Correct by contruction

Author:
J. Maree FOURIE

December 17, 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

1 Correct by Construction (CbyC) 1
- 1.1 Overview . . . 1
- 1.2 Specification and Design . . . 1
  - 1.2.1 TLA$^+$ . . . 2
- 1.3 Construction . . . 2
  - 1.3.1 C# . . . 2
    - Design by contract . . . 2
    - Property based testing . . . 2

2 Agile and Correct by Construction 3
- 2.1 Manifesto for Agile Software Development . . . 3
- 2.2 Correct by Construction development workflow . . . 3

3 Applying the methodology 5
- 3.1 First iteration . . . 5
- 3.2 Second iteration . . . 5
- 3.3 The rest of the life cycle . . . 5

A TLA+ 7
- A.1 TLA$^+$ Toolbox . . . 7
- A.2 More information . . . 7

B FsCheck 9
- B.1 Installing . . . 9
- B.2 More information . . . 9

C Hoare Logic 11
- C.1 Program execution . . . 11
  - C.1.1 Axion of Assignment . . . 11
  - C.1.2 Rule of Consequence . . . 11
  - C.1.3 Rule of Composition . . . 11
  - C.1.4 Rule of Iteration . . . 11

Bibliography 13

List of Figures

2.1 CbyC development process. 4

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 [4]:

- 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 [4].

Formal specifications and designs are more precise and this forces us to understand issues and questions before we start coding [6].

1.2.1 TLA$^+$

TLA$^+$ has a large community and is proven useful in industry [8]. 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 [9].

1.3 Construction

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 [4].

CbyC suggests the generation of evidence of correctness by using programming languages and tools, that allow for verification and analysis [4].

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 [6].

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 [6].

1.3.1 C#

Design by contract

CbyC program design is based on information flow expressed as code contracts [4]. C# does not have built 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 [2] [7]. We will be using the FsCheck framework for our property based tests [5].

Chapter 2

Agile and Correct by Construction

2.1 Manifesto for Agile Software Development

The manifesto sets out the overarching principles of agile software development [1]:

> We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
>
> - **Individuals and interactions** *over* processes and tools
> - **Working software** *over* comprehensive documentation
> - **Customer collaboration** *over* contract negotiation
> - **Responding to change** *over* following a plan
>
> That is, while there is value in the items on the right, we value the items on the left more.

2.2 Correct by Construction development workflow

The main properties of the CbyC development process are [3]:

- each life-cycle phase can be validated;
- the semantic gap between life-cycle phases are reduced, making it possible to show the conformance of later life-cycle phases with earlier phases.

The CbyC development process products are shown in Figure 2.1.

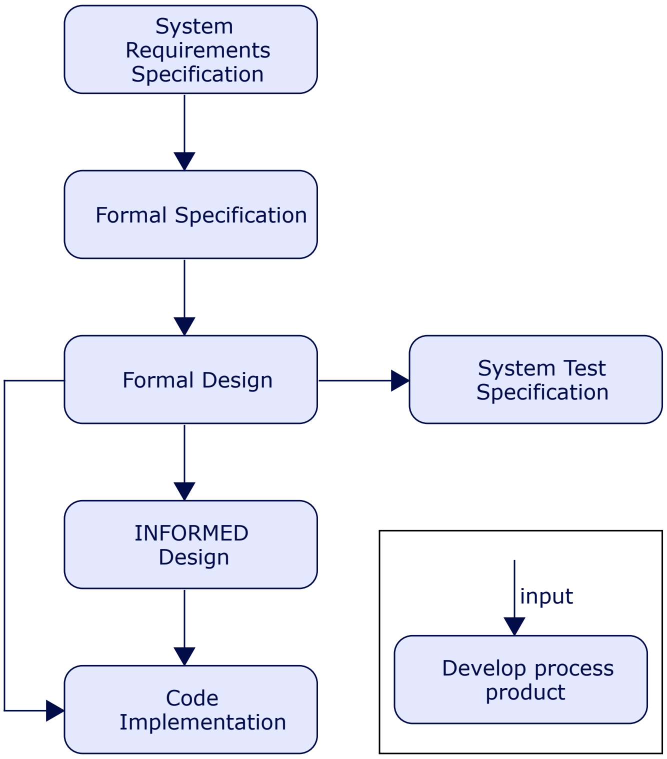
System Requirements Specification
Formal Specification
Formal Design
System Test Specification
INFORMED Design
Code Implementation
input
Develop process product

FIGURE 2.1: CbyC development process.

- r.

Chapter 3

Applying the methodology

3.1 First iteration

3.2 Second iteration

3.3 The rest of the life cycle

Appendix A

TLA+

A.1 TLA$^+$ Toolbox

Use the TLA$^+$ Toolbox to evaluate your specifications. You can download the latest toolbox from the tlaplus GitHub repository [11].

There is more information about the toolbox on the TLA+ website [10]

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] Kent M. Beck et al. *Manifesto for Agile Software Development*. `http://agilemanifesto.org`. [Online; accessed 7-December-2019]. 2001.

[2] 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`.

[3] David Cooper. *Tokeneer ID Station EAL5 Demonstrator: Summary Report*. `https://www.adacore.com/tokeneer`. [Online; accessed 12-December-2019]. Aug. 2008.

[4] 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.

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

[6] Anthony Hall and Roderick Chapman. "Correctness by Construction: Developing a Commercial Secure System". In: *IEEE Software* 19.1 (Jan. 2002), pp. 18–25.

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

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

[9] Leslie Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN: 032114306X.

[10] Leslie Lamport. *The TLA$^+$ Toolbox*. `https://lamport.azurewebsites.net/tla/toolbox.html`. [Online; accessed 4-December-2019]. 2019.

[11] *TLA$^+$ GitHub release*. `https://github.com/tlaplus/tlaplus/releases/latest`. [Online; accessed 4-December-2019].