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, Smart, 2014], 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 based on a quantitative approach aimed at evaluating the degree to which a system meets its non-functional requirements or process-oriented, qualitative in nature and where NFRs are used to drive the software design process.

One example of the latter category, is the NFR Framework [Chung et al., 2000], a structured approach to represent and reason about non-functional requirements.

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

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.

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. This extension include concepts and techniques from Goal Oriented Requirements Engineering (GORE) [Van Lamsweerde, 2001], and specifically, it allows the specification of goals in BDD and modelling and analysis in Goal Requirements Language (GRL) [Amyot et al., 2010]. This is achieved by integrating goals specifications in Gherkin [Wynne and Hellesoy, 2012], a domain specific language for the representation and specification of requirements, and presenting a translator from Gherkin to GRL.

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.

The importance of requirements in software engineering cannot be understated. In his essay *No Silver Bullet* [Brooks, 1987], Fred Brooks 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 et al., 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 another paper, the same author arguments why this notion of non-functional requirements is flawed and present a new classification which is based on four facets: *kind* (e.g. function, performance, or constraint), *representation* (e.g. operational, quantitative or qualitative), *satisfaction* (hard or soft), and *role* (e.g. prescriptive or assumptive). The author points out issues with 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*).

The context just described applies to both traditional and agile software development pro-

cesses. Genrically, 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 et al., 2003].

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

In this these, we do not address the open research question of requirements classification schemes and the, sometimes flawed[Glinz, 2007], separation of functional and non-functional requirements. Instead, we adopt the notion of goals as an objective the system under consideration should achieve. Goal formulations thus refer to intended properties to be ensured; they are optative statements as opposed to indicative ones, and bounded by the subject matter. Goals also 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]

TODO Rephrase:Extracted from Engineering and Managing Software Requirements

The main obstacle in decision-driven RE research resides in the desire to solve any problem formally and rigorously. This presumes that each problem can be properly described by a formal model and is solved just using this model. It equates decision making with finding the optimal solution. According to Roy [51] and Schrlig [57], such thinking was also once dominant in management science and operations research. The reality was that despite enormous progress in optimization and operational research, many questions in the real world could not be answered in a satisfactory way. In many real situations, insisting on establishing the ideal model and searching for the numerically optimal solution eventually ends in a deadlock. Such problems have been characterized as wicked problems by Rittel and Webber [47].

1.3 Contributions

A reinterpretation of behaviours in BDD as not just specifications of functionality of a system but as statements of goals in the spirit of GORE.

We address the vague nature of early requirements.

1.4 Overview of Contents

- Outline the problem
- Explain the aim of the research
- Set any limits to the scope of the work
- Significance of the study

- Provide an overview of the thesis structure

- Define Requirements Engineering
- Define Software Practices and contrast Traditional vs Agile
- Expand on Agile RE
- Introduce BDD
- Highlight issues with current approaches for RE and BDD
- Expand on Functional vs Non-Functional Requirements
- Highlight approaches for handling NFRs
- Describe GORE and highlight benefits
- List Research questions
- Describe Thesis Structure

2 Background

Start of chapter

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

2.1 Agile Requirements Engineering

2.2 BDD

2.3 NFR Research Overview

2.4 Goal Oriented Requirements Engineering

On Non-Functional Requirements [Glinz, 2007]

- Definition of Requirements Engineering
- Multiple Requirements Classification
- Formal vs Informal specifications
- Functional vs Non-Functional Requirements
- Definition of N FRs
- Issues with current definition of NFRs

  Definition

  - Terminology
  - Scope
  - Misconceptions

  Classification

  - Sub-Classification
  - Concepts

Representation

- Representation-Dependant
- Location
- Traceability

A Survey of Non-Functional Requirements in Software Development Process [Matoussi and Laleau, 2008]

- Definition of Requirements Engineering
- Formal vs Informal specifications
- Definition of NFRs
- Issues with current definition of NFRs

NFR Background Information

Requirements Analysis and Specification Level

- Informal and semi-formal
  - NFR Framework
  - Dissatisfaction driven approach
  - CEI's Quality Attribute Taxonomy
  - Use Cases based
  - PREM
  - Misuse Cases
  - KAOS
- Formal
  - FDAF (Formal Design Analysis Framework)
  - Specification Languages
  - NoFun
  - Z

Design Level

- Use Cases and Goal Driven
- OONFR
- UML based

Implementation Level

- Constraint and Object Oriented Programming
- ADA
- Exception Handling
- Combining Rules and Object Orientation

Complete approaches

- FRIDA
- NFR Reuse Process
- BMethod
- Tropos / Formal Tropos

On non-functional requirements in software engineering [Chung and do Prado Leite, 2009]

- Functional vs Non-Functional Requirements
- Definition of NFRs
- Classification Schemes
  - Basic and extra quality
  - Concerns
  - ISO/IEC 9126
  - NFR Frameowork based
  - Roman's Taxonomy
  - Software Quality Trees
  - FURPS
- Issues with Classification Schemes
- Representations

  Textual

  - IEEE 830
  - Volere
  - System Analysis (SADT)

  Trees and Lists

  Use Cases and Misuse Cases

  Restrictions over Scenarios

  Goal oriented approaches

  - KAOS
  - NFR Framework
  - i* Family
    * i*
    * Tropos
    * GRL

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.

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

3 Application and reflection

4 Conclusion

- Benefits of new terminology defined [Glinz, 2007]
- Benefits of Reuse Process [Yu et al., 2005]

Bibliography

[Amyot et al., 2010] Amyot, D., Ghanavati, S., Horkoff, J., Mussbacher, G., Peyton, L., and Yu, E. (2010). Evaluating goal models within the goal-oriented requirement language. *International Journal of Intelligent Systems*, 25(8):841.

[Brooks, 1987] Brooks, F. P. (1987). No silver bullet essence and accidents of software engineering. *Computer*, 20(4).

[Chung and do Prado Leite, 2009] Chung, L. and do Prado Leite, J. (2009). On non-functional requirements in software engineering. *Conceptual modeling: Foundations and ...*.

[Chung et al., 2000] Chung, L., Nixon, B. A., Yu, E., and Mylopoulos, J. (2000). *Non-Functional Requirements in Software Engineering*. Springer Science + Business Media.

[Cugola and Ghezzi, 1998] Cugola, G. and Ghezzi, C. (1998). Software processes: a retrospective and a path to the future. *Software Process: Improvement and Practice*, 4(3):101.

[Davis, 2005] Davis, A. M. (2005). *Just Enough Requirements Management: Where Software Development Meets Marketing*. Dorset House.

[Glinz, 2007] Glinz, M. (2007). On non-functional requirements. *15th IEEE International Requirements Engineering Conference (RE 2007)*, pages 21–26.

[Hull et al., 2011] Hull, E., Jackson, K., and Dick, J. (2011). *Requirements Engineering*. Springer Science.

[Matoussi and Laleau, 2008] Matoussi, A. and Laleau, R. (2008). A survey of non-functional requirements in software development process. *Departement d'Informatique Universite ...*.

[North, 2006] North, D. (2006). Introducing bdd.

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

[Paetsch et al., 2003] Paetsch, F., Eberlein, A., and Maurer, F. (2003). Requirements engineering and agile software development. *12th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'03)*.

[Smart, 2014] Smart, J. F. (2014). *BDD in Action: Behavior-driven development for the whole software lifecycle*. Manning Publications, 1 edition.

[Van Lamsweerde, 2001] Van Lamsweerde, A. (2001). Goal-oriented requirements engineering: A guided tour. *Requirements Engineering*.

[Wiegers and Beatty, 2013] Wiegers, K. and Beatty, J. (2013). *Software Requirements (3rd Edition) (Developer Best Practices)*. Microsoft Press.

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

[Yu et al., 2005] Yu, Y., Liu, L., Eric, S. K., and do Prado Leite, J. (2005). Quality-based software reuse. *Advanced Information ....*