Correct by contruction

Author:
J. Maree FOURIE

December 3, 2019

"Writing is hard; writing is hard because thinking is hard. It is easier to think you are thinking."

Leslie Lamport

Abstract

Correct by contruction

by J. Maree FOURIE

ToDo. ...

Contents

List of Figures

List of Tables

List of Abbreviations

CbyC	**C**orrect **by** **C**onstruction
TLA	**T**emporal **L**ogic **A**ctions

Chapter 1

Correct by Construction (CbyC)

1.1 Overview

A lot of time and money is lost dealing with the consequences of software bugs. The ”Correct by Construction” software development methodology proposes to limit software bug by:

1. making it difficult to introduce errors, and
2. detecting and removing errors as early as possible.

To achieve this we apply the following strategies [2]:

- use formal notation for deliverables;
- use tool-supported methods to validate deliverables;
- carry out small steps and validate the deliverables for each step;
- sate things only once;
- design software that is easy to validate;
- do the hard things first.

Software development can be divided in to two parts: specification and design; and implementation. To make this applicable to my work environment I chose to use TLA+ as a specification language and C# as a programming language.

1.2 Specification and Design

Regular language is inherently imprecise. Writing a specification and design in regular language leaves the documents open to conflicting interpretations.

Using a mathematical, formal, language to define the specification and high-level design of a system provides a precise description of the system’s behaviour and a precise model of its characteristics. Using a mathematical language also enables the use of automated tools to verify the specification and design [2].

Formal specifications and designs are more precise and this forces us to understand issues and questions before we start coding [4].

1.2.1 TLA$^+$

TLA$^+$ has a large community and is proven useful in industry [6]. TLA$^+$ was created by Leslie Lamport in the late 1980's. TLA (Temporal Logic of Actions) is a simple variant of Pnueli's temporal logic. Most TLA specifications consists of ordinary non-temporal mathematics. Temporal logic is only used when appropriate. TLA is written in an assertional reasoning style [7].

1.3 Implementation

CbyC design methodology is based on information flow. The information flow is defined using a contract-based notation. The contract-based notation is used to define the abstract state of the program and the information relationships across boundaries [2].

CbyC suggests the generation of evidence of correctness by using programming languages and tools, that allow for verification and analysis [2].

Testing is usually the main method of verification and validation. The normal testing method follows these steps:

1. test individual units;
2. integrate them and test the integration;
3. then test the system as a whole.

This approach is inefficient because unit testing is ineffective and expensive. Unit testing is ineffective because most errors are interface errors, not internal errors in units. Unit testing is expensive because you have to build test harnesses to test units in isolation [4].

A more efficient and effective approach is to incrementally built the system from the top down. Each build is a real (if small) system and the system can be completely exercised in a real environment. This reduces the integration risk [4].

1.3.1 C#

Design by contract

CbyC program design is based on information flow expressed as code contracts [2]. C# does not have build in code contracts any more, but it is very simple to implement code contracts using standard C# language features.

Property based testing

C# has very little static analysis tools able to mathematically prove correctness. As an alternative we will use property based testing to exercise the code and show correctness [1] [5]. We will be using the FsCheck framework for our property based tests [3].

Chapter 2

Specification and Design

2.1 TLA+ overview

2.2 High level specification

2.3 Refining the specification

Chapter 3

Development

3.1 Design by contract

3.2 Property based testing

Chapter 4

Formal methods in Agile

4.1 ...

Appendix A

TLA+

A.1 Installing

A.2 More information

Appendix B

FsCheck

B.1 Installing

B.2 More information

Appendix C

Hoare Logic

C.1 Program execution

$$\{P\}Q\{R\}$$

If the assertion P is true before the program Q executes, then the assertion R will be true after Q has executed.

C.1.1 Axion of Assignment

$$\vdash P_0\{x := f\}P$$

x is a variable identifier;

f is an expression;

P_0 is obtained from P by substituting x with f;

C.1.2 Rule of Consequence

$$\text{If } \vdash \{P\}Q\{R\} \text{ and } \vdash R \supset S \text{ then } \vdash \{P\}Q\{S\}$$

$$\text{If } \vdash \{P\}Q\{R\} \text{ and } \vdash S \supset P \text{ then } \vdash \{S\}Q\{R\}$$

This rule allows the strengthening of the precondition and/or the weakening of the postcondition.

C.1.3 Rule of Composition

$$\text{If } \vdash \{P\}Q_1\{R_1\} \text{ and } \vdash \{R_1\}Q_2\{R\} \text{ then } \vdash \{P\}(Q_1; Q_2)\{R\}$$

C.1.4 Rule of Iteration

$$\text{If } \vdash \{P \wedge B\}S\{P\} \text{ then } \vdash \{P\} \text{ while } B \text{ do } S\{\neg B \wedge P\}$$

Bibliography

[1] Koen Claessen and John Hughes. "QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs". In: *Proceedings of the ACM SIGPLAN International Conference on Functional Programming, ICFP* 46 (Jan. 2000). DOI: 10.1145/1988042.1988046.

[2] Martin Croxford and Dr. Roderick Chapman. "Correctness by Construction: A Manifesto for High-Integrity Software". In: *The Journal of Defense Software Engineering* 18.12 (Dec. 2005), pp. 5–8.

[3] *FsCheck*. https://fscheck.github.io/FsCheck/index.html. [Online; accessed 3-December-2019].

[4] Anthony Hall and Roderick Chapman. "Correctness by Construction: A Manifesto for High-Integrity Software". In: *IEEE Software* 19.1 (Jan. 2002), pp. 18–25.

[5] Richard Hamlet. "Random Testing". In: *Encyclopedia of Software Engineering*. Wiley, 1994, pp. 970–978.

[6] Leslie Lamport. *Industrial Use of TLA+*. https://lamport.azurewebsites.net/tla/industrial-use.html. [Online; accessed 3-December-2019]. 2018.

[7] Leslie Lamport. *Specifying Systems*. Boston, Massachusetts: Pearson Education, 2003.