Chris Sigmund
Jesse Hunt
Kyle Brooks

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. Our team injected a fault into each of the 5 methods tested.

The source code files are saved in the Python library. To access these files you must navigate to the appropriate folder either via file manager or terminal.

For File Manager:
Navigate to ‘computer’
Click the ‘usr’ folder
Click the ‘local’ folder
Click the ‘lib’ folder.
Click the 'python2.7' folder
Click the ‘'dist-packages' folder
Click the ‘mercurial’ folder.

For Terminal:
Navigate to /usr/local/lib/python2.7/dist-packages/mercurial/

After navigating to the mercurial folder, you will need to modify the permissions of the files that are to be edited.

Modifying Permissions In Terminal:
While in the mercurial folder, execute the following commands:
sudo chown <user> progress.py
sudo chown <user> templatefilters.py
sudo chown <user> simplemerge.py
sudo chown <user> namespaces.py
sudo chown <user> worker.py

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 < 60:
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 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 []
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() except NotImplementedError:

return 1 //MODIFIED CODE HERE
```