Extending BDD
A systematic approach to handling non-functional requirements

Pedro Moreira

Kellogg College
University of Oxford

A dissertation submitted for the
MSc in Software Engineering

Abstract

Software engineering methods have evolved from having a prescribed and sequential nature to using more adaptable and iterative approaches. Such is the case with Behaviour Driven Development (BDD) (North, 2006), a recent member of the family of agile methodologies addressing the correct specification of the behaviour characteristics of a system, by focusing on close collaboration and identification of examples.

Whilst BDD is very successful in ensuring developed software meets its functional requirements, it is largely silent regarding the systematic treatment of its non-functional counterparts, descriptions of how the system should behave with respect to some quality attribute such as performance, reusability, etc.

Historically, the systematic treatment of non-functional requirements (NFRs) in software engineering is categorised as being either product-oriented and based on a quantitative approach aimed at evaluating the degree to which a system meets its NFRs or process-oriented, qualitative in nature and where they are used to drive the software design process. Examples of the latter category, are the NFR Framework (Lawrence Chung et al., 2000) – a structured approach to represent and reason about non-functional requirements – and the goal-oriented requirements language (GRL) (Amyot et al., 2010) that provides support for evaluation and analysis of the most appropriate trade-offs among (often conflicting) goals of stakeholders.

In this thesis, we investigate the extent to which goal-oriented principles can be integrated in BDD, with the aim of handling non-functional requirements in an explicit and systematic way, whilst respecting the principles and philosophy behind agile development.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Dr Jeremy Gibbons, for his guidance, support, comments and encouragement.

I would also like to thank my family for their constant support and love, and in particular my wife, Tamara Moreira, for her endless patience whenever I so often disappeared to my office to work on this thesis.

The author confirms that: this dissertation does not contain material previously submitted for another degree or academic award; and the work presented here is the author's own, except where otherwise stated.

Contents

1 Introduction

This thesis presents an extension to Behaviour Driven Development (BDD) (North, 2006) to support the elicitation, communication, modelling and analysis of non-functional requirements. It includes concepts and techniques from goal-oriented requirements engineering (GORE) (Van Lamsweerde, 2001), and more specifically, allows the definition of goals in BDD and modelling and analysis in Goal Requirements Language (GRL) (Amyot et al., 2010). This is achieved by integrating notions of goals in Gherkin (Wynne and Hellesoy, 2012) – a domain specific language for the representation and specification of requirements. We also present a translator from Gherkin to GRL, allowing Gherkin-defined actors and goals satisfactions levels to be subject to qualitative and quantitative analysis in a GRL tool.

1.1 Motivation

The primary measure of success of a software system is the degree to which it meets the purpose for which it was intended (Nuseibeh and Easterbrook, 2000). Shortcomings in the ways that people learn about, document, agree upon and modify such statements of intent are known causes to many of the problems in software development (Wiegers and Beatty, 2013). We informally refer to these statements of intent as Requirements and the engineering process to elicit, document, verify, validate and manage them as Requirements Engineering [1].

The importance of requirements in software engineering cannot be understated. In his essay *No Silver Bullet*, Frederick P. Brooks (1987) states that '*The hardest single part of building a software system is deciding precisely what to build*'. More recently, Davis (2005) reveals that errors introduced during requirements activities account for 40 to 50 percent of all defects found in a software product. When arguing for the importance of requirements, Hull, Jackson, and Dick (2011) reason that to be well understood by everybody they are generally expressed in natural language and herein lies the challenge: to capture the need or problem completely and unambiguously without resorting to specialist jargon or conventions. The authors follow by positing these needs may not be clearly defined at the start, may conflict or be constrained by factors outside their control or may be influenced by other goals which themselves change in the course of time.

Furthermore, requirements can be classified in multiple and at times conflicting ways. Glinz (2007) points out that in every current classification scheme there is a distinction between requirements concerning the functionality of a system and all other, often referred to as non-functional requirements. In "Rethinking the notion of non-functional requirements", the same author points out issues with current classification schemes such as sub-classification, terminology and satisfaction level whereby some requirements are considered *'soft'* in the sense that they can be weakly or strongly satisfied (e.g *The system shall have a good performance*; or *The System shall be secure*). L Chung and Prado Leite (2009) contribute that, in spite of this

[1] These topics will be explored in depth in Chapter 2

separation, most existing requirement models and requirements specification languages lack a proper treatment of non-functional requirements.

In addition, this separation of functional and non-functional requirements has lead to the latter being either neglected, addressed later in a project or completely ignored. This problem applies to both traditional and agile software development processes. Generically, a software process is defined as a set of activities, methods, practices, and transformations that are used to develop and maintain software and its associated products (Cugola and Ghezzi, 1998). Agile software development approaches have become more popular during the last few years. Several methods have been developed with the aim to be able to deliver software faster and to ensure that the software meets customer changing needs. All these approaches share some common principles: Improved customer satisfaction, adopting requirements, frequently to changing delivering working software, and close collaboration of business people and developers (Paetsch, Eberlein, and Maurer, 2003).

One of such agile approaches is Behaviour Driven Development (BDD) (North, 2006). The understanding of BDD is far from clear and unanimous (Solis and Wang, 2011). Some authors refer to BDD as a development process (Smart, 2014), others state that it is not a fully fledged software development methodology but rather '*supplement other methodologies, provide rigour in specifications and testing, enhance communication between various stakeholders and members of software development teams, reduce unnecessary rework, and facilitate change*' (Adzic, 2011).

In spite of the above mentioned differences of interpretation, it is unanimously accepted that BDD focus on taking business goals into a sufficiently set of software features that contribute to achieve these business goals. This process makes use of Gherkin (Wynne and Hellesoy, 2012) – a domain specific language which promotes the use of a ubiquitous language[2] that business people can understand – to describe and model a system. However the focus has been on functionality and quality characteristics such as performance, security, maintainability are not explicitly addressed. To the best of the author's knowledge, the single exception to the above is the work of Barmi and Ebrahimi (2011) but with restricted applicability to probabilistic – those that can be written using probabilistic statements (Grunske, 2008) – non-functional requirements only.

None of these agile practices however, treat non-functional requirements in a systematic way, certainly not in a way that allows reasoning about which requirements interdependencies may exist, and the positive or negative influences each may have on each other. Among many proposals, goal-oriented approaches were the first to treat non-functional requirements as first-class citizens. Mylopoulos, Lawrence Chung, and E. Yu (1999) observe that goal-oriented requirements engineering is generally complementary to other approaches and, in particular, is well suited to analyzing requirements early in the software development cycle, especially with respect to non-functional requirements and the evaluation of alternatives.

It seems only logical and expectable that, improvements to the discovery and communication of requirements, and in particular non-functional requirements, will lead to an increase in success rates of software projects.

1.2 Aim and limitations of study

The context described in the previous section justify research aimed at capturing, documenting and communicating requirements using natural language tools and techniques in a precise,

[2]Evans (2004) first introduced that term in *Domain-driven Design: Tackling Complexity in the Heart of Software*

complete and unambiguous way, but also with the flexibility and adaptability to allow requirements to change and evolve through the course of time.

In this thesis, we investigate the extent to which goal-oriented principles can be integrated in BDD, with the aim of handling non-functional requirements in an explicit and systematic way, whilst respecting the principles and philosophy behind agile development. In particular, we consider how BDD can be extended and in particular Gherkin can be modified to incorporate actor and goal concepts, as defined and treated in GRL.

We do not however investigate the integration of GRL with use case maps (UCM), as part of the User Requirements Notation (Liu and E. Yu, 2004). UCM targets modelling scenarios of functional or operational requirements and performance and architectural reasoning. This is left as an area for further research.

We also do not aim at providing another classification scheme and address the, sometimes artificial, separation of functional and non-functional requirements. Instead, we adopt the notion of goals as an objective the system under consideration should achieve and goals formulations as properties to be ensured. We share the view that goals cover different types of concerns: functional which are associated with the services to be provided, and non-functional concerns associated with quality of service such as safety, security, accuracy, performance, and so forth (Van Lamsweerde, 2001).

Finally, we do not apply this technique to a specific non-functional requirement or use any particular taxonomy as our approach is independent of the NFR being addressed or taxonomy chosen.

1.3 Significance of the study

By reinterpreting behaviours in BDD as not just specifications of functionality of a system but as statements of goals, this thesis brings the following contributions to BDD:

- Allows non-functional requirements to be specified in natural language form in Gherkin
- Allows Gherkin specifications to be converted into goal models and imported and used in jUCMNav
- Allows BDD to consider all non-functional requirements, not just those that are technical, but still relevant for a successful product delivery
- Brings to BDD the capability to perform qualitative and quantitative satisfaction levels of actors and goals

By allowing goals to be elicited and specified in Gherkin, this thesis brings the following contributions to goal-oriented requirements engineering:

- Allows goals elicitation to occur in Gherkin using natural language and therefore more suitable for discussion and fostering communication
- Brings the benefits of executable specifications in BDD to goal formulations

1.4 Overview of Contents

We have now reviewed the motivation for this study, stated the aim of the research and identified the contributions our work brings to BDD and goal-oriented requirements engineering and the research community in general. The rest of the thesis is organised as follows:

Chapter 2 contains all the necessary background material to understand the remainder of the thesis and includes: section 2.1 on requirements engineering and in particular, the approaches taken by agile processes; section 2.2 on behaviour-driven development, describing the principles and practices of this popular agile process; section 2.3 presenting an overview of the research concerning ways to handle non-functional requirements in software engineering and section 2.4 on goal-oriented requirements engineering with a focus on GRL and with a description of jUCMNav (Amyot et al., 2010), an editor for GRL models.

Chapter 3 is the core of the thesis and contains details of extensions to Gherkin; mapping of Gherkin elements to GRL such as actors and intentional elements and a description of a translator from Gherkin to an XML-based interchange format to be used in jUCMNav.

Chapter 4 contains implications of findings, concluding thoughts, identifies limitations of study and suggests topics for future research.

2 Background

2.1 Agile Requirements Engineering

2.2 BDD

2.3 NFR Research Overview

2.4 Goal Oriented Requirements Engineering

3 Extending behaviour-driven development

4 Conclusion

A TO be Removed: typical chapter contents

Start of chapter

General advice Link back to previous parts in particular previous chapter
State the aim of the chapter
Outline how you intend to achieve this aim in the form of an overview of contents

Contents

Discussion or Analysis

- What's important
- What overall themes can be identified
- What can be observed or learned
- What limitations or shortcomings have been identified
- Situate the chapter within the whole thesis

Summary

- Replies to the introduction by briefly identifying the chapter's achievements and sets the scene for the next chapter

End of chapter

General advice Start with a strong summary of the main findings of this chapter with academic references and relate it with current theory.
Relates this chapter results to earlier analysis.
End with a strong lead into next chapter.

Bibliography

Adzic, Gojko (2011). *Specification by Example: How Successful Teams Deliver the Right Software*. 1st ed. Manning Publications.

Amyot, Daniel et al. (2010). “Evaluating goal models within the goal-oriented requirement language”. In: *International Journal of Intelligent Systems* 25.8, p. 841.

Barmi, Zeinab Alizadeh and Amir Hossein Ebrahimi (2011). “Automated testing of non-functional requirements based on behavioural scripts”. Chalmers University of Technology.

Chung, L and JCS do Prado Leite (2009). “On Non-Functional Requirements in Software Engineering”. In: *Conceptual modeling: Foundations and ...*

Chung, Lawrence et al. (2000). *Non-Functional Requirements in Software Engineering*. Springer Science + Business Media.

Cugola, Gianpaolo and Carlo Ghezzi (1998). “Software processes: a retrospective and a path to the future”. In: *Software Process: Improvement and Practice* 4.3, p. 101.

Davis, Alan Mark (2005). *Just Enough Requirements Management: Where Software Development Meets Marketing*. Dorset House.

Evans, E. (2004). *Domain-driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley.

Frederick P. Brooks (1987). “No Silver Bullet Essence and Accidents of Software Engineering”. In: 20.4.

Glinz, M (2005). “Rethinking the notion of non-functional requirements”. In: *Proc. Third World Congress for Software Quality*.

— (2007). “On Non-Functional Requirements”. In: *15th IEEE International Requirements Engineering Conference (RE 2007)*, pp. 21–26.

Grunske, Lars (2008). “Specification patterns for probabilistic quality properties”. In: *Software Engineering, 2008. ICSE’08. ACM/IEEE 30th International Conference on*. IEEE, pp. 31–40.

Hull, Elizabeth, Ken Jackson, and Jeremy Dick (2011). *Requirements Engineering*. Springer Science.

Liu, Lin and Eric Yu (2004). “Designing information systems in social context: a goal and scenario modelling approach”. In: *Information systems* 29.2, pp. 187–203.

Mylopoulos, John, Lawrence Chung, and Eric Yu (1999). “From object-oriented to goal-oriented requirements analysis”. In: *Communications of the ACM* 42.1, p. 31.

North, Dan (2006). *Introducing BDD*. URL: `http://dannorth.net/introducing-bdd/` (visited on 07/20/2015).

Nuseibeh, B and S Easterbrook (2000). “Requirements engineering: a roadmap”. In: *On the Future of Software Engineering*.

Paetsch, F, A Eberlein, and F Maurer (2003). “Requirements engineering and agile software development”. In: *12th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE’03)*.

Smart, John Ferguson (2014). *BDD in Action: Behavior-driven development for the whole software lifecycle*. 1st ed. Manning Publications.

Solis, Carlos and Xiaofeng Wang (2011). “A Study of the Characteristics of Behaviour Driven Development”. In: *Proceedings of the 2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications*. SEAA ’11. Washington, DC, USA: IEEE Computer Society, pp. 383–387.

Van Lamsweerde, A (2001). “Goal-oriented requirements engineering: A guided tour”. In: *Requirements Engineering*.

Wiegers, Karl and Joy Beatty (2013). *Software Requirements*. 3rd. Developer Best Practices. Microsoft Press.

Wynne, M. and A. Hellesoy (2012). *The Cucumber Book: Behaviour-Driven Development for Testers and Developers*. The pragmatic programmers. Pragmatic Bookshelf.

Yu, Y et al. (2005). “Quality-based software reuse”. In: *Advanced Information . . .*