Chapter 5

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 four 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 the 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.

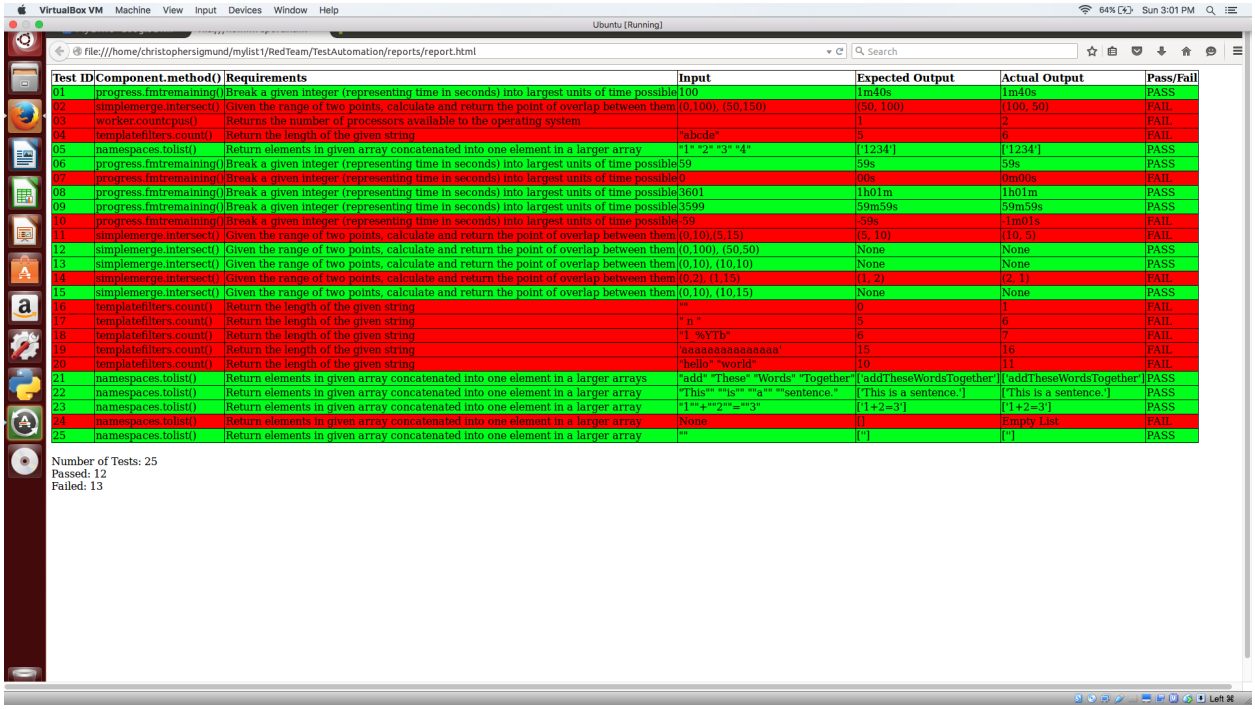
```
def count(i):
    """:count: List or text. Returns the length as an integer."""
    return len(i) + 1
```

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’
    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()
    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	simplemerge.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.countcpus()	Returns the number of processors 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	3599	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	simplemerge.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	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)	(2, 1)	FAIL
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	1	FAIL
17	templatefilters.count()	Return the length of the given string	" n "	5	6	FAIL
18	templatefilters.count()	Return the length of the given string	"1_%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" "world"	10	11	FAIL
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	[]	Empty List	FAIL
25	namespaces.tolist()	Return elements in given array concatenated into one element in a larger array	""	['']	['']	PASS

Number of Tests: 25
Passed: 12
Failed: 13

We have 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 two tests has 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.