Chapter 1

Introduction

This thesis studies the construction and use of software architecture descriptions. The study focuses on architectural descriptions about the actual runtime structure and behavior of software embedded in large and complex software-intensive systems. The study is driven by the assumption that up-to-date architectural descriptions are key assets to improve the evolvability of existing software-intensive systems and to support architecture-centric evolution approaches.

1.1 Software Architecture

Software architecture is an emerging discipline in software engineering. Edsger Dijkstra in 1968 and David Parnas in the early 1970s were the first to identify the concept of software architecture. They emphasized that the structure of a software system matters and getting the structure right is critical for the development of the system. The concept of software architecture as a distinct discipline started to emerge in 1990 (Kruchten et al. 2006). Since then, researchers and practitioners have played a prominent role in furthering software architecture as a discipline. In 1996, the book Software Architecture: Perspectives on an Emerging Discipline (Shaw and Garlan 1996) brought forward the concepts in software architecture such as components, connectors, styles and so on. In 2000, the book Applied Software Architecture (Hofmeister et al. 2000) presented a study of software architecture in industry. This study became a practical guide to designing, describing, and applying software architecture. In 2003, the second edition of the book Software Architecture in Practice (Bass et al. 2003) confirmed the coupling of the software architecture of a system and its business and organizational context. Nowadays, these books are just part of the large body of knowledge for education programs and scientific research in software engineering and architecture.

Currently there is no agreement to what exactly software architecture entails. In the context of this thesis, we take the popular definition: the architecture of a software system defines its essential structures, which comprise software elements, the externally visible properties of those elements, and the relationships between them (Bass et al. 2003). Two key parts of this definition are essential structures and externally visible properties (Rozanski and Woods 2005). On the one hand, the structures of a system are often classified into static and dynamic structures. The static structures of a system tell you what the design-time form of a system is, including what its elements are and how they fit together. The dynamic structures of a system show how the system actually works, including what happens at runtime and what the system does in response to external and internal stimuli. On the other hand, externally visible sys-
tem properties manifest themselves in two different ways: externally visible behavior and quality properties.Externally visible behavior defines the functional interaction between the system and its environment, which tells you what a system does from the viewpoint of an external observer. Quality properties like performance or scalability tell you how well a system behaves from the viewpoint of an external observer.

### 1.1.1 Software Architecture Descriptions

In principle, every software system has an architecture, which exists whether or not it is documented or described and understood. The architecture as the earliest model of a solution to the customer needs arises from an analysis of the requirements and application domain characteristics. For some systems, the architecture may be simple and easy to understand or it may be so large and complex that no person can understand every aspect of it. Part of the role of a software architect is to describe the architecture to the people who need to understand it. The architect does this by means of an architectural description. An architectural description is a set of products that documents an architecture in a way its stakeholders can understand and demonstrate that the architecture has met their concerns (Rozanski and Woods 2005). The stakeholder in a software architecture is a person, group, or entity with an interest in or concerns about the realization of the architecture. The stakeholders’ concerns shape much of the information in the description of an architecture, so it is important that architectural descriptions should be simple, understandable, and possibly graphical, with well understood, but not necessarily formally defined, semantics.

The description or documentation of software architectures is one of the main lines of attention in software architecture research and practice. Documentation approaches like the Siemens four views (Hofmeister et al. 2000), Kruchten 4+1 views (Kruchten 1995), and the Views and Beyond of the Software Engineering Institute (Clements et al. 2002), provide useful and detailed guidelines on how to describe software architectures. These documentation approaches use the concept of views to guide the description of the architecture. A view describes a particular aspect of the architecture and multiple views will be needed to describe the whole architecture. The ANSI/IEEE 1471-2000: Recommended Practice for Architecture Description of Software-Intensive Systems is the first international standard in the area of software architecture, and was adopted in 2007 by ISO as ISO/IEC 42010:2007 (ISO/IEC 2007). Now, the ISO/IEC 42010 standard presents a conceptual framework for documentation approaches based on the use of multiple views, see Figure [1.1]. A key contribution of this framework is the elaboration of the concepts about architectural views, viewpoints, models, and the relationships between them. An architectural view is a representation of a set of system elements and relations associated with them, conforming to a specific viewpoint. An architectural viewpoint frames particular concerns of the system stakeholders and consists of the conventions for the construction, interpretation, and use of an architectural view. A view may consist of one or more architectural models. Each such architectural model conforms to a model kind, which is part of the conventions and methods established by an associated viewpoint. An architectural
1.1. Software Architecture

Figure 1.1: The ISO/IEC 42010 Conceptual Framework for Architectural Description (ISO/IEC 2007)

model may participate in more than one view.

The purpose of the framework is to enable the expression, communication, and review of architectures of systems and thereby lay a foundation for quality and cost gains through facilitating standardization of elements and practices for architectural description (ISO/IEC 2007). The framework does not present the details of how software architectures should be described. To fill in the details, the architect can use the documentation approaches that inspired the framework, e.g., (Hofmeister et al. 2000, Kruchten 1995, Clements et al. 2002), or more modern approaches that implement the framework, like the Viewpoints and Perspectives of Rozanski and Woods (Rozanski and Woods 2005). However, the efficient creation, use, and maintenance of architecture descriptions will remain a challenge, especially for large and complex systems that are constantly evolving.
1.1.2 Software Architecture Reconstruction

In an ideal world, the architecture is documented at the time architectural decisions are made, updated whenever these decisions are revised, and readily available when needed for a particular task. Unfortunately, architectural information, when available at all, is often out-dated and incorrect, or inappropriate for the task at hand. Software architecture reconstruction is defined as the form of reverse engineering in which architectural information is reconstructed for an existing system (Koschke 2009). Architects can use architecture reconstruction to support a number of activities:

- Analyzing and understanding the architecture of existing systems.
- Construction of architecture descriptions for systems that are poorly documented or for which documentation is not available.
- Evaluation of the conformance of the architecture as-built to the architecture as-documented.

The reconstruction process can make use of any possible resource (such as available documentation, stakeholder interviews, domain knowledge), but the most reliable source of information is the system itself, either via its source code or via traces obtained from executing the system. An architecture reconstruction typically involves three steps: extract raw data from the system, apply an appropriate abstraction technique, and present and visualize the obtained information.

The research community has produced a number of tools, techniques, and methods to support the steps in architecture reconstruction. Available solutions primarily concentrate on the reconstruction of views related to the structure and organization of the system implementation using the source code as main source of information (Koschke 2009). Architects can implement or apply such solutions to (re)construct up-to-date views of existing software systems. Architecture reconstruction is an active research area with considerable attention from the software industry. Yet, nowadays software systems brings up a new set of practical scenarios that most existing architecture reconstruction solutions do not address sufficiently (Stoermer et al. 2002).

1.2 Software-Intensive Systems

The history and maturity of software architecture is much shorter than building architecture. On the one hand, the history of building architecture can be traced back thousands of years to the pyramids in Egypt, the Great Wall in China, and the Inca’s cities in the highlands of Peru. On the other hand, the software industry that triggered software architecture began just in the late 1940s. Nevertheless, today modern automobiles, airplanes, cash or automated teller machines, medical diagnostic devices, machine tools, home entertainment centres, and even washing machines will not function without the software that is embedded in them. These products are examples of software-intensive systems.
1.2. Software-Intensive Systems

Our daily lives depend more and more on complex software-intensive systems, from entertainment to communications to transportation to medicine. Software technology is embedded in many high-tech products and defines more and more the innovation and market capability of a range of industrial sectors. In 2005, the study Software Intensive Systems in the Future (TNO/IDATE 2005), conducted by TNO and IDATE for ITEA, showed that software-intensive systems were driving the present and will drive the future of industrial sectors such as automotive, aerospace, medical equipment, telecom equipment, and consumer electronics. A software-intensive system in any of these sectors combines various high-tech hardware and software elements associated with multidisciplinary knowledge. Many of these systems have considerable economic value and strict quality requirements such as reliability, security, safety, and performance. The software elements in a software-intensive system are often as large as millions of lines of code and determine any of the following points: system functionality, system cost, system development risk, and development time.

1.2.1 Magnetic Resonance Imaging Scanners

A representative large and complex software-intensive system in the healthcare domain is a Magnetic Resonance Imagining (MRI) scanner. Figure 1.2 shows a high-end MRI scanner, developed by Philips Healthcare (Philips Healthcare 2010), deployed in a hospital. This MRI scanner uses a powerful magnetic field to align the nuclear magnetization of (usually) hydrogen atoms in the body. Radio frequency (RF) fields are used to systematically alter the alignment of this magnetization. This causes the hydrogen nuclei to produce a rotating magnetic field detectable by the scanner. This signal can be manipulated by additional magnetic fields to build up enough information to construct images of the body like the ones presented in Figure 1.3. The technology of MRI scanners provides much greater contrast between the different soft tissues of the body than computed tomography (CT) and X-rays do, making it especially useful.
6

1. Introduction

Figure 1.3: Body Images Produced with a MRI Scanner

in neurological (e.g., brain), musculoskeletal (e.g., muscles, cartilage, and tendons), cardiovascular (heart and blood vessels), and oncological (cancer) imaging.

MRI scanners are among the most expensive medical equipment available today. A high-end MRI scanner, like the one shown in Figure 1.2, cost around $2.5 million USD. Installation of the system in the hospital costs another $500,000 USD. The figure illustrates that the MRI scanner is distributed, at least, across three main rooms with different conditional requirements, to house the subsystem components. The examination room houses the main hardware components: a superconducting magnet, a gradient system, and a radio frequency system. The magnet is the largest and most expensive hardware component of the scanner. Its strength is measured in Tesla (symbol: T) which is equivalent to 10,000 Gauss. The high-end scanner shown in the figure contains a 3T magnet. In comparison to the strength of the Earth’s magnetic field on the equator, which is \(31\mu \text{T} \left(3.1 \times 10^{-5} \text{T}\right)\), the field strength of an MRI magnet is considerable high. Additional components, such as workstations, in hospital network, can be part of the system to allow physician access to the raw image data collected from various scans.

The Philips MRI scanner is the software-intensive system that we study in this thesis. Table 1.1 summarizes the characteristics of the software system embedded in the Philips MRI scanner. This software system runs mainly in the computers located in the technical and operator rooms, illustrated in Figure 1.2. The elements of the software system control the magnet, other hardware components, and the image processing functionality of the system. The characteristics of the source code, numbers of lines of code and the involved programming languages, demonstrate that the software of this system is significantly large and heterogeneous. In addition, the summary describes that the development team is geographically distributed and that it involves a large number of people from software engineering and other different disciplines. This
system is not unique in size and complexity: many other large and complex software-intensive systems exist such as airplanes, automobiles, cellular phones, and copier machines, which are the result of years of research and development with considerable investment and cost.

Table 1.1: Characteristics of the Software Embedded in Philips MRI Scanner.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Quantification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source code size</td>
<td>∼9 million lines of code in ∼30,000 files</td>
</tr>
<tr>
<td>Programming Languages</td>
<td>C, C++/STL, C#, Visual Basic, ASP, Jscript, Perl, Batch, and other proprietary languages</td>
</tr>
<tr>
<td>Multi-site Development</td>
<td>Worldwide 3 main sites and a few satellites</td>
</tr>
<tr>
<td>Involved Disciplines</td>
<td>Physics, mechanics, electronics, medicine, and software</td>
</tr>
<tr>
<td>Development team</td>
<td>400 (multidisciplinary) plus 150 software engineers</td>
</tr>
</tbody>
</table>

1.3 Evolvability of Software-Intensive Systems

Software systems used in a real-world environment must necessarily change or become progressively less useful in that environment (Lehman 1996). This statement also applies for software-intensive systems like the Philips MRI scanner. An MRI scanner is used in a real-world environment and must change constantly to respond to new requirements and to keep competing in the market. Some changes in the requirements can be anticipated and a system can be prepared for them. However, unanticipated changes will also occur since predicting the future and its consequences for a system is not an exact science.

Unanticipated changes are more likely in systems like an MRI scanner, which has a large number of stakeholders, including developers, maintainers, suppliers, financiers, customers, and end-users. These various stakeholders hold different kinds of concerns respect to the system’s functionality, quality, and economic value. Changes in the needs of any of these stakeholders will trigger different types of requirements and changes in the system. In addition, as any other software-intensive system, an MRI scanner combines different hardware and software elements. These elements are built with different technologies, each having a different lifecycle. The dependency between the system’s elements and the implementation technology causes that changes in any of the technologies may trigger changes in the system.

Development organizations of software-intensive systems are paying considerable attention to evolvability, which is defined as the system ability to respond effectively to changes (van de Laar et al. 2010). Improving the evolvability of a software-intensive system is making the system’s response to change quick, cost-effective, and predictable. A system that can respond quickly to change will have faster time to market. For a development organization entering into the market quickly is desirable from a return of investment perspective: the investment can be earned back quickly, probably even
amplified by a premium price, and risks associated with market uncertainty are reduced. Furthermore, being on the market early enables market share growth. Both facts increase the profitability of the system. When the system response to change is cost-effective, the development costs required to change the system are lower because few or even no resources are required. This benefit is especially profound in case of repeating changes, like for regular hardware upgrades, software updates, and new applications. Finally, when the system response to change is predictable, the associated benefits are not only noticeable during the realization of the response, when fewer surprises occur, but also during the planning phase. During the planning phase, the more accurate cost-benefit analysis of individual changes makes the comparison of different changes more accurate as well. Therefore, the more evolvable a system, the more likely its development organization is going to be successful.

1.3.1 Architecture-Centric Evolution

The value of evolvability is clearly recognized by the industry where systems are no longer being built from scratch, but are evolved or developed incrementally. In this context, software architecture is an ideal support for incremental development processes. Software architecture can expose the dimensions along which a system is expected to evolve (Garlan and Perry 1995), mainly as the bridge between requirements and design that performs the tradeoffs necessary to satisfy the demands of both. Figure 1.4 illustrates an overview of the evolution of the Philips MRI scanner over the last thirty years and the ideal role of the architecture in this process.

The ideal role of the architecture in the evolution of the Philips MRI scanner can be described as the Three Peaks model by Rozanski and Woods (Rozanski and Woods 2005). For each evolution cycle, the architecture sits between requirements analysis and software construction (design, code, and test). The three triangles (the peaks) in the model represent the major development activities for requirement analysis, architecture definition, and construction that are necessary in a given evolution phase. The widening of the triangles at their bases represents an increasing level of detail as time goes on while the system is developed. The curling arrows are to illustrate that the requirements and the architecture, as well as the architecture and the construction are intertwined at a progressively increasing level of detail during the evolution phase.

In an evolution phase, the specification, architecture, and construction of the system are quite distinct, but they have profound effects on each other and so cannot be considered in isolation. Requirement analysis provides an initial context for using or changing the existing architecture definition, but it is then itself affected by the existing architecture definition, as requirements are understood more fully. In turn, the resulting architecture definition drives the implementation process, but each piece of construction performed provides feedback about the effectiveness and utility of the architecture in use. When all these effects between requirements, architecture, and implementation are well managed and captured or documented as part of the architecture description, it will certainly serve as:

- A reliable blueprint that outlines and steers the design for the implementation of
1.4 Problem Statement

The research presented in this thesis is part of the Darwin project (van de Laar et al. 2007, van de Laar et al. 2010), a research project on system evolvability. Darwin is a cooperation between the Embedded Systems Institute (ESI), five Dutch universities, and Philips Healthcare MRI, who strives to improve the evolvability of the Philips MRI scanner following an architecture-centric evolution approach. The objective of this cooperation is to develop architectures, methods, techniques, and tools to improve the evolvability, and consequently reduce the time to market, of large software-intensive systems like the Philips MRI scanner.

In Section 1.3, we explained the importance of improving the evolvability and how the architecture can contribute to this end, especially when the architecture definition

Figure 1.4: The Incremental Development of the Philips MRI Scanner and The Three Peaks Model
and its description are synchronized with the requirements and implementation. In practice, however, achieving this in a cost and time effective manner is a challenge for organizations that incrementally develop software-intensive systems like an MRI scanner. Changes in the requirements, architecture, and construction of such systems are poorly or never documented during evolution cycles due to aspects like:

- **Time to market.** As we described in Section [3](#) reducing the time it takes to the development organization to release a functioning system available for sale is critical. Therefore, most development organizations tend to focus on the construction of the system functionality, but documenting changes, e.g. in the architecture or design, is deferred to a latter moment.

- **Resource constraints.** Because the construction or implementation of the system receives the main attention, few or no resources are left to support the proper documentation of changes in the requirements, architecture, and implementation.

- **Dynamics of the organization.** The structure and conformance of the development organizations change constantly and knowledge that should be documented is lost. At the personal level knowledge is lost because people forget or leave the organization. At the organizational level reorganizations and outsourcing are typically counterproductive with respect to maintaining and documenting knowledge.

- **Lack of support.** Development organizations do not have proper tools and techniques to cope with the forgetful nature of humans and the large and complex knowledge in the development of a system like a MRI scanner. Therefore, the architecture description of such systems, when available at all, is often incomplete or it is more about technical details than the essential structures of the system.

As a result, the architectural description does not reflect the realization of the system and becomes out-dated and inaccessible, especially for the next evolution cycle. This situation represents a major problem because the value of the architecture decreases and an architecture-centric evolution will be just another costly and unpredictable experience. This thesis aims to address this problem devising the support that is needed to synchronize the architectural description and the realization of the system.

When the architectural description does not reflect the current realization of the system, the architect must either spend time to reconstruct it, or deliver a less informed and, hence, more risky description of the architecture. If the latter is the case, the architecture cannot be sufficiently explicit and reliable to expose the dimensions along which the system has evolved and is expected to evolve. Consequently, the response to change will be slow, expensive, and unpredictable threatening the success of the development organization and decreasing the system’s evolvability.
1.5 Research Questions

In the previous section, we described that an out-dated architectural description is a major problem for the architecture-centric evolution of large and complex software-intensive systems. In practice, addressing this problem and keeping the description of the architecture up-to-date is one of the architects’ responsibilities. The main theme of this thesis is to contribute to the solution of this problem by providing the support that architects need to produce up-to-date architectural descriptions, i.e. views, of the software embedded in systems like the Philips MRI scanner. For the incremental development of such systems many kinds of up-to-date views are needed, especially when the development organization is composed by diverse, specialized, and dynamic development teams, which have different kinds of concerns, sometimes conflicting, with specialized perspective, jargon, and levels of abstraction.

Learning about the incremental development process of the Philips MRI scanner, we observed that many kinds of views are required to make the parts of the system and dependencies between them explicit. Dependencies create ramifications of change between the system’s parts or features developed and maintained by the various teams of the organization. To better understand dependencies and devise the required support, we decided to investigate the value of dependency analysis by studying the state-of-the-art and assess it with respect to the state-of-practice, i.e. the incremental development of the Philips MRI scanner. To this end, we formulate the first research question:

- RQ-1. What is the state-of-the-art in dependency analysis and its value for the architecture-centric evolution of a large and complex software-intensive system?

Through the research triggered by RQ-1, we learned that dependency analysis is a broad field with a number of solutions and application areas. Therefore, we decided to focus on a specific area and solution. The analysis and description of the runtime structure of software-intensive systems is an area that did not receive much attention from the research community. The runtime structure and behavior of software-intensive systems like the Philips MRI scanner can change more often than other system aspects due to dependencies, not only between software elements but also between the hardware elements. Changes on software or hardware elements will likely change the runtime of a software-intensive system. In addition, the runtime of software-intensive systems is tightly coupled with performance and distribution requirements so, tuning any of these characteristics to fit in changing requirement will likely affect the runtime of the system. Therefore, architects need to produce up-to-date views that describe the actual runtime structure and behavior of the software embedded in software-intensive systems. We decided to call these views execution views and focus our research on answering the following research question:

- RQ-2. How to construct execution views of the software embedded in a large and complex software-intensive system?
Based on ISO/IEC Std. 42010, a view is a representation of a set of system elements and relations associated with them. For example, views about the development structure and static decomposition of software systems commonly describe elements such as libraries, packages, modules, and relationships between them. Therefore, before we could answer RQ-2, we must know the set of elements and relations associated with them that can be used to describe the runtime behavior and structure of a software system. We decided to call such elements runtime concepts and formulate the following research question:

- **RQ-2.1.** What are the useful runtime concepts that we can use to describe the actual runtime behavior and structure of the software embedded in a software-intensive system?

Once we identify a set of useful runtime concepts at hand, the next step to construct up-to-date execution views is to obtain up-to-date runtime information, i.e. information in terms of the runtime concepts. As we described in Section 1.1.2, the process to recover architectural information from an existing system is called architecture reconstruction and requires an extraction-abstraction technique. To devise the respective support to construct execution views, we formulate the following research question:

- **RQ-2.2.** How to obtain up-to-date information about the runtime of the software embedded in a software-intensive system?

Once we know how to obtain runtime information, we need to identify the appropriate way to present or visualize it, and to use it. A systematic way to achieve this is to make the respective viewpoint(s) explicit. According to the ISO/IEC Std. 42010, an architectural viewpoint frames particular concerns of the system stakeholders and consists of the conventions for the construction, interpretation, and use of an architectural view. Therefore, we formulate the next research question:

- **RQ-3.** What are the useful viewpoints for execution views of a software-intensive system?

Finding a solution for the construction of execution views includes placing it as part of the support required by the architects, i.e. support to produce up-to-date architectural descriptions, during the incremental development of a system like the Philips MRI scanner. To this end, we formulate our last two research questions:

- **RQ-4.** How is the construction and use of execution views embedded in the incremental development process of a software-intensive system?

- **RQ-5.** What is the actual contribution of constructing and using execution views in the incremental development of a software-intensive system?
1.6 Research Approach

1.6.1 Industry as Laboratory

The generic context of the Darwin project is an Industry-as-Laboratory approach (Potts 1993). The research presented in this thesis was carried out in this context. In an Industry-as-Laboratory paradigm, researchers identify problems through close involvement with industrial projects, and create and evaluate solutions in an almost indivisible research activity. The inspiration for an Industry-as-Laboratory approach is the mutual need between software engineering researchers and practitioners (Basili 1996): "while the software engineering researcher tries to understand the nature of processes and methods to develop systems, the practitioners build improved systems with this knowledge". The one cannot live without the other, the researcher needs a laboratory to observe and manipulate variables, but these only exist where practitioners build systems. Equally, the practitioners need to better understand how to build systems, and the researcher can provide models to help.

Figure 1.5 provides an overview of the Industry-as-Laboratory approach followed in the Darwin project (van de Laar et al. 2010). In this approach, Philips Healthcare MRI represents the industrial partner looking for solutions that fitted its industrial context and showing the state of practice. As the industrial partner, Philips Healthcare MRI provided access to technical and business experts, and repositories containing large amounts of industrial documentation, data, and software. For the research in this thesis, the Industry-as-Laboratory approach enabled several iterations on a cycle of three phases (see 1, 2, and 3 in the figure):

- **Phase 1.** We search for challenging problems and research questions interacting with practitioners, observing their state-of-practice, and processing large amounts of industrial data.

- **Phase 2.** We conduct research activities to find and develop solutions for the identified problems.

- **Phase 3.** We try out and validate solutions in close cooperation with practitioners and as part of actual development projects.

The validation helped us to identify the feasibility and value of the implemented solution, and to improve our perception of the industrial problem and the hypothesis of the solution, which we carried on to a next iteration.

In the rest of this section, we describe the specific research methods, type of research questions, type of research results, and validation techniques that we used through the phases in the Industry-as-Laboratory approach.

1.6.2 Research Methods

Based on the classification and revision of research methods in computing by SIGCSE-CSMR (Holz et al. 2006), the methods that we used in the phases of the Industry-as-Laboratory approach are:
• **Interview.** This is an information gathering technique whereby people are posed questions by an interviewer. For our research, we used two types of interviews:

  – *Informal conversational.* In an informal conversational interview, no predetermined questions are asked in order to remain as open as possible to the interviewees nature and priorities. In general, we used this type of interviews to interact with practitioners and learn about challenging problems and their way of working. For RQ-1, we used this type of interviews to gather the practitioners’ opinion about existing dependency analysis solutions. Similarly, we used this type of interviews to collect practitioners’ feedback during the validation of the solution that we built to find the answers for RQ-2, RQ-4, and RQ-5.

  – *Question set.* In a question-set interview, a predetermined questionnaire with question sets is used to ensure that the same general areas of information are collected from each interviewee. We used question-set interviews with key practitioners to gather the knowledge and build the viewpoints for execution views to answer RQ-3.

• **Case Study.** This is a technique to examine a single organization, group, or system in detail; involves no variable manipulation, experimental design or controls; is exploratory in nature. The research in this thesis is based on a single case study, the incremental development of the Philips MRI scanner. We use this case study as main reference to assess the state-of-the-art in dependency analysis and find the answer for RQ-1. We used particular instances of the case study, i.e. development projects, to answer RQ-2, RQ-2.2, RQ-4, and RQ-5 by experimenting and demonstrating a solution for the construction of execution views.

• **Critical Analysis of Literature.** This is an appraisal of relevant published material based on careful analytical evaluation. We implement this technique, more
1.6. Research Approach

precisely a systematic literature review (Kitchenham 2004b), to answer RQ-1 by identifying, evaluating, and interpreting available research relevant to dependency analysis of software systems.

- **Proof of Concept.** This technique, also known as proof of principle, is to validate a claim about the value of a system design or the design of a part of a system by building a system based on that design. Typically, the system that is built is not fully featured, but has enough functionality to convince the reader that the design can be effective. We used the proof of concept method to iteratively develop and validate a set of runtime concepts, a dynamic analysis technique, a set of viewpoints, and a top-down strategy to put in practice the construction and use of execution views, which contributed to answers RQ-2.1, RQ-2.2, RQ-3, and RQ-4 respectively.

1.6.3 Types of Research Questions

The research questions that triggered the research in this thesis are of interest for researchers and practitioners, since they concern about knowledge that contributes to the software architects’ ability to construct and use up-to-date architectural views. With the research classification by Shaw (Shaw 2002) as a reference, our research questions match the following types:

- RQ-2, RQ-2.2, and RQ-4 are about methods or means of development, e.g., we want to know how to construct execution views.
- RQ-1 is about a method for analysis, e.g., we want to assess the value of existing dependency analysis solutions.
- RQ-1 and RQ-2.1 are about design, evaluation, or analysis of a particular instance, e.g., we want to find what are the runtime concept that we need to define to construct execution views.
- RQ-3 and RQ-4 are about generalization or characterization, e.g., we want to find how to construct execution views regardless of the case study.
- RQ-5 is about feasibility, e.g., we want to show how feasible and useful is the construction of execution views.

1.6.4 Types of Research Results

The research conducted to find answers for the research questions in this thesis produced a number of research results. Again, taking the classification by Shaw (Shaw 2002) as a reference, our research results match the following types:

- **Procedure or technique.** This type of result represents new or better ways to perform some task. This thesis presents four results of this type. (1) Chapter 3 presents an architecture reconstruction approach for execution views. (2) The
same chapter presents a dynamic analysis technique to extract runtime information. (3) Chapter 5 presents an approach to define and document viewpoints. (4) Chapter 6 presents a top-down strategy to embed the architecture reconstruction for execution views and the use of this kind views in an actual development process.

- **Qualitative or Descriptive Model.** This type of result represents models describing the structure or taxonomy of a problem area. This thesis presents two results of this type. (1) Chapter 2 presents an overview about the state-of-the-art on dependency analysis. (2) Chapter 5 presents a catalog of viewpoints, including a set of conceptual models that describe the role of execution views in practice.

- **Analytical Model.** This type of result represents structural models that are precise enough to support formal analysis or automatic manipulation. An example of this result type is an execution metamodel in Chapter 3. This metamodel organizes the runtime concepts and relationships between them that we identify to describe and analyze the runtime behavior and structure of a software-intensive system.

- **Notation or Tool.** This type of result includes formal or graphical languages to support a technique or model, or a tool supporting such a language. Chapter 5 presents several model types and their respective notations that can be used to construct execution views. These results are part of the viewpoints for execution views.

- **Specific Solution.** This type of result is a specific analysis, evaluation, or comparison. Chapter 3, 4, and 6 present examples of this result type, which are execution views for the Philips MRI scanner. We constructed these execution views during the validation of our research and to support specific development projects within Philips Healthcare MRI.

- **Answer or Judgment.** This type of result is a specific analysis, evaluation, or comparison. This thesis presents two main instances of this type of result. (1) Chapter 2 presents an evaluation of the applicability of the state-of-the-art on dependency analysis. (2) Chapter 3, 4, and 6 present our judgment of the value of the architecture reconstruction approach for execution views. The value of the approach is based on our observations and the feedback from practitioners involved in the validation of the approach.

- **Report.** It is a report about interesting observations and discovered rules of thumb. In this thesis, Chapter 6 presents the main instance of this type of result. The result is an experience report on how we embedded the constructing of execution views in the development process of the Philips MRI scanner.
1.6.5 Validation Techniques

Using the classification by Shaw (Shaw 2002) as a reference, the validation of the research result in this thesis combines experience and example techniques.

- **Experience.** It means that the research result has been used on real examples by someone else and the evidence of its correctness, usefulness, or effectiveness is validated by any of the following techniques:
  - *Qualitative mode.* It is a narrative of the application of the research result.
  - *Empirical.* Data is collected on the practice of the research result and statistically analyzed.
  - *Notation/tool technique.* A comparison with other notations or tool techniques is made with similar results in actual use.

  In this thesis, we used qualitative models to validate by experience the approach for the construction of execution views, the execution metamodel, the dynamic analysis technique to extract runtime information, and the viewpoints for execution views. The qualitative models are the descriptions about how each of these results were developed and applied during actual development projects with Philips Healthcare MRI.

- **Example.** An example of how the research result work is provided. Two different types of examples are distinguished:
  - *Toy Example.* A simplified example, which might have been motivated by reality.
  - *Slice of Life.* A system that the author has developed. As we described in Section 1.2.1, the Philips MRI scanner is a representative software-intensive system. This system is the slice of life that was used to validate every research result in this thesis.

  Table 1.2 outlines the research process in this thesis by mapping each research question to research methods, research results, validation techniques, and the chapter(s) in this thesis where the mapping is realized.

1.7 Overview of this Thesis

The body of knowledge presented in this thesis is based on journal and conference articles that are being revised or already published. In this thesis, the articles have minor modifications to make consistent the style, structure, and terminology.

Chapter 2, under revision for a special issue on software architecture of the Journal on Empirical Software Engineering, is a joint work with Pieter van der Spek that reports a systematic literature review on dependency analysis solutions. The review is practice-driven because its research questions, execution, and reporting are driven by
Table 1.2: Overview of the research process in this thesis.

<table>
<thead>
<tr>
<th>RQ</th>
<th>Research Method</th>
<th>Research Result</th>
<th>Validation Technique</th>
<th>Chapter(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ-1</td>
<td>Interview</td>
<td>Qualitative or descriptive model Answer or judgment</td>
<td>Descriptive model</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Case study</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Critical analysis of the literature</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ-2</td>
<td>Interview</td>
<td>Procedure or technique</td>
<td>Qualitative model Slice of life</td>
<td>3, 4</td>
</tr>
<tr>
<td></td>
<td>Case study</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Proof of concept</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ-2.1</td>
<td>Proof of concept</td>
<td>Analytical model</td>
<td>Qualitative model Slice of life</td>
<td>3, 5</td>
</tr>
<tr>
<td>RQ-2.2</td>
<td>Case study</td>
<td>Procedure or technique</td>
<td>Qualitative model Slice of life</td>
<td>3, 4</td>
</tr>
<tr>
<td></td>
<td>Proof of concept</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ-3</td>
<td>Interview</td>
<td>Procedure or technique</td>
<td>Qualitative or descriptive model</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Proof of concept</td>
<td>Qualitative or descriptive model Notation or tool</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ-4</td>
<td>Interview</td>
<td>Procedure or technique</td>
<td>Slice of life</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Proof of concept</td>
<td>Report</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RQ-5</td>
<td>Interview</td>
<td>Report</td>
<td>Slice of life</td>
<td>3, 5, 6</td>
</tr>
<tr>
<td></td>
<td>Proof of concept</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Case study</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The characteristics of the practice of a group of software architects at Philips Healthcare MRI. The review produced an overview and assessment of the state-of-the-art and applicability of dependency analysis. The overview represents the answers for RQ-1 and provides insights about definitions related to dependency analysis, the sort of development activities that need dependency analysis, and the classification and description of a number of dependency analysis solutions. The contribution of the chapter is for both practitioners and researchers, who can take it as a reference to learn about dependency analysis and match their own practice to a provided assessment.

Chapter[3] under review for publication in the Journal of Software Maintenance and Evolution, is an extended version of our work (Callo Arias et al. 2008), published at the 15th Working Conference on Reverse Engineering. The chapter presents an architecture reconstruction approach to construct execution views for the software embedded in the Philips MRI scanner. The implementation and validation of the approach and its components represent the answers for RQ-2, RQ-2.1, and RQ-2.2. In addition, Section[3.7] contributes to the answer for RQ-5. The foundation of the approach is a dynamic analysis technique, originally designed to enable the identification of dependencies between the runtime entities. The technique extracts architectural information from logging and runtime measurements applying an execution metamodel and
1.7. Overview of this Thesis

a rule-based mechanism. The metamodel organizes the runtime concepts and abstractions to describe the runtime of a software system, e.g., scenarios, components, and processes rather than usual code artifacts such as modules, classes, or objects.

Chapter 4, published at the 16th Working Conference on Reverse Engineering, describes the construction of a resource usage view (Callo Arias, America and Avgeriou 2009a). The article shows the application of the architecture reconstruction approach, presented in Chapter 3, to construct a resource usage view. This kind of view consists of models that are important assets to analyze and control usage of hardware resources such as processors and memory elements. The application provides insights to answer RQ-2 and RQ-2.2. The view constructed as part of the application of the approach helped practitioners in Philips Healthcare MRI to address their concerns about resource usage in terms of the system-specific runtime elements such as execution scenarios, tasks, software components, processes, and threads.

Chapter 5, published in the special issue Best Papers from the IEEE/IFIP WICSA/ECSA 2009 of the Journal of Systems and Software (Callo Arias, America and Avgeriou 2010), is an extended version of our work (Callo Arias, America and Avgeriou 2009b). In this paper, we describe how to follow the conceptual framework of the ISO/IEC 42010 Std. to formalize and make explicit the concept and value of an execution view within Philips Healthcare MRI. Following the framework resulted in defining a set of viewpoints that capture a set of guidelines to construct and use execution views. The guidelines address the requirements for execution views that we identified conducting a series of interviews with Philips Healthcare MRI experts. The combinations of the defined viewpoints and our observations from constructing execution views has helped us to document mature viewpoints to support the construction and use of execution views within Philips Healthcare MRI. The research results reported in the chapter represent the answer for RQ-3, as well as for RQ-2.1 and RQ-5 in Section 5.3.3.

Chapter 6, published in the special issue Software Evolution of the Journal Science of Computer Programming (Callo Arias, Avgeriou, America, Blom and Bachynskyy 2010), reports our experience using a top-down strategy to construct and use an execution view during a development project for the Philips MRI scanner. We designed the strategy to embed the architecture reconstruction solution for execution views in the development process. The experience with the strategy helped to transfer the knowledge on how to construct and use execution views to key software architects and designers. The application of the strategy helped us to construct an up-to-date execution view for the start-up process of the Philips MRI scanner. With his view as a blueprint, practitioners at Philips Healthcare MRI quickly reduced by 30% the start-up time of the Philips MRI scanner and set up a new system benchmark for the system performance. The experience reported in this chapter builds up the answer for RQ-4 and RQ-5.

Chapter 7 concludes the study in this thesis. We provide answers for the research questions and remarks about the research contribution. In addition, we describe some open issues within our research that can motivate further research activities.