KORAL HASSAN

INTERIM REPORT

August 26, 2018

Short-code: kbh15
CID number: 01096803
Academic Supervisor: Esther Perea
University: Imperial College London
Placement Company: Metaswitch Networks

Contents

Summary

Overview of Metaswitch

Metaswitch Networks (formerly Data Connection Ltd) is a private UK-based company that designs, develops, manufactures, and markets telecommunications software to communication service providers, equipment manufacturers and large enterprises. Metaswitch's award-winning solutions are powering more than 1,000 service providers in today's global, ultra-competitive and rapidly changing communications marketplace.

Division at the Company

Within Metaswitch, I have been working in the Fixed Voice Systems Testing division. Fixed Voice oversees the part of the business related to landline calls, which even today makes up a majority of the company's business. The Systems Testing division ensures that the company's excellence standards are met in every product even with tight schedules and daily upgrades to the software.

Overview of Projects

The main projects I have so far been involved with are;

- Product Requirement Definition Dashboard Information Prioritisation,
- SFR Dashboard Front-End,
- Automatically Raised MVS SFR's,
- MetaGo.

Reflection on Placement

The time I have spent at Metaswitch has been greatly educational. I have gained familiarity with many industry practices and standards, including virtualisation and the Linux kernel among other things. I have also had the chance to network with other software engineers and gain insight to the day-to-day routines of such a job which has helped me with solidifying a long-term career path.

Context

History of Metaswitch

The company was founded in 1981 by seven former IBM employees led by Ian Ferguson, who remains on the board of directors. The company's earliest business areas included IBM Systems Network Architecture and retail point of sale systems. In the 1990's, the company began developing network protocol software. In 2000, the company launched the Metaswitch brand, which provided soft-switches and network management systems designed to enable telephone service providers to migrate to Voice over IP (VoIP) networks while still supporting legacy telephone technologies.

They have a 30+ year history of providing high-performance, hardware-independent software to the communications industry, while solving its most difficult problems. Their communications software solutions are critical elements of the equipment of the leading industry Telecom equipment manufacturers, as well as the networks of the world's leading network operators, underpinning their most advanced data and voice services.

Metaswitch's success stems from a unique combination of their deep experience in software engineering, with superbly agile development capabilities; an extremely highly-regarded support team; and financial and organizational stability, based on a foundation of recruiting and retaining talented and enthusiastic employees, and rewarding them fairly for their contributions.

The company actively seeks out students from a handful of elite universities across England for their internship and graduate roles. Imperial is one of these universities; in fact, although they are not affiliated with the EE Department's 6-month placement scheme, several students before me have chosen this company as part of their 3rd year in the course. Even during my last month here, I have been able to collaborate with Imperial engineers from other disciplines in a way that was never available to me before. I personally discovered the company through one of the annual Career Fairs held on campus.

Metaswitch does not request any prior software knowledge from their interns, even though the work pivots around programming. What they look for at the interview stage is critical thinking and an ability to absorb new information at great velocity. In these areas, the many individual and group projects I have done at Imperial allowed me to shine through.

The Placement

I started my placement on the 23rd of April. I will be finishing it on the 23rd of September.

My team, Good Omens, consists of 6 people, and focuses on building testing tools. Previous courses I have taken in EEE has helped me tremendously in knowing how to design a piece of Software to be used as a tool, particularly High Level Programming. All our code is written in Python, using Object Oriented Programming. I had extensive familiarity with both the language and the paradigm from my studies.

Good Omens follows an agile product management framework called Scrum. This framework is the context in which all future project planning and past project accomplishments are described for the rest of this report. It is essential to the way Good Omens functions and it has led to vast amounts of professional and personal development for me. The first-year university module called Professional Engineering helped me a lot while learning to adopt a managerial system that I was completely unfamiliar with.

Purpose of Projects

Product Requirement Definition Dashboard Information Prioritisation

At Metaswitch, new software for all their products are released in synchronisation through what is called a release. The duration of time between releases is typically in the order of several months.

Product Requirement Definitions are various enhancements that the company intends to implement in the future. Product Line Managers prioritise the PRD's and schedule which ones will be implemented in each release. The development of these new features and enhancements is tracked very stringently and is a big part of the jobs of middle-managers.

The PRD dashboard is a web-app created to allow users to track PRD's. It displays the key information about all PRD's - such as deadlines, progress and owners - on one page.

I enhanced the dashboard with the purpose of making the relevant information available in a more easily digestible format. This saved time for the management of the company.

SFR Dashboard Front-End

The System Fault Reports (SFR's) are product defects, and can be raised and handled by Development, Systems Testing (ST) or Support. The raising and handling of SFR's is a constantly ongoing affair. It is crucial to maintaining a high standard of product development and not letting the company's customer experience regress.

The SFR dashboard is a web-app created to allow users to track current SFR data.

I designed and coded a front-end for the dashboard so that it could display the key information about all current SFR's - such as their current state and who is responsible for handling them - on one page, in an easily digestible format. This saved time for the management of the company.

Automatically Raised MVS SFR's

SFR's - as described above - are used for the raising and handling product defects.

MetaView Servers (MVS's) are a type of server that is capable of hosting an instance of MetaView Web. MetaView Web is a Web-based application that provides an interface for creating, configuring and administering subscribers and business groups on a range of Metaswitch hardware.

Bricks is a set of automation tools. It is used for several purposes:

- It can monitor system resources (e.g. servers that are running instances of Metaswitch software products) and track events on those resources.
- It can execute what are called runs, where it will execute a series of tests. The tests will usually involve putting certain resources under some kind of load (e.g. making them connect a certain number of phone calls per second). It will then save the results of the tests in a database for posterity.
- It will automatically run any commands you want on remote resources - such as gathering diagrams to an SFR, turning on IPS trace, upgrading resources, etc.

Bricks comes in the form of a Python package, readily extensible, and with an API supported by the Systems Testing tools team. It provides a database and a statistics User Interface

for anyone to use - either on the main Bricks server, or on the user's own computer. It supports all Fixed Voice products.

Bricks includes a series of pre-canned scripts, ready to use at the moment the package is installed. I have been extending the functionality of one of these scripts as one of my projects, in an effort to increase the automation of work within the Systems Testing Division.

The script is called RunSFRManager. It was created to automatically go through the events that have occurred in a particular Bricks monitored run and raise new SFR's or update SFR's that are "awaiting reproduction". It also copies diagnostics to the relevant SFR folders.

Before I started to work on the script, it was only capable of raising SFR's for 3 event types that are called asserts, checks, and exceptions. Also, the script could only be used on a family of resources of a particular type, called "PES". MVS's - which is what I extended the script to use on - do not belong to the "PES" family of resources.

MetaGo

In Metaswitch offices, there exists a tea-room on every floor. These rooms serve the purpose of providing refreshments as well as a space for breaks and socialising. Every tea-room contains drinks, fruits, biscuits and chocolates. These items all have an associated price. Also in every tea-room is a small tablet. These tablets run an app that was designed specifically around billing employees for tea-room edibles. Currently, the system is such that when an employee takes an item, they will find their profile on this app, they will enter what items they have taken, and the costs will be taken out of their next salary accordingly.

Every year Metaswitch holds what is called an interns week, which will be full of activities to participate in. The biggest of these activities is called the Vacathon. It is a 3-day-long hackathon where interns will form into groups and create the necessary software for an idea their group wants to realise. It is an intensive time period that encourages innovation, team-spirit, and initiative. It also gives the interns an opportunity to demonstrate the technical skills they have developed over the course of their internship. This year's Vacathon was held from the 8th of August to the 10th of August. I participated within a group of 5 people.

My group focused on a particular problem we wanted to solve; we wanted to transform tea-room transactions into a faster and smoother process. The solution we decided on was a web-app called MetaGo. MetaGo automatically processes transactions by watching a room

via web-cameras and identifying which person took what item.

Deliverables

PRD Dashboard Information Prioritisation

Deliverables relating to the PRD dashboard information prioritisation were;

- gathered requirements and feedback from the managers who use the tool the most,
- a proposed solution,
- a high-level design of the code,
- an implementation of the solution decided on,
- a hosting method for the implementation that makes the solution accessible to all employees.

SFR Dashboard Front-End

Deliverables relating to the SFR dashboard front-end were;

- an interface between the front-end and back-end that the front-end and back-end engineer have agreed upon,
- a proposed user interface along with a sketch,
- an initial protoype,
- a Minimum Viable Product (MVP),
- a fully featured product with complete functionality and no known bugs.

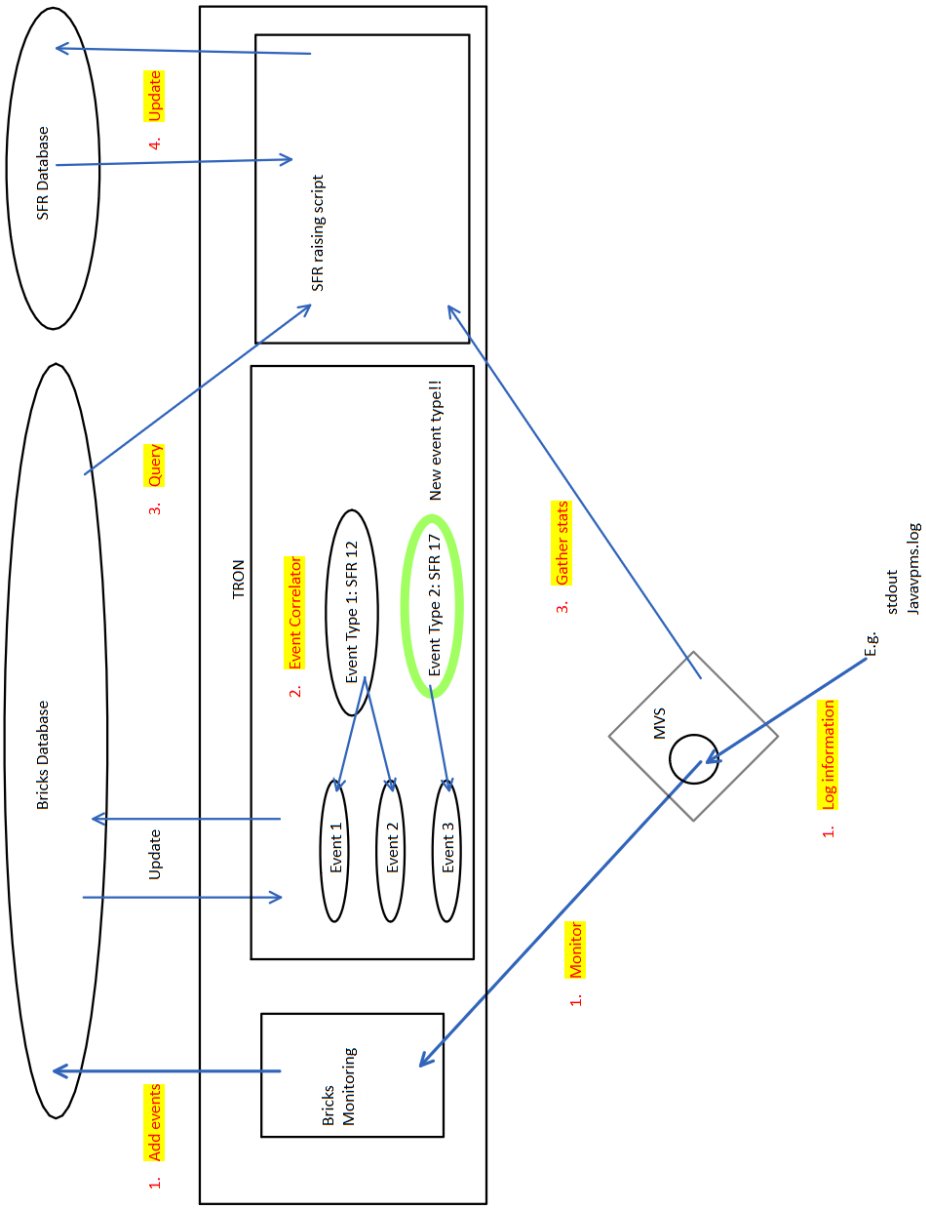
Automatically Raised MVS SFR's

Deliverables relating to automatically raised MVS SFR's were;

- test session feedback for new stacktrace delimiters in Java VPMS logs,
- implementation of monitoring java errors, java exceptions and java L4 logs,
- implementation of pre-processing and saving to a database of java errors, java exceptions and java L4 logs,

- implementation of correlating new java errors, java exceptions and java L4 logs to previously encountered ones and updating a database accordingly,
- implementation of raising and updating SFR's in the SFR database according to the correlation information,
- activating the new automatic SFR raising on all regularly used testing hardware.

Figure 1: RunSFRManager Data Flow



MetaGo

Delivarables relating to the MetaGo web-app were;

- a working prototype of proof-of-concept,
- a Pitch Deck,
- a presentation,

- an advertisement video,
- the project codebase.

Progress since Preliminary Report

PRD Dashboard Information Prioritisation

At the time of writing the Preliminary Placement Report, this project had been completed already. However, its technical details will be discussed further in this report.

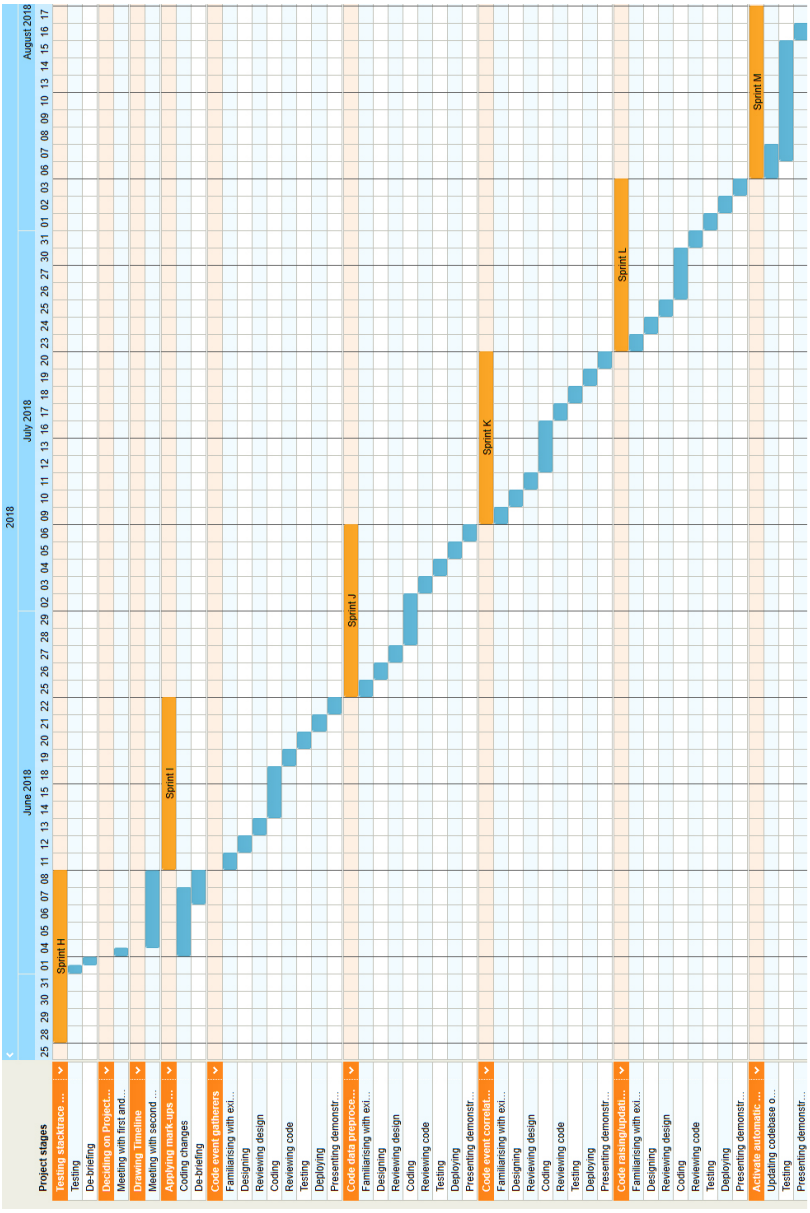
SFR Dashboard Front-End

At the time of writing the Preliminary Placement Report, this project had been completed already. However, its technical details will be discussed further in this report.

Automatically Raised MVS SFR's

At the time of writing the Preliminary Placement Report, this project was in the early planning stages and was briefly mentioned. Since then the following timeline was planned and executed.

Figure 2: RunSFRManager Gantt Chart



MetaGo

At the time of writing the Preliminary Placement Report, the date of the 2018 Metaswitch Networks Vacathon had been confirmed. Since then, MetaGo has been made into a fully functioning website, which allowed our team to win the Vacathon.

Projects

PRD Dashboard Information Prioritisation

Description

The purpose of this project was to make the relevant PRD information available in a more easily digestible format to users of the PRD dashboard. This saved time for the management of the company.

First, I gathered requirements and feedback from the managers who use the tool the most. This surveying group consisted of my second line manager and the manager of Systems Testing. The information I acquired was the following:

- The dashboard colour coding worked very well for highlighting PRD's that were struggling with deadlines or otherwise needed careful attention.
- At the time, the dashboard showed too many PRD's top pay attention to at once. While they liked having the option to see all of them, they felt that this wasn't always necessary.
- The managers had to scroll through the page to find the interesting PRD's. What they wanted to be able to do was just glance at the webpage in passing (without ever touching the mouse) and immediately see if everything was okay or if there was a problem.

With these requirements in mind I proposed the following solution:

- The PRD dashboard will have a default sorting when the web-page is loaded. The sorting order will mimic the colour coding we already use to indicate the level of urgent action required for a PRD.
- The PRD dashboard will have several filtering options. There will be a default filter on when the web-page is first loaded. This default filter will hide the PRD's that are most likely to be uninteresting (e.g. PRD's that have been completely closed with no issues of any kind).
- After the page is first loaded, the filtering options will be completely adjustable to meet the user's wishes. The method of adjustment will be easy-to-use and intuitive.

After my proposed solution was approved, I drafted a high-level design of the functionalities. The dashboard would have the following filtering options:

- initials - Input Metaswitch initials to only show PRDs owned in some way by that person. E.g. ABC. Or a group of people, e.g. ABC,HP1,JK (must be a comma separated list with no whitespace).
- release - Input a release to only show PRDs in that release. E.g. V9.4.20. Or a comma-separated list of releases e.g. V9.4.20,V9.4.30.
- show-summary - Input False to hide the PRD summaries on the dashboard.
- prd-numbers - Input a comma-separated list of PRD numbers to only show those PRDs. E.g. 1111,12231,12392.
- priority - Input the priorities to display separated by commas to only show PRDs set with that priority. E.g. 1,2.
- scheduled - Input True to only show scheduled PRDs. Input False to only show non-scheduled PRDs.
- show-closed - Input True to also show closed PRDs.
- show-only-st-tracked - Input False to also show PRDs not explicitly tracked by ST as per this article using the additional information entry ST-PRD-Tracking.
- colour-blind-mode-off - Input True to remove the symbols by the PRD number on the dashboard.
- custom-sort - Default is to sort by colour. Set to "release,prd-number" to sort instead by release and then PRD ID.
- show-all-initials - Input True to show all PRDs.
- hide-options - Input True to hide the options GUI on the PRD dashboard.

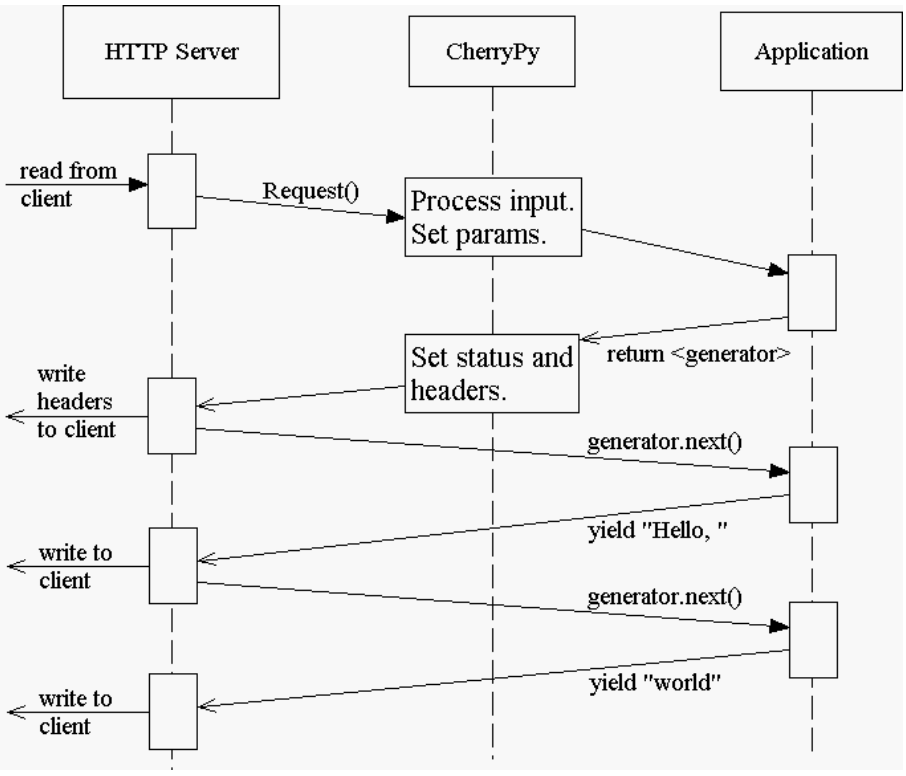
The dashboard would be sorted as;

1. Scheduled PRDs with missing dates,
2. PRDs which have missed their current deadline,
3. PRDs with at least one deadline this work week,
4. PRDs that have a deadline locked in this work week,

5. PRDs with at least one date filled in,
6. Unscheduled PRDs with no dates filled in,
7. PRDs requiring no dates,
8. Completed PRDs.

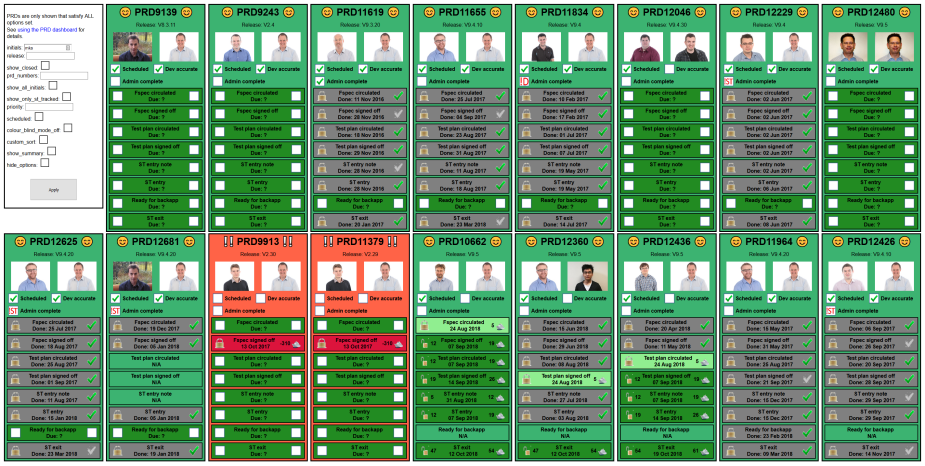
The PRD dashboard codebase leverages a Python framework called CherrPy as its Web Server Gateway Interface (WSGI). CherryPy allows developers to build web applications using object-oriented programming. I had to get familiar with this framework to make the necessary code changes.

Figure 3: CherryPy Response Stream



After a code review some testing, I downloaded my code changes to the live server, so they were in use immediately.

Figure 4: Current iteration of the PRD Dashboard



Reflection

A challenge during the requirements gathering stage of this project was liaising with several managers. Specifically, arranging a meeting around multiple busy schedules was not very straightforward. I solved this problem by applying a scheduling strategy I had learned while studying decision mathematics. I asked them to inform me of all their available time-slots, I then went through each manager from the least available to the most available and arranged a meeting with them at our first matching available time-slot. My professionalism also helped a lot by ensuring I communicated clearly, arranged liaison significantly early, and was empathetic to the demands and constraints of their managerial duties.

During this project I had the opportunity to educate myself on how web servers functioned from a networking perspective. I also had to reinforce my knowledge of Python which is a very common language in the industry and also helped me further down the line during my placement.

Future improvements that could be made to the dashboard include a visual way of adjusting the filters that have been implemented. The success of this feature would involve a lot of re-iteration based on feedback, so long periods of elapsed time would be expected.

The communication theory I studied during my university degree also helped me a lot during this project. It exposed the rationale behind the industry's design decisions when it comes to computer networking; which made new concepts more intuitive.

SFR Dashboard Front-End

Description

I designed and coded a front-end for the dashboard so that it could display the key information about all current SFR's - such as their current state and who is responsible for handling them - on one page, in an easily digestible format. This saved time for the management of the company.

The project started with me talking to my team members about what data I needed to be passed to the front-end. We also discussed what data structure best represented type of data we were going to pass through. One point we paid careful attention to was that no data processing was to be done on the front-end. All business logic and variable manipulation should be on the back-end with the front-end just displaying the data it has received. This is a good design principle that allows clear separation of functionality requirements.

After we had reached an agreement, I made a rough sketch of the layout I was thinking of. I showed this to my team-mates and discussed possible improvements.

After I was happy with what the best way to display the relevant data was, I began programming. I started by writing in Hyper Text Markup Language (HTML) to display the data on the web-page. I also added hyperlinks to the relevant SFR's for when a statistic was clicked on. Once that was passing my tests, I presented a demonstration of it to my team.

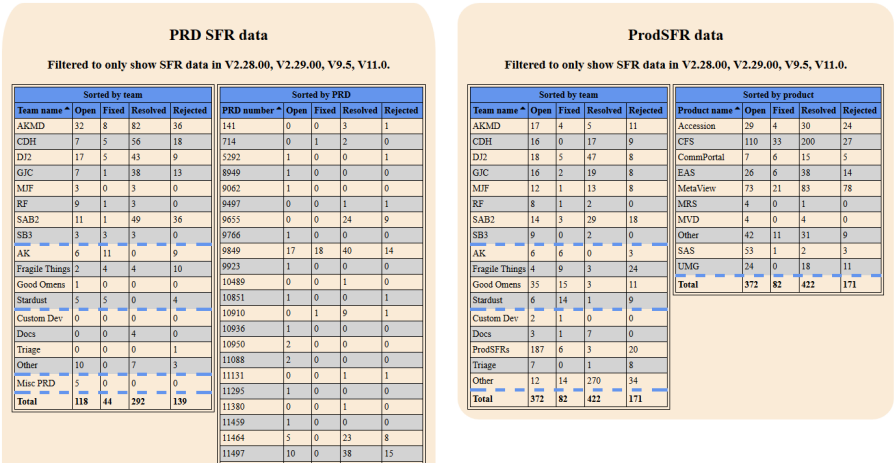
The next thing I did was writing the Cascading Style Sheets (CSS) to modify the colours, sizes and general appearance of the objects on the web-page. At this point my static page was capable of displaying the data and was formatted to be easy to read and navigate. I had a minimum viable product that could theoretically be shipped right away and meet all minimum requirements. This is when I presented a demonstration to my second line manager as he was one of the people that would be using the tool frequently. Moreover, he was colourblind so I checked with him that the styling of the page made it easy for him to differentiate distinct parts of the page.

After getting my second line manager's feedback and approval of the MVP, I started writing the JavaScript that would make the page dynamic and responsive to the user's inputs. I added highlighting to any rows or cells the mouse was hovering over. I also added pop-up bubbles when the mouse hovered over a team's name. These pop-ups gave a list of the team

members. The final interactive feature was the ability sort the tables on the page by any column, ascending or descending, by clicking on the respective table headers.

The finishing touches to the product included adding filtering options for which SFR's were included in the statistics being shown, extensive testing, and integration with the back-end.

Figure 5: Current iteration of the SFR Dashboard



PRD SFR data

Filtered to only show SFR data in V2.28.00, V2.29.00, V9.5, V11.0.

Sorted by team

Team name	Open	Fixed	Resolved	Rejected
AKMD	32	8	82	36
CDH	7	5	56	18
DJ2	17	5	43	9
GJC	7	1	38	13
MJF	3	0	3	0
RF	9	1	3	0
SAB2	11	1	49	36
SB3	3	3	3	0
AK	6	11	0	9
Fragile Things	2	4	4	10
Good Omens	1	0	0	0
Stardust	5	5	0	4
Custom Dev	0	0	0	0
Docs	0	0	4	0
Triage	0	0	0	1
Other	10	0	7	3
Misc PRD	5	0	0	0
Total	**118**	**44**	**292**	**139**

Sorted by PRD

PRD number	Open	Fixed	Resolved	Rejected
141	0	0	3	1
714	0	1	2	0
5292	1	0	0	1
8949	1	0	0	0
9062	1	0	0	0
9497	0	0	1	1
9655	0	0	24	9
9766	1	0	0	0
9849	17	18	40	14
9923	1	0	0	0
10489	0	0	1	0
10851	1	0	0	1
10910	0	1	9	1
10936	1	0	0	0
10950	2	0	0	0
11088	2	0	0	0
11131	0	0	1	1
11295	1	0	0	0
11380	0	0	1	0
11459	1	0	0	0
11464	5	0	23	8
11497	10	0	38	15

ProdSFR data

Filtered to only show SFR data in V2.28.00, V2.29.00, V9.5, V11.0.

Sorted by team

Team name	Open	Fixed	Resolved	Rejected
AKMD	17	4	5	11
CDH	16	0	17	9
DJ2	18	5	47	8
GJC	16	2	19	8
MJF	12	1	13	8
RF	8	1	2	0
SAB2	14	3	29	18
SB3	9	0	2	0
AK	6	6	0	3
Fragile Things	4	9	3	24
Good Omens	35	15	3	11
Stardust	6	14	1	9
Custom Dev	2	1	0	0
Docs	3	1	7	0
ProdSFRs	187	6	3	20
Triage	7	0	1	8
Other	12	14	270	34
Total	**372**	**82**	**422**	**171**

Sorted by product

Product name	Open	Fixed	Resolved	Rejected
Accession	29	4	30	24
CFS	110	33	200	27
CommPortal	7	6	15	5
EAS	26	6	38	14
MetaView	73	21	83	78
MRS	4	0	1	0
MVD	4	0	4	0
Other	42	11	31	9
SAS	53	1	2	3
UMG	24	0	18	11
Total	**372**	**82**	**422**	**171**

Reflection

There was a particular problem I was struggling with at the beginning of this project. There were several tables on the web-page I was building whose structures and code looked very similar and repetitive. I was having trouble with maintaining the "Dont Repeat Yourself" (abbreviated as "DRY") coding principle. I began searching for a solution. That is when I found Cheetah, an open source template engine and code-generation tool written in Python. Learning how to use Cheetah allowed me to build my tables in an iterative manner. This improved the readability, maintainability and extensibility of my code. It also saved me time and effort.

This project emphasised to me, the importance of clear communication and specifications between engineers. Doing this successfully saved me many hours in compatibility with the back-end and designing the website to be colurblind friendly from the beginning.

The most useful enhancement to website right now would be to make the columns and rows of the tables collapsible and capable of being re-arranged. This functionality will require

a significant level of familiarity with JavaScript to implement in a neat and maintainable way. This will probably not happen soon since I am the only one on the team with experience in using JavaScript.

I had previously acquired many of the front-end design concepts I utilised for this project, while building a professional portfolio for my university industrial placement. This was a big help.

Automatically Raised MVS SFR's

Description

I extended the functionality of Brick's pre-canned RunSFRManager script as one of my projects, in an effort to increase the automation of work within the Systems Testing Division. The script was created to automatically go through the events that have occurred in a particular Bricks monitored run and raise new SFR's or update SFR's that are "awaiting reproduction". After my changes the script can also be used on events of the types java error, java exception and java L4 log, which are events that occur on MVS's.

I started by testing a new feature that the Development Operations had created for us that allowed us to identify when one event on an MVS ended and another started. It did this by inserting a special string of characters - "===" - in the Java VPMS logs when recording a new event. After I made sure that we were happy with the new feature and that it served our purposes, I wrote a de-brief in the SFR that tracked the development of this new feature, detailing my testing and conclusions.

Then I programmed a system that automatically sets up the necessary settings on MVS's before a Bricks run in order to successfully monitor for the new event types.

The third step in the development was to decide on a way of processing the data so that it could later be associated to past instances of the same problem. I decided to have two data fields for each event instance, called "Detail1" and "Detail2". I used Python regular expressions (regex's) to process the stacktraces of events, since it was a widely used tool for data extraction from text.

After relevant data gets extracted events, they must be compared to events of the past to see if there have been any previous encounters. If compared information that was too specific like the exact state of the computer at the time of the failure, then the script would

fail to ever recognise past instances of the same problem as identical. On the other hand, it could lead to a lot of bugs being missed if the script only compared very general data like the type of failure and from then on discarded all future instances that type of failure as having been dealt with already. I decided to compare events on their event types, the Python module that caused the event, and the trace of function calls that triggered the event. The compared data had to be identical between two events for them to be considered different instances of the same events.

After events are correlated, the relevant SFR's must get raised or updated. This part of the process had been implemented previously already. However, I added some enhancements. The notes section of an SFR now specifies when exactly the issue was hit. This makes it easier to investigate what caused the issue. The notes section now also informs the reader of which folder the relevant diagnostics and diagrams have been gathered to.

Reflection

There was an unexpected new requirement during the execution of the project. The closure of the project consisted of updating all instances of the Bricks codebase on regularly used testing hardware, for my changes to take effect. Furthermore, the planning allowed for a one week period after activation where the effects of the script would be observed for quality assurance purposes. During this observation week, a team from the Development division informed us that there was an entire class of significant events happening on MVS's that we had been completely unaware of in the past. They had discovered this family of events through an SFR that had been automatically raised thanks to the code changes of this project. As a response to this, we created a new event type for monitoring called Java L4 logs. We also started automatically raising SFR's for encountered instances of Java L4 logs. These code changes did not take too much time since the script had already been modified to work on MVS's.

This incident highlights the usefulness of the project as a whole. Furthermore, it emphasised the importance of buffer zones in a timeline, from a project management perspective.

This incident suggests that further useful work could be done investigating other possible events happening on MVS's that could be of interest to the company, and so should be tracked. The most efficient way of achieving this goal would be to practicing vigilance and looking for patterns when analysing future SFR's.

The Embedded Systems university module helped me a lot during this project, because it gave me familiarity with MicroPython. MicroPython is very similar to Python, which was the sole programming language used for this project.

MetaGo

Description

My Vacathon group focused on a particular problem we wanted to solve; we wanted to transform tea-room transactions into a faster and smoother process.

By the time our 5-person team had been formed and the Vacathon had started, we had already had a brainstorming session and decided on the solution to the problem that we would attempt. This saved us time, which is the most valuable resource in hackathons. The solution we decided on was a web-app called MetaGo. MetaGo automatically processes transactions by watching a room via web-cameras and identifying which person took what item.

We started the hackathon by dividing the code into 5 distinct sections that could be developed in parallel and integrated later. My responsibility was setting up the back-end of our website, which would use Django. Django is a free and open-source web framework, written in Python, which follows the model-view-template (MVT) architectural pattern. I was also in charge finding a website hosting solution and getting our website a domain name. I decided to use Heroku for hosting because it was highly compatible with Python, had a free hosting tier, and most importantly was designed for rapid app deployment. Our website domain name was "metago.herokuapp.com".

At noon of the second day, we had a meeting for each member to de-brief the team on progress made. At this point we felt that we were ready to start integrating the different sections of code we had been developing. Since I was familiar with the back-end and the hosting platform, I started to work with individual team members to inject their respective functions into the existing codebase.

At noon of the third day, all our sections had been integrated and we started working on our Pitch Deck, demonstration, presentation and advertisement video. At 4 PM that day we introduced our product to the judges of the hackathon and the other teams. We won the hackathon.

Reflection

The biggest challenge we faced during the hackathon was integrating the different parts of our web-app after developing them. In particular, the API we used for facial recognition utilised a C library; however the rest of our web-app was written in Python. When the problem was encountered, it became the highest priority task for our group to solve it. We immediately brainstormed 5 possible solutions and then each started looking into one of these solutions. In the end the solutions. The solution that worked in the end was to install what was called a buildpack on our Heroku account. Buildpacks are scripts that are run when an app is deployed. They are used to install dependencies for the app and configure the environment.

For me, this problem we ran into showed the importance of clear communication, especially under stress and time constraints. We should have established early in the project that if any programming language other than Python is being utilised at all - even if not explicitly being programmed in - this must be immediately declared to the team for approval.

Future improvements to the project could include developing an accompanying mobile app for the Apple App Store and the Google Play Store; so that the project doesn't purely depend on a website designed with laptops and desktop PC's in mind. This would require familiarity with either Java or Kotlin and either C# or Swift. The apps being developed to full functionality with the necessary documentation, comments and logs would take around 40 hours each for one person.

The university modules that helped me most in this hackathon were Entrepreneurship Online and Machine Learning. Entrepreneurship Online taught me how to identify needs in the market, come up with a solution, and pitch a product. Machine Learning taught me the underlying principles behind facial recognition, which was useful when I was helping my team-mates make a well-informed decision on with facial recogniiton library to use.

Conclusion

All in all, the industrial placement has been an incredibly positive experience for my development, both technically and personally. It has equipped with the skills necessary when applying to Software Engineering positions. It has also allowed me a glimpse into corporate culture and engineering ethics. I now feel much more prepared to define my future career goals and take the necessary actions to achieve them.

Bibliography

Metaswitch [online] Available at: <https://www.metaswitch.com/>

The Future of Metaswitch [online] Available at: <http://www.lightreading.com>

Metaswitch Networks LTD [online] Available at: <http://www.companiesintheuk.co.uk/ltd/metaswitch-networks>

Metaswitch opens NFV code as Project Calico [online] Available at: <https://www.theregister.co.uk>

The Home of Scrum [online] Available at: <https://www.scrum.org/>