# 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), 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 — a quantitative approach aiming at evaluating the degree to which a system meets its non-functional requirements — and a process-oriented one, qualitative in nature and where non-functional requirements 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

Thanks.

# Contents

# 1 Introduction

# 2 Background

*On Non-Functional Requirements* [Glinz, 2007]

- Definition of Requirements Engineering
- Multiple Requirements Classification
- Formal vs Informal specifications
- Functional vs Non-Functional Requirements
- Definition of NFRs
- 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

# 3 Application and reflection

# 4 Conclusion

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

# Bibliography

[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.

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

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

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

# A Notes on LaTeX

There are books that you can buy on LaTeX, but you are unlikely to need any of them: the commands mentioned here should see you through. If there's something else that you think we should add, let us know!

## A.1 Letters, words, lines, and paragraphs

Remember, don't break your chapters up like this unless you really are presenting something along the lines of a manual.

### A.1.1 Special characters

LaTeX treats the following characters as special cases:

- `\` is the escape character, and indicates the beginning of a command, as in `\chapter`; if you want a backslash symbol, write `\backslash` *in math mode (see below!)*
- `%` comments out the remainder of the line, so if you need a percent symbol, write `\%`
- `$` puts you into math mode, which you probably won't need unless you're typesetting some mathematics; if you want a dollar symbol, write `\$`. If you want the pound sign, type `\pounds`.
- `{` opens a group of characters for interpretation, and `}` closes one. This can be used to delimit an argument to a command: e.g. `\chapter{Name}`. It can also be used to limit the scope of some declaration: e.g. `{\bfseries some text in bold}`. If you want the braces themselves, you'll need `\{` and `\}`.
- `&` is used as a column separator in the `\tabular` environment: see Section A.2.2 below. If you need an ampersand symbol, you can write `\&`.
- `[` and `]` after a command are used to delimit an optional argument. If you find that this is happening when you don't want it to, then you can put `\relax` between the command and the opening square bracket—e.g.
  `\item \relax [some text in brackets]`
- `` ` `` and `'` produce opening and closing quotation marks, respectively; they do the right thing when doubled. Take care to use these correctly!

### A.1.2 Breaking

LaTeX does this kind of thing for you, but you need to show it how your text is divided into paragraphs. Just leave a blank line to do this. If you can't leave a blank line (really?) you can type `\par` instead.

In running text, a new paragraph should start with an indented line, unless it's the first paragraph after a heading. This will happen automatically. Don't mess with this: that is, don't redefine `\parindent` and `\parskip`. If you need to tell a specific paragraph *not* to indent the first line, and you really mean to do that, then you can type `\noindent` just before it starts.

- `\\` produces an immediate line break. Do not use this in running text unless you absolutely have to; never use it to create paragraph breaks—just use a blank line.
- Use `\clearpage` to produce a page break. You shouldn't really do this kind of thing until you're putting the finishing touches to your document.

If LaTeX is complaining about "overfull hboxs", that's probably because it can't break a line where it wants to. If the box in question is overfull by more than 10pt (take a look at the typeset version and see if you can spot it) then try reorganising the line or paragraph.

### A.1.3 Fonts

You might not need to insert any font commands in your running text, but in case you want to:

- `\emph` puts its argument in emphatic font—usually italic or slanted: e.g. "`\emph{your} mileage`" produces "*your* mileage".
- `\verb` does more than just put its argument in typewriter font, it renders it verbatim: spaces matter. Exceptionally, this command requires that its argument be delimited by the next symbol it sees: e.g. `\verb|st uff|` will produce `st uff`. Also, the argument to `\verb` can't include a linebreak: for that, you need a `verbatim` environment.
- `\textbf`, `\textsl`, `\textit`, `\textsf`, and `\texttt` put their arguments in bold, slanted, italic, sansserif, or typewriter font, respectively.
- `\bfseries`, `\slshape`, `\itfamily`, `\sffamily`, and `\ttfamily` are the corresponding declarations, and apply for the rest of the current enclosing group.
- `\small` and `\large` do what their names suggest: you might want to use `\small` to shrink a table to fit, but otherwise you shouldn't need to use these.

## A.2 Environments

### A.2.1 Lists

Environment commands in LaTeX are more like the familiar markup of HTML or XML: they have a `\begin` and an `\end`.

- `\begin{enumerate} \item ... \item ... \end{enumerate}` produces an enumerated list, with command `\item` marking the beginning of a new item with a label.

- `\begin{itemize} \item ... \item ... \end{itemize}` does the same, but the labels are all bullets, or dashes, or circles, depending upon how deep you go.
- `\begin{verbatim} ... \end{verbatim}` renders everything inside it, well, in verbatim: in typewriter font, with newlines and spaces, and even the special symbols, appearing as they are. This is ideal for presenting program code or model text.
- `\begin{quote} ... \end{quote}` indents the enclosed material to the same extent as an itemized or enumerated list.

The LaTeX `description` environment is also available, but is a little awkward in comparison to those above. A better solution may be to use a `\subsubsection` or `\paragraph` command to introduce each of the items, if you absolutely have to do this.

### A.2.2 Figures and tables

Figures and tables are *floats* in LaTeX: they will appear near to the surrounding text if they can, but they might also float away only to wash up later in the document.

- A figure environment can be used to import a graphic into the document (don't use the archaic LaTeX drawing commands unless you have a good reason to). Any graphic will do, and you shouldn't need to specify an extension.

```
\begin{figure}[t]
  \centering
  \includegraphics{layercake}

  \caption{How to have your cake and eat it}
  \label{fig:layercake}
\end{figure}
```

  The `t` option says 'put this at the top of a page'. `h` says 'put it here instead'. You can add an exclamation mark in each case to (try to) insist: e.g. `t!`. You shouldn't have to, though.

  The `\caption` command creates a caption using the supplied text, and the `\label` command defines a label for referring to this figure in the text (see Section A.3.2 below).

- The `\includegraphics` command accepts optional arguments in square brackets before the name of the graphic file. These can be used to determine the size and orientation of the picture. The most useful are:
  - `\includegraphics[scale=0.8]{...}` to scale the graphic by, for example, `0.8`, or 80%.
  - `\includegraphics[width=6cm]{...}` to set the width of the graphic to, for example, 6cm.
  - `\includegraphics[height=6cm]{...}` to set the height of the graphic to, for example, 6cm.

- A table environment is pretty much the same thing, although you're quite likely to be using LaTeX's table making commands instead of importing a table drawn in another package and exported as a graphic.

```
\begin{table}[t]
  \centering
  \begin{tabular}{l|l}
    Person & Score \\ \hline
    Susan & 30 \\
    Peter & 40
  \end{tabular}
  \caption{Tennis score when rain stopped play}
  \label{tab:tennis}
\end{table}
```

- You can also set tables as displayed material in running text: in this case, you want to use the `tabular` environment, perhaps inserted within a `quote` environment or a `center` environment (note the spelling) to get the alignment correct.

```
\begin{quote}
  \begin{tabular}{l|l}
    Person & Score \\ \hline
    Susan & 30 \\
    Peter & 40
  \end{tabular}
  \caption{Tennis score when rain stopped play}
  \label{tab:tennis}
\end{quote}
```

  produces:

| Person | Score |
|---|---|
| Susan | 30 |
| Peter | 40 |

  Note that there is no caption here, nor is there a label.

- The `tabular` command takes a 'table specification' as a mandatory argument:
  - `l`, `c`, and `r` declare columns with material to be aligned to the left, centred, and aligned to the right, respectively.
  - `|` denotes a vertical line.

  After this argument comes the beginning of the first row.

  Each row can contain up to $n - 1$ ampersands ('`&`'), where $n$ is the number of columns declared: these act as delimiters between column material. A row is ended by a double backslash ('`\\`'), which takes an optional argument indicating any additional vertical space required before the next row: e.g. `\\[2cm]`. A `\hline` after `\\` produces a horizontal rule across all of the columns before the next row. You can use two adjacent `\hline` commands to produce a double rule.

## A.3 Sections, labels, references, and citations

### A.3.1 Sections

Your dissertation should be divided into a number of chapters, each introduced using a `\chapter` command with a single, mandatory argument: the name of the chapter. For example, this chapter, in the appendix of the template document, begins with the command

```
\chapter{Notes on \LaTeX}
```

Your chapters can be divided into sections, and into subsections, using the `\section` and `subsection` commands, respectively. Again, these take the name of the section or subsection as their single, mandatory argument.

Further levels of sectioning are available—`\subsubsection`, `paragraph`, and `subparagraph`—but try not to use these unless what you are writing is intended to resemble a reference manual.

### A.3.2 Labels and (cross-)references

You should use labels to identify any chapters, sections, subsections, items (in enumerated lists), figures, or tables that you wish to refer to. You can do this using the `\label` command, with whatever text you wish to use as a key: for example,

```
\chapter{Notes on \LaTeX}
\label{chap:latex}
```

might be an appropriate label for this chapter. You can then refer to this chapter by writing

```
Chapter~\ref{chap:latex}
```

and LaTeX will insert the appropriate number (provided that you've run it at least once since inserting the corresponding `label`). Since we're in the first chapter of the Appendix here, this is going to produce

> Chapter A

For floats—figures and tables—you should insert the `\label` command after the `\caption`. Note that since you are referring to a specific item, you should capitalise 'Chapter', 'Section', 'Figure', or 'Table'. It is sufficient to say 'Section' when referring to smaller divisions: e.g. subsections.

### A.3.3 Citations

Although you can construct and maintain a bibliography by hand, it is far more efficient to make use of the `BibTeX` program. To do this, you create a `.bib` file to hold the references that you might wish to use. You then place the commands

```
\bibliographystyle{plain}
\bibliography{example}
```

at the appropriate point in your document, where `example` is the name of your `.bib` file (the extension is not required).

Each entry in your `.bib` file will start with a key (after an `@` command identifying the document type). For example,

```
@article{exampleref,
  author = {Barry Andrews},
  title = {{Preparing an MSc Project Proposal}},
  journal = {What Project? Monthly},
  publisher = {Random House},
  year = {2009}
}
```

The key here is '`exampleref`'. We can now 'cite' this article in our document, simply by using this key as an argument to the `\cite` command. For example,

```
most of this information will be useful to those preparing a project
proposal~\cite{exampleref}
```

will produce

> most of this information will be useful to those preparing a project proposal [**?**]

To cite more than one reference at the same point, simply include the respective keys in the argument to `\cite`, separated by commas.

## A.4 Extras

If you want to use mathematics, you should obtain the `zed.sty` package: email `Jim.Davies@comlab.ox.ac.uk` to ask. If you want to use fonts other than the default *Computer Modern* (CM), then you might like to try adding

```
\usepackage{times}
```

between then `\documentclass` and `\begin{document}` commands. `newcent`, `utopia`, `palatino` are other packages that define alternative fontsets. It's probably best to stick with CM or Times, however.