Final IT Report

Vivian Ye-Ting Dang (z5118468)

April 14, 2019

Contents

Abstract

This report contains details relating to the compulsory industrial training that I have carried out as required for the Engineering degree at the University of New South Wales. It is written in the format provided by the UNSW Engineering Industrial Training Guideline. I have completed a total of 60 days (12 weeks) of industrial training over one full-time placement.

Chapter 1

Introduction

I completed my industrial training internship with the programming languages (PL) and formal methods (FM) team at Cog Systems Pty Ltd in Sydney. Cog Systems leads the industry in secure connected device implementations across world governments, defense organizations and corporate enterprises. They have adopted an embedded solution built on modularity, proactive security, trustworthiness, and adaptability to enable highly secure connected devices.

The PL and FM team at Cog Systems makes use of the latest academic research as well as industry practices to enable IoT components to be provably free of software bugs and of security vulnerabilities.

I worked with Kai Engelhardt, a senior principal engineer at Cog and also the leader of the PL and FM team. As a software engineering intern, I was given a wide variety of tasks and responsibilities to ensure that I have as much exposure to real world industry practice as possible. Some of these tasks included designing, building and testing backends of a domain specific language (DSL) compiler, a library for inter-process communication (IPC) as well as the implementation of an input/output primitive for the language. These tasks required predominantly programming, debugging and testing as well as frequent discussions with my project lead and extensive research from academic sources and documentation.

I worked full-time between the 19th of November 2018 and 15th February 2019 (with a week worth of leave in between) for a total of 60 days.

Chapter 2

Engineers Australia Competencies

2.1 Knowledge and Skill Base

(1.2) Conceptual understanding of the mathematics, numerical analysis, statistics, and computer and information sciences which underpin the engineering discipline

I was given the opportunity to work on developing a DSL that describes a concurrent system based on message-passing IPC. These systems are typically described in the literature using Kripke structures [Kripke, 1963] and reasoned about using Linear Temporal Logic (LTL) [Pnueli, 1977]. An example of such a system is given in Figure 2.1. There are two processes, one representing a traffic light and the other representing a vehicle. The vehicle is only able to drive while the light is green. This property can be expressed in LTL as

$$\square(\text{Drive} \Rightarrow \text{Green})$$

PROMELA (Process or Protocol Meta Language) is a verification modelling language that allows the modelling of concurrent processes. The process can then be visualised and automatically verified using Spin, which is a tool to conduct logical verification of concurrent software. I developed part of the compiler of the DSL that translated from the DSL to PROMELA.

I undertook a relevant course at UNSW, Concepts of Programming Languages (COMP3161), which taught me useful skills that I could apply in the development of a programming language. This involves being able to work with a parser and typechecker, which is what I was required to do in the DSL compiler in order to translate from the DSL to PROMELA.

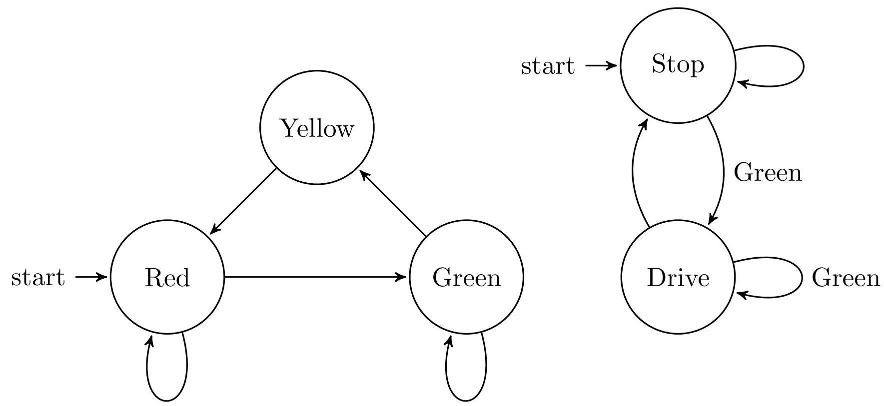


Figure 2.1: An example of a Kripke Structure that models a traffic light

In addition, I also developed a message-passing IPC library to allow the users to simulate their DSL programs based on the concept of channels similar to Hoare [1985]. For this, I used the Portable Operating System Interface (POSIX) standardised library `pthreads` [Mueller, 1993].

(1.4) Discernment of knowledge development and research direction within the engineering discipline

Over the last decade, formal methods aimed at complete, end-to-end verification of software systems have become more popular, with many successes including verified compilers for C [Leroy, 2009] and ML [Kumar et al., 2014], verified theorem provers Milawa [Davis and Myreen, 2015] and Candle [Kumar et al., 2016], a verified conference system [Kanav et al., 2014], a crash-resistant file system [Chen et al., 2015], the concurrency verification in CertiKOS [Gu et al., 2016], the verified cryptographic routines of OpenSSL HMAC [Beringer et al., 2015], the verified distributed system Ironfleet [Hawblitzel et al., 2015] and many more.

The aforementioned IPC library was intended to facilitate the compilation of the DSL to the new research language Cogent. Cogent is a recent research language aimed at reducing the cost of end-to-end formal verification of systems [Amani et al., 2016]. This opens up a new frontier for verified systems, by allowing users to easily prove once and for all that their software meets requirements in the form of a correctness specification. This proof also integrates with the world-famous verified seL4 microkernel [Klein et al., 2009], itself a game-changer in the world of high-assurance systems software.

Cogent is a purely functional language based on uniqueness types [O'Connor et al., 2016]. This type system allows specifications written in a mathematical style to be compiled to efficient C implementations [Wadler, 1990]. Cog envisions connecting their DSL up to the Cogent language, thus enabling their software to be formally verified to be free of correctness bugs.

(2.2) Fluent application of engineering techniques, tools, and resources.

My role involved a lot of programming, specifically in Haskell [2010], which is a functional programming language that is seeing increasing use in high-assurance softare development. Partly for this reason, Cog decided to use Haskell for the DSL development. I completed the course Software System Design and Implementation (COMP3141) at UNSW which equipped me with Haskell proficiency as well as best practice in high-assurance software engineering.

As mentioned in the previous section, I also learnt PROMELA and how to use SPIN during my placement in order to be able to implement the part of the compiler that translates the DSL into PROMELA. I achieved this mainly through reading documentation and asking for help from my direct supervisor at work when I got stuck. This is notably different from the learning environment at UNSW, where skills are imparted directly by teaching staff. I had to get used to considerable self-directed learning, a point I will expand upon in Chapter 3.

Another tool I used is Git, which is a common collaboration tool in the industry, used to track multiple versions of software source code and their authorship, allowing multiple users to collaborate on the same code base and to isolate when bugs are introduced in the development process. While I was exposed to this tool in many courses that I have completed throughout my degree at UNSW, this is one of the first times I used it in a large-scale professional context.

(2.4) Application of systematic approaches to the conduct and management of engineering projects

Our project followed the Agile method [Beck et al., 2001], which involves repeating the Software Development Life Cycle (SDLC); a summary of the cycle is shown in Figure 2.2.

The cycle starts with requirements analysis, this includes identifying the problem statement with my supervisor and other members of the company and come up with specification that contains the acceptance criteria for all the features that have to be completed in the tasks assigned to me.

The next stage is design, which means evaluating different approaches to solving the problem and weighing the pros and cons for each approach to decide which one seems best for fulfilling the requirements, also taking time and resources into consideration.

Then I move on to the development phase, where I implement a minimal viable product (MVP) that satisfies the previously stated acceptance criteria. This is a minimal implementation that *only* satisfies the acceptance criteria, and is intentionally kept small to aid maintainability and flexibility for future changes.

After this, the product undergoes testing. This testing is based on the acceptance criteria, as well as other internal engineering concerns, such as maintanability and performance. Based on the information gained through testing, as

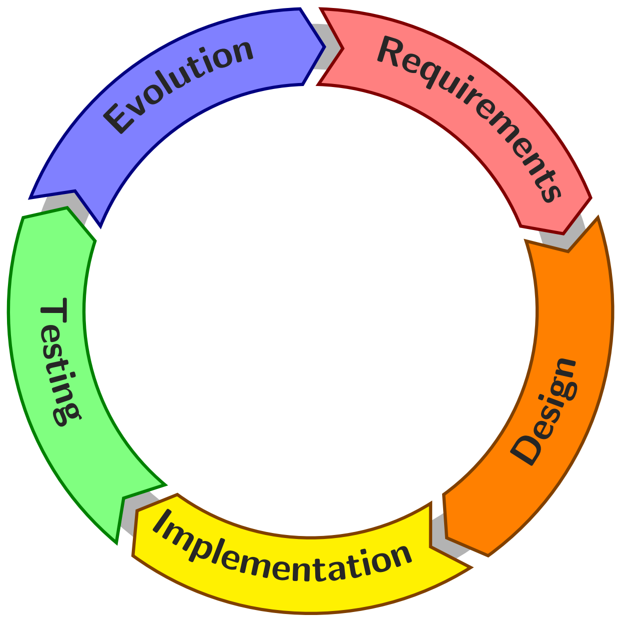


Figure 2.2: The Software Development Life Cycle (SDLC)

well as feedback from stakeholders, changes are made to the specification, and faults are isolated and fixed. Once the appropriate specification changes have been made, the cycle repeats again, this time focusing on the implementation of new features or other requested changes.

(3.5) Orderly management of self, and professional conduct

While this report may seem to indicate that I was given a number of concrete tasks to achieve during my placement, the tasks I completed were mostly the result of collaborative research efforts with the rest of my team.

This meant that it was crucial for me to concretize and reify the requirements placed on my team into specific tasks that could be completed with precise evaluation and acceptance criteria.

As this project had a considerable focus on new research, it was vital that I maintained my connections with UNSW research staff, with whom Cog collaborated for this project. This also enabled me to further my own studies in this discipline, and explore new avenues for future research that I am carrying out in my undergraduate thesis.

This was a fortunate confluence of ambitions, as I wished to improve my knowledge in the formal methods and programming languages spaces, Cog wished to improve their products using this knowledge, and my UNSW friends and colleagues were able to assist both me personally and Cog professionally in this context.

In order to ensure that my tasks were completed in a timely manner, I broke down my tasks into small, achievable components with self-set deadlines. This deadline pressure is an effective way to ensure that I do not let myself become distracted by interesting research problems that do not align with organisational objectives.

Chapter 3

Reflection & Conclusion

My experience in this industrial placement brought into stark relief several interesting distinctions between learning in an academic context and through an internship position.

One of the main aspects of the industry that I noticed is the emphasis on self-directed learning and development. In a university course, I could reasonably expect to find all the information necessary for understanding a particular topic to be available from the provided course materials. By contrast, the industrial setting poses much more open-ended problems where the solutions are not necessarily known, even by more senior members of staff. Therefore, I had to undertake significant study projects independently in order to gain the skills necessary to achieve the milestones set for me. If I needed assistance, I would have to take initiative to seek it out myself, rather than rely on my supervisors or other superiors to provide me with information.

While this is significantly different from most UNSW coursework, it is not that dissimilar to the undergraduate thesis project I am now undertaking. This project, being a research thesis, also involves a number of open-ended problems, and independent, self-directed study.

Another aspect of industry that I observed is that team-work is the default. Individual, independent projects are exceptionally rare in the software industry, as opposed to a university setting, where, despite some group-work projects, the majority of work is necessarily conducted individually. Because of this, it is necessary to forge social relationships with colleagues in an industry environment, whereas in university, such relationships are strictly an extra-curricular activity.

The structure of an industry workplace is also markedly different from that of university. For example, an eight-hour work day took considerable getting used to, coming from the sporadic class timetables of university life. Also, the physical work environment differs too, as I was given a fixed desk in an open-plan office, whereas at UNSW, work is conducted wherever I can find a place to sit.[1]

I also was able to significantly expand my technical knowledge during this placement, which has in turn assisted me in my coursework and undergraduate thesis. Seeing as my thesis also involves the Cogent language, and Programming Languages more generally, much of what I learned at Cog applies directly to my current studies.

Overall, I feel that this placement was immensely valuable to me. It helped me to develop my professional communication skills, widen my technical knowledge, and forge new personal relationships.

[1] Also, this is becoming increasingly difficult.

Bibliography

Saul A. Kripke. Semantical considerations on modal logic. *Acta Philosophica Fennica*, 16(1963):83–94, 1963.

Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57, 1977. doi: 10.1109/SFCS.1977.32.

C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Inc., 1985. ISBN 0-13-153271-5.

Frank Mueller. A library implementation of posix threads under unix. In *USENIX Conference 93*, pages 29–41, 1993.

Xavier Leroy. A formally verified compiler back-end. *Journal of Automated Reasoning*, 43(4):363–446, December 2009. ISSN 0168-7433. doi: 10.1007/s10817-009-9155-4.

Ramana Kumar, Magnus Myreen, Michael Norrish, and Scott Owens. CakeML: A verified implementation of ML. In Peter Sewell, editor, *Symposium on Principles of Programming Languages*, pages 179–191, San Diego, January 2014. ACM. doi: 10.1145/2535838.2535841.

Jared Davis and Magnus O. Myreen. The reflective Milawa theorem prover is sound (down to the machine code that runs it). *Journal of Automated Reasoning*, 55(2):117–183, 2015. doi: 10.1007/s10817-015-9324-6.

Ramana Kumar, Rob Arthan, Magnus O. Myreen, and Scott Owens. Self-formalisation of higher-order logic semantics, soundness, and a verified implementation. *Journal of Automated Reasoning*, 56(3):221–259, 2016. doi: 10.1007/s10817-015-9357-x.

Sudeep Kanav, Peter Lammich, and Andrei Popescu. A conference management system with verified document confidentiality. In *International Conference on Computer Aided Verification*, volume 8559 of *Lecture Notes in Computer Science*, pages 167–183. Springer, 2014.

Haogang Chen, Daniel Ziegler, Tej Chajed, Adam Chlipala, M. Frans Kaashoek, and Nickolai Zeldovich. Using Crash Hoare logic for certifying the FSCQ file system. In *Symposium on Operating Systems Principles*, pages 18–37, Monterey, CA, October 2015. ACM.

Ronghui Gu, Zhong Shao, Hao Chen, Xiongnan (Newman) Wu, Jieung Kim, Vilhelm Sjöberg, and David Costanzo. CertiKOS: An extensible architecture for building certified concurrent OS kernels. In *Symposium on Operating Systems Design and Implementation*. ACM, November 2016.

Lennart Beringer, Adam Petcher, Katherine Q. Ye, and Andrew W. Appel. Verified correctness and security of OpenSSL HMAC. In *Security Symposium*, pages 207–221, Washington, DC, US, August 2015. USENIX. ISBN 978-1-931971-232.

Chris Hawblitzel, Jon Howell, Manos Kapritsos, Jacob R. Lorch, Bryan Parno, Michael L. Roberts, Srinath T. V. Setty, and Brian Zill. IronFleet: proving practical distributed systems correct. In *Symposium on Operating Systems Principles*, pages 1–17, Monterey, CA, USA, October 2015. ACM. doi: 10.1145/2815400.2815428.

Sidney Amani, Alex Hixon, Zilin Chen, Christine Rizkallah, Peter Chubb, Liam O'Connor, Joel Beeren, Yutaka Nagashima, Japheth Lim, Thomas Sewell, Joseph Tuong, Gabriele Keller, Toby Murray, Gerwin Klein, and Gernot Heiser. Cogent: Verifying high-assurance file system implementations. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 175–188, Atlanta, GA, USA, April 2016. doi: 10.1145/2872362.2872404.

Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. sel4: Formal verification of an os kernel. In *Symposium on Operating Systems Principles*, pages 207–220, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-752-3. doi: 10.1145/1629575.1629596.

Liam O'Connor, Zilin Chen, Christine Rizkallah, Sidney Amani, Japheth Lim, Toby Murray, Yutaka Nagashima, Thomas Sewell, and Gerwin Klein. Refinement through restraint: Bringing down the cost of verification. In *International Conference on Functional Programming*, Nara, Japan, September 2016.

Philip Wadler. Linear types can change the world! In *Programming Concepts and Methods*, 1990.

Haskell. Haskell 2010 language report, 2010. URL `https://www.haskell.org/onlinereport/haskell2010/`.

Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken

Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development, 2001. URL http://www.agilemanifesto.org/.

Chapter 4

Appendix

UNSW SYDNEY | Australia's Global University

UNSW Engineering

Industrial Training Employer Evaluation Form

Complete this Employer Evaluation Form for every Industrial Training placement you undertake.

1. At the start of the placement, the supervisor and student create up to 3 goals to be achieved.
2. Arrange a date to meet at the end of the placement
3. Supervisor and student review the 3 goals that were created – both providing comments in needed.
4. Supervisor to rate the student's professional attributes using Engineer's Australia Table 3 Professional and Personal Attributes: Elements and Indicators
5. Supervisor to complete the total days worked and provide proof via company email or company letter to student
6. Student to complete student reflection
7. Student to upload this form and proof of total days worked (company email or company Letter) to Moodle

Placement Information

Company Name:	Cog Systems Pty Ltd	Supervisor Name:	Kai Engelhardt
Supervisor Email:	kai@cog.systems	Supervisor Phone:	N/A
Student Name:	Vivian Ye-Ting Dang	UNSW zID:	z5118468
Start of placement:	19th Nov 2018	End of placement:	15th Feb 2019
Student's Job Title:	Software Engineer		

Goals (To be agreed upon at the start of the placement by the supervisor and student)

Use the SMART criteria to create the goals:

SPECIFIC	MEASURABLE	ATTAINABLE	RELEVANT	TIME-FRAME
Define the goal as much as possible	Quantify or suggest an indicator of progress	Make sure the goal is not out-of-reach or below standard performance	How does the goal tie into your key responsibilities?	The goal should have a time limit

Goal 1: Write a backend for a DSL compiler to translate from one programming language's features to another.
Estimated time: 5 weeks.

Goal 2: Write a programming language library for inter-process communications (IPC).
Estimated time: 5 weeks

Goal 3: Extend a DSL with debugging functionalities.
Estimated time: 2 weeks.

Review of goals (To be completed by supervisor at the end of the placement)

Poor = Development below expectations

Fair = Would benefit from more experience

Good = Mostly competent in this area

Excellent = Demonstrates excellent competence in this area

Goals	Poor	Fair	Good	Excellent
Write a backend for a DSL compiler to translate from one programming language's features to another. Estimated time: 5 weeks.	○	○	○	◉
Supervisor Comments: works perfectly, completed on-time.				
Student Comments: hands-on experience on the application of Programming Languages concepts.				
Write a programming language library for inter-process communications (IPC). Estimated time: 5 weeks	○	○	○	◉
Supervisor Comments: Served as a good basis for future work on libraries.				
Student Comments: Required to learn to navigate poorly documented technologies.				
Extend a DSL with debugging functionalities. Estimated time: 2 weeks.	○	○	◉	○
Supervisor Comments: Basic prototype works, still more work to be done in the future.				
Student Comments: Learnt more about language development				

Total Days worked (To be completed by supervisor)

The days that are written here are used to credit the student towards the 60-day requirement to complete Industrial Training.

In addition, please supply the student with evidence confirming the total days worked, by either:

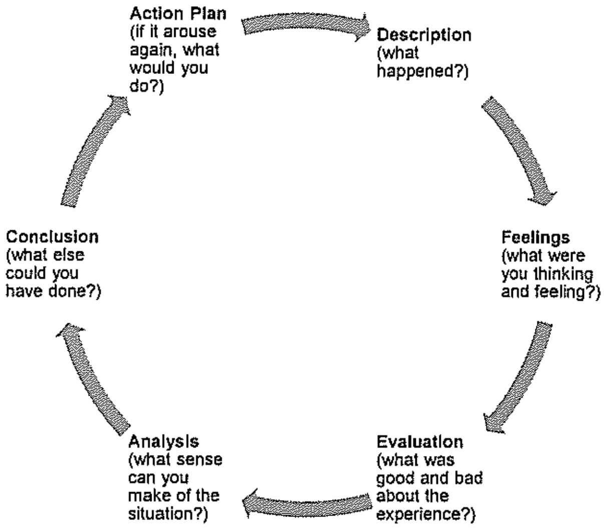
- An email with company signature (translated if required)
- Letter on company letterhead (translated if required)

TOTAL NUMBER OF DAYS WORKED: 60

Student Reflection **(To be completed the student at the end of the placement)**

Use the questions and flow chart below to help you write a brief reflection on your placement and the feedback given by your supervisor.

This reflection will help you write your Written Report.



Gibbs G. (1988) *Learning by Doing: A guide to teaching and learning methods.* London: Further Education Unit.

1. A description of what happened during your placement.

Worked as a software engineer at Cog systems, designing, implementing, and testing a DSL prototype.

2. How did the feedback from your supervisor make you think and feel?

Stimulate learning opportunities and gives me a wide variety of tasks to do.
Encourage curiosity and experimentation.

3. What was good and bad about your placement?

Good: friendly working environment, flexible hours, technically interesting work.
Bad: Poorly documented codebase.

4. What have you learnt from the placement?

Many aspects of compiler and DSL design. Software practices like version controls. Isolating and detecting bugs in large codebases.

5. What do you now need to develop, learn or change now you have had this experience?

Independent research from academic sources and online materials.

6. What actions are you now going to put into place before you graduate?

I will be starting an honours thesis this year in this area of study. Which will improve the aforementioned research skills.

Student's Professional Attributes (To be completed by supervisor at the end of the placement)

Refer to Engineer's Australia Table 3 Professional and Personal Attributes: Elements and Indicators

Poor = Development below expectations | Good = Mostly competent in this area

Fair = Would benefit from more experience | Excellent = Demonstrates excellent competence in this area

Student's Professional Attributes	N/A	Strongly Disagree	Disagree	Agree	Strongly Agree
Ethical conduct and professional accountability.	○	○	○	○	●
Supervisor Comments: n.a.					
Effective oral and written communication in professional and lay domains.	○	○	○	○	●
Supervisor Comments: n.a.					
Creative, innovative and pro-active demeanour.	○	○	○	●	○
Supervisor Comments: n.a.					
Professional use and management of information.	○	○	○	○	●
Supervisor Comments: n.a.					
Orderly management of self and professional conduct.	○	○	○	○	●
Supervisor Comments: n.a.					
Effective team membership and team leadership.	○	○	○	●	○
Supervisor Comments: n.a.					

Supervisor's signature: Date: 2019/02/15

Student's signature: Date: 2019/02/15