# Application to Fortescue Metals Group

Shane Mulligan

*<2020-01-07 Tue>*

## Contents



**Other formats** `Word` application-to-fortescue-metals-group.doc
`PDF` application-to-fortescue-metals-group.pdf

To Fortescue Metals Group,

Please consider me for the position of Senior Software Tester.

In the specific area of HIL Testing I have 2 years of solid experience from working at Crown Robotics Technology Center ( RTC ) in Penrose, Auckland. I also have a year of building and testing a C++ application for TracMap

(details below). I have also accrued a significant amount of testing experience over the last 8 years working as a Software Engineer on embedded software, software as a service (SASS) and continuous integration (CI) as a service.

I have been a practicing software engineer for over a decade now with my first job in web development in 2005. I've been programming for 20 years altogether. I hope that you will look past the chink in my armour which is the amount of HIL Testing experience I have and help me to take the next step in my career.

I believe I can perform this task and will continue to make myself useful after the first year.

Yours sincerely,

Shane Mulligan

## 0.1 Testing experience

| | Company | Context | |
|---|---|---|---|
| **HIL Testing** | Crown | Localisation, smoke, integration tests. | 2 years (2016, 201 |
| **C++ testing** | TracMap | C++ application testing | 20 |
| **JavaScript testing** | TracMap | SAAS testing | 20 |
| **Python testing** | TracMap | SAAS testing | 20 |
| **C++ unit tests** | Crown | C++ application dev | 2015, 20 |
| **Python testing** | Crown | Selenium, HIL | 2016, 20 |
| **Golang testing** | CodeLingo | | 20 |
| **Experience with Gitlab CI/CD tools** | CodeLingo | Automating CodeLingo | 8 months (2018, 201 |
| **Experience with code change tracking** | Crown | Jenkins | 20 |

## 0.2 Summary of Crown work

**Preamble** While I was at Crown I managed to automate everything that I did and put it all under my terminal system; I can't demonstrate this but from my blog you may infer what I have the potential to make.

I turned managing the HIL rigs into a fully automated process. I was very proud of my work. When I started out, the HIL rigs were a black box; An engineer would bring a keyboard to a HIL rig, connect a monitor to it and work through a featureless shell.

Once I had fully automated the HIL rigs, I could launch any jenkins job from my terminal, search for and relaunch jobs using a fuzzy finder, restart hardware components, log into various components via other components, even when IP addresses changed, etc.; I logged all HIL rig jobs.

Also, I was able to integrate all of the research trucks seamlessly into the same environment which I used to manage the HIL rigs. I refrained from doing so, however, unless there was a request to do so.

The trucks then benefited from the same search and debugging tooling I had made; If anyone required logs generated from EBL, I would know exactly how to find them, which ones were the newest, etc., what was inside them (I had an H5 log parser).

I was not assigned an assistant so the grunt work of managing the HIL rigs and automating them was solely my responsibility.

The majority of my experience in HIL testing comes from working at Crown on their localisation software; EBL and Dual-Mode. For the my first 2 years there I recall the following responsibilities and tasks:

- Create documentation on the Wiring Loom, and components (software and hardware).
- Document and be the source of information on the capabilities of each of the 8 HIL rigs.
- Create specifications for new HIL rigs to be built to requirements
- Maintain the operational status of the HIL rigs.
- Maintain the existing HIL tests.

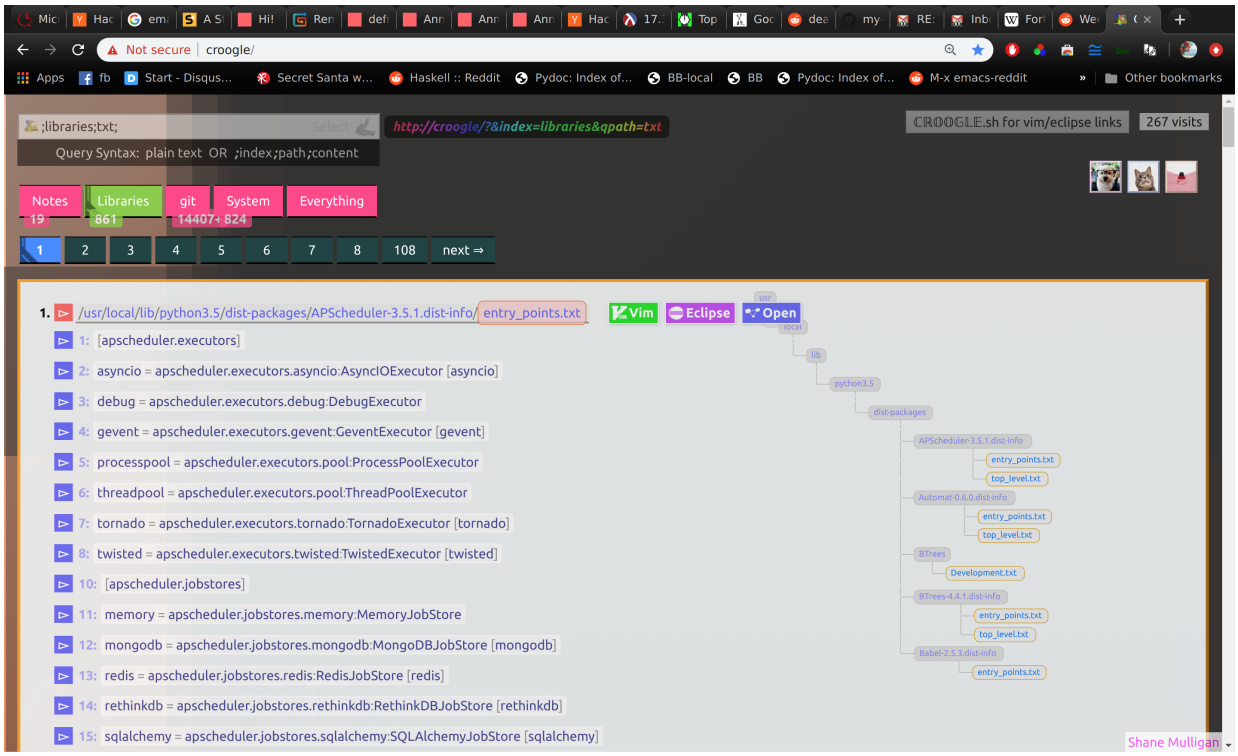
  - Update system software and HIL test dependencies, fix broken code as a result of updates, new code, patches or new hardware.

- Report on the operational status of the HIL rigs.
- Classifying and debugging test failures on the HIL rigs.
  - RTM computer (several revisions with varying capabilities)
    * EBL (C++ application)
    * Other servers running on the RTM
  - Selenium (EBL web front-end automation)
  - HIL components
    * Arduino
    * Wifi attenuator
  - Logging everything and ensuring those logs are searchable. I used SphinxSearch for this.
- Creating workarounds for HIL rig failures
  - Lighting problems.
- Find the root cause of failures which may have gone back months or hundreds of builds/revisions ago. I built a database for this and search commands.
- Debug network faults in the HIL rigs.
- Debug failing Python tests.
- Analyse logs from the HIL rigs
- Create new jenkins jobs for new tests including:
  - The testing of new hardware.
  - The testing of Gazebo simulations.
- Integrate new jobs for and including:
  - Dual Mode.
  - Projects from the interns.
- Run a development/experimental jenkins server.
- Look into Atlassian Bamboo.
- Look into ElasticSearch and LogStash.
- Manage jenkins:
  - Extending Jenkins to provide more information for developers.
  - C++ application compilation / build matrix
  - Restart servers.
- Analyse logs to
  - Look for the cause of failure.
  - Find the point of failure from the top down.
    * Jenkins > Python > C++ > Hardware.
- Delegate tasks to other developers for fixing tests.

### 0.2.1 Some innovations I made

I automated the HIL rigs so that everything was operational from my own machine. I could inspect any part of the HIL rig system in seconds without opening up a web browser. I built emacs modes and a filesystem.

1. Search engine




This included:

- A query language
- Generated graphs to keep track of and correlate errors


rtcseleniumerror: timed out wa
fail: test_bootstraptruck (tes
no ebl position messages recei
timed out waiting for css sele
Autotest Build
attributeerror: 'eblstartuptes
exception: timed out waiting f
fail: teststopviolationtrigger
assertionerror: number of dete
error: test_1_odomandimucalibr
file /usr/lib/python2.7/socke
hil 1
hil 2
hil 3
hil 4
3347
3346
3345
3344
3342
3341
3340
3339
3338
3337
3336
3334
3333
3332
3328
3326
3325
3324
3323
3321
3320
3319
3318
3317
3316
3315
3314
3313
3312
3311
3310
3308
3307
3306
3304
3303
3301
3300
3299
3298
3297
3296
3295
3293
3292
3291
3290
3289
3285
3283
3282

I built my own LogStash-like database.

I set up this search engine to assist me with debugging errors from many different sources. This is before LogStash was an option.

2. I built new HIL test tooling for debugging and management

   - I automated the HIL rigs for remote management.
   - Record video of test rigs with Python traces as subtitles.
   - Allow for interactively debugging the HIL tests, remotely. This was done on-site initially.

#### 0.2.2 My typical day

On a daily basis I would be writing mainly Python and bash, but also JavaScript, awk and GNU Makefiles, assisting Java and C++ developers with running their code on the HIL rigs.

#### 0.2.3 Other languages I encountered

I also had to use Plan9 Makefiles, C, C++, Golang and Groovy (for jenkins).

#### 0.2.4 InfoLink

In the last 6 months of working at Crown I was involved in the development of their mounted control panel, InfoLink. I made the startup screen in C, the software update system in bash and javascript and helped a colleague with a Basler 3D camera in C++.

While working on InfoLink I got a small amount of experience writing tests with the mocking framework `trompeloeil`.

1. Testing InfoLink on actual Trucks

   The process of testing out new versions of the InfoLink firmware was very tedious and a little bit of a black box which means frequent trips to and from the truck. But I automated building and deployment of the `js/node/electron` application to the truck and also the logging to and from the InfoLink server. This was so I was able to see the messages to and from the application and the server. Keep in mind that this is all happening across multiple machines, but the information is right at hand while I am debugging.

#### 0.2.5 EBL application testing

I had experience writing unit tests in C++ using the `catch` library.

#### 0.2.6 EBL HIL testing

I maintained HIL integration tests that utilised the `unittest` library.

#### 0.2.7 Debugging

I was able to combine all text logs including:

- python unittest
- application build versions from git
- jenkins metadata
- selenium
- firefox and chrome errors
- hardware versions

- EBL application errors
- C++ tracing
- javascript (v8) logging
- kernel messages
- etc.

into a single log streaming on the screen while testing.

I built the same environment to also provide me with a python shell mid-test as soon as there was an error on jenkins.

I interfaced my debugging environment to jenkins' build directories.

To make these things I used:

- UNIX tools to achieve my logging goals (ssh, awk, jq, sed, etc.).
- Python interact, python trace, Xpra, Xephyr, etc. to achieve my interactivity for HIL tests.

#### 0.2.8 C++

While I was at Crown I was also the person people came to when they needed to find their C++ build logs, C++ coverage data, `h5` logs (usually for searching for `lost events`).

### 0.3 Summary of TracMap work

My first exposure to HIL and SIL was at TracMap where I developed application firmware in C++ without supervision under various combinations of hardware and software components.

#### 0.3.1 TM4/TM5 application firmware

The test matrix factors would have looked something like this:

| Build toolchain | Application Firmware | Head unit type | GPS type | Dis |
|---|---|---|---|---|
| metabuild (busybox shell) | Flight v1 | TM4 (real) | gpssim (c/ncurses, flight sim + speader) | ven |
| sysman (perl) | Flight v2 | TM5 (real) | gpssim (javascript, replaying logs) | ven |
| | Ground v1 | TM4 (VirtualBox) | real (garmin) | ven |
| | Ground v2 | TM5 (VirtualBox) | real (other vendors, etc.) | |

#### 0.3.2 Broken headunits

I also diagnosed problems with broken headunits as they were returned to TracMap for servicing.

#### 0.3.3 TracLink - Development and Testing

This is an online service for remotely interacting with trucks, reporting and planning.

I wrote unit tests in JavaScript using the `jasmine` testing framework.

#### 0.3.4 SAAS

I had a year of web development developing TracLink, a SAAS product for interacting with the fleet.

This involved:

- writing database migration scripts in SQL
- extending the database
- writing jasmine unit tests

#### 0.3.5 Automation

I had automated the entire building, testing and debugging process of the TM5 head unit; It could all be done remotely via a terminal, including the GPS/flight simulation software which was an ncurses program.

I stuck to using terminal-based applications for everything I could, including miniterm for interacting with the GPS while decoding messages.

With regard to testing the head unit's user interface, I took extra time to get the application running on VirtualBox so could then stream it over SSH, and work from home if I wanted.

Over New Year's Day I took my laptop with me to a beach in the far reaches of NZ and did a little testing. One day I'd like to be controlling robots on the moon!

### 0.4 Summary of work at CodeLingo

**Preamble** I started automating CodeLingo when I began the job.

> Just like my work at Crown I managed to automate the entire platform except much quicker than last time.
>
> I made 410 CodeLingo-specific scripts while I was working there and 2000 scripts for general automation of my terminal.
>
> It's hard to talk about but I can provide provide access to my scripts on GitHub on request.
>
> In the first month I automated the repository research phase of CodeLingo, before which relevant repositories had to be found manually. I used my GitHub search engine which I had built while studying a PgDip in 2018 to find repositories which were both active and valid and had the user defined specifications we needed to target. I then proceeded to automate the platform to run on these repositories.
>
> Once it was established that I had automated the entire system we started a marketing campaign which lasted a month. We made automated bug fixes and pull requests to over 1000 repositories.
>
> I was offered a permanent position but I declined citing it was too stressful and I needed to be paid more for the work I was doing.

—

I will defer to my blog articles about CodeLingo for the TLDR.

CodeLingo vs Linters: TLDR // Bodacious Blog

I helped CodeLingo to document bugs, automate their platform, emails and also worked on a lexicon which can be imported into their DSL, CLQL.

I did some code generation, Google BigQuery, Golang, Python and bash.

Tremendous Task: Searching for code on GitHub with BigQuery and GHTorrent // Bodacious Blog

## 1 Job description requirements

### 1.1 Develop testing procedures for new projects

Yes, at Crown.

### 1.2 Experience with Gitlab CI/CD tools

- Yes, I have automated lengthy processes with GitHub and Gitlab.
- Also, I have scripts, CLI tools and emacs plugins which I use to interact with both GitHub and Gitlab.

### 1.3 Experience with testing C++ code base

Yes.

### 1.4 Experience with gtest suite and gmock

I believe we used `catch` as our c++ testing framework and `trompeloeil` for mocking.

I was involved in creating tests but not selecting the testing technology.

### 1.5 Experience with docker

I started with linux containers circa. 2011. I have built scripts to deal with docker containers and kubernetes.

Here is an example of a workaround to do port forwarding for k8s when `kubefwd` would time out unexpectedly.

I had to build this in order to automate CodeLingo. Also I designed it for the other developers to use in their own workflows.

I've deemed the following code to be insensitive, but made some text replacements to be sure.

```
#!/bin/bash
export TTY

read -r -d '' hosts <<HEREDOC
127.1.27.1 codelingo-ast-common codelingo-ast-common.lexicon codelingo-ast-common.lexicon.svc.clus
127.1.27.2 codelingo-ast-cpp codelingo-ast-cpp.lexicon codelingo-ast-cpp.lexicon.svc.cluster.local
127.1.27.3 codelingo-ast-csharp codelingo-ast-csharp.lexicon codelingo-ast-csharp.lexicon.svc.clus
127.1.27.4 codelingo-ast-go codelingo-ast-go.lexicon codelingo-ast-go.lexicon.svc.cluster.local
127.1.27.5 codelingo-ast-golint codelingo-ast-golint.lexicon codelingo-ast-golint.lexicon.svc.clus
127.1.27.6 codelingo-ast-php codelingo-ast-php.lexicon codelingo-ast-php.lexicon.svc.cluster.local
127.1.27.7 codelingo-ast-phplint codelingo-ast-phplint.lexicon codelingo-ast-phplint.lexicon.svc.c
127.1.27.8 codelingo-ast-python27 codelingo-ast-python27.lexicon codelingo-ast-python27.lexicon.sv
127.1.27.9 codelingo-ast-python36 codelingo-ast-python36.lexicon codelingo-ast-python36.lexicon.sv
127.1.27.10 codelingo-vcs-git codelingo-vcs-git.lexicon codelingo-vcs-git.lexicon.svc.cluster.loca
HEREDOC

cathosts() {
    printf -- "%s\n" "$hosts"
}

sudo killall -9 kubefwd

gateway_host="$1"
: ${gateway_host:="XXXXXXXXXXXXXXXXXX.XXXXXXXXX.compute.amazonaws.com"}

kubectl get pods --all-namespaces | grep "^lexicon" | grep "Running" | awk '{print $2}' | awk 1 |
(
exec 0</dev/null
pod_name="$(printf -- "%s" "$pod_hash" | cut -d - -f1-3)"
local_ip="$(cathosts | grep -P "\b$pod_name\b" | awk '{print $1}')"
remote_ip="$(kubectl describe pods "$pod_hash" -n lexicon | grep "^IP:" | awk '{print $2}')"
ssh -fNT -L $local_ip:9999:$remote_ip:9999 "$gateway_host"
ssh -fNT -L $local_ip:8888:$remote_ip:8888 "$gateway_host"
)
done
```

### 1.6 SIL and HIL testing experience

#### 1.6.1 CodeLingo

- Automated git bisect

### 1.7 Experience with C++ static analysis tools

- Bullseye (C++ code coverage)
- cppcheck
- CodeLingo
- Semmle

### 1.8 Experience with code change tracking, traceability and auditing.

Yes, my debugging environment helped me to chase down errors to the specific git revision of a given project which caused them.

We used a git repository submodule system but given someone's jenkins build configuration, I was able to recombine it to test other parameters, including combinations of different application components.

- Jenkins (C++, python)
- Go Modules (golang)

While at CodeLingo, I maintained multiple builds of the CodeLingo system which meant that one of my machines would usually be operational during times when everyone else's was broken.

### 1.9 Release management

I was not reponsible for release management at Crown but I was responsible for caring for both the development and main jenkins servers. The development server moved at a different pace but I had to correlate errors between both.

I was however, involved in setting up jobs surrounding different stages of release and was the person who provided off-hand information on the current state.

### 1.10 Simulation based testing

I have done some automation around Gazebo.

I helped in setting up Gazebo for use in the HIL rigs at Crown.

I automated all of the Xterms and logging from Gazebo. One problem I solved was with the Xterms actually.

Debugging Dual-Mode simulations with Gazebo remotely on HIL rigs was very impractical. The job ran autonomously but couldn't be interacted with.

I started by making it work on my own machine with all Xterms defering to run tmux instead. I did a similar thing at `CodeLingo`.

This replaces the `mate-terminal` program.

```
 1  #!/bin/bash
 2  export TTY
 3
 4  ( hs "$(basename "$0")" "$@" 0</dev/null ) &>/dev/null
 5
 6  pane_id="$(TMUX= tmux neww -n platform -P bash)"
 7
 8  export DISABLE_COLORIZE=y
 9
10  # Do not kill here unless I plan on restarting it
11  docker kill $(docker ps -q)
12  docker run -d -p 9411:9411 openzipkin/zipkin
13
```

```
14  while [ $# -gt 0 ]; do opt="$1"; case "$opt" in
15      --tab) {
16          shift
17
18          name=
19          command=zsh
20          while [ $# -gt 0 ]; do opt="$1"; case "$opt" in
21              -t) {
22                  name="$2"
23                  shift
24                  shift
25              }
26              ;;
27
28              -e) {
29                  command="$2"
30                  shift
31                  shift
32              }
33              ;;
34
35              *) break;
36          esac; done
37
38          # Must use tm splits (beacuse it traps INT)
39
40          new_pane_id="$(TMUX= tmux splitw -F "#{pane_id}" -P -t "$pane_id" "trap '' INT; stty stop
41          TMUX= tmux select-layout -t "${pane_id}" tiled
42
43          echo "$new_pane_id" | ds -s -q "matepane$name"
44
45          echo "$name:$new_pane_id"
46      }
47      ;;
48
49      *) break;
50  esac; done
51
52  TMUX= tmux kill-pane -t "${pane_id}" # The original pane
53  TMUX= tmux select-layout -t "${pane_id}" tiled
```

## 2 In summary

In each of the 3 jobs listed that I have worked (TracMap, Crown and CodeLingo) I have managed to get to the stage where I could do my all my work through a single terminal window, which is how I like it!

Thanks for reading!