Programming by Contract-based Behavior Driven Development Framework

Ahmad Salim Al-Sibahi, Hildur Uffe Flemberg, Søren Sønderby Nielsen, and Nicolai Skovvart*

IT University of Copenhagen

* Supervisors: Andrzej Wasowski and Rolf-Helge Pfeiffer

Abstract. TODO
The four sentence method
- State the problem
- Say why it's an interesting problem
- Say what your solution achieves
- Say what follows from your solution

Maximum 150 words in this course.
Example: Feature modeling is an important element of development of Software Product Lines. Much research exists on feature modeling languages, and feature modeling tools. It is surprising though, that no experience reports exist of using feature modeling in industrial practice, and no characteristics of industrial models is openly available. In this work we identify, present, and analyze two real life feature models of operating system kernels, eCos and Linux. We find that many assumptions about feature models made by tool builders in academia do not hold in reality, and thus many tools are potentially not useful; real models are substantially larger, and with higher density of constraints than previously anticipated. 109 words

Table of Contents

1 Introduction

- General statement introducing the area; You can most likely start with the first paragraph from your project description and evolve it.
- Explanation of the specific problem and why do we care about the problem.
- Explanation of your solution, and how it improves on the work by others. Relation to related work can be very brief, given that you have a separate extensive section devoted to this.
- A hint on how the solution was evaluated and what was the outcome of this evaluation.
- A summary (a ”map”) of how the paper is organized.

2 Background

3 Real stuff comes here

4 Technical Solution

5 Analysis

6 Evaluation

- **Subjects** objects, or persons, being studied, whose properties or behavior is analyzed through the experiment. In the example subjects are programmers.
- **Factor** is the property, a variable, whose correlation we study. For example speed of programming in C vs speed of programing in C#
- **Treatment** one possible value of a factor, so in the example C and C# are treatments.
- **Outcome** result; numbers, other data, or qualitative information, indicating a correlation between treatments and observed variables. Example: correlation (or no) of speed of programming and the language.

7 Threats to Validity

- Validity of the experiment itself -¿ Internal Threats
- Generalizability of the results -¿ External Threats
- Other types possible (conclusion validity, construct validity)

8 Internal Validity

- Is the correlation between the treatment and the outcome casual,
- Accidental?
- Caused by some third variable that has not been observed.
- Example: all programmers in C were faster than programmers in C#
- But ...all the programmers in C# took the experiment very late at night, when they were tired.
- or all the C programmers were experienced engineers, and the C# programmers were newbies.

8.1 Main Internal Threats

- History – treatments are applied to subject in order at different times. Timing influences results.
- Learning – subjects learn the task over time.
- Testing – subjects should not know the results of the test on the fly (the ”election poll” effect)
- Mortality – subjects dropping out during experiment. Is the sample still representative?
- Intrusive instrumentation – the measurement or observing changes the variable being measured.
- Statistical insignificance – the sample is too small. Check experimentation handbooks.
- Direction of correlation – whether A causes B, or B causes A; are there tertiary reasons that cause both A and B.

Example 1 An internal threat is that our statistics are incorrect. To reduce this risk, we instrumented the native tools to gather the statistics rather than building our own parsers. We thoroughly tested our infrastructure using synthetic test cases and cross-checked overlapping statistics. We tested our formal semantics specification against the native configurators and cross-reviewed the specifications. We used the Boolean abstraction of the semantics to translate both models into Boolean formulas and run a SAT solver on them to find dead features. We found 114 dead features in Linux and 28 in eCos. We manually confirmed that all of them are indeed dead, either because they depended on features from another architecture or they were intentionally deactivated.

Example 2 Git allows rewriting histories in order to amend existing commits, for example to add forgotten files and improve comments. Since we study the final version of the history, we might miss some aspects of the evolution that has been rewritten using this capability. However, we believe that this is not a major threat, as the final version is what best reflects the intention of developers. Still, we may be missing some errors and problems appearing in the evolution, if they were corrected using history rewriting. This does not invalidate any of our findings, but may mean that more problems exist in practice.

8.2 External Validity

- How far examples are generalizable?
- Does the sample of subjects allow to conlcude anything about the broader population?
  - Tested programming speed on students, can we conclude about professionals?
  - Tested programming speed on small system level programs; can we conclude about writing web based applications?
- Do you expect similar results in slightly modified conditions?
- Normally the question of external validity has to do with interacion of the treatment with the chose sample:
- Will treatment give a different outcome for a different sample? (different results expected at ITU and in an EE school)
- Does the context interact with the treatment ? (listening to music gives different impression indoors and outdoors)

9 Related Work

To describe contracts there are two methods that come to mind: Object Constraint Language(OCL) [Object Management Group (2006)], and Business Object Notation(BON) [Walden, Marc-Nerson (1994)]. OCL only allows for formal specifications of contracts, but BON both allows and encourages the use of both informal and formal specifications. BON's informal specifications are similar to our project's behaviors, as they both allow description of behaviors in natural language. Our project proposes a way to express the behaviors in a more flexible and natural language only requiring a loose structure, where informal BON requires the use of technical terms such as classes, inheritance and composition.

Hubert Bauermeister[Bauermeister (2004)] wrote an interesting paper about refactoring test-cases written in a test driven development workflow to formal contracts. This is similar to our project in how it attempts to leverage the ease of writing test-like behaviors and translating them to allow for use in critical systems. It also presents a simple DSL which allows for simple transformations between normal tests and contracts. While many of the refactoring techniques could be used in our case for generalising step-based behaviors, the current implementation of the Bauermeisters DSL lack many of the advantages of behavior driven development, and unlike our project it does not seem to offer easily readable documents that can be used in other domains than computer science.

An alternative approach to specifying contracts textually is specifying them visually, as suggested in Model-driven Monitoring: Generating Assertions from Visual Contracts [Lohmann et al. (2006)]. While visual approaches can be easier to understand they are often cumbersome to use, and in the case of software the behaviour is already specified. Our project allows for an easier transition between loosely defined specification to structured contracts and does not require extensive tooling.

We could have used a better integration of natural language in our transformation language so rules were more reusable and flexible. The input rules could be made more flexible using natural language processing to interpret the behaviors, in a similar manner to Schulze et al. [Schulze et al. (2012)] who attempt to keep Literate Modeling models and UML diagrams synchronized by using natural language processing and OCL model querying. It could be possible to infer the method specification from the behavior definitions in natural language such as what is proposed by Pandita et al. [Pandita et al. (2012)]. A potential problem with increased flexibility in this manner is less reliability which could prevent it from being used in critical systems where reliability is paramount. When matching sentences using a subset of regular language this problem should be minimized if not completely eliminated.

10 Conclusion

– Summarize the most important outcomes
– Make sentences sound very strong. What should stay in reader's (examiner's) memory?
– Include a paragraph about future work. What is the next step in this research ?
– Delay writing the abstract, conclusion and the introduction to the late phases of writing
– Write the story first!

11 Reflection

What did we learn in the process