Framework comparison method

Comparing two frameworks based on technical domains, focussing on customisability and modifiability

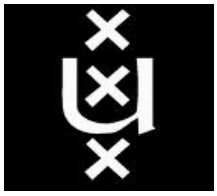
Anton Gerdessen, 13 August 2007
Master course software engineering

Thesis Supervisor: Dr. J. Vinju
First internship Supervisor: Ir. Onno de Groote
Second Internship Supervisor Mark Mastop MsC
Company: Everest bv
Availability: Public

Everest BV

EVEREST
REACHING SUMMITS

University of Amsterdam (UvA)



Abstract

More and more frameworks emerge each year. Yet the question why they emerge is never answered, are the older ones no longer sufficient, or are the newer ones 'better'? Even the question what a framework is exactly, has never been answered. With this wide field of frameworks and no clear definition, how can a framework be picked over another framework? How can be determined which is more suitable?

Everest has its applications build upon the Sun Blueprints framework. Everest uses an older version of this Blueprints framework. This older version has a very clear separation of the business, data and presentation layer. The newer version of the Blueprints framework follows a different methodology, claiming to focus more of the ease of development. The Spring framework is another framework also claiming the same ease of development. The question that Everest has is, would changing to Spring be possible and in what advantages or disadvantages would this result compared to the old situation.

This question is motivated by the observation that Blueprints itself is moving towards the same methodology as Spring. The Spring framework has gone through a tremendous growth in the last few years. Is this by pure chance or are the underlying principles sound and should a change of framework be applied for Everest's applications.

There are no methods available to determine which frameworks would be best in the given context; there are no real methodologies for framework comparison at all. In order to solve this problem, we first need to define what a framework is exactly. Following this definition, we can select criteria on which to compare the frameworks. This approach is based upon abstraction, comparing a framework as whole is far to complex; by selecting criteria and comparing these criteria individually we reduce the complexity. We propose a theoretical framework model for a framework and base our criteria and comparison on this model.

We will define these criteria as domains and infer the domains of a framework and their different types. We will make a distinction between the purpose of a framework, the vertical domain, and the way this purpose is achieved by covering horizontal domains.

The method first proposes a theoretical model that gives us the criteria for a framework in general. This model has been composed by examining the literature available for frameworks and extracting terms related to frameworks and filtering these until we can construct a theoretical model. Following we can use these terms as criteria and iterate through these and compare each criteria separately. In this research we will focus on customisability and modifiability, this because Everest indicated that these are the two criteria they find most important since they are dealing with applications which often change and these changes need to be applied rapidly.

Customisability and modifiability will be the two criteria on which we will focus; these criteria will be evaluated by using feature description diagrams. This has been done because both customisability and modifiability are changes in the features or the features behaviour. We will define the difference between customisability and modifiability and propose a method for comparison based primarily upon feature description diagrams. The diagrams will be complemented with metrics extracted from the source code for each framework.

The results indicate that the Spring framework holds a higher customisability compared to Spring. However, Blueprints holds a slightly higher modifiability than Spring. This indicates a slight advantage for Spring for the current comparison. Nevertheless, in order to completely evaluate both frameworks, more domains need to be compared.

The contribution of this research is threefold; we provide a theoretical model for frameworks in general. Secondly, we provide a method for comparing frameworks based upon this theoretical model. Thirdly, we propose a method for comparing customisability and modifiability based upon features diagrams and metrics.

We state that if the change of framework is made, a clear set of features will need to be defined which are allowed to be used and which are not. This due to the fact that Blueprints is more a framework of limitations and thus limiting complexity. Spring however is a framework of possibilities.

Acknowledgements

This thesis is the result of my graduation project for the title of Master of Science at the university of Amsterdam.

This thesis has been shaped by the influence of several people. First my university supervisor, Jurgen Vinju, who was always available for questions or discussion about the thesis. Secondly, Everest, who offered me the opportunity to perform this research.
Also the supervisor at Everest itself, Onno de Groote and Mark Mastop, who provided me with suggestions and insight in the more practical issues for this thesis.

List of contents

Figure index

1 Introduction

This section introduces the background of this thesis. The context and basic concepts involved will be explained. In addition, we will define the scope of this research project and identify the research questions. We will also discuss why this research is important and how it contributes to the field of software engineering. Finally, we will display the fields that influence this research project and the derived research method.

1.1 Context

There have been a great number of changes in the field of software engineering, yet the design and implementation of complex systems remains expensive and time consuming. Much of this effort rests in rediscovering of basic components and concepts at the core of all applications. Application frameworks, hereafter simply referred to as frameworks, are a technique designed to remove the rediscovering aspect.

The question what a framework exactly is has never been answered. With this wide field of frameworks and no clear definition, how can a framework be picked over another framework? How can be determined which is more suitable?

Frameworks are often defined as reuse technique; they provide the basis implementation for an application. This is done by providing a base skeleton code that implements basic component and concepts so they do not need to be reinvented each time. The already implemented areas are called *frozen spots* [Mar00], the framework kernel in essence. The points were a framework could be extended to fit the specific needs for applications are called *hot spots* [Mar00]. These *hot spots* are also called *hooks* [Fro97]. A framework on its own is not a working application.

Currently there are a large number of frameworks available. Frameworks are often directly related to object-oriented applications, although frameworks are not restricted to object-oriented programming [Fay99]. Frameworks are often intended for a specific programming language, this is a direct result of these frozen spots. Frozen spots are parts of the framework that are already implemented in a language. The choice of framework therefore directly relates to the language of the application. For some languages, there are a number of different frameworks available. Yet little to no effort has been put into comparing these frameworks.

Most frameworks are directed at a specific application domain. This is a result of a frameworks purpose; preventing the rediscovery of basic concepts. These basic concepts and aspects are domain related and only valid within that specific domain. For this reason frameworks are mainly categorised for their purpose, in [Fay97] frameworks are categorized into three categories:

- *Enterprise frameworks* mainly used for enterprise applications, much broader then the next two.
- *System infrastructure frameworks*, mainly used for internal applications.
- *Middleware infrastructure frameworks*, mainly used for distributed applications.

Little effort has been put into comparing frameworks on a conceptual level. One type of comparison can be found in [Pri06]. This comparison is based at the architectural level, not on a conceptual level. Architecture is what a system should do, not what it does. We want to focus on what a framework does and not what it should. This leaves the question, how can frameworks be compared on a conceptual level?

1.2 Everest's interest

Everest is a company which target branch is the financial and banking sector. Currently they offer a service that can quickly accommodate changes and support large quantity of concurrent users. These two aspects are important for the target branch that has a high rate of change and a large number of concurrent users. For example, new types of insurance policies are brought on the marked frequently. Everest's product can quickly incorporate these changes, this way the time to market is a lower. The current product is based upon Sun Blueprints [Blr07]. In the last several years, the Spring framework [Spr07] has gained popularity and has been incorporated in a large number of products. This combined with the direction that Blueprints is also moving more towards the ideas incorporated in Spring, resulted in the question if a change of framework for Everest is desirable.

1.3 Scope

In order to compare frameworks, several aspects need to be taken into consideration. These aspects can roughly be divided into two groups:

- Technical aspects
- Non-technical aspects (composite aspects)

Technical aspects are aspects like scalability, customisability, etc. Aspects that can be directly related to source code. The second group are the non-technical aspects; these are aspects like the learning curve required for a developer to use the framework, the evolution of the framework, etc. These are aspects that can be related to one or more technical aspects as well as documentation and architectural description etc.
For example, maintainability is a non-technical aspect influenced by, modifiability, duplication of code, the level of abstraction used, the documentation etc.

This thesis will focus on the technical aspects of frameworks. However, this research can be used as input for reasoning about non-technical aspects since they highly related to technical aspects, for example the maintainability aspect mentioned before.

1.4 Research Question

Everest requires a method to determine if their application can be converted from Sun Blueprints framework to the Spring framework. By converted we mean preserving its features, what features are exactly will be defined later in this thesis. Moreover, what positive and negative effects will this have on the application itself. The question which of the two frameworks is better in general is not the primary concern right now. The question that follows is:

RQ-1 *"Can application X conform framework A be converted to framework B"*

Sub questions that arise from the main question are:

- **RQ-2** *"Does application type for both frameworks overlap"*
- **RQ-3** *"How to determine which criteria to use for comparing the frameworks"*
    - **RQ-4** *"In which advantages and disadvantages will this conversion result"*
        - **RQ-5** *"Is the change of framework desirable"*

The last question can only be answered by combining the technical analysis that this research delivers and a non-technical analysis that this research does not deliver. This leads to the conclusion that the last research question is partially outside the scope of this research.

This does however leave some additional questions, especially in for the criteria selection, which can either be answered by the literature available, or at least be given a direction by the literature. These questions will be discussed in the next section.

1.4.1 Literature questions

In order to answer the questions mentioned in the previous section, some information is needed from the literature.
The first question that we need to answer is what are frameworks exactly? Is there a precise definition we can work with, this brings us to the first literature question.

LQ-1 *"What is the definition of a framework in the software engineering context?"*

Once we have a definition of a framework, we need a basis on which to compare frameworks on a conceptual level. Frameworks by itself are a complex structure of many complex factors; we need a way to reduce this complexity. In other words, we need an abstraction from a framework. In order to do this we need to make an assumption.

AS-1 *"A framework can be divided into smaller parts, we can then compare these individual parts separately."*

Now that we made this assumption, we have to classify these individual parts. In [Cza00] and [Sim95], a classification is made for frameworks divided into smaller parts called domains. These domains are divided into two groups, vertical and horizontal domains. Where a vertical domain is the domain for which the framework is intended. Horizontal domains are the domains that the framework has to address in order to cover the vertical domain.
For example if a framework is designed to control robots in an automobile factory, robot control for an automobile factory is it vertical domain. Horizontal domains can be controlling the robot, reading the specifications for the automobile etc. From the previous information, we can infer our secondary literature question:

LQ-2 *"Can we create a framework abstraction based upon horizontal domains?"*

A question in extend of this one is:

LQ-3 *"Can be describe these horizontal domains"*

Once we have the answer to these questions, we have a basis for comparing frameworks.

1.5 Contribution

The contribution of this thesis is threefold. First, this thesis will provide a conceptual model of frameworks in general. Secondly, with the theoretical model, we can provide a method for comparison. Later in this thesis, we will identify several horizontal domains; these domains will be prioritized by Everest. The result can be seen in figure 11. We will compare two of these domains, customisability and modifiability; we will propose a method for comparison based on features and feature diagrams, which is the third contribution.

1.6 Research influences

Figure 1 shows how the research is structured. It shows which literature was used and how they are all related.

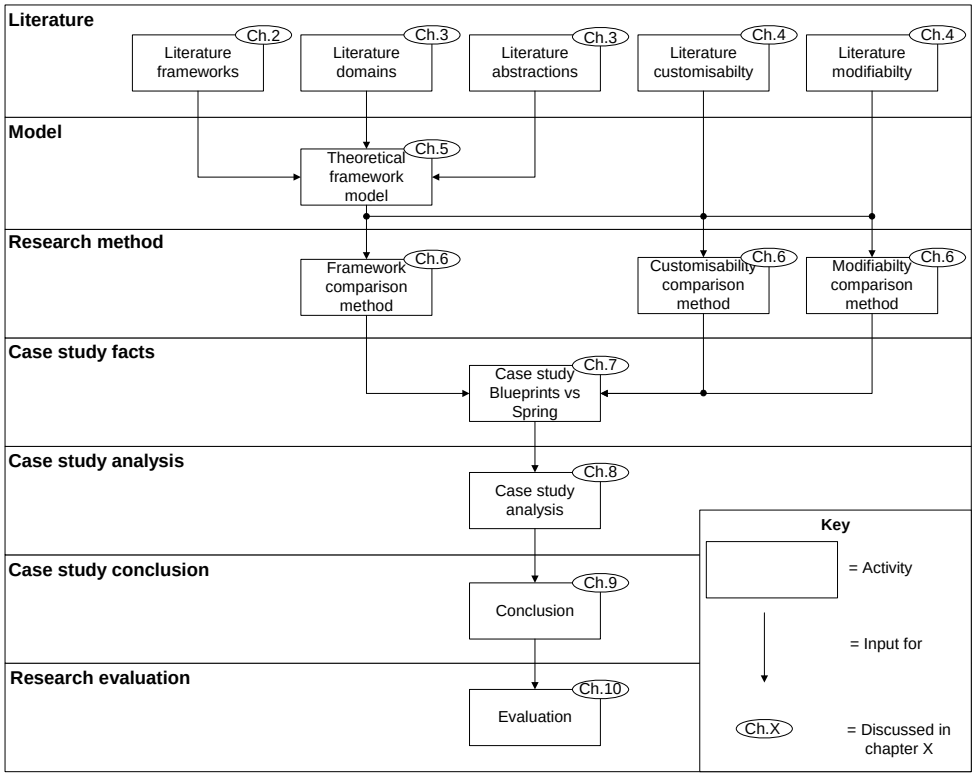


Figure 1: Research approach

Figure 1 is not consistent with the order in which the actual research has been performed. We decided that it was best to group al the literature in one place for easier reading. We started with chapter one to three. Following we created a crude model and research method, which are chapter four and six. We than continued to prioritize the domains, which is chapter seven. Once this was done we went performed chapter four. This resulted in a refinement of chapter five and six. After that we completed the rest of the research according to the chapter numbers, chapter seven and further.

1.7 Summary

In this chapter, we gave an introduction on frameworks. Moreover, we identified why this research is important for Everest, which is currently investigating a change of framework. We also identified the scope of the research; we will focus on technical aspects. We identified the research questions **(RQ 1-5)** and the literature questions **(LQ 1-3)**. In addition, we presented the roadmap displayed in figure 1 for this research.

2 Frameworks

This chapter discusses frameworks in general. We will start with the question why to use frameworks at all, what is the purpose of doing so. After discussing the purpose of frameworks, we will define what a framework is. The next section will list possible options on how frameworks can be compared. The last section will explain what a framework is composed of and how it relates to terms like components and domain specific languages.

2.1 Framework purpose

To the question why frameworks emerged, a number of answers are presented in the literature. [Joh97a] states:
"Frameworks are an object oriented reuse technique".
Another purpose mentioned in [Fay97] is:
"Frameworks are a promising technology for reifying proven software designs and implementations in order to reduce the cost and improve the quality of software."
This purpose also holds the reusable aspect. The purpose used in [Fro00] is:
"Frameworks enable developers to rapidly produce new applications."
The reason for this rapid production is not mentioned in this paper. In light of the other purposes mentioned, we can assume this is by reusability.

The next question is, what reuse is exactly, in [Kru92] reuse is defined as:
"Software reuse is the process of creating software systems from existing software rather than building software systems from scratch."

We can now conclude that the most important purpose of frameworks is the reuse of design and/or code.
The reason for reuse varies. Other minor arguments are to improve quality, or to rapidly produce new software.

The most important purpose of frameworks is the reuse of design and/or code.

2.2 Framework definition

In the literature, a large number of definitions are available for frameworks. A definition used in [Joh97a] is:
"*A framework is a reusable design for all or parts of a system that is represented by a set of abstract classes and they way their instances interacts."*
This definition reflects the overall purpose of frameworks, the reusability component.
Another definition also given in [Joh97a] is:
"A framework is the skeleton of an application that can be customised by an application developer."
This definition reveals another aspect of frameworks, apparently a framework is not a product by itself it is only a skeleton. A definition given in [Fay00] is:
"Frameworks are specialized for a narrow range of applications; each model of interaction if domain-specific, e.g. designed to solve a narrow set of problems."
This definition states that frameworks are domain specific, implicitly a conclusion can be drawn that not every framework is applicable to every domain. In [Fay02] another definition is given:
"A set of related functions is the set that defines the area of expertise or competencies of the framework."
This definition states that a framework holds a set of related functions; this amplifies the earlier statement that a framework is domain specific. The definition used in [Joh98] is:
"A software framework is a reusable design for a software system (or subsystem). This is expressed as a set of abstract classes and the way their instances collaborate for a specific type".
This definition holds aspects mentioned in earlier definitions, reusable design, specific domain but it also adds implementation design specific elements like abstract classes. Another definition found in [Joh97a] is:
"A framework is a larger-scale design that describes how a program is decomposed into a set of interacting objects."
This definition also mentions interaction.

These definitions lead to the definition that will be used in this thesis:
"A framework is a reusable skeleton for applications, targeting a specific domain, covering certain key areas and the interaction between them within that specific domain."

This definition is stronger then the definitions given before because it holds all the criteria specified before:

- reusable
- domain specific
- skeleton of an application
- various coverage per framework
- interaction

The definition does not cover the abstract classes mentioned in an earlier definition. That is because this is an implementation detail. It also suggests that all frameworks are object-oriented based since abstract classes originate in the object-oriented corner, which is incorrect [Fay99].

A framework is a reusable skeleton for applications, targeting a specific domain, covering certain key areas and the interaction between them within that specific domain.

2.3 Framework comparison options

In the literature, very little guidelines are given for comparing frameworks; some comparison is done based upon the architectural design of the framework itself [Pri06]. This however does not compare the implementation of the framework. Architecture is how a system should be composed, but often not how it is composed in reality. Thus, it is not valid as a comparison option since we want facts about **what a framework does**, not what **a framework should do**.

We feel a generative approach could be used. Instead of comparing the frameworks to each other, we want to compare how well framework A compares to framework B on specific criteria. This leaves the question on what criteria will we compare these frameworks. We can now return to our definition *"A framework is a reusable skeleton for applications, targeting a specific domain, covering certain key areas and the interaction between them within that specific domain."* This definition defines *"certain key areas*". If we can define these areas, we can compare the frameworks on those areas.

The idea of dividing into smaller areas evolved from a number of observed facts. In the paper [Fay00b] a list of characteristics is given for enterprise application frameworks, these characteristics include scalability, platform independency and extendibility. This gives an indication there is a set of criteria that all enterprise application frameworks cover. This paper however, focuses solely on enterprise application.
Combining the previous facts with the vertical and horizontal domains mentioned in [Cza00] and [Sin95]. These stated that a framework covers a specific vertical domain by implementing several horizontal domains. This results in the idea that if we can find these horizontal domains we can base the comparison on them.

This leaves one problem however, not all frameworks cover all horizontal domains, only those required to cover the vertical domain. In [Alp03] the philosophy behind what humans see as abstractions is explained. A conclusion here is that flying is directly connected to the abstraction bird, although only 90% of the birds can actually fly. Which gives some leverage to create a framework abstraction where not all areas are covered by all frameworks.
In essence, if we have a list of areas covered by framework A and a list for areas covered by framework B, we can compare the frameworks by comparing the implementation of the covered areas individually.

A criterion for abstraction is valid, as long as it encompasses 90% or more of the subjects under the abstraction.

2.4 Terms related to frameworks

A question that remains is what constitutes these frameworks? The literature for this subject is rather scarce.
In [Joh97a] the statement is given that frameworks are components combined with patterns. This view is consolidated by [Joh97b], which is by the same author; this paper relates components and patterns to frameworks. In [Rup96] a statement is given that frameworks are patterns in general.
From all this we can at least conclude that patterns and components are related to frameworks. The question remains related how. The following figure displays several factors mentioned in the literature related to frameworks.
This view is based upon functionality compared to generic or specific. Each term will be defined and its position in the figure explained.

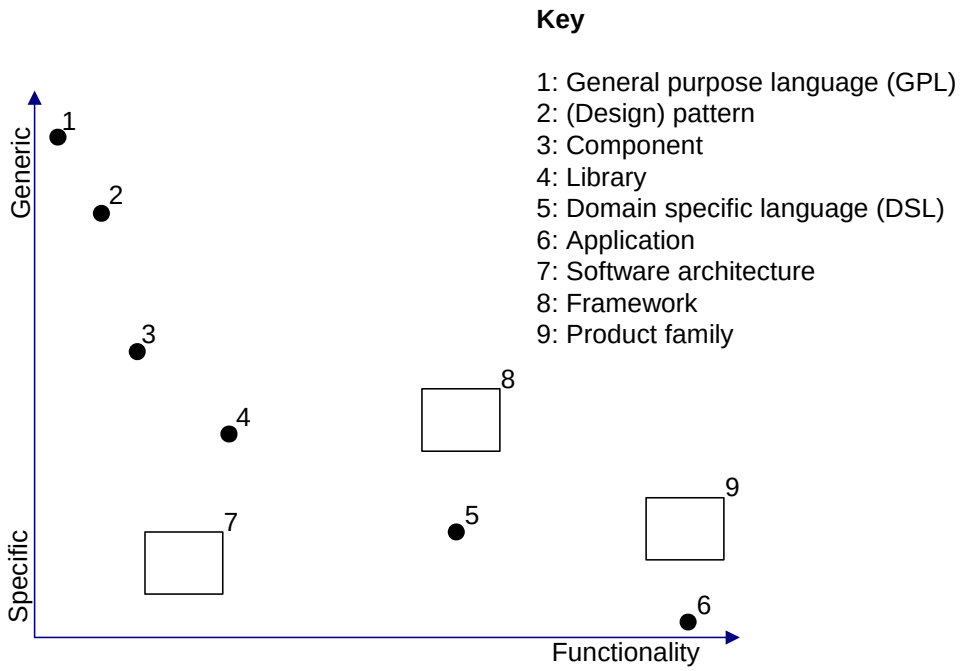


Figure 2: Generity versus functionality

2.4.1 General purpose language (GPL)

There is no real definition of general purpose languages, often the term *"not related to a specific domain"* is used, although this is no definition. General purpose languages are languages like assembler, Java and C Sharp. General purpose languages are generic; they have little restriction on the field they can be used. They can be used from embedded system to enterprise web applications. They offer little functionality, the only functionality offered, is normally high-level function to communicate with the underlying hardware. This is the reason why general purpose languages are positioned high in the generic scale and low on the functionality scale.

2.4.2 (Design) pattern

In [Fay99] a design pattern is defined as:
"A pattern is an essay which describes a problem to be solved, a solution, and the context in which the solution works."
A similar definition is given in [Bas03]. Examples of design patterns are the factory pattern and the model view controller (MVC) pattern. These patterns offer a generic solution to solve a specific problem. However, a design pattern does not dictate the precise implementation. This is the reason a design pattern still scores reasonable high on the generic scale. By providing the generic solution, they offer some functionality, thus score higher then a general purpose language for functionality.

2.4.3 Components

In [Cza00] components are defined as:
"Building blocks from which different software systems can be composed."
These building blocks come in the form of code and thus are more specific than design patterns. They offer limited functionality for a specific task. Thus, components score higher than patterns on the functionality scale. However, they are less generic because they come in the form of code. Therefore, they score less on the generic scale than design patterns.

2.4.4 Library

A library is defined in [Cza00] as:
"Library's package general and domain specific reusable abstractions together with the code supporting it."
This definition gives us some insight in where to place libraries. Libraries supply code, which means they become more specific and offer more functionality at the same time. Library's can be seen as a collection of components. This results in libraries being placed below components on the generity scale; by combining several components, the resulting library is more specific. For the functionality scale, it is placed between components and domain specific languages. This is because one library is not near a fully functional application; it does however offer a more functionality then a single component. A domain specific language will offer all or most functionality required to create an application, a single library is however not enough to create an entire application.

2.4.5 Domain specific languages

In [Mer05] a domain specific language is defined as:
"Domain specific languages are languages tailored to a specific domain"
This definition is short, but it does give us a lot of information on where to place domain specific languages.
A domain specific language is tailored for a specific domain, so it becomes a lot more specific. However it is not a working application by itself, it is the language in which to create the application. For this reason domain specific language are placed between library and application on the generity scale. For example, a graphical library to draw graphs can be used in several applications spreading several domains, but a domain specific language is just for one domain. Domain specific languages offer all functionality to create an application; however, programming in this domain specific language is required to make it a fully functional application. For this reason, we place domain specific languages between application and library.

2.4.6 Application

An application is normally the product of a software development project. It offers specific functionality required for the needs of a specific customer. It can offer a little configurability to tailor the application, but in essence, it is completely specific. This is the reason to place applications very low on the generic scale and very high on the functionality scale.

Until now, all elements have been represented with a dot, architecture, product family and framework are represented with a rectangle. The reason for this is that they are open definitions and each architecture can differ in functionality and generic properties. The same reasoning applies to frameworks and product families. Although the same could be said for the other five terms, no application offers the same functionality as the next, they represent as much smaller field then the terms, frameworks, product family and architecture.

2.4.7 Software architecture

When discussing software architectures we define software architectures [Bas03] as:
"The software architecture of a program or computing system is the structure pr structures of the system, which comprise the software elements, the externally visible properties of those elements, and the relationships among them."
This definition gives us some insight in where to place architectures. Software architecture by itself is not a working application, it is however for a specific application, for this reason it is not generic and rather specific. It is less specific then a domain specific language because it will have properties for non-functional requirements (performance, security etc) and not for functional requirements that a domain specific language does have. It does offer more functionality than a component, a component often offers functionality for a specific functional requirements while software architecture offers multiple non-functional requirements. For this reason, software architecture is placed between components and library on the functionality scale. A software architecture is for just one application, thus it is even less generic that a domain specific language and placed between a domain specific language and an application on the generity scale.

2.4.8 Framework

The next piece of the puzzle, are frameworks themselves. Frameworks are more generic then domain specific languages because a domain specific language might target data entry applications or banking applications. A framework however is more like to target enterprise applications or embedded applications and is on a higher level than a domain specific language and it is not near as generic as a pattern. For this reason, it is placed higher on the generic scale than domain specific languages. On the functionality scale, it is placed close to domain specific languages. If offers all the functionality to implement an application but still there is a need to write code to create a specific application. This is the same for a domain specific language. For this reason, it is placed on the same range as domain specific languages.

2.4.9 Product family

Product families or software product lines are defined in [Cle02] as:
"A set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and are developed from a common set of core assets in a prescribed way."
Product families can be seen as applications that are not yet completed; they need to be configured before it is a final application. Nevertheless, it does contain all the functionality required for an application or even more applications. For that reason product families are placed along the same line as applications in terms of functionality. They are less specific then software architecture, because software architecture is only for one application. Product families are targeting a specific domain however, for example a product line of mobiles phones. For this reason product families have been places at the same line as domain specific languages on the generic scale.

2.5 Summary

This chapter discussed why we would want to use a framework in the first place. The purpose of frameworks was identified as the re-use of existing design and/or code. In addition, the answer to **LQ-1** was provided: *"A framework is a reusable skeleton for applications, targeting a specific domain, covering certain key areas and the interaction between them within that specific domain."* In extend of the definition we discussed that frameworks cover a vertical domain (purpose) by covering key areas. These areas give us a starting point for comparison since if several frameworks are designed specifically for a vertical domain; those frameworks will all need to cover the same areas. Additionally, we positioned frameworks within the context of terms like architecture, applications, library's, etc. The result is displayed in figure 2.

3 Domains and abstractions

This chapter will focus on the smaller distinct areas mentioned in the framework literature. We will classify these areas as domains. First, we will discuss why to use domains at all and define the type of different domains. Secondly, we describe what a domain is. Next, we will describe the relation between domains and abstractions.

3.1 Domain purpose

We have identified that if we can divide the framework into smaller more manageable areas we can compare these areas individually. This is where domains come into play. The reason to use domains (and abstractions) is to reduce complexity. By dividing a problem into sub problems (domains), the complexity is reduced. Domains allow us to place certain characteristics within a boundary; this boundary is called a domain. We can then reason about the domains themselves without going directly to the details of the specific characteristics. One problem with domains and abstraction in general is that you are removing the details. A popular saying goes *"The devil is in the details"*. This is a thought to keep in mind while using domains and abstractions.

3.1.1 Vertical versus horizontal

Two types of domains are normally related to frameworks, vertical and horizontal. These definitions are explained in [Cza00] and [Sim95]. For the remainder of this thesis, we will use the following definitions.
"A vertical domain is a domain for which the framework is intended".
For example if a framework is designed to control robots in an automobile factory, that is it vertical domain.
"In order to cover a vertical domain a number of horizontal domains will need to be covered".
These horizontal domains can be domains like loading the controlling the robot, reading the specifications for the automobile etc.

3.1.2 Technical, non technical, vertical and horizontal

We have mentioned four types of domains in the context of frameworks now. We can place them as follows:

- Vertical
  - Technical / Horizontal
- Non-technical (composite, emerging)

A vertical domain is the domain for which a framework is intended, an example is data entry applications. Covering this domain is done by covering several technical (synonym: horizontal) domains like persistence, modifiability and performance. Technical domains are concrete and explicit.
By covering these domains, we influence several non-technical domains, as example we can use maintainability. Maintainability is non-technical; it is however, influenced by technical domains like modifiability and separation of concern. There are others factors like documentation and architecture that influence maintainability as well.
In essence, we can say that by covering the technical / horizontal domains, non-technical domains are influenced. Thus, non-technical domains can also be classified as emergent or implicit.

We use this classification to be able to reason about a frameworks purpose (vertical domain) independent from its implementation (horizontal domain). Also by separating non-technical, we reduce to complexity and focus on the aspects we can answer directly. Since non-technical domains, are composite and we could need to examine other aspects like documentation and architecture. These domains are very different and are outside the scope of this research.

A vertical domain is the domain for which the framework is intended.

Horizontal domains are the domains that need to be implemented in order to cover a specific vertical domain.

3.2 What is a domain

With domains in the context of software engineering, terms like domain engineering and domain specific languages surface quickly, although we want a more general description of a domain. In the context of domain specific languages, a domain is often a synonym for *application oriented, task-specific, problem-oriented or specialized* [Mer05]. This however does not bring us closer to a more general description of what a domain is. In [Bjr01] the following definition is given:
"*A domain is the universe of some specific application.*"
This definition and characteristics give the following elements for domains:

- Targeting a specific delimited area
- Task specific or problem specific or specialized

Another definition found in [Cza00] is:
"A domain is an area of knowledge or activity characterized by a set of concepts and terminology understood by practitioners in that area."
In this definition, we again find the specific area. Additionally we can derive these characteristic:

- Terminology understood by practitioners in that area
- A (limited) set

This leads us to the following definition of a domain:
"A domain is an area of knowledge for a specific problem or task characterized by a limited set of concepts and terminology."
In this definition, we can find the following characteristics:

- Specialized
- Task or problem specific
- Terminology for a specific domain
- A limited set

A domain is an area of knowledge for a specific problem or task characterized by a limited set of concepts and terminology.

3.3 Abstractions and domains

In one of the earlier chapters, abstractions were also identified as an influence to this research. However, no specific information has been provided for abstractions until this point. The reason for this is that abstractions and domains are tightly coupled. By identifying a domain within an overall structure, an abstraction is made.
A definition used for abstractions [Sai01] is:
"Abstraction, intended as the ability to dismiss irrelevant details and focus on the relevant information only"
A key aspect of abstraction is not mentioned however. In [Kno90] this aspect is mentioned:
"Problem solving uses abstractions to reduce the complexity of search by dividing up a problem into smaller sub problems."
This is exactly the reason to use abstractions and domains this research project. By reducing the complexity by dividing the framework into multiple smaller domains, the complexity is reduced.

Abstraction is a tool for solving a problem, not a goal by itself.

3.4 Summary

In this chapter we described why to use domains, the reason to use them is to reduce the complexity of the problem and divide it into parts that are more manageable. Furthermore, we gave the definition of a domain, which is "a *domain is an area of knowledge for a specific problem or task characterized by a limited set of concepts and terminology."* We also discussed the relation between abstractions and domains. By creating an abstraction for a problem, you are already dividing the problem into smaller areas, domains.

4 Technical domains

This chapter will discuss the literature for the domains we will compare. For this thesis, we will research modifiability and customisability. These two domains were given the highest priority by Everest, see figure 11. The reasoning behind the high priority for these two domains is discussed in section 7.3. We will first discuss features, which are the basis for both domains. Following we will discuss customisability and modifiability.

4.1 Features

Modifiability and customisability are two concepts closely related. First, we will explain what we see as customisability and what we see as modifiability. We see both concepts as changes in the features of the application. Where we define feature [Cza00] as:
"A property of a domain concept, which is relevant to some domain stakeholder and is used to discriminate between concept instances."
In [Bat05] features are defined as:
"A feature is an increment in program functionality."
Another definition found in [Eis01] is:
"A feature is a realized (functional as well as non-functional) requirement."
This is rather vague, but does add non-functional requirements to features.

From these definitions, we can infer the following characteristics:

- Relevant to a stakeholder
- Realised functional and non-functional requirements

From these characteristics, we can define features as:
"Features are realized requirements, functional and non-functional, relevant to a stakeholder."

From these characteristics, we can conclude that features describe the entire system. Any change in the code or configuration will result in a change in the features. Therefore, any modification or customization will be visible by a change in features or their behaviour.

Features are realized requirements, functional and non-functional, relevant to a stakeholder.

4.1.1 Feature binding time

Both customisability and modifiability are visible by changes in the features or their behaviour, but how do we separate the two. The difference is the point in time at which the change of feature is bound. By bound, we mean how do we make the actual change happen, can we do this after the product is released by changing some configuration parameter? Alternatively, does the source code need to be changed in order to encompass the change? This shows similarities with the definition of timeline variability [Dol03]:
"Certain features can be bound at several stages of the life-cycle."
From this definition, we can derive the difference between modifiability and customisability. The following figure displays what stages we define as modifiability and what stages we define as customisability.

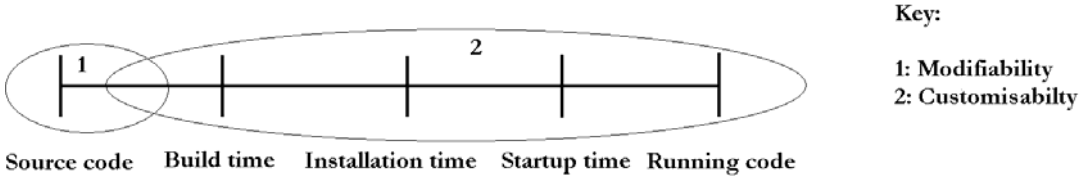


Figure 3: Binding timeline

The field of modifiability only holds source code. Where source code is changes to the general programming language code itself. Customisability holds build time, installation time, start-up time and the running code. With build time, we mean build parameters if a tool like Apache Ant [Apa07] is used. With installation time, we mean parameters that are given during installation; this can be the users name and initials for a word processor to system information for an operating system. With start-up time, we mean the time when the application is started. For example in the Apache HTTP server [Aph07], configuration files can be changed to configure which modules to load etc. Changes here can range from setting GUI parameters like the default background colour to loading new modules for the application.

More bound times can be named like deploy time, link time etc, but for the difference between modifiability and customisability these times have no influence.

We can see that there is some overlap between modifiability and customisability. This is something which cannot be avoided. Another way to look at customisability and modifiability is the following. Modifiability can be seen as changing the implementation of a specific functionality (features) in the code, this can even be interface changes. This view strengthen are view that modifiability is changes in source code. Customisability can than be seen extending upon the already present functionality, these wont be interface changes. This is where we see the overlapping area because these changes can also be caused by source code changes.

For the remainder of this thesis however, we will hold to the definition that modifiability are changes in source code and customisability changes without changing source code.

The difference between customisability and modifiability is the point in time at which the change of feature is bound.

4.1.2 Customisability and modifiability defined

From the previous, we can derive that customisability is:
Customisability is changing features or the behaviour of features without changing the source code.

While modifiability is:
Modifiability is changing features or the behaviour of features by changing the source code.

4.2 Customisability

The following section will discuss the literature used to form the method to compare the customisability domains.

4.2.1 Application customisation versus framework customization

We emphasise that there is a difference between customising the framework and customising the resulting application. A customization to the resulting application can be the creation of a new template and using this in the applications, the application is changed in terms of feature behaviour, and the look has changed for example. However, **this is not customization of the framework**. A customization of the framework would for example be a change in the elements the template engine handles. A new element or a change in elements, **this is framework customization.**
Another example is setting the transaction start point with declarative AOP for example; this is customizing the application, **not a customization of the framework**. **A customization of the framework would** be the type of transaction management used.

From the two examples above, we could infer that framework customization is done without any visual effect on the resulting application. This is valid for customization in the data and model layer, but this clear separation fades at the view level. For example adding support for more tags to the template engine could result in a change in the look of the application.

We make this distinction because we want to compare the frameworks customisability and not the resulting applications customisability. If we do not make this distinction, we do not answer the research question but a total different question, the question of customisability of the resulting application. The confusion here is caused because the framework itself is not a runnable but changes are always made while the framework is being used together with custom code which makeup the resulting application.
This indicates that every resulting application build by the same framework can still exhibit different customisability; this is caused by the custom code for each individual application. However, we are interested in the base customisability of the framework.

4.2.2 Customisability defined

From the feature binding time section, we can now use the definition:
Customisability is changing features or the behaviour of features without changing the source code.
However, very little literature is available for customisability in a technical context. A recurring paper on the subject is [Pin93]. The focus however, is financial. The paper focuses on when is it financial healthy to add customisability in a product. This topic is of no interest for us now. We have an existing product and want to evaluate how customisable it is, not if adding this customisability was financial sound. The do however state that customisability is added by creating variations. In [Jia03] an index is proposed to measure if a customization is financial sound, this part is again of no interest to us. The paper also identifies two aspects of customisability, which are of interest:

- What can be customised
- How easy can it be customised

For what can be customised, we need a way to record customization options, again [Jia03] supplies us with an option, and they list customization with variation. This results in a link to product families [Bas03].
In [Gur01] variability is defined as:
"Variability is the ability to change or customise a system."
This leads to the definition of a variation point [Jac97]:
"A Variation point identifies one or more locations at which the variation will occur."
This is inline with our definition of customisability. In [Jac97], the link between abstractions and variation is also made. However, no distinction between abstraction types is made in [Gur01], two types of abstraction points are identified:

- Open abstraction points, new features can be added.
- Closed abstraction points, no new features can be added.

4.3.3 Subtypes of features

Besides the general types we mentioned earlier, mandatory and optional, there are three more subtypes of features. Mandatory and optional describe how and if the feature is present in the product, while these sub types define, what the feature is composed of and how it can be used.
These types used are:

- Composite
- Atomic
- Alternate variant

Composite features:
In [Deu02] composite features are defined as:
"Features that are defined in terms of other features are composite features."
This indicates that a composite features value comes from it children. We can conclude that for composite features, we determine the value by evaluating its children.

Atomic features:
In [Deu02] composite features are defined as:
"Atomic features are features which cannot be further subdivided in other features."
For atomic features, we have to make a distinction between mandatory and optional features. Optional features require and additional step which is how do we enable this feature. After that step, we can treat all atomic feature the same and the question lies in the customisability of the feature.

Alternate variants:
In [Bat05] alternates are defined as:
"Only one feature can be selected of this group."
Alternate variants are sub features of a feature of which only one can be selected. The question of customisability with alternate variants lies in how we can enable the variants.

We can record this information in feature diagrams [Bat05].

This results in the following diagram:

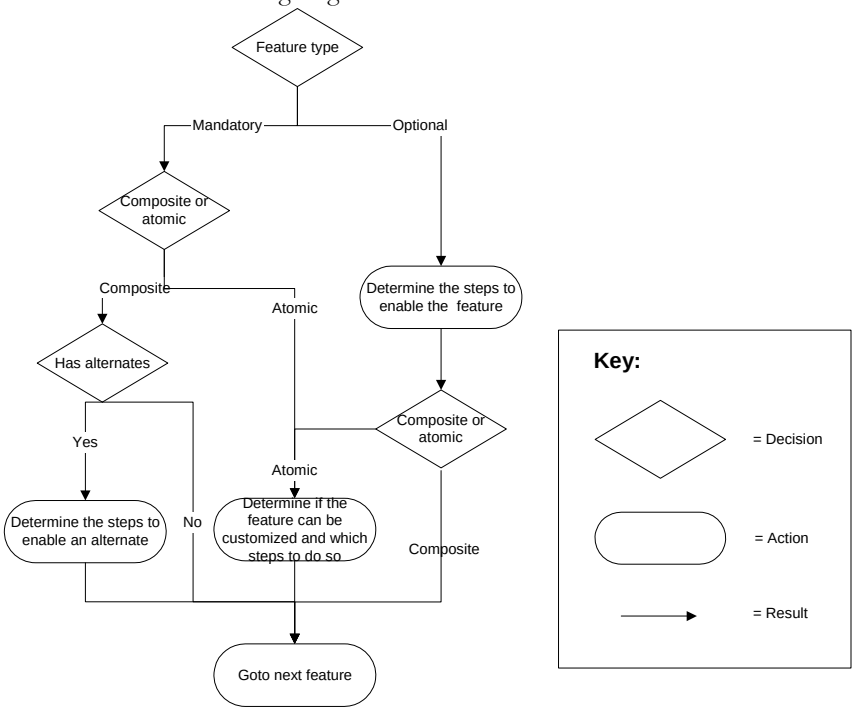


Figure 4: Steps for customisation per feature type

If we take the definition for customisability, again we can conclude:
Customisability is changing features or the behaviour of features without changing the source code.

- Change of feature -> Alternate variants, or optional features
- Change in behaviour -> Customizing features.

4.3.4 Abstraction types

The question remains, how can we identify open and closed abstractions. This can be done based upon if the applications architects **designed** the application in such a way that new features can be added. Abstraction creates a layer between the features and the rest of the applications. As mentioned in [Lin02]:
"Providing more stability than using the components directly; often enabling different implementations of the same functionality."
Since both frameworks are build on the Java general purpose language, abstractions layers can be identified by a set of abstract classes and/or interfaces with an optional base implementation.
Adding just the layer is only half the task; the children also need to use this layer instead of directly communicating with the rest of the application. The question remains, how we can measure this. From the previous, we can identify two characteristics:

- An abstraction layer.
- All underlying children should use this abstraction layer to communicate with the rest of the application.

We now have basic characteristics to look for but still no method to measure this. How do we measure this abstraction layer? As mentioned before this abstraction is commonly achieved by interfaces and abstract classes. Secondly if it is an abstraction layer it should have low dependencies, you cannot be abstract and rely on many other packages. In addition, the creation the abstraction layer is only half of the journey; each variant also has to use the abstraction layer to access the rest of the software instead of communication directly. We now have some characteristics we can measure:

- Abstraction level (ratio of interfaces/abstract classes to implementation)
- Each child should use the abstraction layer (depend on abstraction layer)

Because not all classes in the abstraction layer will be declared abstract. Some classes will have a base implementation. Because the abstraction layer is in essence the glue between the variant and the rest of the software, the classes in the abstraction layer should exhibit the following behaviour, which we identified in the previous section.

- A low efferent coupling
- On average a low ncss per method
- On average a high JavaDoc per method

Where ncss [Jnc07] is:
"None comment source statements."
Where efferent coupling is [Rei02]:
"The number of classes from other packages that the classes within the package depend upon."

This leaves the question of the ease of customisation. Ease is an open term, what is easy for person A can be hard for person B and visa versa. For this reason, we will have to identify criteria on which to determine the ease. Terms used in combination with configuration languages are *"configuration model, satisfying the requirements and valid"* [Soi99] and relates this to the tools to support this.

- Tools support for the underlying language, validity, completeness.

In [Leo00] another aspect is mentioned, centralising the configuration.

- Number of different places to edit the configuration (centralized or not)

We however feel an important criterion is missing here, which is:

- The time (or steps) it takes to customise something

Therefore, these three will be the criteria for comparison.

4.3 Modifiability

For modifiability, we can reuse and build upon the literature for customisability; however, there are distinct differences that we will discuss.
Mapping the initial criteria for customisability on modifiability, we get:

- What can be modified?
- How easy can it be modified?

From the feature binding time section, we can now use the definition:
Modifiability is changing features or the behaviour of features by changing the source code.

Following this definition, we can be divide modifiability into two groups

- Adding/removing features
- Modifying the behaviour of features

For what feature can be added and removed, we can use abstraction points in the code. Points that we identified in the customisation section. In particular, we are looking for open abstractions. Points in the code that are **intended to be extended**.

This leaves us with the last group, modifying the behaviour of features. In order to modify a feature, we first need to find it, which gives us the first indication of how we can make modifiability measurable:

- Finding the feature

Finding a feature can be translated to the modularity of the code. If the code is modular, we will be able to find the location of a specific feature faster than with code that is not modular. In [Loh84] this is confirmed. Which leaves us with the question, how can be measure modularity. For this, we can use [Rei02], which defines a number of criteria:

- Afferent coupling
- Efferent coupling
- Abstractness
- Instability

Where efferent coupling [Jde07] is:
"The number of classes from other packages that the classes within the package depend upon."
Efferent coupling [Jde07] is:
"The number of classes from other packages that depend on the classes within the package".
Abstractness [Jde07] is:
"The abstractness of a package is defined as the ratio of abstract classes / interfaces to the total number of classes."
And instability [Jde07]:
"The instability of a package is defined as the ratio of efferent coupling to total coupling."
These are all measurable criteria.

Once we have located the feature, the next step is to understand the code. This brings us to understandability of the code. Understandability is defined as follows in [Mcc04]:
"The ease of with which you cam comprehend a system at both the system-organizational and detailed-statement levels. Understandability also has to do with the coherence of the system at a more general level than readability does."
In addition, readability is defined as:
The ease with which you cam read and understand the source code of a system, especially at the detailed-statement level."
From this, we can infer some characteristics that the system should have:

- Coding standard, coherence of the system, which include naming conventions
- Low complexity, understand the source code

Coding standard cannot be measured, but we can examine a sample group for each application and determine if a coding standard was used.
Complexity however can be measured. For this, we will use the McCabe complexity [Mcc76] or often called the cyclomatic complexity. Although this method is sometimes criticized [Wey88] and alternatives are available, dataflow, lines of code, Hallstead's computer science etc. cyclomatic complexity is the de facto standard.

The concept of code understandability is an entire field on its own. Including several human interpretable factors, for our current research we pick the measurable and the most important aspects according to [Mcc04]. This is the complexity.

4.4 Summary

In this chapter, we discussed the two domains we will research, customisability and modifiability. We provided the basis that we will use for both comparisons, which are features. We defined features as *"features are realized requirements, functional and non-functional, relevant to a stakeholder."* Additionally we described the different between customisability and modifiability as *"the difference between customisability and modifiability is the point in time at which the change of feature is bound."*
Which lead to the definition for customisability as *"Customisability is changing features or the behaviour of features without changing the source code."* And the definition for modifiability as *"Modifiability is changing features or the behaviour of features by changing the source code."* Moreover, we made the concepts related to customisability and modifiability measurable.

5 Theoretical framework model

In this chapter, we will introduce a framework model; we will describe how this model came to be.

5.1 Vertical domain of a framework

As mentioned before the vertical domains of a framework is the application type for which the framework is intended. Application type can for example be data entry, in this application type the focus will mainly be on how the data is entered and how it is stored, which can be classified as the technical domains. Another type can be online banking applications, for this type the focus will be more on security and concurrent users as well as the data storage. These can again be classified as the technical domains of the application type.

This tells us that the vertical domain depends on the technical/horizontal domains of the framework. For this reason the focus is on the technical domains

5.2 Technical domains of a framework

In order to divide a framework into separate domains the following approach has been used. During the initial literature study terms have been collected which are related to frameworks.
These terms were chosen on the following criteria:

- They were related to framework characteristics / properties
- They were mentioned in more than two separate articles
- Both articles were from different authors

5.2.1 Technical domains versus characteristics / properties

We state that technical domains and characteristics are equal. We do this on the basis that when we start designing a system we will want the system to exhibit certain properties (requirements). However, it is not always true that the resulting software will meet these requirements. We intended the system to handle all payments automatically, but it does not do that. However, it does have certain characteristics; it handles the payment until a signature is required.
This is distinction. **Requirements are what we want the system to do**. While **characteristics are, what the system actually does**. Thus, characteristics / properties are equal to the technical domains because the system exhibits this and this can be found in the source code.
In essence, requirements, non-technical requirements and architecture are what you **want the system to do**.
Characteristics are **what the system actually does.**
Thus, we want characteristics and not the others; we want what the system does and use that for comparison.

5.3 Initial list

The initial list after this process in no particular order is:

Term	**Reference**
Authentication	[Fay99] [Blu07] [Spr07] [Wes04]
Authorization	[Fay99] [Blu07] [Spr07] [Wes04]
Availability	[Pri06] [Fay99] [Spr07]
Complexity	[Fro97] [Fay99] [Blu07] [Spr07] [Mar00] [Joh97a] [Bak06] [Wes04]
Configuration	[Pri06] [Fay99] [Blu07] [Spr07] [Mar00] [Bak06] [Cas05]
Connection pooling	[Blu07] [Spr07]
Customisability	[Joh97b] [Fro97] [Fay99] [Spr07] [Mar00] [Fro00] [Joh97a]
Dependencies	[Fro97] [Fay99] [Blu07] [Spr07] [Mar00] [Bak06]
Distributed	[Fay97] [Fay99] [Blu07] [Spr07] [Zar04]
Error/Exception handling	[Blu07] [Spr07] [Wes04]
Extendibility	[Fay97] [Fro97] [Fay99] [Blu07] [Spr07] [Mar00] [Joh97a] [Bak06]
Fault tolerance	[Fay99] [Blu07] [Spr07] [Bak06]
Flexibility	[Pri06] [Fay97] [Fay99] [Bak06]
Internationalization	[Pri06] [Blu07] [Spr07] [Bak06]
Interoperability	[Fay97] [Fay99] [Cas05]
Logging	[Pri06] [Fay99] [Blu07] [Spr07] [Wes04]
Modifiability	[Fay97] [Fay99] [Blu07] [Spr07] [Fro00]
Modularity	[Fay97] [Fay99] [Blu07] [Spr07] [Zar04]

Openness	[Fay97] [Fay99] [Fro00] [Zar04]
Performance	[Fay97] [Pri06] [Fay99] [Blu07] [Spr07] [Zar04]
Persistency	[Pri06] [Fay99] [Blu07] [Spr07]
Portability	[Pri06] [Fay99] [Spr07]
Reliability	[Fay97] [Fay99] [Blu07] [Spr07]
Scalability	[Pri06] [Fay99] [Blu07] [Spr07] [Zar04]
Security	[Pri06] [Fay99] [Blu07] [Spr07]
Separation of concern	[Fro97] [Fay99] [Mar00] [Bak06] [Wes04] [Cas05]
Session handling	[Fay99] [Blu07] [Spr07]
Skeleton implementation	[Fro97] [Fay99] [Mar00] [Bak06] [Zar04] [Bos97]
Tailorability	[Fay99] [Wes04]
Testability	[Pri06] [Fay99] [Spr07] [Bak06] [Wes04] [Bos97]
Transaction management	[Blu07] [Spr07] [Zar04]
Transparency	[Fay99] [Zar04] [Wes04]
Unit testing	[Pri06] [Spr07]

Figure 5: Initial characteristics of a framework

Note that for terms that already had several references, no extra ones were collected. At first glance, this list has several elements that overlap, for example security and authentication. Absolutely no pre selection has been made here. In addition, it has elements that are not at the same 'level'. For example, authentication and authorization can be seen as 'children' of security. Therefore, this list is only the start of the process; the next step is to group these elements to identify hierarchies and overlap.

5.4 Grouping of domains

In this section, we will group the related elements. This has been done based on several criteria:

- Context of the element in the literature
- Definitions of the element

All elements related will be in the same row, no ordering within rows will be done.

Scalability	Distributed	Transaction management		
Persistency				
Security	Authentication	Authorization	Logging	
Error/Exception handling	Reliability	Availability	Fault tolerance	
Customisability	Configuration	Tailorability		
Openness	Interoperability	Transparency		
Skeleton implementation				
Performance	Connection pooling	Session handling		
Separation of concern	Complexity	Modularity	Dependencies	
Modifiability	Flexibility	Extendibility	Portability	Internationalization
Testability	Unit testing			

Figure 6: Grouped characteristics of a framework

The next step is the filtering of the elements.

5.5 Filtering of elements

This section will filter the remaining element. The rationale can be found in Addendum A. This has been done in the following way:

- Removal of duplicates or synonyms (Strikethrough)
- Identifying the 'top' level, horizontal domains (Bold, first column)
- Invalid terms (Underlined and strikethrough)

Horizontal domain	Possible implementations			
Scalability	Distributed	Transaction management		
Persistency				
Security	Authentication	Authorization	Logging	
Availability	~~Reliability~~	Error/Exception handling	Fault tolerance	
Customisability	Configuration	~~Tailorability~~		
Interoperability	Openness	~~Transparency~~		
~~Skeleton implementation~~				
Performance	Connection pooling	Session handling		
Separation of concern	Complexity	Modularity	Dependencies	
Modifiability	~~Flexibility~~	Extendibility	Portability	Internationalization
Testability	Unit testing			

Figure 7: Filtered horizontal domains of a framework

The next section will describe the domain and discuss the reasoning behind it. It will also discuss why certain elements have been removed.

5.6 Technical domains of a framework

This section will list the technical domains of a framework inferred and list the description for each domain.
The reasoning behind each technical domain can be found in Addendum A. that will also discuss the reasoning behind the decision to call it a framework domain. Only the top levels are important and can be recognized as a domain. For example in the security row, authentication is one option to make systems more secure; it is not the only option. For this reason, filling columns besides the most left one is not important for this research.

For several of these domains the abstraction guidelines in [Alp03] are used. The 90% rule applies here. Not all domains described here are valid for all frameworks. Nevertheless, if a great portion of frameworks exhibit a certain domain; they are part of the model. See the example of birds and flying in [Alp03].
Skeleton implementation has been removed because this is an implementation of the base architecture, this base architecture should encompass all framework domains. Therefore, by comparing all the technical domains in a framework we already compare the skeleton implementation implicitly.

At first glance, these domains appear to overlap with non-functional requirements or quality attributes. However, architecture is in essence an abstraction of the software so it is only natural they should surface when creating and abstraction of a framework that is also software. If a software system has characteristics attached to it, its only natural the architecture (abstraction from the system) also has characteristics attached to it.
The key difference is that these domains will be compared on implementation level, not on an architectural level. None the less, it looks like frameworks try to make the implicit attributes more explicit.

Frameworks try to make non-functional attributes more explicit.

Horizontal domain	Description
Scalability [A-1]	*Scalability is the ability of a computer application or product to continue to function well as it (or its context) is changed in size or volume. Typically, the rescaling is to a larger size or volume.*
Persistency [A-2]	*Persistency refers to the characteristic of data that outlives the execution of the program that created it.*
Security [A-3]	*Security is the measure of the system's ability to resist unauthorized usage while still providing its service to legitimate users.*
Availability [A-4]	*Availability is the probability that the system is operating at a specified time.*
Customisability [A-5]	*The ability and ease for software to be changed by the user.*
Interoperability [A-6]	*The ability of systems, units, or forces to provide services to and accept services from other systems, units or forces and to use the services so exchanged to enable them to operate effectively together.*
Performance [A-7]	*Performance is the degree to which a software system or component meets its objectives for timeliness. Thus, performance is any characteristic of a software product that you could, in principle, measure by sitting at the computer with a stopwatch in your hand.*
Separation of concerns [A-8]	*Separation of concerns is s the process of breaking a program into distinct features that overlap in functionality as little as possible. A concern is any piece of interest or focus in a program. Typically, concerns are synonymous with features or behaviours.*
Modifiability [A-9]	*Maintainability is the capability of the software product to be modified. Modifications may include corrections, improvements or adaptations of the software to change the environment, and in requirements and function specification*
Testability [A-10]	*The likelihood that the software will expose a failure under testing, if it is faulty and the degree to which it is easy to fulfil a given test coverage."*

Figure 8: Horizontal domain list of a framework

5.7 Summary

This chapter introduce a framework model on which we will base our comparison. The 10 technical domains are introduced which along with their definition which can be found in figure 7.
This answers **LQ-2** and **LQ-3**. We also explained the reasoning behind the theoretical model.

6 Research method

This chapter will identify the cohesion of the research questions, secondly we will describe the research method for comparing frameworks. In addition, we will describe the method for comparing the customisability and modifiability domain.

6.1 Research method demands

For the research method to be useful in the first place we will impose some constraints.

- Repeatable, the method must be repeatable by a different executor and yield the same result
- Time demand, the method should be executed in one month, in order to allow it to be useful in any industrial setting
- Validation, the method should provide a validation for the result, in order to provide a sound conclusion

6.2 research cohesion

This section will describe the cohesion of the individual research questions, how they will lead to an answer.

6.2.1 Research questions recap

The research questions are:
RQ-1 *'Can application X conform framework A be converted to framework B"*

Sub questions that arise from the main question are:

- **RQ-2** *"Does application type for both frameworks overlap"*
- **RQ-3** *"How to determine which criteria to use for comparing the frameworks"*
    - **RQ-4** *"In which advantages and disadvantages will this conversion result"*
        - **RQ-5** *"Is the change of Framework desirable"*

As can be seen from the indentation, **RQ-2** serves as input for **RQ-3, RQ-3** as input for **RQ-4** etc.

6.2.1.1 Research question 2

RQ-2 can be answered by comparing the vertical domain of framework A with the vertical domain of framework B. Basically this is the application type for which the framework was intended. This type can often be found in the frameworks documentation.
As mentioned earlier, only when the application overlap is a comparison feasible. If this condition is not true, work will be done which will be pointless. If this is not the case**, RQ-1** can be answered with **No**.
The overlap will never be 100%, this can be compared to the architecture of any software application. There are always tradeoffs between certain aspects. Again we can use the 90% abstraction guideline.
This question is crucial because if this one is answered with a no, proceeding is pointless, since if the application types do not overlap, the horizontal domains will not overlap either. Thus, we have no common horizontal domains and we cannot compare anything.

6.2.1.2 Research question 3

For **RQ-3,** we can use the framework domains identified in **LQ-2**, which are displayed in figure 7.

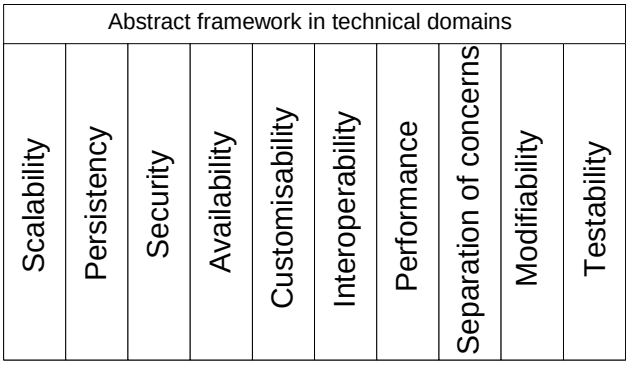


Figure 9: Framework domains

First, we have to identify which domains are important for application X. This can be done by using the high-level requirements for application X. These high-level requirements can be documented or they can be obtained from the applications creator and/or architects etc.
The next step is to prioritize these domains, since there are not that many framework domains; a number from on to ten for each domain will suffice. Once this list is created, a separate minor research project can be conducted for each of these domains.

6.2.1.3 Research question 4

The results of these small research projects will be analysed and a comparison can be made how well framework A covers a given domain versus framework B. Since each domain will be analysed separately, the analysis will results in coverage for each domain. The positive and negative effects are the advantages and disadvantages mentioned in **RQ-4**. Important factor here is that this is an **iterative process for each framework domain.** This, because by prioritizing and iterating we ensure the most important domains are covered first.

6.2.1.4 Research question 5

As mentioned in chapter 1, **RQ-5** is influences by non-technical factors as well. In order to answered the question if the change of framework is desirable we will take the results for all framework domains from both framework and compare them. This will allow us to answer **RQ-5** from a technical perspective. In order to get a definitive answer the none-technical aspects need to be added to the equation. This is however outside the scope of the current research and therefore we **only partially answer RQ-5**.

6.2.1.5 Research question 1

With all data gathered in the **RQ 2-5,** we can give a final verdict on if an application X can be converted from framework A to framework B. In this verdict, we can mention the results from **RQ2-5**. **RQ-1** can be seen as the conclusion for **RQ2-5**.

6.3 Research steps

The following figure displays how each RQ relates:

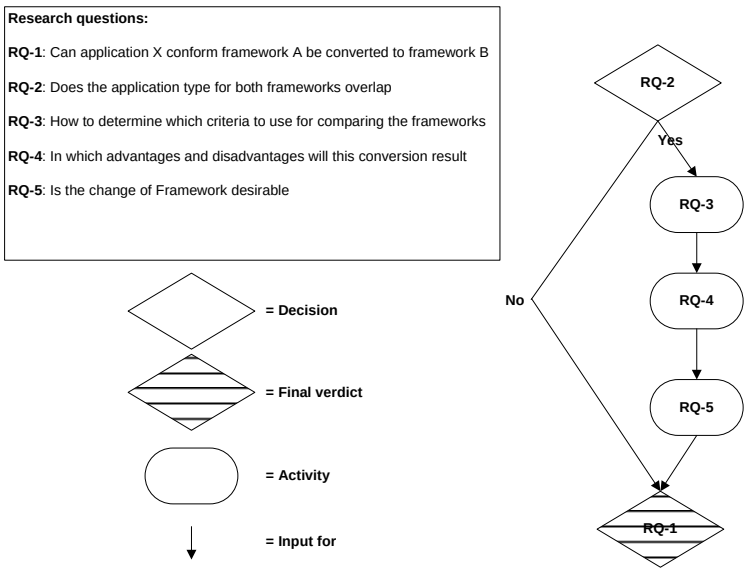


Figure 10: Research steps

6.4 Technical domains research

This section will discuss how each of the technical domains will be compared and in which groups we can split these comparisons.

6.4.1 Comparison groups

First, we will divide the horizontal domains into two groups for comparison:

- Dynamic
- Static

The difference between these two groups is that the dynamic group will be mainly compared by experiments and prototypes. The static group will be compared by a number of techniques like metrics, DSL comparison etc.
It is entirely possible that a domain in the dynamic group also has a static component and visa versa, but the emphasis will be on the group in which it has been categorized.

Under **dynamic,** we can position the following the domains:

- Scalability
- Availability
- Performance

Under **static,** we can position the domains:

- Persistency
- Security
- Customisability
- Interoperability
- Separation of concerns
- Modifiability
- Testability

6.4.2 Comparison domains

For all the technical domains that will be compared dynamically, one addition note applies. Most of these frameworks C#, J2EE, run on a specific type of application server or a specific plug-in (.net runtime environment, EJB container etc). These applications servers or plug-ins themselves also effect the availability (for example auto restart on error), scalability (clustering) and performance (load balancing). Moreover, since a framework itself is not a runnable application, a prototype will have to be made and this prototype will run on an application or plug-in. In addition, this application server or plug-in will in turn run on a specific OS. **In essence your testing the combination of framework, with application server / plug-in and the underlying OS with dynamic comparison**. This important factor should not be overlooked.

In the above example we used the most extremes we could find. Since C# is highly dependant on the underlying OS since it almost exclusively runs on Windows platforms. For C# we could say we are testing the framework + OS. This is different for J2EE, J2EE strives to be platform independent. This results that by testing J2EE frameworks, you are mainly testing the framework + application server.

6.5 Customisability

This section will describe the research method we will use to compare customisation.

6.5.1 Research method

From the customisation literature combined with the theoretical framework model, we can now propose a method for comparison that can be divided in several steps. Step 1 through 4 will measure what can be customised; step 5 will measure the ease of change.

Step 1, feature extraction
Determine the features and their types in both frameworks.
This will be done based on the feature diagram. Most likely, these are not available and will be extracted from the architecture and documentation.

We need to make one enhancement to feature diagrams for our purpose. Mandatory features are defined as features that are required to get a working application. In terms of frameworks, we need to expand this a little. There are features which are not required to get a working application but they are always present.
For example, the feature in Spring for dynamic language support. This is always available; however, you do not need to use dynamic languages at all.
Therefore, for the context of frameworks we redefine the definition of mandatory features:
Mandatory features are features that are always present and cannot be removed.

For completeness, we will also define optional features as:
Optional features are feature we have to be enabled to be used, but are not required.

Step 2, abstraction types
This step will identify open and closed abstractions points. We use the composite features here and not just alternate variants because composite features hold the alternate variants and any feature composed of children. These two groups can have an abstraction layer. For example with remoting, this is not an alternate variant, but it is very likely all these types of remoting will use a similar way of communication with the rest of the application. Thus, an abstraction layer could be present. This however, is not a rule, for example enterprise utilities, although they are grouped together, it is unlikely that they will all communicate in a similar matter.

This step is not directly related to customisability, but it is a natural step, a top down approach. It will also provide a form of validation for the first step. By examining the package structure deeper, we can determine if we missed and features and if there are features in the diagram which do not belong their since they cannot be found in the code and libraries.

The last step is tools, which enable us to collect this data. For this we can used JDepend [Jde07]. One note on the use of JDepend, it also measures standard libraries. This is not what we want so we excluded the following packages from analysis:

- Standard libraries (Java.*, javax.*, sun.*, sun.com.*)
- Tests (vary per framework)
- Logging (org.apache.commons.logging)

For measuring the base implementation, we used a second tool JavaNcss [Jnc07]. This tool allows us to measure the ncss (none commenting source statements), JavaDoc statements, methods and classes per package.

All these metrics use the package as parent, which means we have to pinpoint each feature to a specific package or sub packages; this will be listed together with the features.

Note that not all features will be present in the source code listing presented, the features missing are libraries supplied with the framework in non-source code format. We did verify that these libraries are present, as mentioned before if the feature is not present in the source code or in the libraries, it does not exist.

Step 3, customisation identification
Now that we have identified the features, we can determine what can be customised.
Important here is that we are going to answer the question:
What (feature / components) can be customised in the framework?
Not the question:
To what degree can each individual component be customised?
There is a huge difference between these two questions. The first one can be answered by a 'yes' or 'no', the second cannot be answered like that. Due to the scope of this research, we will limit ourselves to the first question.

We will now make a forward reference to the next step in which we will determine how many steps we need to do in order to use a feature. Because **step 5, ease of change, has overlap with this step**, we will conduct step 5 first. Once we complete step 5 we can identify if any of the steps taken in the process are customisability steps, and if they are we can add a 'yes' to the feature.

We also need to identify if the customization is added by the framework or if it facilitates this customization by, for example, providing a way to provide parameters to the feature, or if the feature already had this customization option by default. If the feature already has this customization option, it does not count towards the customisability of the framework. This can also be taken from step 5, the comment *'[C]'* is added if the feature had this customization by default.

In addition, we separate the types of features into groups:

- All features
- Skipped features
- Optional features
- Features with alternate variants

This has been done so that we have a clearer picture about all the aspects of customization. These categories are based upon figure 4. We also add another distinction for these groups, if the customization is achieved solely by customization (pure) or if it is a mix (mixed) between customization and source code. For this, we add a weight column. In the first case we set a **1** for the weight factor, in the second case we set a **0,5**.

If multiple options are possible to customise a features, we will check if there is a customization option which is pure customization first, if this is possible we set a *'1'* to the weight factor, if not we add a *'0,5'* for a mixed.

Step 4, tool support
Tools support is important for configuration files, errors are easily made in manually editing configuration files.
In the step, we will count the number of tools available and list these tools combined with the number of configuration files it can manipulate.

Step 5, ease of change
The next step is identifying for each feature what steps need to be taken in order to change or enable the feature. This way we will measure the ease of change. Since most steps are a mix of customisation and modification, separating the two here is not possible. Hence, we will measure the ease of change. The way we measure this is straightforward. Suppose we want to enable or change feature X, what do we need to do? Which steps would we need to take? In addition, what type of steps? Are there multiple ways of achieving the change?
We use two step types:

- Source code step, creating or editing one file in the general programming language.
- Configuration step, creating or editing one configuration file.

We will record the following information:

- Feature, the feature name
- Step category, enabling or changing the feature
- Step number, how many steps do we need to go trough, are there multiple ways to reach the same goal?
- Step description, what did we do this step
- Step type, source code or configuration

These prototypes do not have to be fully functional programs, as long as they enable us to answer our question what needs to be done in order to select or use the feature. If the feature, for example, is Hibernate, we only need to determine what we need to do in order to use this type of object relation mapping, we do not need to write and entire application to test the mapping itself.

For determining if a feature already had this possibility for change, we use the documentation of the feature. If the change is mentioned there; it was already present and the framework does not add any value.

Note that we do not measure step in time, this is because the length it will take person X to complete a step can differ greatly from the time it takes person Y to complete a step. The number of steps for each person stays the same however.

6.5.2 Validation

Validation can be split into two sections, result validation and method validation, we will discuss method validation first. Scenarios were used to validate both the method and the result. Three scenario's where chosen which were applicable to both frameworks. This number was chosen based on the available time in relation to statistical chance. Also we used scenario's because metrics and diagrams are all theoretical, performing the customisation itself far more practical. If both theoretical and practical suggest the same result we can offer a far more sound conclusion.

6.5.2.1 Method validation

For each step in the research method we will list the assumptions made and the performed validation for the step.

Step 1, feature extraction
A large part of the result depends on the validity of the feature diagram. If a feature was not located it will not be counted for any of the remaining steps at all. The feature diagram was validated in the form of an interview. For Spring the interviewee were the creators of Spring, interface 21. For blueprints the interviewee were the employee of Everest which have considerable experience with Blueprints.

The feature diagram was given and the following questions were asked:

- Are any features missing
- Are any features present which should not be
- Are the features in the correct tree (child/parent order)
- Are the features types marked correctly

Assumptions made in this phase were:

- All features present will be listed in the documentation
- If a feature exists, it will be visible in either the source code or libraries.

We cannot offer much validation for the first assumption, we state that if a feature is present it will be listed in the documentation. If this is not the case the features will not be used since the application developer cannot find it. This validation was performed in step 2, abstraction types, where we examined the code deeper and identified if all features which we extracted from documentation were in fact present in the code or libraries.

Following these interviews minor corrections were made to the feature diagram until both parties agreed on the validity.

Step 2, abstraction types
Several assumptions were made in this step, these assumptions were:

- Each package can be pinpointed to a single feature
- Abstractness, method to NCSS, method to doc and efferent coupling determine the abstraction type

That each package can be pinpointed to a single feature is suggested by the package structure of point frameworks, however we cannot offer and validation for this. This is also one of the discussion points mentioned later in this thesis. The use of specifically these metrics originates in the literature, which can be seen as the validation for using these metrics

Step 3, customisation identification
The following assumptions were made in this step:

- If a customisation is possible it will be mentioned in the documentation
- If a customisation is possible, but it was already possible without the framework (for integrated features like Freemarker, Velocity, etc) in this care the features documentation will list this
- The feature type dictates how to identify the customisation

The first assumption was made specifically due to the time constraints. Although the literature suggests it to be true. Some feature will be customisable while the documentation does not list it, there is nothing we can do about that. The second assumption followed during the fact collection for this step, almost every feature was customisable, we than realised most of those features were integrated products. The customisability of these integrated products does not count towards to customisability of the framework, this is in the same order as customisability for the final product does not count for the customisability of the framework.

The last assumption is again supported by the literature for customisability.

Step 4, tool support
Assumptions made in this step were:

- Tools provide a faster way of customising than by hand.

This assumption is supported by the literature.

Step 5, ease of change
The following assumptions were made:

- The ease of change depends on the number of steps to achieve the change
- The complex single step counts for less than two simple steps

The number of steps will be validates by actually performing the customisation for the three scenario's. The results confirm our assumption, the more steps it takes the more time is required, see figure 16.
The same can be said for the second assumption, each time if the number of steps was higher the time required went up. No single step scenario was found which took longer than any two step scenario. Although this a weak validation of this assumption, due to the small sample group, we can offer no better validation at this time.

6.5.2.2 Result validation

We selected a sample group applicable to both frameworks. This sample group consist of framework customisation, and not application customisation.
We did use an application as basis for the comparison; this application was the pet store. This since the pet store was available for both Spring and Blueprints. We used the following customizations, which all have at least one customization step according to addendum H and I, note that we only use this data to select the scenario's, we do not use it in any other way for validation.

- 1, Logging all incoming requests to the command line, framework requests cycle
- 2, Redirect all 404 errors to an error page, frameworks request cycle
- 3, After entering the password wrong 3 types, alert an administrator (via email), framework security

We recorded the following information for validation:

- Lines of code changes
- Minutes taken to complete
- Steps taken to complete

This is the first time we touch the time aspect; we tried to avoid this because it varies greatly depending on the person doing the experiment. All customisation were done by the author of the thesis, to which both frameworks are new, although he has two years of J2EE experience. All times are excluding tests and deploy times, just the time spend on the customisation.

6.6 Modifiability

This section will describe the research method we will use to compare modifiability.

6.6.1 Research method

From the modifiability literature combined with the theoretical framework model, we can now propose a method for comparison that can be divided in several steps. Step 1 through 4 will measure what can be customised; step 5 will measure the ease of modifiability.

Step 1, extension points
Determine which extension points the system has, for this we can reuse the data from step two in customization. These points identify if features were intended to be added there. For removing features, the abstraction layer will make removing easier, since the feature only communicates with the abstraction layer. If no abstraction later is present, removing a feature will result in changes spreading more locations than with the use of an abstraction layer.

Step 2, finding features
In the context of finding the feature, we want data about the modularity of the system, for this, we gather data about the following characteristics:

- Afferent coupling
- Efferent coupling
- Abstractness
- Instability

All these characteristics can be measured with JDepend [Jde07] and the same extracted fact file was used which we extracted for customisation, we only applied a different XSLT to extract the required data.

Step 3, understanding features
In the context of understanding the feature, determine the following:

- Is a coding standard used and visible within the code, take a sample group for this.
- Determine the code complexity by using the cyclomatic complexity metric.

The coding standard step will have to be done manual, for the cyclomatic complexity, we used Cobertura [Cob07].

Step 4, feature dependencies
Once we have located the feature and understand it, we have to perform one additional step before we can modify the feature. In the context of the impact of a chance we need to identify dependencies, we can determine this from the data for step two; we can use the efferent coupling, which are the dependencies outside the package.

- Impact of the modification

For this, we can use a metric we mentioned earlier, the efferent coupling, but this time we will not evaluate other properties, we are only interested in the number of dependencies, in other words the efferent coupling.

Step 5, ease of change
This leaves us with the last question, the ease of modifiability. For this, we can reuse the section for the ease in customisability, step 5. Since we ignored the step type in that part of customizability, we already measured the easy of modification/customisation combined, the ease of change.

6.6.2 Validation

Validation can be split into two sections, result validation and method validation, we will discuss method validation first. Scenarios were used to validate both the method and the result. Three scenario's where chosen which were applicable to both frameworks. This number was chosen based on the available time in relation to statistical chance. Also we used scenario's because metrics and diagrams are all theoretical, performing the modification itself far more practical. If both theoretical and practical suggest the same result we can offer a far more sound conclusion.

6.6.2.1 Method validation

For each step in the research method we will list the assumptions made and the performed validation for the step.
The validation data used van be found in figure 21, as well as the performed steps.

Step 1, extension points
This step has been discussed in step 2, abstraction types for customisation.

Step 2, finding features
The following 3 steps all focus on changing the feature, first we have to find it, than we need to understand it and finally we need to understand what impact the change will have. The assumption here was:

- The following metrics determine the modularity of a system (abstractness, instability, where instability is the ration between CE/CA)
- The modularity of a system determines how easy it is to find a feature

Literature offers validation for the both assumption as can be found in the literature for modifiability.

Step 3, understanding features
For understanding we made the following assumption

- A feature is harder to change if it is harder to understand (more complex)

We can validate the assumptions for the last three steps with our validation data.
Scenario one took 5 minutes per step for both frameworks. The complexity for blueprints was 1 (com.sun.j2ee.blueprints.encodingfilter.web) and for Spring this was 1,59(org,springframework,web,filter)

The second adjustment took 10 minutes per step for Spring 15 per step for Blueprints. Thus we should expect the complexity to be higher. For blueprints the complexity was 2,63 (com.sun.j2ee.blueprints.waf.controller.web) for Spring it was 2,08 (org,springframework,transaction,support). This supports the assumption.

The last scenario took 15 minutes per step for both frameworks, we expect around the same complexity as with scenario two. The complexity calculations are a bit different here because the are the averages of the steps. For Blueprints this was 2,4 (3,64 + 1,29 / 2), for Spring 2,0 (1 + 1,70 + 3,2 / 4), which is inline with what we expected.

Step 4, feature dependencies
For dependencies we made the following dependencies

- The more dependencies a feature has, the more impact a change will have

This assumption can be validated by the literature which suggests the same.

Step 5, ease of change
This step has been discussed in step 5, ease of change for customisation.

6.2.2.2 Result validation

For the result validation, we will use a similar approach as the one used to validate customisability. We will select a sample group of three scenarios for modification. According to our data, Blueprints does not offer extension points to remove or add new features. Thus, we are limited to changing the current functionality. Again, we will use the pet store application and basis.

The selection of the group was rather hard, but eventually we used the following scenarios, which all have at least one modifiability step according to addendum H and I, again this data has only been used to create the scenario's no other data has been used from these addenda.

- 1, Setting the response encoding to UTF-8, front controller
- 2, Extend transactions, log all transactions, data layer separation (Transaction / business controller)
- 3, Display a different interface depending on the logged in user role, page flow

We recorded the following information for these tests

- Lines of code changes
- Minutes taken to complete
- Steps taken to complete

7 Case Study

This chapter will give a brief introduction of both frameworks used in this case study and determine if they both are frameworks by the definition given in chapter two. Next, we will prioritize the technical domains and discuss customisability and modifiability.

7.1 Frameworks introduction

This section will briefly cover the two frameworks under comparison. We will discuss if they even fall under the category framework or not.

7.1.1 Sun Blueprints

Sun Blueprints does not have framework in its name . According to [Blue07] it is reusable code and patterns combined with best practices; however, it does exhibit the characteristics mentioned in the framework description. Sun Blueprints was created because developing J2EE applications was hard; there was no starting point. Sun Blueprints was the first to offer guidelines in the process of J2EE application creation. The definition was *"A framework is a reusable skeleton for applications, targeting a specific domain, covering certain key areas and the interaction between them within that specific domain."*
From the information on [Blue07], the following characterises can be retrieved:

- Aimed at enterprise applications => domain
- Reusable patterns and code => reuse
- Covering key areas with code and patterns => covering key area's, interaction

This leads to the conclusion that Sun Blueprints is a framework.

7.1.2 Spring

Spring is a much younger then Blueprints. The motivation behind Spring is that development for J2EE applications should be a lot easier. The Spring framework has framework in its name. The question is, is this valid? From the websites [Spr07] and [Tss07], the following characteristics have been retrieved:

- Aimed at business logic applications => domain
- Reusable => reuse
- base architecture => interaction, skeleton

Thus, Spring is a framework.

7.1.3 Comparing apples and oranges

Are these two frameworks even comparable? Spring is a J2EE framework while Blueprints is more of a set of guidelines for developing J2EE applications. Although as determined in the previous two sections, they are both frameworks.
Both frameworks build a 'layer' on top of the J2EE specification, although the layer that Spring builds is 'thinker' than the Blueprints layer. Blueprints still uses some aspects of J2EE without adding any layer, while Spring allows this (using EJB beans), but recommends the usage of the provided abstraction layer.
We do not evaluate the J2EE component in Spring, so we must also not evaluate the J2EE component in Blueprints.

7.1.4 Versions

For Sun Blueprints the following versions were used:

- Java EE 1.3
- Pet store demo 1.3.1.02

For Spring the following versions were used:

- Spring framework 2.0.4 (The base framework, extra plug-in modules are not evaluated)
- Spring reference manual 2.0

7.2 Is the change of framework possible

As mentioned in chapter 6, the first question is if both frameworks cover the same vertical domain and are applicable to the current application. To answer this question, we can use the documentation for both frameworks.
From the Spring reference manual [Rsp07] the following domain can be derived:
"Java applications (a loose term that runs the gamut from constrained applets to full-fledged n-tier server-side enterprise applications)"
Sun Blueprints follows the J2EE specifications, under the documentation [Fbl07] for these J2EE specifications we find:
"A set of coordinated specifications and practices that together enable solutions for developing, deploying, and managing multi-tier server-centric java applications."
Just from the definitions, we can infer the following characteristics:

- Java based
- Multi tier(n –tier) applications
- Client/server
- Scalable (fully fledged/server centric)
- Based upon the J2EE specifications

These hold for both definitions, so we can answer **yes** to **RQ-2**. Both frameworks cover the same application type. Thus, we can now continue with the next step and analyse the frameworks in more detail.

7.3 Horizontal domain iterations

Next, we prioritized the domains, this in order to ensure the most important aspects are covered first. For the prioritizing, we asked one of the application creators to prioritize the technical domain list. As mentioned before Everest creates applications that are focussed at the banking and insurance sector. These sectors are liable to rapid and frequent changes as well as large quantities of users. For these reasons, domains related to change are high on the list. The horizontal domains on a scale from one to ten are rated as following:

Technical domain	Priority
Customisability	1
Modifiability	2
Scalability	3
Security	4
Performance	5
Persistency	6
Testability	7
Separation of concerns	8
Availability	9
Interoperability	10

Figure 11: Prioritised domains for the case study

This gave us an order in which to compare the domains. Iteration has been chosen to ensure the most important domains are compared first.

7.3.1. Extracted facts customisability

This section will list the extracted facts for both frameworks.

Step 1, feature extraction
The feature diagram for Spring can be found in addendum B, the diagram for Blueprints in addendum C. The execution method for both frameworks slightly differ, this will be shortly discussed below.

Step 1, Spring, feature extraction
The Spring diagram has initially been composed by using the reference manual [Rsp07]. After its initial creation, we asked the creators of the Spring framework to provide some feedback, which was given and resulted in several communications back and forth and several changes to the diagram. Next, the packages structure of the framework has been analyzed to confirm that all features in the diagram are actually present in the source code or included libraries. The diagram evolved during the first two steps, after that it remained relatively the same.

Step 1, Blueprints, feature extraction
For Blueprints, the process was different. There is no skeleton source code available for Blueprints. Blueprints comes in the form of patterns and guidelines. These guidelines and patterns can be found in the book [Kas02], which is also available online at [Blr07]. However, the skeleton code is not distributed separately. There is a sample application available, that was created by the engineers of Blueprints and follows their guidelines. Blueprints also relies heavily on the J2EE specifications [J2E07]. Blueprints can be seen as an extension to the J2EE specifications, which means that in the feature diagram features from both J2EE in general and Blueprints will be displayed. However, only the features that Blueprints add count for Blueprints. If we would add the features from J2EE, we would be comparing J2EE + Blueprints, not just Blueprints. The features are displayed however, to leave no gaps in the features, since some rely on base code from J2EE. Spring does this as well, but created layers above each J2EE feature so no J2EE feature are visible. The initial version was created by examining the reference guide at [Blr07] and several other online resources. After this, the diagram was discussed with employees within Everest who have experience with the pet store application. Furthermore, the diagram has been further refined in the second step when the package structure of the demo application was examined in detail.

Note that we do not claim completeness for correctness for the features listed as features from J2EE. These features are nearly added to provide a more cohesive picture.

Step 2, abstraction types
From most of the package names, the feature was directly derivable. For example, *org.framework.aop.aspectj*, the resulting feature is aspectJ under AOP support in the tree. For other packages the JavaDoc and reference guide have been used. The packages to supply to the tools have been based upon relevance, mainly samples and tests have been left out. These do not influence frameworks features.

The packages used for Spring are:

- /aspectJ/src
- /dist
- /lib
- /src
- /target
- /tiger/src

The packages for Blueprints are:

- /src/waf
- /src/components/encodingfilter
- /src/components/mailer
- /src/components/servicelocator
- /src/components/signon
- /src/components/uidgen

This list has been composed by taking the reusable component from both frameworks. For Spring we only had to remove tests and mock code. For Blueprints we consulted the employees of Everest to clarify which packages where reusable code and which belonged exclusively to the pet store application.

These metrics were gathered using the following parameters on the tools:
javancss.Main -recursive -gui –all PATH
jdepend.xmlui.JDepend -file jdepend.xml PATH
Where path are the packages mentioned above.
The sources need to be compiled as well, make sure that only the given path is compiled to avoid tests and mock code in the packages, which could change the metrics considerably.

JavaNcss has a save function to store the package information, which is all displayed. On the xml file, an XLST [Xsl07] transformation has been used to filter the data.

The data for Spring can be found in addendum D, the data for Blueprints are in addendum E.

From the data in Addendum C, we can now determine the types of abstraction layers for each composite feature are. We can do this by using the table and performing the following steps:.

Determine which packages have the 'top' feature as feature in the table column.
From these packages filter the packages that exhibit the following criteria discussed in the research method:

- Abstractness ('A') is **higher** than average

Alternatively, the package exhibits the following characteristics

- Methods to NCSS is **lower** than average
- Method to JavaDoc is **higher** than average
- Package CE is **lower** than average

From observation of the package structure of Spring, we can conclude that the support package appears almost consistently trough the code that hints us that this package name could be an indication of an abstraction layer.
In addition, the root package often holds the exceptions for the entire package, which also hints to an abstraction layer.

An exempt is made for the following packages:

- Annotation packages, these packages often fit the criteria, but this is because of the nature annotations, no concrete classes.
- EJB packages, EJB works by defining two interface classes and 1 implementation class, which makes the package two third abstract by default, EJB packages are only counted if they are 100% abstract.

Next, we take the children below the current feature and determine if they use the abstraction layer.
If they do we have an open abstraction, if they do not it is closed. By using we mean does the 'top' package or packages use (depend upon) the identified abstraction packages. For this, we can use the report generated by JDepend.

We will discuss this here completely for the AOP package tree, for the other packages only the results are listed.
The packages listed as AOP are:

- aop
    - aop.config
    - aop.support
        - aop.support.annotation

The packages left after the filter:

- aop
    - aop.support

The features 'below' AOP are:

- AspectJ
- Spring/proxy AOP

We can now take the 'top' packages for these features below AOP, which are:

- aop.aspectj
- aop.framework
- aop.scope
- aop.target

All these packages depend on the identified abstraction packages.
We can now conclude that AOP has an open abstraction because it exhibits both characteristics mentioned before:

- Abstraction layer
- All children communicate through the abstraction layer (depend upon)

For the remaining features, we only list the results.

Features	**Open/closed**	**Comment**
AOP support	Open	
Proxy/Spring AOP	Closed	No abstraction layer could be identified.
Bean factory	Closed	There is an abstraction layer, but not all children depend on it.
Application context	Closed	No abstraction layer could be identified.
Spring container	Open	
DAO	Open	

ORM	Closed	There is an abstraction layer, but Ibatis does not use it, the other do.
Remoting	Open	
Scheduling	Open	
Dynamic language support	Open	
Transaction management	Closed	There is an abstraction layer, but Ibatis does not sure it the others do.
Content generator	Closed	No abstraction layer could be identified.
View abstraction	Closed	There is an abstraction layer, but content generator does not use it the other do.
MVC controller	Closed	No abstraction layer could be identified.
Spring MWC	Open	
Excel	Open	Abstraction in 1 package, implementation in libraries.
Enterprise utilities	Closed	The individual features are present and scattered, but no abstraction layer can be identified anywhere.
Configuration	Closed	Only the features for loading the type of configurations are present, no abstraction layer can be found.
JTA	Closed	No code package found, included as a library
Resource files	Closed	No code package found, most likely hidden in a utility class, which indicates no abstraction layer.

Figure 12: Abstraction layer types Spring

The abstraction layer for Blueprints are:

Composite feature	Open/closed	Comment
MVC controller	Closed	No abstraction layer could be identified.
Security	Closed	No abstraction layer could be identified, no common code for security, only implementation
Enterprise utilities	Closed	No abstraction layer could be identified.
Web container	Closed	No abstraction layer could be identified, no common code for web container
View abstraction	Closed	No abstraction layer could be identified, no common code for web container
Template engine	Closed	No abstraction layer could be identified.

Figure 13: Abstraction layer types Blueprints

Step 3, customisation identification

The results for Spring per feature can be found in addendum F, the results for Blueprints in addendum G.
Note that a distinction is made between a features that were customisable by default, and features to which the framework adds customisability. We are interested in the last category since these add value to customisability of the framework.

Framework	Spring	Blueprints
Number of total features	74	21
Number of mandatory composite non alternate variant features (Skipped)	10	8
Number of features candidate for customisability (figure 11)	64	13
Number of features customisable	15	4
Number of features customisable due to framework	11	4
Number of features pure customisable	11	0
Number of features mixed customisable	0	4
Number of optional features	25	3
Number of optional features which can be enabled per customisability	24	2
Number of optional features which can be enabled pure customisable	24	2
Number of optional features which can be enabled mixed customisable	0	0
Number of alternate variants	7	1
Number of alternate variants which can be enabled by customization	7	1
Number of alternate variants which can be enabled by pure customisable	7	1
Number of alternate variants which can be enabled by mixed customisable	0	0

Figure 14: Summary of what can be customised

Step 4, tool support
Both frameworks of no tools support for editing configuration files.

Step 5, ease of change
The results for Spring per feature can be found in addendum H, the results for Blueprints in addendum I.
The step type can hold two values:

- Enable the feature
- Change the feature

Framework	Spring	Blueprints
Number of configuration points	15	5
Number of enables paths	33	3
Average enable step length	1,06	1
Number of change paths	35	9
Average change step length	1,14	1,56

Figure 15: Summary ease of change

Note that we no longer use the step type in these figures. We are no longer interested in if the steps are source code or configuration related. We are only interested in the step required to enable or change a features in general. We already identified these features as either pure customization or a mix of customization and source code before, this is no longer relevant for the ease of since all steps have to be performed.

The averages are calculated by divided the total step length of all enable paths and divide them by the total number of enable paths. The same applies to the customization average. Where changing is considered anything with step type changing, thus ignoring the difference between customisation and modification. Essentially, we are determining the easy of change. We use this approach because due to the use of mixed steps, it becomes very hard to separate the two. Since the two terms are so related, we decided to leave them coupled and determine the easy of customization / modifiability.

7.3.2.1 Validation

The Following data has been collected for validation:

	Spring	Blueprints	Comment
1 Lines of new code	5	1	
Source code lines	0	1	
Non source code lines	5	0	
1 Time taken in minutes	5	10	In Spring this could be achieved via advice at the bean configuration xml, for blueprints this was an extra line of code in the request processor
1 Steps	1	1	
2 Lines of new code	4	5	
Source code lines	0	1	
Non source code lines	4	4	
2 Time taken in minutes	5	45	Blueprints required changes in two locations, web.xml and the templateServlet, the latter was hard to locate. Spring only required changes in the web.xml
2 Steps	1	2	Excluding the creation of the error page itself
3 Lines of new code	5	15	Excluding the mail sending code
Source code lines	5	15	
Non source code lines	0	0	
3 Time taken in minutes	15	45	Locating the security concepts in both frameworks was relatively easy (SignonFilter, SignonAction)
3 Steps	1	2	For Blueprints, a subclass had to be created, because if we changed the code directly we would break the separation of concerns principle. In Spring, the singon class was purely for that purpose and could be changed

Figure 16: Customisation validation sample

7.3.2 Extracted facts modifiability

This section will list the extracted facts for modifiability.

Step 1, extension points:
Recapping the data collected in step two for customizations

	Spring	Blueprints
Composite features	20	6
Number of closed	12	6
Number of open	8	0

Figure 17: What can be modified

Step 2, finding features:
We extracted the following data

	Spring	Blueprints
Average abstractness	0,29	0,24
Average afferent coupling	5,83	1,75
Average efferent coupling	7,12	1,54
Average instability	0,69	0,44

Figure 18: Modularity metrics

The complete results for these metrics can be found in addendum J for Spring and in addendum K for Blueprints.
Side note here is that only the packages should be compiled for Spring that we listed as the path 5.3.1.3 step 2. This because Spring uses different directories for samples, tests etc, but when compiled they end up in the same directories as the source and thus adding classes to the source which we are not interested in.

Another problem is that JDepend works on the class files, while Cobertura works on the class files and source files, minor discrepancies can occur while merging the results that have to be resolved by hand. These discrepancies can be counting empty directories, missing source code files, only class files available etc. However, these can be resolved by comparing the extracted package structures and file counts for both tools.

Step 3, understanding features
For the coding standard, we used a sample group of four files for each application. We determined of these files used a noticeable naming and coding standard.
Files were selected at random, but rejected if they did not exhibit the following criteria:

- At least ten lines of non comment code
- The file should be in the 'main' package structure (org.springframework / sun.j2ee.blueprints)

The files used for Spring were:

- Beanfactory (package org.springframework.beans.factory)
- IncorrectResultSizeDataAccessException (package org.springframework.dao)
- OpenSessionInViewInterceptor (package org.springframework.orm.hibernate.support)
- WebUtils (package org.springframework.web.util)

The files used for Blueprints were

- URLMappingsXmlDAO (package com.sun.j2ee.blueprints.waf.controller.web)
- I18nUtil (package com.sun.j2ee.blueprints.waf.util)
- EncodingFilter (package com.sun.j2ee.blueprints.encodingfilter.web)
- ServiceLocator (package com.sun.j2ee.blueprints.servicelocator.web)

For both frameworks the result was the same; both frameworks appears to have had naming and coding conventions.
We used the following commands for complexity:
cobertura-instrument.bat --destination d:\tmp PATH
cobertura-report.bat --basedire PATH --destination d:\tmp --format xml
Again and XSLT has been used to filter only the relevant data.

We extracted the following data for complexity

	Spring	Blueprints
Average cyclomatic complexity	1,82	2,09
Maximum cyclomatic complexity	6	5,25
Minimum cyclomatic complexity	0	1
Coding /naming standard used	Yes	Yes

Figure 19: Complexity metrics

The complete results for the complexity metrics for Spring and Blueprints can be found in respectively addendum J and K

Step 4, feature dependencies
We extracted the following data for the impact of a change

	Spring	Blueprints
Maximum efferent coupling	30	11
Minimum efferent coupling	0	0
Average efferent coupling	7,12	1,54
Packages below average efferent coupling	120	19
Packages above average efferent coupling	70	9

Figure 20: Impact of change

The complete data for the dependencies upon a certain package from outside the package is called efferent coupling and is listed in addendum J and K.

Step 5, ease of change
As mentioned before, a combined ease of customization and modifiability has been performed, the result can be found in figure 15.

7.3.2.1 Validation

The Following data has been collected for validation:

	Spring	Blueprints	Comment
1 Lines of new code	1	1	
Source code lines	1	1	
Non source code lines	0	0	
1 Time taken in minutes	5	5	Encoding filter class in Blueprints and Spring
1 Steps	1	1	
2 Lines of new code	2	4	Default requestprocessor for Blueprints, class declaration, default process request, call to log at beginning and end, all steps 1 line. For Spring the 2 lines were added the Transaction manager
Source code lines	2	2	
Non source code lines	0	0	
2 Time taken in minutes	10	15	
2 Steps	1	1	Excluding the log code itself, only the call to the log functionality is counted.
3 Lines of new code	14 + 6*	7 + ~3*	Excluding the interface pages themselves **(1)** Spring, **(2)** Blueprints
Source code lines	13 + 1*	7	
Non source code lines	1 + 5*	~3*	
3 Time taken in minutes	90	45	Adding this with Spring took considerable longer, the number of files to change and to locate them took long
3 Steps	6	3	

Figure 21: Modifiability validation sample

(1)

- 2 lines in the AccountDOA class (getter/setter definition)
- 2 lines to fill the attribute in SqlMapAccountDao class
  - 5 lines to make a of action mapping with setter and getter for user in the action mapping super class
  - 4 lines to overwrite findforward with the requested URL retrieval and adding the user role to it, in the action mapping super class
- 1* line per ActionMapping to extend the new super class in each existing action mapping class
- One line to define the attribute in account xml
  - One line per jsp page to include the user from the session in the struts config xml
  - Four lines per jsp page in the struts config xml

Open bullets indicate that the two actions are in the same file, hence the six total for Spring.

(2)

- Five lines for the userEJB class (getter/setter + attribute)
- Two lines for adding the user to the screen retrieval. In the template servlet class
- ~3* per screen definition in the screen definition xml. (depends on the number of elements in the template we used top, bottom, content frame)

8 Case study analysis

Now that we gathered all the facts, we can analyze and interpret them. We will start with customizability followed by modifiability.

8.1 Customisability

This section will analyze and interpret the results for customisability.

8.1.1 Analysis

We can now compare the extracted facts for both frameworks and analyse the results, we will do this step by step. Afterwards we will discuss the validation.

Step 1, feature extraction

When we look at the features diagrams for both frameworks, we can make a number of observations:

- Spring offers more features than Blueprints (64 to 13)
- Spring offers more choice in alternate variation (7 to 1)
- Spring offers more optional features than Blueprints (24 to 2)

Step 2, abstraction types

This step does not offer us much in terms of customisability, the reason this step was executed was to determine variants and their type. This step was also used as validation for the features inferred in step 1. Observations for this step are:

- Spring offers more open abstractions (8 to 0)

Step 3, customisation identification

In the third step, we determined if a feature achieved its customisability due to the framework or with help of the frameworks. We also identified if the customisability was pure or if it was a mix of customisability and source code. Observations in this step are:

- Blueprints offers a higher rate of features that are customisable (31% to 23%)
- Spring offers a higher pure customization rate (100% to 0%)
- The rate of optional features that can be enabled by pure customization is equal (100% to 100%)
- The rate of alternate variants that can be enabled by pure customization is equal (100% to 100%)

Step 4, tool support

In this step we examined the easy of change, observations for this step are:

- Spring offers more configuration points (15 to 4)
- Blueprints offers a marginally lower enable step length (1 to 1,06)
- Spring offers a lower step length for customization (1,14 to 1,56)
- The configuration point versus feature relation is equal (1 to 1)

Step 5, ease of change

In step five, we determined the tools which the framework offers to assist with customisability and configuration, observations we can make in this step is:

- Both frameworks offer no tools to assist with configuration (0 to 0)

8.1.2 Validation

This section will discuss the validation of the performed research.

The result for the scenarios is:

Total	**Spring**	**Blueprints**
Lines of new code	14	21
Source code lines	5	17
Non source code lines	9	4
Time taken in minutes	25	100
Steps	3	5

Figure 22: Customisability validation summary

These results confirm our data, since Spring performs better on average in all three criteria for this sample group. Less lines of code, more non-source code lines, less time taken and fewer steps performed. Additionally it shows that changes in Blueprints are more often source code changes than changes in Spring. Spring could accommodate two out of three changes without editing source code, while Blueprints could do that for none.

Why are the steps and/or number of steps different from those mentioned in appendix H and I? This is because they list steps to customize the component; it is possible that the specific customisations we picked do not map 100% to the customisation that can be done by performing these steps.

8.2 Modifiability

This section will analyze and interpret the results for modifiability.

8.2.1 Analysis

We can now compare the extracted facts for both frameworks and analyse the results, we will do this step by step. Afterwards we will discuss the validation.

Step 1 extension points
In this step, we identified the points in the code that are open to new features or the removal of features. As mentioned before, Spring offers far more features; it is not surprising that there are more open variants in Spring than in Blueprints:

- Spring offers more open abstractions than Blueprints (8 to 0)

What is awkward is that ratio of open composite features versus closed:

- Spring offers a far better ratio of open versus closed composite features (45% to 0%)

Step 2, finding features
In this step, we gathered information about the modularity of both frameworks, the following facts can be observed:

- Spring offers a slightly higher abstractness ratio, which could indicate that Spring follows the rule of 'programming to an interface' better than Blueprints (0,29 to 0,24)
- Blueprints offer a far lower average afferent coupling than Spring (1,75 to 5,83)
- Blueprints offers a far lower average efferent coupling than Spring (1,54 to 7,12)
- Blueprints offers a lower average instability than Spring (0,69 to 0,44)

Step 3, understanding features
This step gathered information about the understandability of the framework, primarily the readability and the following of naming and coding conventions. The following data was extracted:

- Both frameworks appear to use and follow a naming and coding standard
- Spring offers a lower cyclomatic complexity than Blueprints (1,82 to 2,09)
- Blueprints has a lower maximum cyclomatic complexity than Spring (5,25 to 6)

Step 4, feature dependencies
In this step, we gathered information about the impact of a change and the following data was extracted:

- Blueprints offers a much lower average efferent coupling (1,54 to 7,12)
- Blueprints offers a far lower maximum efferent coupling than Spring (11 to 30)
- Blueprints offers a slightly lower percentage of packages above the average efferent coupling than Spring (32% to 37%)

Step 5 ease of change
This step was performed in the customisation phase and we will recap the data that is relevant to modifiability.

- Blueprints offers a marginally lower enable step length (1 to 1,06)
- Spring offers a low step length for customization (1,14 to 1,56)

8.2.2 Validation

The validation of step one and five have been discussed in the customisation section and will not be repeated here.

The results for the scenarios are:

Total	Spring	Blueprints
Lines of new code	23	13
Source code lines	17	10
Non source code lines	6	3
Time taken in minutes	105	65
Steps	8	5

Figure 23: Modifiability validation summary

These results are inline with what we expected. One scenario had equal results, another favoured Spring slightly and the last highly favoured Blueprints. Overall blueprints performs better on the modifiability scenario's

The last results can be explained due to the centralised nature of Blueprints, single points of control principle is very important for Blueprints.

9 Conclusion

This section will conclude from the previous analysis. First, we provide the overall conclusion, following the conclusions for customisability and modifiability. The last section contains feature work.

9.1 Overall conclusion

The overall conclusions can only be drawn for the two compared domains. In order to provide an overall conclusion, more horizontal domains will need to be compared. We can offer a partial conclusion after comparing these two domains. Overall Spring performs slightly better than Blueprints. This because customisability was given a higher priority than modifiability and Spring offer a higher customisability while blueprints offers a slightly higher modifiability.

9.2 Customisability

In this section, we will discuss what we can conclude from the previous sections. We will also discuss the research in customization in general, what blank spots did we leave and what feature work could be done in this area.

9.2.1 Conclusion

From the previous sections, we can conclude that Spring offers a higher customisability than Blueprints. We base this conclusion on the fact that Spring offers far more features to begin with. In addition, Spring offers more options in terms of alternate variants and optional features while blueprints consist of mainly mandatory features. Thirdly, Spring offers a more centralized customisability; the number of steps that needs to be taken in order to customise a feature is much lower than with Blueprints.

Blueprints has a lower number of configuration points. However, if we hold the number of configuration points versus the number of customisable features / optional features in general, Spring yields a more centralized result. Blueprints offers a higher customisability when comparing the total number of feature to the number of features that can be customised. Nevertheless, most of this Blueprints customisability is mixed, while most Spring customisability is pure.

Therefore, our conclusion is that of these two frameworks, Spring offers a better customisability than Blueprints for the versions mentioned in chapter 7.

Spring offers a better customisability than Blueprints.

9.2.2 Discussion

This section will discuss a few (intentionally) gaps in our current research.

A subject of concern is trying to pinpoint a feature to a package or library in the code. This is a bit short sighted, it is possible that features span across multiple packages not in the same package tree. There is also a possibly than one package holds several features. We only limit a package to one feature. The reason we chose this approach is that we found it worked rather well in practise to pinpoint a single feature to a single package. This could be due to the design of the current frameworks under examination.

Another concern is the measurement if a feature is customisable. We defined this that if a feature had a customise step, it is customisable. We never measure the degree in which this feature is customisable. For example if we take a random spreadsheet product and the only part that can be customised is the title of the spreadsheet, our method calls this customisable. That we cannot customise the data, layout, column, colours, etc, is not taken into account. The reason we chose this approach is our scope, we want to be able to answer several domains and not just the domain of customisability. The domain of customisability alone is enough to create an entire thesis about, but this thesis is about comparing frameworks and not just customisability. Secondly we also feel that we are answering the question how customisable is the feature and not the framework itself, although the features customisability does add to the frameworks customisability.

A third concern is the rule that all variants should use the abstraction layer. If there are twenty variants and nineteen use the abstraction layer and one does not, we call the variant closed. This rule can be seen as strict, but we believe that allowing even one variant not to use the abstraction layer break the abstraction all together.

A large concern the basis behind what is source code and what is not. Throughout this thesis, we consider source code to be the code written in the general programming language of the framework. In this case Java code. The question can be raised if this is valid, are the xml files associated with the framework source code as well?

Another concern is that we did not factor in documentation in any way. There are several reasons for this, measuring the documentation and how long it takes for someone to find it, is different for each person, the measurement we have given are all static measurements and do not vary no matter who takes them. For documentation, this is different, so this has been left outside the scope. However, for determining what has to be done to customise a component we used the documentation.

A concern that can be raised is in step four, we also determined if a component can be changed by modifying the source code. In theory, **any component can be customized by changing the source code**. Why are only a few listed? For this, we used the reference guides as oracle. If the reference guide mentioned options to change to component by changing source code, we added the option. The underlying principle is that the reference guide has been created by the frameworks creators, if they intended this point in the framework to be changed, they will mention this in the reference guide, if they did not intend the framework to be changed there, they will not mention it.

Last concern that can be raised is the appearance of doubles in the feature diagram, for example, toplink in the Spring diagram. The reason for this is that the feature is used in a different perspective and therefore we found it justified to mention the feature twice. It is entirely possible to use this feature at only one part in the framework and not use it the second time. The feature is used in a different context and therefore we mentioned it twice.

9.3 Modifiability

In this section, we will discuss what we can conclude from the previous sections. We will also discuss the research in modifiability in general, what blank spots did we leave and what feature work could be done in this area.

9.3.1 Conclusion

From the previous sections, we can conclude that the complexity of Blueprints and Spring is almost equal. However, the coupling both, efferent and afferent is much lower for Blueprints. This also results in a lower instability for Blueprints. However, Spring offers points in the code where new functionality can be added, and Blueprints does not. The complexity is near equal. In addition, adding functionality will most likely be done by the frameworks developers, and not by the application developer. Because if many changes have to be made in functionality, it is far more likely, a different framework will be chosen. Small changes in functionality of the existing features are far more likely to done by application developers.

Therefore, our conclusion is that of these two frameworks, Blueprints offers a slightly better modifiability than Spring for the versions mentioned in chapter 7.

Blueprints offers a slightly better modifiability than Spring.

9.3.2 Discussion

This section will discuss a few (intentionally) gaps in our current research.

A concern is that we left the human aspects of understandability out of scope. We can count control structures and observe the coding style, but the effect of these on the understandability can differ from person to person. This aspect of understandability can most likely only be tested with the use of a sample group who grades the understandability of several files of both frameworks. Due to our scoping and this being only one of the factors, we decided not to investigate this further.

Another concern is the potential overlap of modifiability with separation of concerns and testing. Modularity is also an important component of separation of concerns. The last part of assessing the impact of a change is testing the change and sees if the application still passes the tests. This overlap is something that will always remain; we cannot eliminate this.

9.4 Feature work

Although all of the topics mentioned under discussion can be seen as feature work, we highlight four which we find most important.

The most obvious feature works is the completion of the other domains, the other horizontal domains could be compared in order to formulate a stronger conclusion.

One option for further work could be the longer study of readability we mentioned in the discussion section. A group of people could judge the understandability of a selection of the frameworks code and determine the understandability more accurately.

Another option is examining the impact of change more accurately. We identified the dependencies as the main factor in the impact of a change. We feel this is the right approach, although other factors like the architecture, contract documentation etc could be examined.

We mentioned the concern about if a feature is customable in the previous section, a way to improve on the current method would be to take this step further. Not just answer the question if a feature is customisable but also in what degree. A possibly option for this would be to measure all which sub-features can be customised and which sub-features cannot be customised. Based on those factors a more precise measurement can be given per feature instead of a *'yes'* and *'no'* answer.

10 Research evaluation

This section will reflect on the performed research. In addition, we will provide a recommendation to Everest in the context of a framework choice.

10.1 Method evaluation

This section will reflect upon the performed research. We will discuss if we answered the research question and what problems we did not solve.

10.1.1 Research questions

Recapping the research questions

RQ-1 *'Can application X conform framework A be converted to framework B"*

Sub questions that arise from the main question are:

- **RQ-2** *"Does the application type for both frameworks overlap?"*
- **RQ-3** *"How to determine which criteria to use for comparing the frameworks"*
    - **RQ-4** *"In which advantages and disadvantages will this conversion result"*
        - **RQ-5** *"Is the change of framework desirable"*

We answered **RQ-2** with yes. We answered **RQ-3** by creating a theoretical framework model in chapter 5.
We can only answer **RQ-4** and **RQ-5** partially. Due to the fact that we only compared two out of ten horizontal domains. In order to provide a more sound comparison more horizontal domains need to be compared.
This also results in a partial answer for **RQ-1.**

10.1.1.1 Literate questions

We answered all three literature questions. **LQ-1** in chapter 2, **LQ-2** and **LQ-3** in chapter 5.

10.1.2 Evaluation

Overall, the research advanced quite rapidly. None the less, we are displeased with the fact that we could only compare two horizontal domains. Thus offering a partial conclusion instead of complete. This is our greatest disappointment.

In more detail, the creation of a theoretical framework model was a good idea, although the overlap between the individual domains makes it very hard to compare them independently. This expressed itself in the two horizontal domains we compared; the easy of change could not be split into customisability and modifiability.

The separation of frameworks in horizontal domains was a good idea for reducing the complexity, although this also increased the total work considerably.

10.2 Application of the method

In this section we will reflect upon the method demands. We stated that the method must be:

- Repeatable
- Time demand
- Validation

Due to the input for most phases being source code, documentation or metrics based upon them, the results are repeatable. On the other hand however, we have the lab tests, which depend upon the executor. A different person could use a different solution to achieve the same results for the lab tests which could influence the results. Overall we can call this method repeatable.
On the subject of time demand however, we initially expected to be able to iterate through more domains. This was not the case. We spend nearly five weeks on two domains, which indicates that in an industrial setting, this method is not near complete. For a sound conclusion more than two domains will need to be examined and the time required is simply to much.
The validation aspect is reasonable covered, we listed the assumptions made for each domain in their respective sections for each step. We also listed if and how we validated these assumptions.

10.3 Recommendation

We can offer a recommendation based upon our research for Everest. Based upon these two horizontal domains we can conclude that a change to Spring is better in the context of these two domains. This however is from a pure technical content. We cannot answer the question if it is feasible in terms of manpower to rewrite the code to Spring.

If the change to Spring is made, we can make the recommendation to set strict guidelines for the usage of features. Spring offers many features, each extra feature providing a potential risk is maintenance. While Blueprints sets a very narrow possibility field, thus limiting possibilities, but also limiting maintainability problems because there are less components interacting with each other.
If the change of framework were made, it would be beneficial to set a strict list of features which can be used and which cannot be used. What the content of this set would be is a very different question that is outside the scope of this research.
Although after the thesis and interviews with Everest employees, the feeling remains that blueprints restrictive view is the best option, which suggests this method is no complete yet.

References

This section will list all resources used in the thesis. Three sections are defined:

- Papers, articles published on scientific platforms or thesis papers
- Books, Published books
- Online articles, articles from the Internet

All references use the same pattern for naming, the first three letters of the primary authors last name, and followed by the last two digits the reference was published. If this results in the same name being used twice, a letter suffix will be used. All references are alphabetically sorted.
For Wikipedia references a 'W' is used and the first two letters of the articles name followed by '07'. Wikipedia references are normally not considered scientific; however, they do represent the opinion of a large group of people.

Papers:

[Alp03] SR Alpert - Abstraction in Concept Map and Coupled Outline Knowledge Representations. - Journal of Interactive Learning Research - 2003 - questia.com

[Bac90] R Bache, M Mullerburg - Measures of testability as a basis for quality assurance - Software Engineering Journal, 1990 - ieeexplore.ieee.org

[Bat05] D Batory - Feature Models, Grammars, and Propositional Formulas - 2005 - hwswworld.com

[Bak06] Paul Bakker – Framework productivity measurement method – master thesis - 2006 – University of Amsterdam

[Bar75] Barlow Proschan - Statistical theory of reliability and life testing: Probability models – 1975 - Research supported by the Boeing Co., U.S. Air Force, and U. S. Navy; New York, Holt

[Bjr01] D Bjrner - Domain Engineering, A Prerequisite for Requirements Engineering Principles and Techniques - 2001 - imm.dtu.dk

[Bos97] J Bosch, P Molin, M Mattsson, P Bengtsson – Object-Oriented Frameworks-Problems & Experiences - 1997 - citeseer.ist.psu.edu

[Cah98] V.J. Cahill, P.A. Nixon, F.A. Rabhi - Object Models for Distributed or Persistent Programming - 1998 - Trinity College, University of Dubli

[Cas05] David Plans Casal – Advanced software development for web applications – 2005 – Luminas internet applications

[Cle97] PC Clements - Coming attractions in software architecture - Parallel and Distributed Real-Time Systems - 1997 ieeexplore.ieee.org

[Cle02] P Clements, L Northrop – Software product lines: practices and patterns - 2002 - Addison-Wesley

[Det01] R Deters - Scalability and information agents - ACM SIGAPP Applied Computing Review - 2001 - portal.acm.org

[Deu02] A van Deursen, P Klint –Domain - Specific Language Design Requires Feature Descriptions - 2002 - cwi.nl

[Dij76] E. W. Dijkstra – A discipline of programming – Prentice hall Englewood cliffs – 1976 - New York

[Dod98] Joint Chiefs of Staff. - Department of Defence Dictionary of Military and Associated Terms, as amended through December 7, 1998 – 1998 - (Joint Publication 1-02).

[Dol03] E Dolstra, G Florijn, M de Jonge, E Visser - Capturing timeline variability with transparent configuration environments - International Workshop on Software Variability Management, 2003 - archive.cs.uu.nl

[Du01] X Du, J Jiao, MM Tseng - Architecture of Product Family: Fundamentals and Methodology - Concurrent Engineering - 2001 - cer.sagepub.com

[Eis01] T Eisenbarth, R Koschke, D Simon - Feature-Driven Program Understanding Using Concept Analysis of Execution Traces – 2001 - doi.ieeecs.org

[Fay97] ME Fayad - Object-oriented application frameworks - DC Schmidt - Communications of the ACM - 1997 - csa.com

[Fay00a] ME Fayad - Introduction to the Computing Surveys' Electronic Symposium on Object - Oriented application frameworks - ACM Computing Surveys (CSUR) - 2000 - portal.acm.org

[Fay00b] ME Fayad, DS Hamu, D Brugali - Enterprise frameworks characteristics, criteria, and challenges - Communications of the ACM - 2000 - portal.acm.org

[Fay02] H. Mili, M. Fayad, D. Brugali, D. Hamu, D. Dori - Enterprise Frameworks: issues and research Directions - Software Practice & Experience Volume 32 - Issue 8 (July 2002) Pages 801-831

[Fro00] G Froehlich, HJ Hoover, PG Sorenson - Choosing an object-oriented domain framework - ACM Computing Surveys (CSUR) - 2000 - portal.acm.org

[Fro97] G Froehlich, HJ Hoover, L Liu, P Sorenson - Hooking into object-oriented application frameworks - Proceedings of the 19th international conference on Software engineering - 1997 - portal.acm.org

[Gur01] J van Gurp, J Bosch, M Svahnberg - On the Notion of Variability in Software Product Lines - Proceedings of the Working IEEE/IFIP Conference on Software engineering – 2001 - doi.ieeecs.org

[Guy91] Neil K. Guy – Community networks: building real communities in virtual space - Thesis - Simon Frazer University

[Guz06] T Guzewich, M Kent, A Pfeifer, K Shank - Software Architecture Case Study Ruby on Rails - 2006 - kyleshank.com

[Iso00] ISO/IEC 2000. Information technology-software product quality - Part 1: Quality model. ISO/IEC FDIS 9126-1:2000(E).

[Jia03] J Jiao, MM Tseng - Customisability Index Based on Information Content - Annals of the CIRP - 2003 - v4web.hmg.inpg.

[Joh97a] RE Johnson – Frameworks = (components+ patterns) - Communications of the ACM - 1997 - portal.acm.org

[Joh97b] RE Johnson - Components, frameworks, patterns - ACM SIGSOFT Software Engineering Notes - 1997 - portal.acm.org

[Joh98] RE Johnson, B Foote - Designing Reusable Classes - Journal of Object-Oriented Programming, 1988 - laputan.org

[Joh05] R Johnson - J2EE development frameworks - Computer - 2005 - ieeexplore.ieee.org

[Kno90] CA Knoblock - Learning abstraction hierarchies for problem solving - Proceedings of the Eighth National Conference on Artificial intelligence - 1990 - isi.edu

[Kru92] CW Krueger - Software reuse - ACM Computing Surveys (CSUR) - 1992 - portal.acm.org

[Lin02] F van der Linden - Software product families in Europe: the Esaps & Cafe projects - IEEE - 2002 - ieeexplore.ieee.org

[Loh84] JB Lohse, SH Zweben - Experimental evaluation of software design principles: an investigation into the effect of module coupling on system modifiability - Journal of Systems and Software - 1984 - portal.acm.org

[Mar00] Marcus Eduardo markiewicz, Carlos J.P. Lucena – Object oriented framework development – 2000 - portal.acm.org

[Mer05] M Mernik, AM Sloane - When and how to develop domain-specific languages - ACM Computing Surveys (CSUR) - 2005 - portal.acm.org

[Mcc76] TJ McCabe – A complexity measure - Proceedings of the 2nd international conference on Software engineering - 1976 - portal.acm.org

[Oss00] H Ossher, P Tarr - Multi-Dimensional Separation of Concerns and The Hyperspace Approach - 2000 - research.ibm.com

[Pin93] Pine BJ - Mass Customization: The New Frontier in Business Competition – 1993 - Hardvard business school press

[Pri06] Tim Prijn – Framework software quality analysis - Master thesis - 2006 - University of Amsterdam.

[Ras01] A Rashid - A Hybrid Approach to Separation of Concerns: The Story of SADES - 2001 - comp.lancs.ac.uk

[Rei02] R Reißing – Towards a Model for Object-Oriented Design Measurement - university of Stuttgart - 2002 - worte-projekt.d

[Rup96] A Rüping - Framework Patterns - Proceedings of the European Conference on Pattern Languages - 1996 - cs.wustl.edu

[Sai01] L Saitta, JD Zucker – A model of abstraction in visual perception - Applied Artificial Intelligence - 2001 - ingentaconnect.com

[Sim95] MA Simos - Organization domain modelling (ODM): formalizing the core domain modelling life cycle - Proceedings of the 1995 Symposium on Software reusability - 1995 - portal.acm.org

[Soi99] T Soininen, I Niemelä - Developing a Declarative Rule Language for Applications in Product Configuration - Proceedings of the First International Workshop on Practical aspects of declarative languages - 1999 - portal.acm.org

[Smi03] CU Smith, LG Williams - Software performance engineering - UML for real: design of embedded real-time - 2003 - portal.acm.org

[Sun05] XH Sun, Y Chen, M Wu - Scalability of Heterogeneous Computing – 2005 -doi.ieeecomputersociety.org

[Voa95] JM Voas, KW Miller - Software testability: the new verification - Software, IEEE, 1995 - ieeexplore.ieee.org

[Wes04] Time Westkamper – Architectural models of J2EE web tier frameworks - Master thesis - 2004 – University of Tampere

[Wey88] EJ Weyuker - Evaluating software complexity measures - IEEE Transactions on Software Engineering, 1988 - doi.ieeecomputersociety.org

[Zar04] Apostolos Zarras – A comparison framework for middleware infrastructure – University of loannina – 2004 – http://www.jot.fm

Books:

[Bas03] L Bass, P Clements, R Kazman - Software Architecture in Practice, second edition - 2003 - Addison Wesley

[Cza00] K Czarnecki, UW Eisenecker - Generative programming: methods, tools, and applications - 2000 - ACM Press/Addison-Wesley Publishing Co. New York, NY, USA

[Fay99] Mohamed E. Fayad, Douglas C. Schmidt, Ralph E. Johnson - Building Application Frameworks: Object-Oriented Foundations of Framework Design – 1999- Wiley computer publishing

[Jac97] Jacobson, Griss, Jonsson - Software Reuse: Architecture Process and Organization for Business Success - 1997 - Addison Wesley Longman

[Kas02] N Kassem, N Kassem, E Team - Designing Enterprise Applications: Java 2 Platform - 2002 - Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA

[Mcc77] JA McCall, PK Richards, GF Walters - Factors in Software Quality - 1977 – NTIS

[Mcc04] S McConnell - Code complete second edition - 2004 - bookpool.com

[Leo00] A Leon - A Guide to Software Configuration Management – 2000 - Artech House Computing Library - ercb.com

Online articles:

[Apa07] http://ant.apache.org/

[Aph07] http://httpd.apache.org/

[BlC07] http://www.sun.com/blueprints/0905/819-4148.pdf

[Blr07] http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/titlepage.html

[Blu07] http://java.sun.com/blueprints/enterprise/index.html

[Cob07] http://cobertura.sourceforge.net/

[Cor07] http://www.omg.org/gettingstarted/corbafaq.htm

[Fbl07] http://java.sun.com/javaee/overview/faq/j2ee.jsp

[JCo07] http://www.theserverside.com/news/thread.tss?thread_id=15228

[Jde07] http://www.clarkware.com/software/JDepend.html

[Jnc07] http://www.kclee.de/clemens/java/javancss/

[J2E07] http://java.sun.com/j2ee/j2ee-1_3-fr-spec.pdf

[Pcm07] http://www.pcmag.com/encyclopedia_term/0,2542,t=customisability&i=40596,00.asp

[Spc07] http://developers.sun.com/learning/javaoneonline/2005/coreenterprise/TS-7695.pdf

[Spr07] http://www.springframework.org/

[Tss07] http://www.theserverside.com/tt/articles/article.tss?l=SpringFramework

[Rsp07] http://static.springframework.org/spring/docs/2.0.x/reference/index.html

[Wmv07] http://en.wikipedia.org/wiki/Model-view-controller

[Xsl07] http://www.w3.org/TR/xslt

Addendum A: Framework technical domain recognition

This addendum describes how the definitions for the framework technical domains were formed.

A-1 Scalability

A definition found in the book [Bas03] for scalability is:
"The system should support variations in load without human intervention."
Another definition found [Det01] is:
"Scalability is the ability of a computer application or product to continue to function well as it (or its context) is changed in size or volume. Typically, the rescaling is to a larger size or volume."
Distribution can be defined as:
"A digital information system dispersed over multiple computers and not centralized at a single location." [Guy91]
From this, we can derive that distribution is an implementation of scalability, by distributing the workload over several machines, the workload is divided over multiple machines and more data can be processed than by using a single machine. Transaction management is vital to using distributed computing and can be seen as a basis requirement for distributed computing. Scalability is also mentioned as one of the characteristics for enterprise applications in [Fay00b]. For these reasons, scalability is the technical domain.

A-2 Persistency

In [Cah98] the following definition is used for persistency:
"Persistency refers to the characteristic of data that outlives the execution of the program that created it."
This description captures the most important characteristic, but another characteristic to identify persistency as a framework domain is the layer it creates between the data storage and the application logic itself. Persistency is related to performance, the type of persistence can have a great impact on the performance. However, persistency covers far more than just performance. In addition, the type of mapping used to change from the internal application structure to data storage is of great importance for persistence. For these reasons, persistency is the technical domain.

A-3 Security

The definition given for security in [Bas03] is the following:
"Security is the measure of the system's ability to resist unauthorized usage while still providing its service to legitimate users".
This definition directly lists some of the implementations for security, authentication due to providing service to legitimate users. The same applies to authorization; deny non-legit users total access and grant legit users access.
Also with increasingly larger application which support different types of users and the grow in Internet applications, security has become an increasing important topic. Another element that can help the security domain is logging. By logging all interactions, a security breach can be detected earlier or can be traced back to the problem later on.
For these reasons, security is the technical domain.

A-4 Availability

Availability is often expressed a percentage. This percentage is the correlation between the uptime and the downtime of a particular system. A definition in [Bar75] for availability is:
"Availability is the probability that the system is operating at a specified time t."
A term that is a synonym of availability is reliability. Two factors that influence the availability are error/exception handling and fault tolerance. If the system is designed with error/exception handling, it will be able to recover from minor errors automatically and therefore have a higher availability. Fault tolerance refers to, for example, the number of input faults the system is able to recover from. Both factors are implementations of availability, this is the reason availability itself is the technical domain.

A-5 Customisability

A definition for customisability is:

"Customisability is the characterized by the easy of a change rather than the extent of a change." [Jia03]

Customisability is defined in [Pcm07] as the following:

"The ability for software to be changed by the user or programmer."

However we feel that the last part is not right 'or programmer' should be removed, since modifying by a programmer is part of the modifiability. De definition we use for customisability is:

"The ability and ease for software to be changed by the user."

In the context of frameworks, the user is the developer of the application with the framework. Not to be confused with the end user of the application. The application that was created by the developer with the framework. In the context of frameworks, the user is the developers and the frameworks the software. The term tailorability is not widely used but is considered a synonym of customisability. On possible implementation of customisability is configuration. Application variables can be altered via configuration. This however is only on side of customisability; the options for customization must also be present in the code. For the reasons mentioned customisability the technical domain.

A-6 Interoperability

Interoperability is defined in [Dod98] as:

"The ability of systems, units, or forces to provide services to and accept services from other systems, units or forces and to use the services so exchanged to enable them to operate effectively together."

On of the techniques to accomplish interoperability is openness, of which transparency is a synonym. They both describe programming to interfaces where these interfaces are public knowledge. Another concept to achieve interoperability is using common object request broker architecture [Cor07]. This list of concepts and techniques in order to achieve interoperability supports that interoperability is the technical domain.

A-7 Performance

Performance is a term widely used in the field of software engineering, yet no standardized definition is available. Two definitions found in the literature are:

"Performance is largely a function of the frequency and nature of inter-component communication, in addition to the performance characteristics of the components themselves, and hence can be predicted by studying the architecture of a system." [Cle97]

"Performance is the degree to which a software system or component meets its objectives for timeliness. Thus, performance is any characteristic of a software product that you could, in principle, measure by sitting at the computer with a stopwatch in your hand." [Smi03]

None of these definitions mentions hardware related issues. This is acceptable in the context of this research, since hardware upgrades is covered by scalability. The two terms mentioned in the same row connection pooling and session handling are two implementation which have a large impact on performance, but these are not the only ones, caching for example is another issue. For these reasons, performance is the technical domain.

A-8 Separation of concerns

One of the most important principles of software engineering is separation of concerns. This principle was already acknowledged in 1976 by [Dij76]. The number of concrete definitions for separation of concerns is very low, despite the fact that the principle is very old. In [Oss00] separation of concerns is defined as:

"Separation of concerns refers to the ability to identify, encapsulate, and manipulate only those parts of software that are relevant to a particular concept."

Another definition in [Ras01] is:

"The separation of concerns principle proposes encapsulating cross-cutting features into separate entities in order to localize changes to these cross-cutting features and deal with one important issue at a time." [Ras01]

Where crosscutting features are features like logging and security. Features which 'cross' more then a single concern or component. From these definitions, we can infer that reducing complexity, increase modularity and reducing dependencies are all means by which separation of concerns can be achieved. For this reasons, separation of concerns is the technical domain.

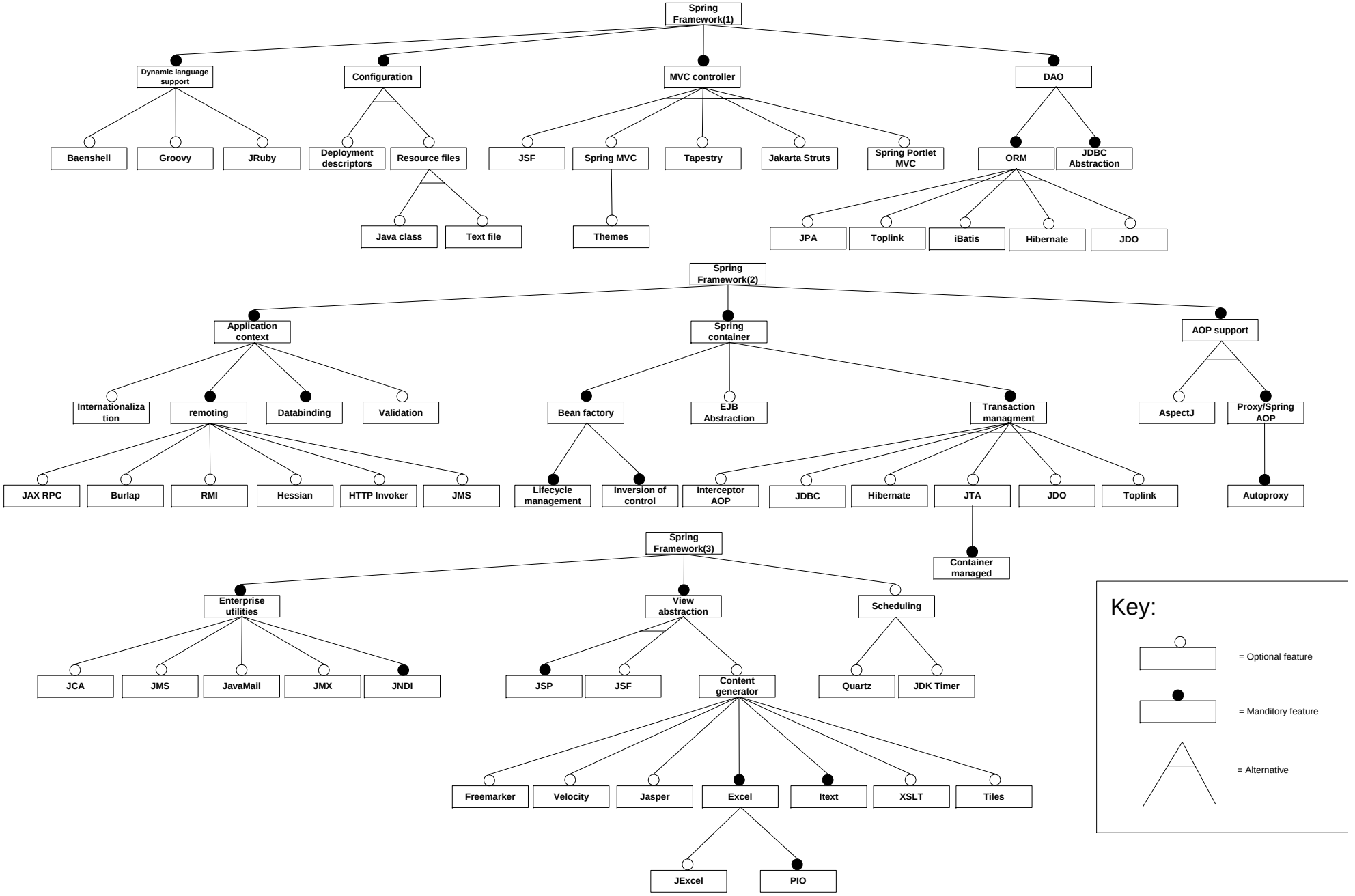
A-9 Modifiability

The definitions for modifiability, sometimes called maintainability, differ greatly. A definition found in [Guz06] is:
"Existing components in the system should be able to easily facilitate the incorporation of changes."
Another definition found in the ISO standard [Iso00] is:
"Maintainability is the capability of the software product to be modified. Modifications may include corrections, improvements or adaptations of the software to change the environment, and in requirements and function specifications."
Flexibility, modifiability and maintainability are considered synonyms. However, in [Mcc77] there is an exception to this rule:
"Maintainability is the effort required to locate and fix and error in an operational program."
"Flexibility is the effort required to modify an operational program."
For this thesis we will consider them synonyms, this is the reason flexibility has been removed. Extendibility is a part of modifiability that can be derived from the second definition, "… *Modifications may include improvements ..."*
Portability is a platform modification and can be derived from the second definition, "…*adaptations of the software to change the environment …"*
Internationalization can be seen as modifying the program to support more languages and is therefore covered by modifiability. For the reasons outlined here, modifiability is the technical domain.

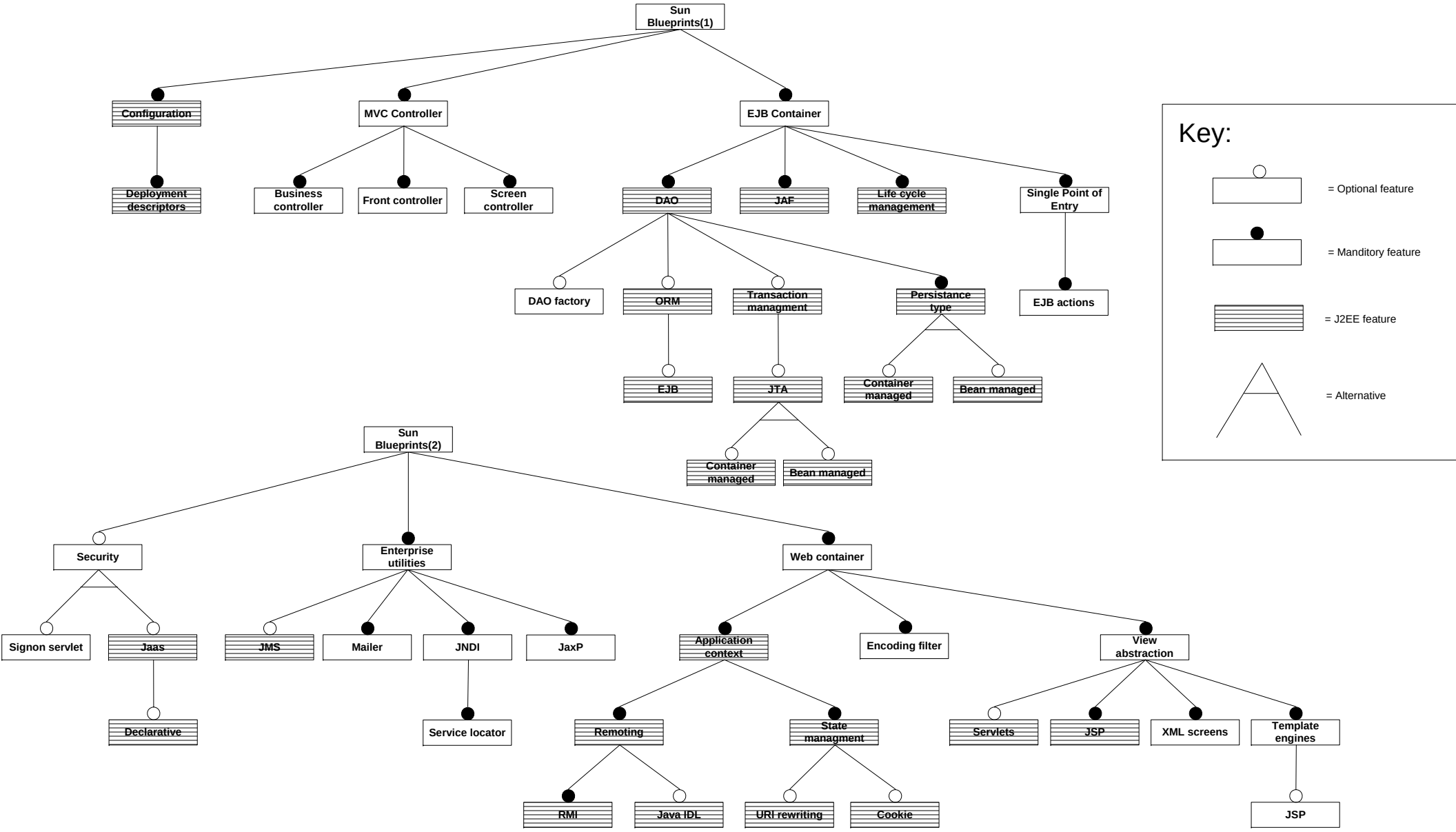
A-10 Testability

Testability is often defined as:
"The likelihood, possibly measured statistically, that the software will expose a failure under testing, if it is faulty." [Voa95]
This is however only one side of the medallion, the other side is less used, but important none the less:
"The degree to which it is easy for software to fulfil a given test coverage criterion." [Bac90]
Software testing is an important factor in quality insurance. Unit testing is only one of the possible tests that can be used to test a software product, others are integration tests, communications tests etc. For this reason, unit tests are a form of testing and testability itself is the technical domain.

Addendum B: Spring feature diagram



Addendum C: Blueprints feature diagram



Addendum D: Spring customisability metrics

	JavaNCSS				Jdepend					
P#	Class	Method	NCSS	Javadoc	Abstractness	Concrete	Abstract	Ce	Package	Feature
1	23	43	124	57	0,74	7	20	4	org.springframework.aop	AOP
2	22	177	1330	99	0,22	42	12	18	org.springframework.aop.aspectj	AspectJ
3	15	88	740	70	0,16	31	6	15	org.springframework.aop.aspectj.annotation	AspectJ
4	2	10	125	6	0,00	13	0	9	org.springframework.aop.aspectj.autoproxy	AspectJ
7	16	70	640	39	0,12	15	2	17	org.springframework.aop.config	AOP
8	23	185	1428	165	0,21	44	12	14	org.springframework.aop.framework	Proxy/Spring AOP
9	12	30	197	26	0,25	9	3	7	org.springframework.aop.framework.adapter	Proxy/Spring AOP
10	7	49	316	45	0,27	11	4	15	org.springframework.aop.framework.autoproxy	Autoproxy
11	3	9	77	7	0,33	2	1	6	org.springframework.aop.framework.autoproxy.target	Autoproxy
12	11	54	367	57	0,18	14	3	8	org.springframework.aop.interceptor	Interceptor
13	3	11	70	8	0,33	2	1	6	org.springframework.aop.scope	Proxy/Spring AOP
14	30	189	997	144	0,40	24	16	7	org.springframework.aop.support	AOP
15	3	10	58	9	0,00	3	0	3	org.springframework.aop.support.annotation	AOP
16	14	100	375	78	0,43	8	6	9	org.springframework.aop.target	Proxy/Spring AOP
17	3	16	64	13	0,67	1	2	3	org.springframework.aop.target.dynamic	Proxy/Spring AOP
18	33	239	1716	209	0,34	31	16	6	org.springframework.beans	Bean factory
19	1	1	19	2	1,00	0	1	2	org.springframework.beans.annotation	Bean factory
20	25	84	320	109	0,39	20	13	4	org.springframework.beans.factory	Lifecycle management / IOC
21	4	13	132	16	0,33	4	2	6	org.springframework.beans.factory.access	Lifecycle management / IOC
22	3	11	90	9	0,50	3	3	6	org.springframework.beans.factory.annotation	Lifecycle management / IOC
23	2	17	165	1	0,67	1	2	4	org.springframework.beans.factory.aspectj	Lifecycle management / IOC
24	38	278	1465	241	0,36	29	16	7	org.springframework.beans.factory.config	Lifecycle management / IOC
25	1	7	29	8	0,33	2	1	3	org.springframework.beans.factory.generic	Lifecycle management / IOC
26	22	100	344	85	0,29	17	7	6	org.springframework.beans.factory.parsing	Configuration
27	33	449	3206	329	0,27	32	12	8	org.springframework.beans.factory.support	Lifecycle management / IOC
28	4	13	76	16	0,50	2	2	3	org.springframework.beans.factory.wiring	Configuration
29	27	167	1591	148	0,30	39	17	12	org.springframework.beans.factory.xml	Configuration
30	20	82	561	71	0,00	20	0	3	org.springframework.beans.propertyeditors	Bean factory
31	8	69	326	71	0,20	8	2	6	org.springframework.beans.support	Bean factory
32	2	27	140	19	0,00	2	0	6	org.springframework.cache.ehcache	Caching(Noreference to this found anywhere)
33	15	35	88	50	0,72	5	13	5	org.springframework.context	Application context
34	4	15	95	17	0,00	4	0	8	org.springframework.context.access	Application context
35	7	21	98	19	0,25	6	2	6	org.springframework.context.event	Application context
36	3	10	44	12	0,67	1	2	1	org.springframework.context.i18n	Internationalisation
37	18	219	1291	176	0,19	22	5	13	org.springframework.context.support	Application context
38	25	147	1204	169	0,49	19	18	5	org.springframework.core	Spring container
39	2	4	42	5	0,67	1	2	1	org.springframework.core.annotation	Spring container
40	10	38	182	30	0,46	7	6	2	org.springframework.core.enums	Spring container
41	13	89	340	97	0,31	9	4	1	org.springframework.core.io	Spring container
42	8	50	411	56	0,50	4	4	3	org.springframework.core.io.support	Spring container
43	6	43	178	28	0,50	3	3	2	org.springframework.core.style	Spring container
44	6	15	73	22	0,18	9	2	2	org.springframework.core.task	Spring container
45	19	39	127	58	0,11	17	2	1	org.springframework.dao	DAO
46	2	10	68	5	0,00	2	0	12	org.springframework.dao.annotation	DAO
47	5	23	142	23	0,60	2	3	5	org.springframework.dao.support	DAO
48	7	43	260	38	0,29	5	2	8	org.springframework.ejb.access	EJB abstraction
49	5	6	46	5	0,20	4	1	8	org.springframework.ejb.config	EJB abstraction
50	7	20	96	27	0,88	1	7	5	org.springframework.ejb.support	EJB abstraction
51	1	2	8	3	0,00	1	0	0	org.springframework.instrument	AOP

52	9	46	248	39		0,10	9	1	3	org.springframework.instrument.classloading	AOP
53	2	8	39	4		0,00	2	0	3	org.springframework.instrument.classloading.glassfish	AOP
54	2	9	46	5		0,00	2	0	3	org.springframework.instrument.classloading.oc4j	AOP
55	1	8	61	4		0,00	1	0	2	org.springframework.instrument.classloading.tomcat	AOP
56	5	5	30	10		0,00	5	0	1	org.springframework.jca.cci	JCA
57	8	57	380	40		0,07	13	1	5	org.springframework.jca.cci.connection	JCA
58	6	36	188	34		0,62	3	5	4	org.springframework.jca.cci.core	JCA
59	2	19	60	12		0,50	1	1	5	org.springframework.jca.cci.core.support	JCA
60	4	22	92	25		0,50	3	3	4	org.springframework.jca.cci.object	JCA
61	3	21	90	14		0,00	3	0	3	org.springframework.jca.support	JCA
62	3	29	193	18		0,00	4	0	3	org.springframework.jca.work	JCA
63	8	24	94	31		0,00	8	0	1	org.springframework.jdbc	JDBC abstraction
64	35	248	1364	222		0,39	31	20	10	org.springframework.jdbc.core	JDBC abstraction
65	9	92	390	73		0,44	5	4	8	org.springframework.jdbc.core.namedparam	JDBC abstraction
66	4	23	60	16		0,50	2	2	3	org.springframework.jdbc.core.simple	JDBC abstraction
67	7	38	200	42		0,62	3	5	9	org.springframework.jdbc.core.support	JDBC abstraction
68	17	142	948	133		0,26	17	6	6	org.springframework.jdbc.datasource	JDBC abstraction
69	8	29	156	28		0,22	7	2	9	org.springframework.jdbc.datasource.lookup	JDBC abstraction
70	11	137	571	151		0,57	6	8	7	org.springframework.jdbc.object	JDBC abstraction
71	13	84	654	70		0,33	10	5	10	org.springframework.jdbc.support	JDBC abstraction
72	8	41	217	37		0,38	5	3	4	org.springframework.jdbc.support.incrementer	JDBC abstraction
73	8	45	335	35		0,24	16	5	5	org.springframework.jdbc.support.lob	JDBC abstraction
74	9	57	277	52		0,22	7	2	4	org.springframework.jdbc.support.nativejdbc	JDBC abstraction
75	4	135	372	123		0,50	2	2	1	org.springframework.jdbc.support.rowset	JDBC abstraction
76	14	20	78	19		0,07	14	1	1	org.springframework.jms	JMS
77	13	123	855	99		0,16	21	4	5	org.springframework.jms.connection	JMS
78	7	102	417	85		0,26	14	5	7	org.springframework.jms.core	JMS
79	1	7	27	7		1,00	0	1	2	org.springframework.jms.core.support	JMS
80	8	135	927	138		0,21	15	4	12	org.springframework.jms.listener	JMS
81	3	33	195	36		0,40	6	4	6	org.springframework.jms.listener.adapter	JMS
82	7	47	280	56		0,19	13	3	7	org.springframework.jms.listener.serversession	JMS
83	3	29	172	23		0,00	3	0	11	org.springframework.jms.remoting	JMS
84	2	21	145	22		1,00	0	2	4	org.springframework.jms.support	JMS
85	4	15	106	19		0,25	3	1	2	org.springframework.jms.support.converter	JMS
86	6	20	115	23		0,43	4	3	4	org.springframework.jms.support.destination	JMS
87	2	4	13	6		0,00	2	0	1	org.springframework.jmx	JMX
88	5	25	215	26		0,00	6	0	7	org.springframework.jmx.access	JMX
89	7	68	433	72		0,23	10	3	11	org.springframework.jmx.export	JMX
90	1	5	136	1		0,88	1	7	4	org.springframework.jmx.export.annotation	JMX
91	10	85	517	81		0,58	5	7	8	org.springframework.jmx.export.assembler	JMX
92	10	53	209	30		0,20	8	2	5	org.springframework.jmx.export.metadata	JMX
93	5	10	89	15		0,40	3	2	7	org.springframework.jmx.export.naming	JMX
94	4	7	44	11		0,50	2	2	2	org.springframework.jmx.export.notification	JMX
95	9	72	479	61		0,08	12	1	7	org.springframework.jmx.support	JMX
96	10	49	281	49		0,20	12	3	6	org.springframework.jndi	JNDI
97	8	58	229	27		0,38	5	3	2	org.springframework.mail	JavaMial
98	1	3	44	2		0,00	1	0	2	org.springframework.mail.cos	JavaMial
99	8	129	622	87		0,22	7	2	4	org.springframework.mail.javamail	JavaMial
100	1	6	11	7		1,00	0	1	0	org.springframework.metadata	Databinding
101	1	6	18	1		0,00	2	0	7	org.springframework.metadata.commons	Databinding
103	2	18	58	20		0,00	2	0	1	org.springframework.orm	ORM
104	18	278	1599	189		0,08	48	4	23	org.springframework.orm.hibernate	Hibernate
105	8	66	364	51		0,25	6	2	12	org.springframework.orm.hibernate.support	Hibernate
106	22	346	2026	250		0,07	63	5	30	org.springframework.orm.hibernate3	hHibernate
107	1	6	28	5		0,00	1	0	3	org.springframework.orm.hibernate3.annotation	Hibernate
108	10	73	400	59		0,20	8	2	19	org.springframework.orm.hibernate3.support	Hibernate
109	4	68	269	47		0,13	13	2	14	org.springframework.orm.ibatis	Ibatis

110	5	30	148	25		0,33	4	2	6		org.springframework.orm.ibatis.support	Ibatis
111	17	176	976	139		0,09	48	5	12		org.springframework.orm.jdo	JDO
112	3	21	120	19		0,33	2	1	9		org.springframework.orm.jdo.support	JDO
113	25	168	1172	128		0,32	32	15	15		org.springframework.orm.jpa	JPA
115	7	69	431	36		0,29	5	2	10		org.springframework.orm.jpa.persistenceunit	JPA
116	5	40	377	30		0,12	7	1	14		org.springframework.orm.jpa.support	JPA
117	8	36	292	22		0,08	11	1	11		org.springframework.orm.jpa.vendor	JPA
118	21	196	950	150		0,16	41	8	22		org.springframework.orm.toplink	Toplink
119	4	26	182	24		0,25	3	1	7		org.springframework.orm.toplink.support	Toplink
120	4	6	21	10		0,00	4	0	1		org.springframework.remoting	Remoting
121	9	44	315	33		0,11	8	1	14		org.springframework.remoting.caucho	Hessian / Burlap
122	8	66	320	64		0,38	5	3	12		org.springframework.remoting.httpinvoker	HHTP invoker
123	6	79	399	76		0,33	4	2	13		org.springframework.remoting.jaxrpc	JAX
124	1	10	71	10		0,00	1	0	4		org.springframework.remoting.jaxrpc.support	JAX
125	12	100	729	94		0,18	14	3	11		org.springframework.remoting.rmi	RMI
126	14	54	274	60		0,57	6	8	6		org.springframework.remoting.support	Remoting
127	3	4	14	7		0,67	1	2	2		org.springframework.scheduling	Scheduling
128	5	57	233	55		0,00	5	0	5		org.springframework.scheduling.backportconcurrent	Scheduling
129	5	46	191	41		0,00	5	0	8		org.springframework.scheduling.commonj	Common J
130	5	57	233	55		0,00	5	0	4		org.springframework.scheduling.concurrent	Scheduling
131	15	113	739	90		0,15	17	3	20		org.springframework.scheduling.quartz	Quartz
132	1	5	25	1		0,00	1	0	3		org.springframework.scheduling.support	Scheduling
133	5	44	169	39		0,00	5	0	5		org.springframework.scheduling.timer	JDK timer
134	3	9	20	12		0,80	2	8	3		org.springframework.scripting	Dynamic language support
135	2	14	130	11		0,20	4	1	6		org.springframework.scripting.bsh	Beanshell
136	2	6	78	6		0,00	2	0	7		org.springframework.scripting.config	Dynamic language support
137	2	11	76	9		0,50	1	1	5		org.springframework.scripting.groovy	Groovy
138	2	13	151	11		0,62	3	5	10		org.springframework.scripting.jruby	Jruby
139	4	30	269	22		0,00	5	0	16		org.springframework.scripting.support	Dynamic language support
140	0	0	8	0		1,00	0	1	0		org.springframework.stereotype	DAO
141	17	42	133	58		0,25	15	5	2		org.springframework.transaction	Transaction management
142	3	12	90	7		0,25	3	1	2		org.springframework.transaction.annotation	Transaction management
143	6	30	153	4		0,50	1	1	4		org.springframework.transaction.aspectj	Delcarative AOP
144	4	9	137	5		0,00	5	0	11		org.springframework.transaction.config	Transaction management
145	19	113	794	95		0,17	25	5	12		org.springframework.transaction.interceptor	Delcarative AOP
146	10	92	692	64		0,00	10	0	6		org.springframework.transaction.jta	JTA
147	17	163	832	159		0,68	6	13	4		org.springframework.transaction.support	Transaction management
148	1	7	31	8		0,00	1	0	2		org.springframework.ui	Content generator
149	3	5	12	8		1,00	0	3	1		org.springframework.ui.context	Content generator
150	4	13	108	10		0,25	3	1	3		org.springframework.ui.context.support	Content generator
151	4	29	179	25		0,25	3	1	7		org.springframework.ui.freemarker	Freemaker
152	1	11	74	12		0,33	2	1	3		org.springframework.ui.jasperreports	Jasper
153	5	28	234	23		0,20	4	1	13		org.springframework.ui.velocity	Velocity
154	26	338	2446	324		0,57	18	24	3		org.springframework.util	Utilities, not a feature
155	5	37	166	25		0,00	5	0	1		org.springframework.util.comparator	Utilities, not a feature
156	4	21	129	18		0,25	3	1	3		org.springframework.util.xml	Utilities, not a feature
157	18	217	748	148		0,47	10	9	3		org.springframework.validation	Validation
158	3	5	23	8		0,33	2	1	0		org.springframework.web	View abstraction
159	10	122	530	99		0,28	13	5	5		org.springframework.web.bind	View abstraction
160	7	25	166	31		0,40	6	4	7		org.springframework.web.context	View abstraction
161	11	63	290	49		0,50	6	6	7		org.springframework.web.context.request	View abstraction
162	19	115	593	90		0,16	16	3	17		org.springframework.web.context.support	View abstraction
163	9	48	281	53		0,30	7	3	10		org.springframework.web.filter	View abstraction
164	6	31	173	34		0,25	6	2	4		org.springframework.web.jsf	JSF
165	5	19	44	23		0,60	2	3	1		org.springframework.web.multipart	Multipart resolver
166	3	34	230	26		0,25	3	1	9		org.springframework.web.multipart.commons	Multipart resolver
167	2	21	173	16		0,00	3	0	7		org.springframework.web.multipart.cos	Multipart resolver

168	5	21	142	15		0,20	4	1	5		org.springframework.web.multipart.support	Multipart resolver
169	10	102	741	113		0,25	21	7	20		org.springframework.web.portlet	Spring portlet MVC
170	5	43	256	47		0,15	11	2	5		org.springframework.web.portlet.bind	Spring portlet MVC
171	15	100	508	66		0,35	11	6	13		org.springframework.web.portlet.context	Spring portlet MVC
172	14	92	523	86		0,25	12	4	10		org.springframework.web.portlet.handler	Spring portlet MVC
173	4	22	107	15		0,50	2	2	7		org.springframework.web.portlet.multipart	Spring portlet MVC
174	11	178	851	172		0,46	7	6	9		org.springframework.web.portlet.mvc	Spring portlet MVC
175	3	59	227	22		0,33	2	1	2		org.springframework.web.portlet.util	Spring portlet MVC
176	17	130	893	145		0,32	25	12	28		org.springframework.web.servlet	MVC controller
177	11	74	435	66		0,25	9	3	12		org.springframework.web.servlet.handler	MVC controller
178	4	21	99	15		0,17	5	1	9		org.springframework.web.servlet.handler.metadata	MVC controller
179	6	17	118	14		0,17	5	1	6		org.springframework.web.servlet.i18n	MVC controller
180	17	176	769	180		0,42	11	8	10		org.springframework.web.servlet.mvc	Spring MVC
181	7	52	347	56		0,29	5	2	9		org.springframework.web.servlet.mvc.multiaction	Spring MVC
182	1	8	66	9		0,00	1	0	8		org.springframework.web.servlet.mvc.support	Spring MVC
183	2	8	32	9		0,50	1	1	2		org.springframework.web.servlet.mvc.throwaway	Spring MVC
184	6	92	480	96		0,50	3	3	13		org.springframework.web.servlet.support	MVC controller
185	10	56	367	51		0,20	8	2	6		org.springframework.web.servlet.tags	MVC controller
186	22	238	1175	231		0,22	21	6	8		org.springframework.web.servlet.tags.form	MVC controller
187	5	13	81	10		0,20	4	1	4		org.springframework.web.servlet.theme	Themes
188	14	133	691	127		0,27	11	4	13		org.springframework.web.servlet.view	Content generator
189	3	20	138	22		1,00	0	3	10		org.springframework.web.servlet.view.document	Excel
190	4	25	140	29		0,20	4	1	12		org.springframework.web.servlet.view.freemarker	Freemaker
191	9	65	436	61		0,22	7	2	13		org.springframework.web.servlet.view.jasperreports	Jasper
192	4	24	141	25		0,25	3	1	12		org.springframework.web.servlet.view.tiles	Tiles
193	7	53	306	56		0,10	9	1	19		org.springframework.web.servlet.view.velocity	Velocity
194	4	60	382	53		0,50	2	2	8		org.springframework.web.servlet.view.xslt	XSLT
195	13	98	581	103		0,31	11	5	15		org.springframework.web.struts	Struts
196	16	97	862	94		0,25	18	6	4		org.springframework.web.util	MVC controller
Total	1648	12026	70815	10528			1695	671				
Avg.	8,58	62,64	368,83	54,83		0,30	8,83	3,49	7,08			

Class averages	
Methods	7,49
NCSS	43,59
Javadoc	6,35
Method averages	
NCSS	5,82
Javadoc	0,85

Addendum E: Blueprints customisability metrics

	JavaNCSS					Jdepend						
P#	Classes	Methods	NCSS	Javadocs		Abstracness	Concrete	Abstract	Ce		Package	Feature
1	1	3	25	0		0,00	1	0	0		com.sun.j2ee.blueprints.encodingfilter.web	Encodingfilter
2	4	30	192	5		0,00	4	0	3		com.sun.j2ee.blueprints.mailer.ejb	Mailer
3	1	2	5	3		0,00	1	0	0		com.sun.j2ee.blueprints.mailer.exceptions	Mailer
4	1	1	4	1		0,00	1	0	0		com.sun.j2ee.blueprints.mailer.util	Mailer
5	1	6	23	6		0,00	1	0	0		com.sun.j2ee.blueprints.servicelocator	Servicelocator
6	1	11	104	11		0,00	1	0	1		com.sun.j2ee.blueprints.servicelocator.ejb	Servicelocator
7	1	12	144	11		0,00	1	0	1		com.sun.j2ee.blueprints.servicelocator.web	Servicelocator
8	3	10	44	5		0,67	1	2	1		com.sun.j2ee.blueprints.signon.ejb	Singon servlet
9	3	21	51	2		1,00	0	3	0		com.sun.j2ee.blueprints.signon.user.ejb	Singon servlet
10	4	22	308	2		0,00	4	0	3		com.sun.j2ee.blueprints.signon.web	Singon servlet
11	3	17	37	0		1,00	0	3	0		com.sun.j2ee.blueprints.uidgen.counter.ejb	Utility
12	3	9	40	1		0,67	1	2	1		com.sun.j2ee.blueprints.uidgen.ejb	Utility
13	4	15	81	6		0,50	2	2	2		com.sun.j2ee.blueprints.waf.controller.ejb	Business controller
14	2	7	18	0		1,00	0	2	2		com.sun.j2ee.blueprints.waf.controller.ejb.action	Business controller
15	1	1	12	0		0,00	1	0	4		com.sun.j2ee.blueprints.waf.controller.ejb.action.actions	Business controller
16	9	52	517	21		0,22	7	2	11		com.sun.j2ee.blueprints.waf.controller.web	Front controller
17	3	9	26	3		0,67	1	2	1		com.sun.j2ee.blueprints.waf.controller.web.action	Front controller
18	1	1	26	1		0,00	1	0	4		com.sun.j2ee.blueprints.waf.controller.web.action.actions	Front controller
19	4	17	113	7		0,25	3	1	1		com.sun.j2ee.blueprints.waf.controller.web.flow	Flow controller
20	1	3	38	1		0,00	1	0	2		com.sun.j2ee.blueprints.waf.controller.web.flow.handlers	Flow controller
21	1	1	19	1		0,00	1	0	0		com.sun.j2ee.blueprints.waf.controller.web.util	Flow controller
22	5	11	29	9		0,60	2	3	0		com.sun.j2ee.blueprints.waf.event	EJB actions
23	1	3	12	1		0,00	1	0	1		com.sun.j2ee.blueprints.waf.event.impl	EJB actions
24	2	4	14	1		0,00	2	0	0		com.sun.j2ee.blueprints.waf.exceptions	MVC controller
25	2	9	114	4		0,00	2	0	1		com.sun.j2ee.blueprints.waf.util	MVC controller
26	12	52	470	12		0,00	13	0	1		com.sun.j2ee.blueprints.waf.view.taglibs.smart	Template engine
27	5	36	437	2		0,00	5	0	2		com.sun.j2ee.blueprints.waf.view.template	Template engine
28	1	4	39	2		0,00	1	0	1		com.sun.j2ee.blueprints.waf.view.template.tags	Template engine
Total	80	369	2942	118			59	22				
Avg.	2,86	13,18	105,07	4,21		0,24	2,11	0,79	1,54			

Class averages	
Methods	4,61
NCSS	36,8
Javadoc	1,48
Method averages	
NCSS	7,98
Javadoc	0,32

Addendum F: Spring feature analysis

F#	Feature	Comment	Customziable?	Framework adds value	Weight	Alternate variant enabled by Customization?	Weight	Optional enabled by customization?	Weight
1	Dynamic language support	[S]							
2	Beanshell	[O]	No					Yes	1,0
3	Groovy	[O]	No					Yes	1,0
4	JRuby	[O]	No					Yes	1,0
5	Configuration	[V]	No			Yes	1,0		
6	Desployment descriptors		No						
7	Resource File	[V]	No			Yes	1,0		
8	Text file		No						
9	Java class		No						
10	MVC controller	[V]	No			Yes	1,0		
11	JSF		Yes	No					
12	Spring MVC		No						
13	Themes	[O]	Yes	Yes	1,0			Yes	1,0
14	Tapestry		No						
15	Struts		No						
16	Spring porlet MVC		No						
17	DAO	[S]							
18	ORM	[V]	No			Yes	1,0		
19	JPA		No						
20	Toplink		Yes	Yes	1,0				
21	Ibatis		Yes	Yes	1,0				
22	Hibernate		Yes	Yes	1,0				
23	JDO		Yes	Yes	1,0				
24	JDBC Abstraction		No						
25	Application Context	[S]							
26	Internationalization	[O]	No					Yes	1,0
27	Remoting	[S]	No						
28	JAX RPC	[O]	No					Yes	1,0
29	Burlap	[O]	Yes	Yes	1,0			Yes	1,0
30	RMI	[O]	No					Yes	1,0
31	Hessian	[O]	No					Yes	1,0
32	HTTP Invoker	[O]	No					Yes	1,0
33	JMS	[O]	No					Yes	1,0
34	Databinding		No						
35	Validation	[O]	No					No	
36	Spring container	[S]							
37	Bean factory	[S]							
38	Lifecycle management		No						
39	Inversion of control		No						
40	EJB Abstraction	[O]	No					Yes	1,0
41	Transaction managment	[V]	No			Yes	1,0		
42	Interceptor AOP		Yes	Yes	1,0				
43	JDBC		No						
44	Hibernate		No						
45	JTA		No						
46	Container managed		No						
47	JDO		No						
48	Toplink		No						
49	AOP support	[V]	No			Yes	1,0		
50	AspectJ		No						
51	Proxy/Spring AOP	[S]							
52	Autoproxy		No						
53	Enterprise utilities	[S]							
54	JCA	[O]	No					Yes	1,0
55	JMS	[O]	No					Yes	1,0
56	JavaMail	[O]	No					Yes	1,0
57	JMX	[O]	Yes	Yes	1,0			Yes	1,0
58	JNDI		No						
59	View abstraction	[V]	No			Yes	1,0		
60	JSP		No						
61	JSF		Yes	No					
62	Content generator	[S]							
63	Freemarker	[O]	Yes	Yes	1,0			Yes	1,0
64	Velocity	[O]	Yes	Yes	1,0			Yes	1,0
65	Jasper	[O]	No					Yes	1,0

66	Excel		No						
67	Jexcel	[O]	Yes	No				Yes	1,0
68	PIO		No						
69	iText		Yes	No					
70	XSLT	[O]	Yes	Yes	1,0			Yes	1,0
71	Tiles	[O]	No					Yes	1,0
72	Scheduling	[S]							
73	Quartz	[O]	No					Yes	1,0
74	JDK Timer	[O]	No					Yes	1,0
Number of Yes			15	11		7		24	

Key	Meaning
[S]	Skipped, see step diagram
[V]	Variant
[O]	Optional

Addendum G: Blueprints feature analysis

F#	Feature	Comment	Customziable?	Framework adds value	Weight	Alternate variant enabled by Customization?	Weight	Optional enabled by customization?	Weight
1	MVC controller	[S]							
2	Front Controller		No						
3	Business controller		Yes	Yes	0,5				
4	Screen controller		No						
5	EJB Container	[S]							
6	DAO factory	[O]	No					Yes	1,0
7	Single point of entry	[S]	No						
8	EJB Actions		No						
9	Security	[O]	No			Yes	1,0	Yes	1,0
10	Signon Servlet		Yes	Yes	0,5				
11	Enterprise utilities	[S]							
12	Mailer		Yes	Yes	1,0				
13	JaxP		No						
14	JNDI	[S]							
15	Service locator		No						
16	Web container	[S]							
17	Encoding filter		Yes	Yes	0,5				
18	View abstraction	[S]							
19	XML screens		No						
20	Template engine	[S]							
21	JSP	[O]	No					No	
Number of Yes			4	4		1		2	

Key	Meaning
[S]	Skipped, see step diagram
[V]	Variant
[O]	Optional

Addendum H: Spring feature step analysis

F#	Feature	Step type	n# steps	Step description	Step type	Config point	Comment
1	Dynamic language support						[S]
2	Beanshell						
		Enable	1	Set the bean type to beanshell in the application context file	Configuration	1	
3	Groovy					1	
		Enable	1	Set the bean type to beanshell in the application context file	Configuration	1	
		Changing	1	Creating callback methods in the bean	Source code	1	
4	JRuby						
		Enable	1	Set the bean type to beanshell in the application context file	Configuration	1	
5	Configuration						
		Enable	1	Set the configuration type in the application context file	Configurarion	1	
6	Desployment descriptors						[1][D]
7	Resource File						[1]
		Enable	1	By placing the resource file in the root directory	Configurarion	2	
8	Java class						[1]
9	Text file						[1]
10	MVC controller						
		Enable	1	Configuring the to use MVC the application context file	Configuration	1	
11	JSF						
		Changing	1	Filling parameters in the application context xml file	Configuration	15	[C]
		Changing	1	Creating custom face lifeycle classes	Source code		
12	Spring MVC						
		Changing	2	Creating an abstract controller	Source code		
				Extending the abtstract controller and overriding handlerequest	Source code		
13	Themes						
		Enable	1	Setting the new theme in the application context	Configuration	1	
		Changing	1	Setting pramaters in the theme file	Configuration	3	
14	Tapestry						
15	Struts						
		Changing	1	Writing custom controllers	Source code		[C]
16	Spring porlet MVC						
		Changing	2	Creating an abstract controller	Source code		
				Extending the abtstract controller and overriding handlerequest	Source code		
17	DAO						[S]
18	ORM						
		Enable	1	Enabling the type of mapping in the application context file	Configuration	1	
18	JPA						
		Changing	1	Modifying the bean code	Source code		[C]
20	Toplink						
		Changing	1	Setting parameters in the toplink xml file	Configuration	4	
		Changing	1	Modifying the bean code	Source code		[C]
21	Ibatis						
		Changing	1	Setting parameters in the sqlmap xml file	Configuration	5	
		Changing	1	Setting parameters in the individual baen xml files	Configuration	6	
		Changing	1	Modifying the bean code	Source code		[C]
22	Hibernate						
		Changing	1	Setting parameters in the application context file	Configuration	1	
		Changing	1	Setting parameters in the Hibernate xml file	Configuration	7	
		Changing	1	Modifying the bean code	Source code		[C]
23	JDO						
		Changing	1	Filling parameters in the application context xml file	Configuration	1	
		Changing	1	Modifying the bean code	Source code		
24	JDBC Abstraction						
		Changing	1	Modifying the SQl query in the code	Source code		[C]
25	Application Context						[S]
26	Internationalization						
		Enable	1	By placing the resource file in the root directory	Configuration	2	
27	Remoting						[S]
28	JAX RPC						
		Enable	1	Enabling the type of remoting trough the application context file	Configuration	1	
		Changing	1	Writing a custom handler to process messages	Source code		[C]
29	Burlap						
		Enable	1	Enabling the type of remoting trough the application context file	Configuration	1	
		Changing	1	Setting parameters in the remoting xml file	Configuration	8	
30	RMI						
		Enable	1	Enabling the type of remoting trough the application context file	Configuration	1	
31	Hessian						
		Enable	1	Enabling the type of remoting trough the application context file	Configuration	1	

32	HTTP Invoker							
		Enable		1	Enabling the type of remoting trough the application context file	Configuration	1	
33	JMS							
		Enable		2	By adding a message queue in the application context file	Configuration	1	
					By enabling the message queue in the application server	Configuration	9	
34	Databinding							
35	Validation							
		Enable		1	Enabling by writing custom validator classes for beans	Source code		
		Changing		1	Modifying the validator java classes	Source code		
36	Spring container							[S]
37	Bean factory							[S]
38	Lifecycle management							
		Changing		3	Writing a custom bean post processor	Source code		
					Writing callbacks in the java bean class	Source code		
					Enabling the callbacks in the application context for a bean	Source code		
39	Inversion of control							
40	EJB Abstraction							
		Enable		1	Enabling EJB in the application context for a bean	Configuration	1	
		Changing		1	Writing custom bean code	Source code		[C]
41	Transaction managment							
		Enable		1	Enabling a tranction type on the application context file	Configuration	1	
42	Interceptor AOP							
		Changing		2	Adding advice in the application context	Configuration	1	
					Setting parameters in the application context for advice	Configuration		
43	JDBC							
44	Hibernate							
45	JTA							[2]
46	Container managed							[2]
47	JDO							
48	Toplink							[2]
49	AOP support							
		Enable		1	Enabling the AOP type in application context	Configuration	1	
50	AspectJ							
		Enable		1	Adding advice to the bean declaration in the application context	Configuration	1	
51	Proxy/Spring AOP							[S]
52	Autoproxy							[D]
53	Enterprise utilities							[S]
54	JCA							
		Enable		1	Enabling JCA in the application context file	Configuration	1	
55	JMS							
		Enable		2	By adding a message queue in the application context file	Configuration	1	
					By enabling the message queue in the application server	Configuration	9	
56	JavaMail							
		Enable		1	Enabling a mail sender bean in the application context file	Configuration	1	
		Changing		1	Writing a custom mailsender	Source code		[C]
57	JMX							
		Enable		1	Enabling JMX on a specific bean in the application context file	Configuration	1	
		Changing		1	Filling parameters per bean for JMX	Configuration		
58	JNDI							
59	View abstraction							
		Enable		1	Enabling a view mapping in the application context file	Configuration	1	
60	JSP							
61	JSF							
		Changing		1	Filling parameters in the application context xml file	Configuration	15	[C]
		Changing		1	Creating custom face lifeycle classes	Source code		
62	Content generator							[S]
63	Freemarker							
		Enable		1	Enabling the bean in the application context	Configuration	1	
		Changing		1	Parameters application context	Configuration	1	
64	Velocity							
		Enable		1	Enabling the bean in the application context	Configuration	1	
		Changing		1	Setting the properties in the velocity properties file	Configuration	10	
65	Jasper							
		Enable		1	Enabling the baen in the application context	Configuration	1	
66	Excel							
67	Jexcel							
		Enable		1	Enabling the baen in the application context	Configuration	1	
		Changing		1	Setting the properties in the JExcel properties file	Configuration	12	[C]
68	PIO							
69	iText							
		Changing		1	Setting the properties in the iText properties file	Configuration	13	[C]
70	XSLT							
		Enable		1	Enabling the baen in the application context	Configuration	1	
		Changing		1	Setting the properties in the XSLT file	Configuration	14	
71	Tiles							

		Enable	1	Enabling the baen in the application context	Configuration	1	
72	Scheduling						
73	Quartz						
		Enable	1	Enabling the beans in the application context	Configuration	1	
		Changing	1	Modifying the doIt() method code	Source code		[C]
74	JDK Timer						
		Enable	1	Enabling the bean in the application context	Configuration	1	

Configuration point	Configuration point name
1	Application context
2	Root
3	Theme properties files
4	Toplink xml
5	Sql map xml
6	Beanname xml
7	Hibernate xml
8	Remoting.xml
9	Server context(on the app server!)
10	Velocity properties file
11	Template files
12	JExcel properties file
13	Itest properties file
14	XSLT file
15	Faces config file

Key	Key meaning
[C]	COTS product, already has this customization option
[D]	Default
[S]	Skipped, see step diagram
[1]	Why are there no configuration possibilities here? Because we can not change the way these files are USED, we can not customize the process, the parameters in the deployment descriptors customize others parts of the framework
[2]	Relies on AOP declerative transaction management

Addendum I: Blueprints feature step analysis

F#	Feature	Step type	n# steps	Step description	Step type	Config point	Comment
1	MVC controller						[S]
2	Front Controller						
3	Business controller						
		Changing	2	Writing a custom request processor	Source code		
				Configuraing the request processor in the web xml file	Configuration	1	
4	Flow controller						
		Changing	2	Subclassing the ScreenFlowManager class	Source code		
				Changing the type of ScreenFlowManager in the MainServlet	Source code		
5	EJB Container						[S]
6	DAO factory						
		Enable	1	Creating xml files per EJB baen	Configuration	5	
7	Single point of entry						[S]
8	EJB Actions						
		Changing	1	Modifying the eventsupport code	Source code		
9	Security						
		Enable	1	Enabling the type security type in the web.xml	Configuration	1	
10	Signon Servlet						
		Changing	1	Changing the parameters in the signon-config xml file	Configuration	3	
		Changing	2	Subclassing the Signon servlet class	Source code		
				Enabling the new signon servlet class in the web xml	Configuration	1	
11	Enterprise utilities						[S]
12	Mailer						
		Changing	2	Subclassing the mail class	Source code		
				Enabling the new mailer servlet class in the ejb-jar xml	Configuration	2	
		Changing	1	Filling parameters in the j2ee ri xml file	Configuration	4	
13	JaxP						
14	JNDI						[S]
15	Service locator						
16	Web container						[S]
17	Encoding filter						
		Changing	2	Subclassing the encoding servlet class	Source code		
				Enabling the new encoding servlet class in the web xml	Configuration	1	
18	View abstraction						[S]
19	XML screens						
20	Template engine						[S]
21	JSP						
		Enable	1	Modifying jps pages to use the template files	Source code		
		Changing	1	Wrtiting jsp template files	Source code		

Configuration point	Configuration point name
1	Web xml file
2	EJB-jar xml
3	Signon-config xml file
4	sun-j2ee-ri xml file
5	DAO xml file per bean

Key	Key meaning
[S]	Skipped, see step diagram

Addendum J: Spring modifiability metrics

	Jdepend				Cobertura	
P#	A	Ca	Ce	I	CC	Package
1	0,74	16	4	0,20	1,02	org,springframework,aop
2	0,22	3	18	0,86	2,46	org,springframework,aop,aspectj
3	0,16	0	15	1,00	1,75	org,springframework,aop,aspectj,annotation
4	0,00	2	9	0,82	2,67	org,springframework,aop,aspectj,autoproxy
5	0,12	1	17	0,94	2,10	org,springframework,aop,config
6	0,21	20	14	0,41	2,34	org,springframework,aop,framework
7	0,25	3	7	0,70	2,03	org,springframework,aop,framework,adapter
8	0,27	3	15	0,83	2,10	org,springframework,aop,framework,autoproxy
9	0,33	1	6	0,86	2,56	org,springframework,aop,framework,autoproxy,target
10	0,18	2	8	0,80	1,79	org,springframework,aop,interceptor
11	0,33	2	6	0,75	2,00	org,springframework,aop,scope
12	0,40	22	7	0,24	1,60	org,springframework,aop,support
13	0,00	1	3	0,75	0,00	org,springframework,aop,support,annotation
14	0,43	6	9	0,60	1,32	org,springframework,aop,target
15	0,67	1	3	0,75	1,38	org,springframework,aop,target,dynamic
16	0,34	66	6	0,08	2,77	org,springframework,beans
17	1,00	1	2	0,67	0,00	org,springframework,beans,annotation
18	0,39	97	4	0,04	1,35	org,springframework,beans,factory
19	0,33	3	6	0,67	2,60	org,springframework,beans,factory,access
20	0,50	1	6	0,86	1,67	org,springframework,beans,factory,annotation
21	0,36	33	7	0,17	1,93	org,springframework,beans,factory,config
22	0,33	0	3	1,00	0,00	org,springframework,beans,factory,generic
23	0,29	2	6	0,75	1,12	org,springframework,beans,factory,parsing
24	0,27	16	8	0,33	2,58	org,springframework,beans,factory,support
25	0,50	2	3	0,60	2,08	org,springframework,beans,factory,wiring
26	0,30	10	12	0,55	2,64	org,springframework,beans,factory,xml
27	0,00	6	3	0,33	2,23	org,springframework,beans,propertyeditors
28	0,20	4	6	0,60	1,64	org,springframework,beans,support
29	0,00	0	6	1,00	1,56	org,springframework,cache,ehcache
30	0,72	33	5	0,13	1,00	org,springframework,context
31	0,00	2	8	0,80	1,93	org,springframework,context,access
32	0,25	3	6	0,67	1,50	org,springframework,context,event
33	0,67	5	1	0,17	1,20	org,springframework,context,i18n
34	0,19	12	13	0,52	2,00	org,springframework,context,support
35	0,49	67	5	0,07	2,43	org,springframework,core
36	0,67	4	1	0,20	6,00	org,springframework,core,annotation
37	0,46	2	2	0,50	1,49	org,springframework,core,enums
38	0,31	38	1	0,03	1,42	org,springframework,core,io
39	0,50	21	3	0,12	2,48	org,springframework,core,io,support
40	0,50	0	2	1,00	1,37	org,springframework,core,style
41	0,18	9	2	0,18	1,05	org,springframework,core,task
42	0,11	27	1	0,04	1,00	org,springframework,dao
43	0,00	0	12	1,00	0,00	org,springframework,dao,annotation
44	0,60	16	5	0,24	2,70	org,springframework,dao,support
45	0,29	1	8	0,89	2,86	org,springframework,ejb,access
46	0,20	0	8	1,00	1,40	org,springframework,ejb,config
47	0,88	0	5	1,00	1,23	org,springframework,ejb,support
48	0,00	1	0	0,00	1,00	org,springframework,instrument
49	0,10	5	3	0,38	1,47	org,springframework,instrument,classloading
50	0,00	0	3	1,00	1,83	org,springframework,instrument,classloading,glassfish
51	0,00	0	3	1,00	1,00	org,springframework,instrument,classloading,oc4j
52	0,00	0	2	1,00	0,00	org,springframework,instrument,classloading,tomcat
53	0,00	3	1	0,25	1,00	org,springframework,jca,cci
54	0,07	2	5	0,71	2,19	org,springframework,jca,cci,connection
55	0,62	2	4	0,67	2,03	org,springframework,jca,cci,core
56	0,50	1	5	0,83	1,11	org,springframework,jca,cci,core,support
57	0,50	0	4	1,00	1,40	org,springframework,jca,cci,object
58	0,00	0	3	1,00	1,38	org,springframework,jca,support
59	0,00	0	3	1,00	1,97	org,springframework,jca,work
60	0,00	7	1	0,12	1,00	org,springframework,jdbc
61	0,39	4	10	0,71	1,69	org,springframework,jdbc,core
62	0,44	1	8	0,89	1,43	org,springframework,jdbc,core,namedparam
63	0,50	0	3	1,00	1,00	org,springframework,jdbc,core,simple
64	0,62	2	9	0,82	1,54	org,springframework,jdbc,core,support
65	0,26	14	6	0,30	2,26	org,springframework,jdbc,datasource
66	0,22	2	9	0,82	2,03	org,springframework,jdbc,datasource,lookup
67	0,57	0	7	1,00	1,51	org,springframework,jdbc,object
68	0,33	10	10	0,50	2,88	org,springframework,jdbc,support
69	0,38	0	4	1,00	1,63	org,springframework,jdbc,support,incrementer

70	0,24	7	5	0,42		1,65	org,springframework,jdbc,support,lob
71	0,22	2	4	0,67		1,82	org,springframework,jdbc,support,nativejdbc
72	0,50	2	1	0,33		2,44	org,springframework,jdbc,support,rowset
73	0,07	7	1	0,12		1,20	org,springframework,jms
74	0,16	3	5	0,62		2,17	org,springframework,jms,connection
75	0,26	1	7	0,88		1,43	org,springframework,jms,core
76	1,00	0	2	1,00		1,57	org,springframework,jms,core,support
77	0,21	3	12	0,80		2,26	org,springframework,jms,listener
78	0,40	0	6	1,00		2,00	org,springframework,jms,listener,adapter
79	0,19	0	7	1,00		1,35	org,springframework,jms,listener,serversession
80	0,00	0	11	1,00		2,03	org,springframework,jms,remoting
81	1,00	6	4	0,40		3,19	org,springframework,jms,support
82	0,25	2	2	0,50		2,67	org,springframework,jms,support,converter
83	0,43	4	4	0,50		1,75	org,springframework,jms,support,destination
84	0,00	5	1	0,17		1,00	org,springframework,jmx
85	0,00	0	7	1,00		2,86	org,springframework,jmx,access
86	0,23	0	11	1,00		2,05	org,springframework,jmx,export
87	0,88	0	4	1,00		3,80	org,springframework,jmx,export,annotation
88	0,58	1	8	0,89		2,15	org,springframework,jmx,export,assembler
89	0,20	3	5	0,62		1,64	org,springframework,jmx,export,metadata
90	0,40	1	7	0,88		2,10	org,springframework,jmx,export,naming
91	0,50	1	2	0,67		1,57	org,springframework,jmx,export,notification
92	0,08	4	7	0,64		2,16	org,springframework,jmx,support
93	0,20	10	6	0,38		2,04	org,springframework,jndi
94	0,38	2	2	0,50		1,43	org,springframework,mail
95	0,00	0	2	1,00		5,67	org,springframework,mail,cos
96	0,22	0	4	1,00		1,62	org,springframework,mail,javamail
97	1,00	3	0	0,00		1,00	org,springframework,metadata
98	0,00	0	7	1,00		1,00	org,springframework,metadata,commons
99	0,00	5	1	0,17		1,22	org,springframework,orm
100	0,08	1	23	0,96		2,05	org,springframework,orm,hibernate
101	0,25	0	12	1,00		1,61	org,springframework,orm,hibernate,support
102	0,07	2	30	0,94		2,06	org,springframework,orm,hibernate3
103	0,00	0	3	1,00		2,00	org,springframework,orm,hibernate3,annotation
104	0,20	0	19	1,00		1,59	org,springframework,orm,hibernate3,support
105	0,13	1	14	0,93		1,42	org,springframework,orm,ibatis
106	0,33	0	6	1,00		1,55	org,springframework,orm,ibatis,support
107	0,09	1	12	0,92		1,66	org,springframework,orm,jdo
108	0,33	0	9	1,00		1,38	org,springframework,orm,jdo,support
109	0,32	2	15	0,88		2,36	org,springframework,orm,jpa
110	0,29	1	10	0,91		1,86	org,springframework,orm,jpa,persistenceunit
111	0,12	0	14	1,00		1,82	org,springframework,orm,jpa,support
112	0,08	0	11	1,00		1,00	org,springframework,orm,jpa,vendor
113	0,16	1	22	0,96		1,68	org,springframework,orm,toplink
114	0,25	0	7	1,00		2,77	org,springframework,orm,toplink,support
115	0,00	6	1	0,14		1,00	org,springframework,remoting
116	0,11	0	14	1,00		2,27	org,springframework,remoting,caucho
117	0,38	0	12	1,00		1,55	org,springframework,remoting,httpinvoker
118	0,33	1	13	0,93		2,00	org,springframework,remoting,jaxrpc
119	0,00	0	4	1,00		1,80	org,springframework,remoting,jaxrpc,support
120	0,18	3	11	0,79		3,00	org,springframework,remoting,rmi
121	0,57	4	6	0,60		1,85	org,springframework,remoting,support
122	0,67	7	2	0,22		1,00	org,springframework,scheduling
123	0,00	0	5	1,00		1,30	org,springframework,scheduling,backportconcurrent
124	0,00	0	8	1,00		1,41	org,springframework,scheduling,commonj
125	0,00	0	4	1,00		1,30	org,springframework,scheduling,concurrent
126	0,15	0	20	1,00		2,18	org,springframework,scheduling,quartz
127	0,00	1	3	0,75		1,40	org,springframework,scheduling,support
128	0,00	1	5	0,83		1,23	org,springframework,scheduling,timer
129	0,80	4	3	0,43		1,00	org,springframework,scripting
130	0,20	0	6	1,00		3,17	org,springframework,scripting,bsh
131	0,00	0	7	1,00		3,17	org,springframework,scripting,config
132	0,50	0	5	1,00		2,45	org,springframework,scripting,groovy
133	0,62	0	10	1,00		2,76	org,springframework,scripting,jruby
134	0,00	1	16	0,94		2,30	org,springframework,scripting,support
135	1,00	1	0	0,00		0,00	org,springframework,stereotype
136	0,25	17	2	0,11		1,14	org,springframework,transaction
137	0,25	2	2	0,50		0,00	org,springframework,transaction,annotation
138	0,00	0	11	1,00		2,89	org,springframework,transaction,config
139	0,17	4	12	0,75		2,17	org,springframework,transaction,interceptor
140	0,00	2	6	0,75		2,95	org,springframework,transaction,jta
141	0,68	21	4	0,16		2,08	org,springframework,transaction,support
142	0,00	7	2	0,22		1,71	org,springframework,ui
143	1,00	5	1	0,17		1,00	org,springframework,ui,context
144	0,25	3	3	0,50		2,31	org,springframework,ui,context,support
145	0,25	1	7	0,88		2,00	org,springframework,ui,freemarker
146	0,33	1	3	0,75		1,82	org,springframework,ui,jasperreports

147	0,20	1	13	0,93		2,89	org,springframework,ui,velocity
148	0,57	137	3	0,02		3,13	org,springframework,util
149	0,00	1	1	0,50		1,89	org,springframework,util,comparator
150	0,25	7	3	0,30		2,24	org,springframework,util,xml
151	0,47	9	3	0,25		1,37	org,springframework,validation
152	0,33	7	0	0,00		1,00	org,springframework,web
153	0,28	6	5	0,45		1,71	org,springframework,web,bind
154	0,40	18	7	0,28		2,04	org,springframework,web,context
155	0,50	11	7	0,39		1,56	org,springframework,web,context,request
156	0,16	15	17	0,53		1,57	org,springframework,web,context,support
157	0,30	5	10	0,67		1,59	org,springframework,web,filter
158	0,25	0	4	1,00		1,87	org,springframework,web,jsf
159	0,60	7	1	0,12		1,00	org,springframework,web,multipart
160	0,25	1	9	0,90		2,32	org,springframework,web,multipart,commons
161	0,00	0	7	1,00		2,07	org,springframework,web,multipart,cos
162	0,20	3	5	0,62		1,81	org,springframework,web,multipart,support
163	0,25	2	20	0,91		2,54	org,springframework,web,portlet
164	0,15	2	5	0,71		1,82	org,springframework,web,portlet,bind
165	0,35	4	13	0,76		1,69	org,springframework,web,portlet,context
166	0,25	2	10	0,83		1,71	org,springframework,web,portlet,handler
167	0,50	2	7	0,78		1,55	org,springframework,web,portlet,multipart
168	0,46	1	9	0,90		1,94	org,springframework,web,portlet,mvc
169	0,33	4	2	0,33		1,49	org,springframework,web,portlet,util
170	0,32	11	28	0,72		2,40	org,springframework,web,servlet
171	0,25	5	12	0,71		1,92	org,springframework,web,servlet,handler
172	0,17	0	9	1,00		1,83	org,springframework,web,servlet,handler,metadata
173	0,17	1	6	0,86		1,94	org,springframework,web,servlet,i18n
174	0,42	4	10	0,71		1,67	org,springframework,web,servlet,mvc
175	0,29	1	9	0,90		2,62	org,springframework,web,servlet,mvc,multiaction
176	0,00	0	8	1,00		2,75	org,springframework,web,servlet,mvc,support
177	0,50	1	2	0,67		1,00	org,springframework,web,servlet,mvc,throwaway
178	0,50	13	13	0,50		1,76	org,springframework,web,servlet,support
179	0,20	1	6	0,86		2,07	org,springframework,web,servlet,tags
180	0,22	0	8	1,00		1,52	org,springframework,web,servlet,tags,form
181	0,20	1	4	0,80		1,69	org,springframework,web,servlet,theme
182	0,27	7	13	0,65		1,73	org,springframework,web,servlet,view
183	1,00	0	10	1,00		1,30	org,springframework,web,servlet,view,document
184	0,20	0	12	1,00		1,43	org,springframework,web,servlet,view,freemarker
185	0,22	0	13	1,00		2,48	org,springframework,web,servlet,view,jasperreports
186	0,25	0	12	1,00		1,75	org,springframework,web,servlet,view,tiles
187	0,10	0	19	1,00		1,77	org,springframework,web,servlet,view,velocity
188	0,50	0	8	1,00		2,13	org,springframework,web,servlet,view,xslt
189	0,31	0	15	1,00		1,71	org,springframework,web,struts
190	0,25	26	4	0,13		2,72	org,springframework,web,util
Avg	0,29	5,83	7,12	0,69		1,82	
Max	1,00	137	30	1,00		6,00	
Min	0,00	0	0	0,00		0,00	
Bavg	112	147	120	77		103	
Aavg	78	43	70	113		87	

Key	Meaning
A	Abstractness
Ca	Afferent coupling
Ce	Efferent coupling
I	Instabilty
CC	Cyclomatic complexity
Bavg	Below/equal average
Aavg	Above average

Addendum K: Blueprints modifiability metrics

	Jdepend					Cobertura		
P#	A	Ca	Ce	I		CC		Package
1	0,00	0	0	0,00		1,00		com.sun.j2ee.blueprints.encodingfilter.web
2	0,00	0	3	1,00		1,80		com.sun.j2ee.blueprints.mailer.ejb
3	0,00	1	0	0,00		1,00		com.sun.j2ee.blueprints.mailer.exceptions
4	0,00	0	0	0,00		1,00		com.sun.j2ee.blueprints.mailer.util
5	0,00	5	0	0,00		1,67		com.sun.j2ee.blueprints.servicelocator
6	0,00	2	1	0,33		5,00		com.sun.j2ee.blueprints.servicelocator.ejb
7	0,00	2	1	0,33		5,25		com.sun.j2ee.blueprints.servicelocator.web
8	0,67	1	1	0,50		1,40		com.sun.j2ee.blueprints.signon.ejb
9	1,00	2	0	0,00		1,29		com.sun.j2ee.blueprints.signon.user.ejb
10	0,00	0	3	1,00		3,73		com.sun.j2ee.blueprints.signon.web
11	1,00	1	0	0,00		1,00		com.sun.j2ee.blueprints.uidgen.counter.ejb
12	0,67	0	1	1,00		1,67		com.sun.j2ee.blueprints.uidgen.ejb
13	0,50	5	2	0,29		1,27		com.sun.j2ee.blueprints.waf.controller.ejb
14	1,00	3	2	0,40		1,00		com.sun.j2ee.blueprints.waf.controller.ejb.action
15	0,00	0	4	1,00		1,00		com.sun.j2ee.blueprints.waf.controller.ejb.action.actions
16	0,22	2	11	0,85		2,63		com.sun.j2ee.blueprints.waf.controller.web
17	0,67	4	1	0,20		1,00		com.sun.j2ee.blueprints.waf.controller.web.action
18	0,00	0	4	1,00		4,00		com.sun.j2ee.blueprints.waf.controller.web.action.actions
19	0,25	3	1	0,25		1,83		com.sun.j2ee.blueprints.waf.controller.web.flow
20	0,00	0	2	1,00		3,00		com.sun.j2ee.blueprints.waf.controller.web.flow.handlers
21	0,00	1	0	0,00		1,00		com.sun.j2ee.blueprints.waf.controller.web.util
22	0,60	10	0	0,00		1,00		com.sun.j2ee.blueprints.waf.event
23	0,00	3	1	0,25		1,00		com.sun.j2ee.blueprints.waf.event.impl
24	0,00	1	0	0,00		1,00		com.sun.j2ee.blueprints.waf.exceptions
25	0,00	2	1	0,33		4,00		com.sun.j2ee.blueprints.waf.util
26	0,00	0	1	1,00		1,98		com.sun.j2ee.blueprints.waf.view.taglibs.smart
27	0,00	1	2	0,67		3,64		com.sun.j2ee.blueprints.waf.view.template
28	0,00	0	1	1,00		3,25		com.sun.j2ee.blueprints.waf.view.template.tags
Avg	0,24	1,75	1,54	0,44		2,09		
Max	1,00	10	11	1,00		5,25		
Min	0,00	0	0	0,00		1,00		
Bavg	19	16	19	17		19		
Aavg	9	12	9	11		9		

Key	Meaning
A	Abstractness
Ca	Afferent coupling
Ce	Efferent coupling
I	Instabilty
CC	Cyclomatic complexity
Bavg	Below/equal average
Aavg	Above average