

Mercurial SCM

A Testing Framework by:

Kyle Brooks

Jesse Hunt

Chris Sigmund

Dr. Bowring

CSCI 362

1 December 2015

Table of Contents

Introduction	2
Chapter One: <i>Choosing an H/FOSS Project: Mercurial</i>	3
Chapter Two: <i>Test Plan</i>	5
Chapter Three: <i>Testing Framework Architecture</i>	9
Chapter Four: <i>Test Cases</i>	11
Chapter Five: <i>Fault Injections</i>	20
Chapter Six: <i>Reflections</i>	24
Project Assessment:	26

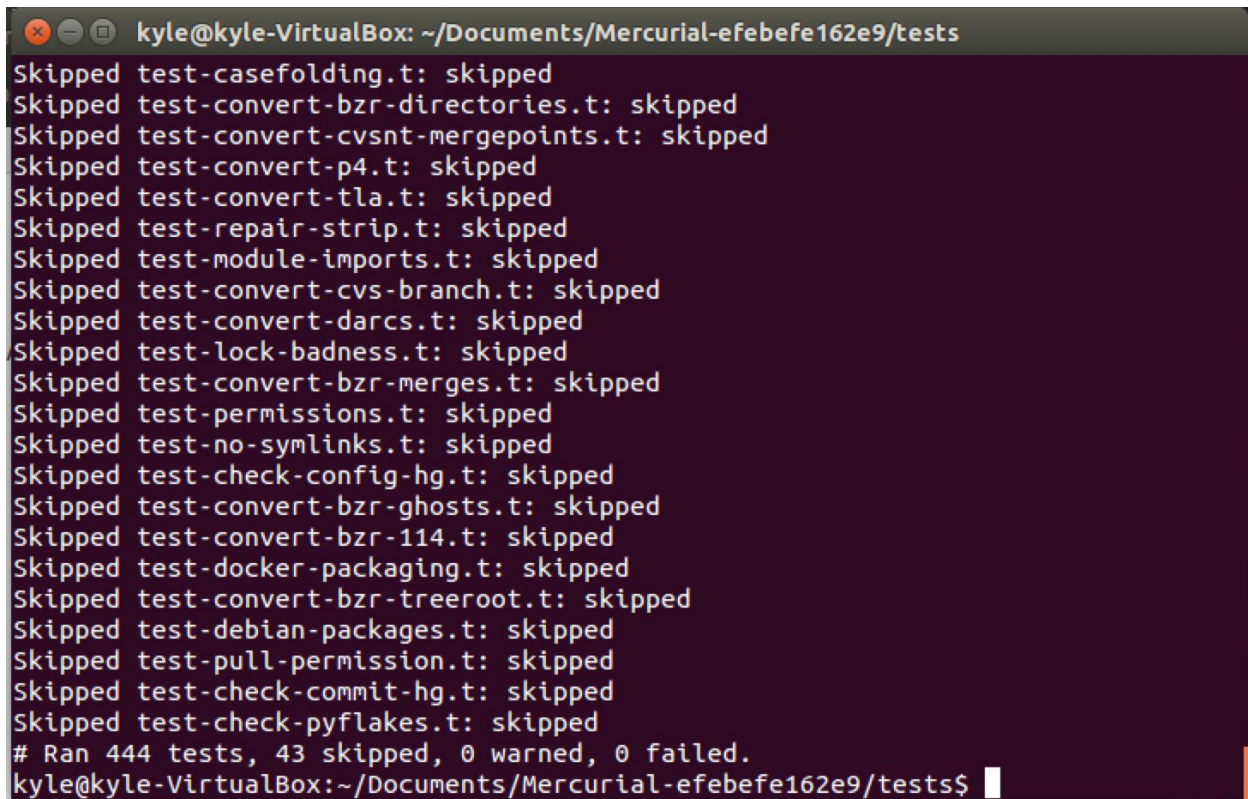
Introduction

CSCI 362 is a course aimed at teaching the intricacies of software engineering. While it covers a wide range of topics, one of the main focuses of the course is on software testing. Software testing is essential in any software development process, yet it is often the step that is rushed or overlooked completely. This being the case, proper software testing is something that is crucial to learn and is becoming a sought-after quality in the job market. Over the course of the semester our class worked in groups on several open source software projects, designing and implementing a testing framework for them. This report aims to present RedTeam's project, Mercurial, and the process we used to complete the assignment.

Chapter One

Choosing an H/FOSS Project: Mercurial

When choosing the project, our team had few restrictions. The only real requirement was that our chosen software must be from a list of open-source projects found on foss2serve.org. Besides that, our team also wanted a project that was well documented. This would theoretically save headaches later on, as well documented code is always easier to work with. With this in mind, our team settled on Mercurial. Mercurial is essentially Git, although not nearly as popular. It allows for version control of code among several members, all able to interact seamlessly with the project. From our research into Mercurial, we found that it was well documented, containing step-by-step instructions on how to download, install, and even how to write your own test cases. Considering this was the essence of our project, it seemed like a solid piece of software to work with. We were easily able to download and compile the source code, as well as run the already established tests (*test results below*).

A terminal window with a dark purple background and light green text. The title bar reads 'kyle@kyle-VirtualBox: ~/Documents/Mercurial-efebefe162e9/tests'. The terminal output lists 25 test cases, all marked as 'skipped'. The tests include: test-casefolding.t, test-convert-bzr-directories.t, test-convert-cvsnt-mergepoints.t, test-convert-p4.t, test-convert-tla.t, test-repair-strip.t, test-module-imports.t, test-convert-cvs-branch.t, test-convert-darcs.t, test-lock-badness.t, test-convert-bzr-merges.t, test-permissions.t, test-no-symlinks.t, test-check-config-hg.t, test-convert-bzr-ghosts.t, test-convert-bzr-114.t, test-docker-packaging.t, test-convert-bzr-treeroot.t, test-debian-packages.t, test-pull-permission.t, test-check-commit-hg.t, and test-check-pyflakes.t. At the bottom, it says '# Ran 444 tests, 43 skipped, 0 warned, 0 failed.' followed by the prompt 'kyle@kyle-VirtualBox:~/Documents/Mercurial-efebefe162e9/tests\$' and a cursor.

```
kyle@kyle-VirtualBox: ~/Documents/Mercurial-efebefe162e9/tests
Skipped test-casefolding.t: skipped
Skipped test-convert-bzr-directories.t: skipped
Skipped test-convert-cvsnt-mergepoints.t: skipped
Skipped test-convert-p4.t: skipped
Skipped test-convert-tla.t: skipped
Skipped test-repair-strip.t: skipped
Skipped test-module-imports.t: skipped
Skipped test-convert-cvs-branch.t: skipped
Skipped test-convert-darcs.t: skipped
Skipped test-lock-badness.t: skipped
Skipped test-convert-bzr-merges.t: skipped
Skipped test-permissions.t: skipped
Skipped test-no-symlinks.t: skipped
Skipped test-check-config-hg.t: skipped
Skipped test-convert-bzr-ghosts.t: skipped
Skipped test-convert-bzr-114.t: skipped
Skipped test-docker-packaging.t: skipped
Skipped test-convert-bzr-treeroot.t: skipped
Skipped test-debian-packages.t: skipped
Skipped test-pull-permission.t: skipped
Skipped test-check-commit-hg.t: skipped
Skipped test-check-pyflakes.t: skipped
# Ran 444 tests, 43 skipped, 0 warned, 0 failed.
kyle@kyle-VirtualBox:~/Documents/Mercurial-efebefe162e9/tests$
```

The README file in the test directory of mercurial includes a link to a site that gives specific instructions on how to create and run your own tests. While none of the tests failed when we ran them, quite a few were skipped. Our plan for the coming days became to come up with a detailed test plan of what we thought necessary to test,

including figuring out why certain tests were skipped. At this time, we agreed that Sunday afternoons worked best for meeting up and determined it would be best to maintain weekly group meetings. Though we had fairly conflicting schedules, we were confident that we would be able to work independently through the week, communicating via text. We agreed on a strategy of dividing the work evenly among ourselves. Each member would be responsible for his part, "presenting" to the others in the weekly meeting. This way, we would all stay informed of the various parts of the project while not necessarily having to be responsible for fully understanding everything.

Chapter Two

Test Plan

The testing process

The initial tests we ran concern various methods used in cloning a project in Mercurial

Test Case	01
Requirement	Break a given integer into the largest increments of time it can be broken into.
Component	progress.py
Method	fmtremaining(int)
Test Input(s)	100
Expected Outcome	1m40s

Test Case	02
Requirement	Given the range of two points, calculate the point of overlap between them
Component	simplemerge.py
Method	intersect(ra, rb)
Test Input(s)	(0,100), (50,150)
Expected Outcome	(50,100)

Test Case	03
Requirement	Returns the number of processors available to the operating system (This test assumes you are running a single-processor virtual machine, so it will fail if more than 1 processor is available to the operating system)
Component	worker.py
Method	countcpus()

Test Input(s)	None
Expected Outcome	1

Test Case	04
Requirement	Return the length of the given string
Component	templatefilters.py
Method	count(String i)
Test Input(s)	"abcde"
Expected Outcome	5

Test Case	05
Requirement	Return elements in an array concatenated into one element in a larger array
Component	namespaces.py
Method	tolist(String)
Test Input(s)	"1", "2", "3", "4"
Expected Outcome	['1234']

Requirements traceability

Users are most interested in the system meeting its requirements and testing should be planned so that all requirements are individually tested.

- Command line arguments generate a repository for use in further tests
- The repository has a data file containing text added to it, then the file is committed, and the repository is checked for the addition of the new files in its file log
- Clones files to a specified local directory (verification only of the command, not of the directory contents)
- Clone command aborts
- Verifies the data was correctly copied over

Tested items

- Test a time output given a number of seconds as input
 - Returns the number of seconds formatted as m/s or h/m or d/h, etc. depending on size of integer input
- Test to find the intersection of two lines.
 - If there is a point of intersection, then that point is returned. Otherwise nothing is returned
- Counts the number of processors available to the operating system
 - returns the number of processors.
- Count the length of a String
 - Given a string as input, return the length of that string
- Return an array as a list
 - Given an array, return all elements concatenated into a single list

Testing schedule

- September 29th: Deliverable #2
 - 5 of 25 test cases developed
 - Detailed Test Plan Due
- October 3rd: Research on how to develop test framework for Mercurial
- October 7th: Begin developing test framework
- October 11th: Continue to develop test framework
- October 22nd: Deliverable #3 Due
 - Testing framework designed and built
- November 12th: Deliverable #4 Due
- November 24th: Deliverable #5 Due

Test recording procedures

It is not enough simply to run tests; the results of the tests must be systematically recorded. It must be possible to audit the testing process to check that it has been carried out correctly.

Test Case: <testNumber>

Requirement: <requirementText>

Component: <componentName>

Method: <methodName>

Test Input: <specified test input>

Expected Outcome: <outcome>

Hardware and software requirements

- Firefox to display results of testing output
- Ubuntu (or other flavor of Linux)
- Python

Constraints

- Scheduling Conflicts
May have other schoolwork priorities
- Travel Constraints
Potential asynchronous meetings off campus in the future.
- Lack of Knowledge
Depending on experience, some tasks require steep learning curve
- Sickness

Chapter Three

Testing Framework Architecture

Architectural Framework

The testing framework is intended as an all-in-one solution that requires only the addition of a single, structured, text-based test file to run the system. Using Python's scripting abilities, the system accomplishes both the set-up and running of tests within the main "runAllTests" script. To describe how the system works, we have the following ordered list of the functions of the runAllTests script:

1. The script sets up its local working directory, noting where tests are coming from (/testCases) and where to place its report (/reports).
2. The script opens the test case directory, lists all of the files, and then puts them in alphanumerical order.
3. The script begins to generate an HTML file to use as the report for the tests.
4. The script then runs through all the files in the testCases directory in which it:
 - i. Lists the identity, requirements, component, method, input, and expected output for a given test case (which is printed to the HTML report file)
 - ii. Parses together the component, method and input into a string
 - iii. Executes the string using an exec method within Python.
 - iv. The system then prints the outcome of the statement to the HTML report file
 - v. If the system encounters an exception, this is then printed to the HTML file and the script moves on.
5. At the end of running through the tests, the script then lists how many tests it ran, how many passed, and how many failed, and prints this information to the HTML document.

How-To Documentation

The process for writing a test for the Mercurial test set is as follows:

1. Create a text file in the /testCases directory. The naming convention is "testCase###.txt," and the system runs through the files in alphanumeric order, so

if you don't follow this convention, be aware of where your test should run based on that order.

2. Within the test case text file, list information in the following order (information with a * indicates it will be parsed, so write it with the intention of that specific code being executed):
 - i. Test Identification
 - ii. Requirement of the method being tested (what's supposed to happen or a quick description)
 - iii. *Component being tested (what you would import from mercurial, i.e. "from mercurial import fmtremaining)
 - iv. *Method being tested (only the method name, no parenthesis or any other symbology)
 - v. *Input being tested
 - vi. *Expected output (exactly as you would expect the system to return it)
3. Once the test case is generated, you can execute "runAllTests.py" within the /scripts directory by typing "python runAllTests.py" in terminal

Test Case Examples

Below is an example to demonstrate test case format (// Indicates comment. Comments not included in actual test cases

testCase02.txt:

2 // Test Case Number

Given the range of two points, calculate the point of overlap between them // Req.

simplemerge // Component required

intersect // Method of component being tested

(0,100), (50,150) // Input

(50,100) // Oracle Value

Chapter Four

Test Cases

By this point, our team had completed the testing framework that will automatically run all test cases and return results in the form of an html table. The framework tests the following methods:

- `fmtremaining` from `progress.py`
- `intersect` from `simplemerge.py`
- `countcpus` from `worker.py`
- `count` from `templatefilters.py`
- `tolist` from `namespaces.py`

In test cases 11-15, we ran into a few issues, as the output only told if the method failed, but not why it was failing. It turned out the quotes we were using in our test case files were somehow italicized, therefore not matching up with the oracle output. Once we discovered this issue, it was a quick fix and we were able to move forward. Below are the 25 test cases we developed.

Test Case	01
Requirement	Break a given integer into the largest increments of time it can be broken into.
Component	<code>progress.py</code>
Method	<code>fmtremaining(int)</code>
Test Input(s)	100
Expected Outcome	1m40s

Test Case	02
Requirement	Glven the range of two points, calculate and return the point of overlap between them
Component	<code>simplemerge.py</code>
Method	<code>intersect(ra, rb)</code>
Test Input(s)	(0,100), (50,150)
Expected Outcome	(50,100)

Test Case	03
Requirement	Returns the number of processors available to the operating system (This test assumes you are running a single-processor virtual machine, so it will fail if more than 1 processor is available to the operating system)
Component	worker.py
Method	countcpus()
Test Input(s)	None
Expected Outcome	1

Test Case	04
Requirement	Return the length of the given string
Component	templatefilters.py
Method	count(String i)
Test Input(s)	"abcde"
Expected Outcome	5

Test Case	05
Requirement	Return elements in given array concatenated into one element in a larger array namespaces
Component	namespaces.py
Method	tolist(String)
Test Input(s)	"1", "2", "3", "4"
Expected Outcome	['1234']

Test Case	06
Requirement	Break a given integer (representing time in seconds) into largest units of time possible
Component	progress.py
Method	fmtremaining(int)
Test Input(s)	59
Expected Outcome	59s

Test Case	07
Requirement	Break a given integer (representing time in seconds) into largest units of time possible
Component	progress.py
Method	fmtremaining(int)
Test Input(s)	0
Expected Outcome	00s

Test Case	08
Requirement	Break a given integer (representing time in seconds) into largest units of time possible
Component	progress.py
Method	fmtremaining(int)
Test Input(s)	3601
Expected Outcome	1h01m

Test Case	09
Requirement	Break a given integer (representing time in seconds) into largest units of time possible
Component	progress.py

Method	fmtremaining(int)
Test Input(s)	3599
Expected Outcome	59m59s

Test Case	10
Requirement	Break a given integer into the largest increments of time it can be broken into.
Component	progress.py
Method	fmtremaining(int)
Test Input(s)	-59
Expected Outcome	-59s

Test Case	11
Requirement	Given the range of two points, calculate and return the point of overlap between them
Component	simplemerge.py
Method	intersect(ra, rb)
Test Input(s)	(0,10), (5,15)
Expected Outcome	(5,10)

Test Case	12
Requirement	Given the range of two points, calculate and return the point of overlap between them
Component	simplemerge.py
Method	intersect(ra, rb)
Test Input(s)	(0,100), (50,50)
Expected Outcome	None

Test Case	13
Requirement	Given the range of two points, calculate and return the point of overlap between them
Component	simplemerge.py
Method	intersect(ra, rb)
Test Input(s)	(0,10), (10,10)
Expected Outcome	None

Test Case	14
Requirement	Given the range of two points, calculate and return the point of overlap between them
Component	simplemerge.py
Method	intersect(ra, rb)
Test Input(s)	(0,2), (1,15)
Expected Outcome	(1,2)

Test Case	15
Requirement	Given the range of two points, calculate and return the point of overlap between them
Component	simplemerge.py
Method	intersect(ra, rb)
Test Input(s)	(0,10), (10,15)
Expected Outcome	None

Test Case	16
Requirement	Return the length of the given string

Component	templatefilters.py
Method	count(String i)
Test Input(s)	""
Expected Outcome	0

Test Case	17
Requirement	Return the length of the given string
Component	templatefilters.py
Method	count(String i)
Test Input(s)	" n "
Expected Outcome	5

Test Case	18
Requirement	Return the length of the given string
Component	templatefilters.py
Method	count(String i)
Test Input(s)	"1_%YTb"
Expected Outcome	6

Test Case	19
Requirement	Return the length of the given string
Component	templatefilters.py
Method	count(String i)
Test Input(s)	"aaaaaaaaaaaaaaaa"

Expected Outcome	15
------------------	----

Test Case	20
Requirement	Return the length of the given string
Component	templatefilters.py
Method	count(String i)
Test Input(s)	"hello" "world"
Expected Outcome	10

Test Case	21
Requirement	Return elements in given array concatenated into one element in a larger array
Component	namespaces.py
Method	tolist(String)
Test Input(s)	"add" "These" "Words" "Together"
Expected Outcome	['addTheseWordsTogether']

Test Case	22
Requirement	Return elements in given array concatenated into one element in a larger array
Component	namespaces.py
Method	tolist(String)
Test Input(s)	"This" "is" "a" "sentence."
Expected Outcome	['This is a sentence.']

Test Case	23
-----------	----

Requirement	Return elements in given array concatenated into one element in a larger array
Component	namespaces.py
Method	tolist(String)
Test Input(s)	"1"+"2"="3"
Expected Outcome	['1+2=3']

Test Case	24
Requirement	Return elements in given array concatenated into one element in a larger array
Component	namespaces.py
Method	tolist(String)
Test Input(s)	None
Expected Outcome	[]

Test Case	25
Requirement	Return elements in given array concatenated into one element in a larger array
Component	namespaces.py
Method	tolist(String)
Test Input(s)	""
Expected Outcome	['']

Finder File Edit View Go Window Help

Ubuntu [Running]

99% Mon 7:30 PM

7:30 PM

file:///home/christophersigund/mylist1/RedTeam/TestAutomation/reports/report.html

Test ID	Component.Method()	Requirements	Input	Expected Output	Actual Output	Pass/Fail
01	progress.fmtremaining()	Break a given integer (representing time in seconds) into largest units of time possible	100	1m40s	1m40s	PASS
02	simplemerge.intersect()	Given the range of two points, calculate and return the point of overlap between them	(0,100), (50,150)	(50, 100)	(50, 100)	PASS
03	worker.countcpus()	Returns the number of processors available to the operating system (This test assumes you are running a single-processor virtual machine, so it will fail if more than 1 processor is available to the operating system)		1	1	PASS
04	templatefilters.count()	Return the length of the given string	"abcde"	5	5	PASS
05	namespaces.tolist()	Return elements in given array concatenated into one element in a larger array	"1" "2" "3" "4"	["1234"]	["1234"]	PASS
06	progress.fmtremaining()	Break a given integer (representing time in seconds) into largest units of time possible	59	59s	59s	PASS
07	progress.fmtremaining()	Break a given integer (representing time in seconds) into largest units of time possible	0	00s	00s	PASS
08	progress.fmtremaining()	Break a given integer (representing time in seconds) into largest units of time possible	3601	1h01m	1h01m	PASS
09	progress.fmtremaining()	Break a given integer (representing time in seconds) into largest units of time possible	3599	59m59s	59m59s	PASS
10	progress.fmtremaining()	Break a given integer (representing time in seconds) into largest units of time possible	59	59s	59s	PASS
11	simplemerge.intersect()	Given the range of two points, calculate and return the point of overlap between them	(0,10), (5,15)	(5, 10)	(5, 10)	PASS
12	simplemerge.intersect()	Given the range of two points, calculate and return the point of overlap between them	(0,100), (50,50)	None	None	PASS
13	simplemerge.intersect()	Given the range of two points, calculate and return the point of overlap between them	(0,10), (10,10)	None	None	PASS
14	simplemerge.intersect()	Given the range of two points, calculate and return the point of overlap between them	(0,2), (1,15)	(1, 2)	(1, 2)	PASS
15	simplemerge.intersect()	Given the range of two points, calculate and return the point of overlap between them	(0,10), (10,15)	None	None	PASS
16	templatefilters.count()	Return the length of the given string	""	0	0	PASS
17	templatefilters.count()	Return the length of the given string	" n "	5	5	PASS
18	templatefilters.count()	Return the length of the given string	"1 %YtB"	6	6	PASS
19	templatefilters.count()	Return the length of the given string	aaaaaaaaaaaaa	15	15	PASS
20	templatefilters.count()	Return the length of the given string	"hello" "world"	10	10	PASS
21	namespaces.tolist()	Return elements in given array concatenated into one element in a larger arrays	"add" "These" "Words" "Together"	["addTheseWordsTogether"]	["addTheseWordsTogether"]	PASS
22	namespaces.tolist()	Return elements in given array concatenated into one element in a larger array	"This" "" "is" "" "a" "" "sentence."	["This is a sentence."]	["This is a sentence."]	PASS
23	namespaces.tolist()	Return elements in given array concatenated into one element in a larger array	"1" "" "2" "" "3"	["1+2=3"]	["1+2=3"]	PASS
24	namespaces.tolist()	Return elements in given array concatenated into one element in a larger array	None	[]	[]	PASS
25	namespaces.tolist()	Return elements in given array concatenated into one element in a larger array	""	[]	[]	PASS

Number of Tests: 25
Passed: 25
Failed: 0

Left 38

Chapter Five

Fault Injection

As a part of our testing method, faults were injected into the source code for Mercurial. It is important to see what would happen if the wrong logic was used in the code, or if there was a simple error in the code. Five different faults were injected into all five of our tested methods.

In progress.py, the fmtremaining method was altered by adding an 'and' to the first "if" statement. Originally, the if statement would take any int less than 60. We changed the condition to > 0 and still less than 60. This should no longer allow the use of negative integers to count as seconds.

```
def fmtremaining(seconds):
```

```
    """format a number of remaining seconds in human readable way
```

```
    This will properly display seconds, minutes, hours, days if needed"""
```

```
    if seconds > 0 and seconds < 60: //MODIFIED CODE HERE
```

```
        # i18n: format XX seconds as "XXs"
```

```
        return _("%02ds") % (seconds)
```

```
    minutes = seconds // 60
```

```
    if minutes < 60:
```

```
        seconds -= minutes * 60
```

```
        # i18n: format X minutes and YY seconds as "XmYYs"
```

```
        return _("%dm%02ds") % (minutes, seconds)
```

```
    # we're going to ignore seconds in this case
```

```
    minutes += 1
```

```
    hours = minutes // 60
```

```
    minutes -= hours * 60
```

```
    if hours < 30:
```

```
        # i18n: format X hours and YY minutes as "XhYYm"
```

```
        return _("%dh%02dm") % (hours, minutes)
```

```
    # we're going to ignore minutes in this case
```

```
    hours += 1
```

```
    days = hours // 24
```

```
    hours -= days * 24
```

```
    if days < 15:
```

```
        # i18n: format X days and YY hours as "XdYYh"
```

```
        return _("%dd%02dh") % (days, hours)
```

```
    # we're going to ignore hours in this case
```

```
    days += 1
```

```

weeks = days // 7
days -= weeks * 7
if weeks < 55:
    # i18n: format X weeks and YY days as "XwYYd"
    return _("%dw%02dd") % (weeks, days)
# we're going to ignore days and treat a year as 52 weeks
weeks += 1
years = weeks // 52
weeks -= years * 52
# i18n: format X years and YY weeks as "XyYYw"
return _("%dy%02dw") % (years, weeks)

```

In `simplemerge.py`, we switched the return values so any intersecting points would be reversed. This fault injection causes some tests to fail but not all.

```

def intersect(ra, rb):
    """Given two ranges return the range where they intersect or None.

    >>> intersect((0, 10), (0, 6))
    (0, 6)
    >>> intersect((0, 10), (5, 15))
    (5, 10)
    >>> intersect((0, 10), (10, 15))
    >>> intersect((0, 9), (10, 15))
    >>> intersect((0, 9), (7, 15))
    (7, 9)
    """
    assert ra[0] <= ra[1]
    assert rb[0] <= rb[1]

    sa = max(ra[0], rb[0])
    sb = min(ra[1], rb[1])
    if sa < sb:
        return sb, sa    //MODIFIED CODE HERE (SWITCH sa AND sb)
    else:
        return None

```

In the `templatefilters.py`, we added 1 to every count. This will cause every test to fail, but the understanding that one small change/mistake in the code can result in failures and gives us a better understanding of the testing process.

```
def count(i):
    """count: List or text. Returns the length as an integer."""
    return len(i) + 1 // MODIFIED CODE HERE
```

In the namespaces.py, we altered the “if” statement to return ‘Empty List’ instead of []. This will only fail if the val inserted is None.

```
def tolist(val):
    """
    a convenience method to return an empty list instead of None
    """
    if val is None:
    return 'Empty List'           //MODIFIED CODE HERE
    else:
    return [val]
```

In worker.py, we altered the try return statement to return double the amount of cpu’s on the system. This will cause all tests to fail, but it was the only way to cause a failure with the given code.

```
def countcpus():
    """try to count the number of CPUs on the system"""
    try:
        return multiprocessing.cpu_count() * 2 //MODIFIED CODE HERE
    except NotImplementedError:
        return 1
```

Test ID	Component	Method()	Requirements	Input	Expected Output	Actual Output	Pass/Fail
01	progress	fmtremaining()	Break a given integer (representing time in seconds) into largest units of time possible	100	1m40s	1m40s	PASS
02	simplerange	intersect()	Given the range of two points, calculate and return the point of overlap between them	(0,100), (50,150)	(50, 100)	(100, 50)	FAIL
03	worker	countcpu()	Returns the number of processors available to the operating system (This test assumes you are running a single-processor virtual machine, so it will fail if more than 1 processor is available to the operating system)		1	2	FAIL
04	templatefilters	count()	Return the length of the given string	"abcde"	5	6	FAIL
05	namespaces	tolist()	Return elements in given array concatenated into one element in a larger array	"1" "2" "3" "4"	["1234"]	["1234"]	PASS
06	progress	fmtremaining()	Break a given integer (representing time in seconds) into largest units of time possible	59	59s	59s	PASS
07	progress	fmtremaining()	Break a given integer (representing time in seconds) into largest units of time possible	0	00s	0m00s	FAIL
08	progress	fmtremaining()	Break a given integer (representing time in seconds) into largest units of time possible	3601	1h01m	1h01m	PASS
09	progress	fmtremaining()	Break a given integer (representing time in seconds) into largest units of time possible	1599	59m59s	59m59s	PASS
10	progress	fmtremaining()	Break a given integer (representing time in seconds) into largest units of time possible	59	59s	1m01s	FAIL
11	simplerange	intersect()	Given the range of two points, calculate and return the point of overlap between them	(0,10), (5,15)	(5, 10)	(10, 5)	FAIL
12	simplerange	intersect()	Given the range of two points, calculate and return the point of overlap between them	(0,100), (50,50)	None	None	PASS
13	simplerange	intersect()	Given the range of two points, calculate and return the point of overlap between them	(0,10), (10,10)	None	None	PASS
14	simplerange	intersect()	Given the range of two points, calculate and return the point of overlap between them	(0,2), (1,15)	(1, 2)	(2, 1)	FAIL
15	simplerange	intersect()	Given the range of two points, calculate and return the point of overlap between them	(0,10), (10,15)	None	None	PASS
16	templatefilters	count()	Return the length of the given string	""	0	1	FAIL
17	templatefilters	count()	Return the length of the given string	" a "	5	6	FAIL
18	templatefilters	count()	Return the length of the given string	"! %YtB"	6	7	FAIL
19	templatefilters	count()	Return the length of the given string	aaaaaaaaaaaaaaa	15	16	FAIL
20	templatefilters	count()	Return the length of the given string	hello's world	10	11	FAIL
21	namespaces	tolist()	Return elements in given array concatenated into one element in a larger array	"add" "These" "Words" "Together"	I add These Words Together	I add These Words Together	PASS
22	namespaces	tolist()	Return elements in given array concatenated into one element in a larger array	"This" "is" "a" "sentence."	[This is a sentence.]	[This is a sentence.]	PASS
23	namespaces	tolist()	Return elements in given array concatenated into one element in a larger array	"1" + "2" = "3"	[1+2=3]	[1+2=3]	PASS
24	namespaces	tolist()	Return elements in given array concatenated into one element in a larger array	None	1	Empty List	FAIL
25	namespaces	tolist()	Return elements in given array concatenated into one element in a larger array	""	1	1	PASS

Number of Tests: 25
Passed: 12
Failed: 13

We observed in the results from the fault injections that there is a problem with the actual output from test number 7 and test number 10. After we injected the fault, the output for the two tests changed. It changed from what the expected output because after the first if statement, the integer representing the number of seconds modular divides with minutes causing the format change. All of the other failures happened as expected.

Chapter Six

Reflections

Kyle:

While I've had several classes that have taught various programming languages, data structures, and the like, few (if any) have really focused on actual applications of what we've learned. Any code written has been for isolated, stand alone programs. Aside from demonstrating a concept, the programs did little else. This class has focused on applying everything we've learned in other classes, along with many things we haven't, to an open source project. In this way I think this project has been one of the most valuable exercises I've done so far. While other classes often stress working alone, this project demands working together with group members, holding each other accountable, and sharing knowledge to get the job done. Although the idea of working in groups may not be ideal in the minds of many, it is regardless how most companies nowadays tend to operate. That being said, I feel it was also a valuable experience to be required to work in groups with assigned teams, similar to an actual job. Overall, I feel the project went incredibly well, especially considering how unsure I was at the beginning of the semester. I am thankful for the team I was placed in, as both more than pulled their weight throughout the course of the semester. Among the three of us, someone was always willing to step up when needed, and I was never worried about being unable to meet the various deadlines of the project.

Jesse:

It seems trite to say, but this project went along pretty painlessly. This group had the good fortune of being focused on making this project come along steadily and we made regular time commitments to work on the project. By spending around 3-5 hours working on the project as a team every Sunday, we were able to approach all of our goals with enough time to overcome hurdles that were unclear or challenging to us. The most challenging part of the experience was setting up a dynamic command for the test cases. Creating a string on the fly that executes a command was not something that I had considered before nor had any experience with. Coupling these dynamic statements with assertions proved to be it's own hurdle as we had to dig and dive deep into documentation to be sure we would be able to get the expected results. As a contrast to most college group projects, this project has been stress free, and every team member stepped up when needed. Everyone didn't have to work at full steam at all times, so having a reliable team was good as some people had to carry weight just

due to how work was split up or prior commitments, and yet I feel very strongly that the team balanced itself very well.

Chris:

Prior to beginning this class, I honestly did not know what to expect. I knew that group work was going to be a huge part but was not sure how much. I do not mind group work though I dislike the awkwardness of forming our own groups. Thankfully we were randomly placed into groups and I feel I was lucky to be put in the group that I was. We decided from the beginning that the best time for us to meet was on Sundays and spending a few hours each week we were able to plan ahead for future assignments as well as tackle current assignments. As a group the work was not overwhelming, but with little experience with testing as a group, we were able to take our time and make sure that we were on point through the semester. As a group I feel our biggest hurdle was developing our testing script. After having that set up, setting up the test cases and running them within the framework was relatively stress free. I feel as a group we all worked extremely well together and when problems arose we were able to handle them with relative ease because of the good communication and ample time that we had set for ourselves throughout the semester. Overall I would work with both partners again without hesitation.

Project Assessment:

This project was a good way to get your foot in the door with open source software. In working with Mercurial, we were exposed to the communities that work on these projects. In many cases, our team utilized Python documentation whenever we were stuck. While the overall project specifications were clear enough, there were a couple of instances where the deliverable descriptions could have been a little more specific to avoid confusion, though there was always ample time to ask questions if necessary. One specific case that appeared to cause quite a bit of confusion was the fault injection deliverable. Understanding which code to modify to inject faults was not well understood by many groups. That being said, the one time when our team was unsure of where to go next, it was easy to set up an appointment.