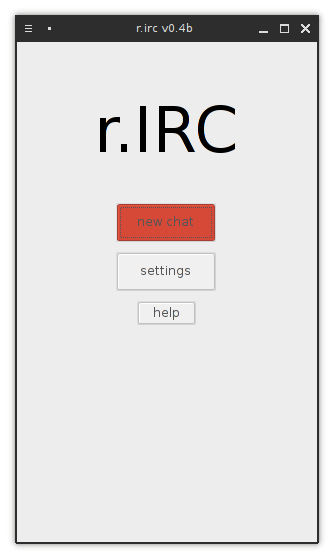
The final level of system testing can be split into two main components: alpha and beta testing. Both of these testing stages will be useful in the final testing process up to the release of r.IRC.

Alpha testing involves the testing of the final complete solution in a simulated environment by internal personnel. Due to the very selective nature of the alpha testers, they are generally better equipped to uncover and report issues and unintended outcomes than the general user base, allowing for most, if not all, bugs and errors still existent to be detected and removed before the next stage of testing. r.IRC’s alpha testing stage will consist of the involvement of two dozen users over two weeks to thoroughly test every possible pathway of the application and especially pathways expected to be most common, in order to detect and subsequently remove a majority of bugs.

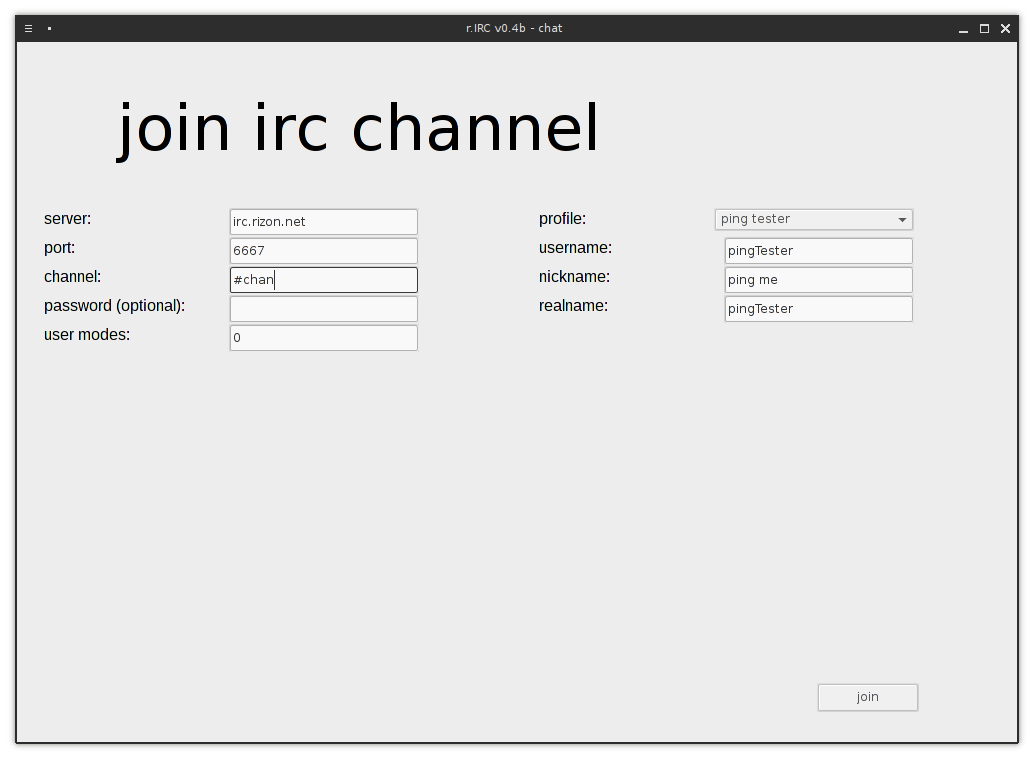
Beta testing, on the other hand, involves the testing of the final solution by a wider audience, usually consisting of internal personnel and enthusiast members of the user base. In comparison to alpha testing, the beta stage of testing consists of a much larger number of testers and thus it is more likely that errors occurring in frequently used aspects of the application will be detected in a shorter period of time. r.IRC’s beta testing will occur after alpha testing has concluded and any issues raised during alpha testing have been addressed, and will involve approximately fifty members of the general user base testing the latest release of r.IRC over three weeks; discovered issues will be resolved and new beta versions released as soon as possible during this period in order to ensure that the application is completely operationally perfect at the conclusion of testing. Following alpha and beta testing and their subsequent patching of issues within the solution, r.IRC will be deemed ready for general release to the public.

# final user interface

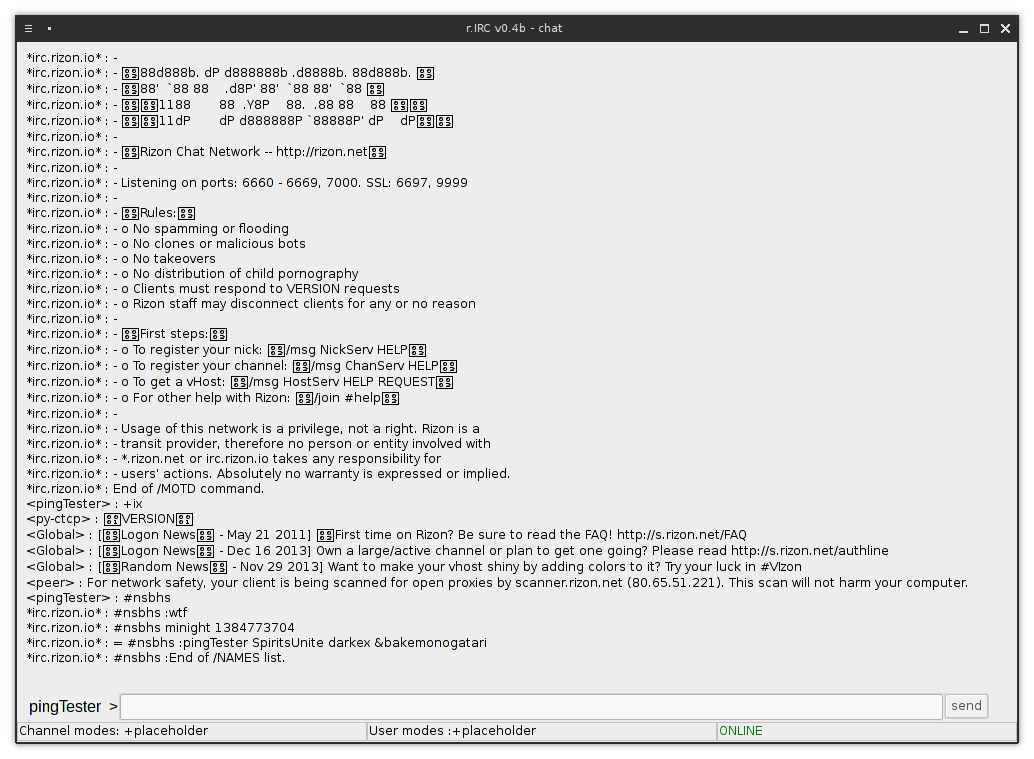
Main menu



New chat configuration



Chat window



maintenance

# case tools

Computer-aided software engineering tools have been and will continue to be extremely useful throughout the development process of r.IRC. Such tools encompass the aspects of code writing, interface design, and version control.

The process of code writing, although the basic foundation of any software development project, is greatly extended and aided by the use of a specialised text editor or IDE. In the case of r.IRC, Sublime Text 3 was the primary tool of choice when writing code, due to its high extensibility leading to an improved workflow, as well as Vim being the secondary text editor for other documents such as note-taking and minor planning. These CASE tools will continue to be useful during the maintenance stage as additional requirements are discovered and code updates are made.

The r.IRC user interface was designed using a combination of traditional paper drawings and CASE interface design tools. The most prominent of the latter of these is draw.io, which allowed for use of rapid prototyping in developing the user interface through its provision of template common graphical elements such as buttons and text boxes. draw.io will remain relevant during the maintenance stage of the software development cycle as new requirements may demand changes to the user interface, as well as providing an interface to users with which they can comment on the effectiveness of a proposed user interface change.

Since r.IRC is a constantly evolving product, version control is of utmost importance. Employing effective version control measures ensures that r.IRC is inclusive of both users that are comfortable with experimental, cutting-edge features as well as users which expect a high degree of stability and reliability, by providing different versions of r.IRC to these respective users. The Git version control system is used in this case, with Github as a remote host to allow prospective and current users the choice of either a stable release or a development release of r.IRC. As well as providing differentiation between versions, the use of Git also allows for the tagging of different versions and updates with relevant information, enlightening users on the nature and details concerning code updates.

# catering for changing requirements

The ever-changing nature of the consumer software environment will lead to inevitable changes to user requirements of r.IRC. This software therefore must be maintained in order to continually meet these new requirements, whether it be done by augmenting the application to provide new features or by modifying existing components to ensure compatibility with newer software and hardware developments.

r.IRC has been developed with future changing requirements in mind. One vital aspect of the application that allows it to be easily adaptable for future requirements is its open source nature. The source code of r.IRC is free and open to anyone who wishes to view it, and third parties are freely able to modify the source code to suit their own needs or the needs of others that have developed in the future. Even individuals, if possessing the technical prowess, can modify the code themselves to cater to their own requirements.

Certainly, r.IRC will be maintained for an extended time period after its release, owing to the volatile nature of consumer software requirements. Any major changes in requirements will prompt a new release as soon as possible; such changes may include resolving incompatibilities caused by upgrades to frameworks, dependencies, or operating systems, as well as implementing highly requested features amongst the user base and fixing critical security vulnerabilities discovered after release.

Furthermore, r.IRC is a completely stand-alone application; it makes use of the Java Runtime Environment, which is platform-independent. The application itself is completely encapsulated in a single directory; installation and uninstallation is unnecessary as all its dependencies apart from Java are packaged with the program. In this way, it is extremely simple to upgrade r.IRC with updated versions in case of changing user requirements; user settings and profiles are stored in their own sub-directory and thus can even be preserved across different versions of r.IRC.

# handling code changes

r.IRC is designed to be used in a constantly changing environment, and thus will inevitably require modifications to the application code to cope with future changes in the system environment.

As mentioned previously, Git has been and will continue to be used as a version control CASE tool for r.IRC. By utilising version control, different versions of r.IRC can be made available for current and new users that are specially developed for situational circumstances. For example, a change in operating system may be accompanied with the incompatibility of a previous version of r.IRC, in which case a new version will be created and made available for users; the previous version would still be available for download by users who have not updated.

Since the executable of r.IRC is self-contained into a single Java executable, it would be impossible to create ‘patches’ to be applied to an existing version for minor application upgrades. Therefore, a completely new executable must be downloaded every time a major update is release. As such, major releases of r.IRC will be purposefully limited due to size limitations.

In order to maintain a degree of stability, changes to the source code will originally be pushed onto a separate non-master branch of the code repository, Nevertheless, advanced users wanting to experience the latest releases will be able to download the source, which will always be updated, and manually compile an executable.

Using a version control system such as Git also provides the benefits of allowing for reversion of the source code. In a case where code containing errors has been pushed to the repository, a version revert can be performed to correct this mistaken update with minimal side effects.