1
Introduction

Modularity is considered an essential concept of modern software design thought. It is defined by the IEEE Standard Glossary of Software Engineering Terminology (IEEE, 1990) as the degree to which a system program is composed of discrete components such that a change to one component has minimal impact on other components. A high degree of modularity is claimed to bring a series of benefits to software design, such as comprehensibility, changeability, adaptability, reusability, and so forth (Parnas, 1972; Booch, 1994; Meyer, 1997). Modularity should be, and usually is, applied at all stages of design, ranging from architecture specification (Bass et al., 2003; Clements et al., 2003) to detailed design and code levels of abstraction. Software engineers consider modularity as a key principle when comparing design alternatives and analyzing architecture degeneration (Eick et al, 2001, Lindvall et al., 2002).

The systematic assessment of modularity plays a pivotal role in the realm of software design. Moreover, assessment and improvement of early design modularity is even more challenging, since early design decisions strongly influence the next stages of development. Therefore, quantitative assessment techniques are needed for evaluating architecture alternatives. Software metrics are a powerful means to provide modularity indicators of software design (Dobrica & Niemela, 2002, Fenton & Pfleeger, 1997). The software metrics community has consistently used notions of module coupling, cohesion and interface size to derive measures of modularity (Briand et al., 1993; Chidamber & Kemerer, 1994; Fenton & Pfleeger, 1997; Lung & Kalaichelvan, 1998; Martin, 1997). Nowadays, a number of tools (e.g. Eclipse plugins (Eclipse Foundation, 2007a)) provide support for measuring these attributes. Also, books on lessons learned and the importance of using metrics in practice have been recently published (Lanza & Marinescu, 2006).

In fact, the conception of the right architectural decomposition is still a deep bottleneck to the software design process, since a number of widely-scoped

concerns need to be simultaneously modularized. A concern is any important property or area of interest of a system that we want to treat in a modular way (Elrad et al., 2001; Tarr et al., 1999). Typical concerns in a software project include features, business rules, non-functional requirements, architectural patterns and design patterns (Elrad et al., 2001; Robillard & Murphy, 2007). Apart from the last category, all the other concerns need to be considered from the architecture design. Distribution, persistence, transaction management, security and caching are examples of concerns found in many software systems.

Much of the complexity of software design is derived from the inadequate modularization of concerns. In practice, it is not trivial to well modularize concerns in a system due to a variety of reasons, including: inadequate initial design of widely-scoped concerns (Robillard & Murphy, 2007); limitations imposed by composition and decomposition mechanisms (Kiczales et al., 1997; Tarr et al., 1999); the emergence of unforeseen concerns as a system evolves (Robillard & Murphy, 2007); and the decay of design structures following repeated changes (Eick et al., 2001; Belady & Lehman, 1976; van Gurb & Bosch, 2001).

In addition, software designers tend to naturally give priority or focus on the modularization of certain concerns, while choosing a certain combination of existing architecture styles or relying on a particular way for design decomposition. As a consequence, a number of concerns end up having a crosscutting impact on the system architectural decomposition, thus systematically affecting the boundaries of several design elements, even elements at the architectural level, such as components and their interfaces (Zhang & Jacobsen, 2004; Greenwood et al., 2007a; Garcia & Lucena, 2008; Kulesza et al., 2006).

When a concrete usage scenario requires the adaptation or reuse of a concern, its adaptability or reusability is hindered if the concern is not well modularized. For instance, Zhang & Jacobsen (2004) claim that the goal of customizing certain concerns in middleware systems had been unattainable because these concerns did not have clear modular boundaries and were tangled with other concerns.

1.1. Problem Statement

Although typical modularity problems are related to the inadequate modularization of concerns, most of the current quantitative assessment approaches do not explicitly consider concern as a measurement abstraction. A number of design quantitative assessment methods are targeted at guiding decisions related to modularity, without calibrating the measurement outcomes to the driving concerns. It imposes certain shortcomings, such as the ineffective identification of desirable and undesirable couplings. For instance, coupling among modules addressing different concerns might hamper reusing or maintaining these concerns separately. On the other hand, coupling among modules addressing the same concern might not hinder the reusability or maintainability of that concern.

In addition, these shortcomings become more apparent in an age that a number of different approaches of design decompositions, such as aspect-oriented software development (Filman et al., 2005), are emerging. Aspect-oriented software development (AOSD) (Filman et al., 2005; Kickzales et al., 1997) is a new paradigm which aims at enhancing design modularization through new compositions mechanisms. However, the achievement of modular aspect-oriented (AO) designs is far from being trivial as the separation of certain concerns based on AO mechanisms can bring more harmful them good results (Filho et al., 2006, 2007; Garcia et al., 2006b). The lack of concern-specific modularity indicators makes it difficult to quantify the impact of contemporary modularization approaches, such as aspect-oriented software development, on system's concerns.

Software engineers, therefore, need quantitative assessment approaches to support them in the identification of modularity anomalies related to (in)adequate modularization of concerns. The lack of a concern-driven quantitative assessment hinders the design modularity analysis, because it makes the analysis of the overall influence of widely-scoped concerns on the software design difficult. The modularity analysis is hindered already from the architectural design. A number of concerns come from the requirements specification, and the architecture is the artifact on which the requirements are first treated. In fact, a number of case

studies have pointed out that detection of certain design flaws can be observed at early design stages (Kulesza et al., 2006; Soares et al., 2002; Filho et al., 2006).

1.2. Limitation of Conventional Measurement Approaches

Current quantitative design assessment approaches (e.g. Chidamber & Kemerer, 1994; Briand et al., 1993; Lung & Kalaichelvan, 1998; Martin, 1997) usually rely on conventional abstractions such as component (or module) and its interfaces in order to undertake the measurements. Based on these abstractions, they define and use metrics for quantifying attributes such as coupling between components, component cohesion, interface complexity, and so forth. Figure 1 depicts an architecture that will serve as a running example throughout this thesis, and as an illustration for the limitations of conventional design metrics. It shows a partial, simplified UML 2.0 (OMG, 2005) representation of the component-and-connector view (Bass et al., 2003) of the architecture description of a real Web-based information system, called Health Watcher (Soares et al., 2002). The design is structured mainly following the layer architectural style (Buschmann et al., 1996). Further information about the Health Watcher system is given in Section 7.3.

In an architecture specification, a concern is addressed (or realized) by a number of architecture elements, such as components, interfaces and operations. The gray boxes in Figure 1 represent the concerns addressed by the Health Watcher architecture elements. For instance, the business concern is addressed by the Business_Rules component and its interfaces, except the useTransaction interface, which addresses the persistence concern. Persistence is also an architectural concern in Health Watcher architecture which is realized by:

- the Data_Manager and Transaction_Control components,
- the useTransaction required interface of the Business_Rules component, and
- the operations transactionExceptionEvent and repositoryExceptionEvent that represent events raised or captured in a number of interfaces, such as savingService.

The exception handling concern is reified by the operations transactionExceptionalEvent, repositoryExceptionalEvent and communicationExceptionalEvent. In the light of this example, the next subsections discuss the limitations of current architecture metrics.

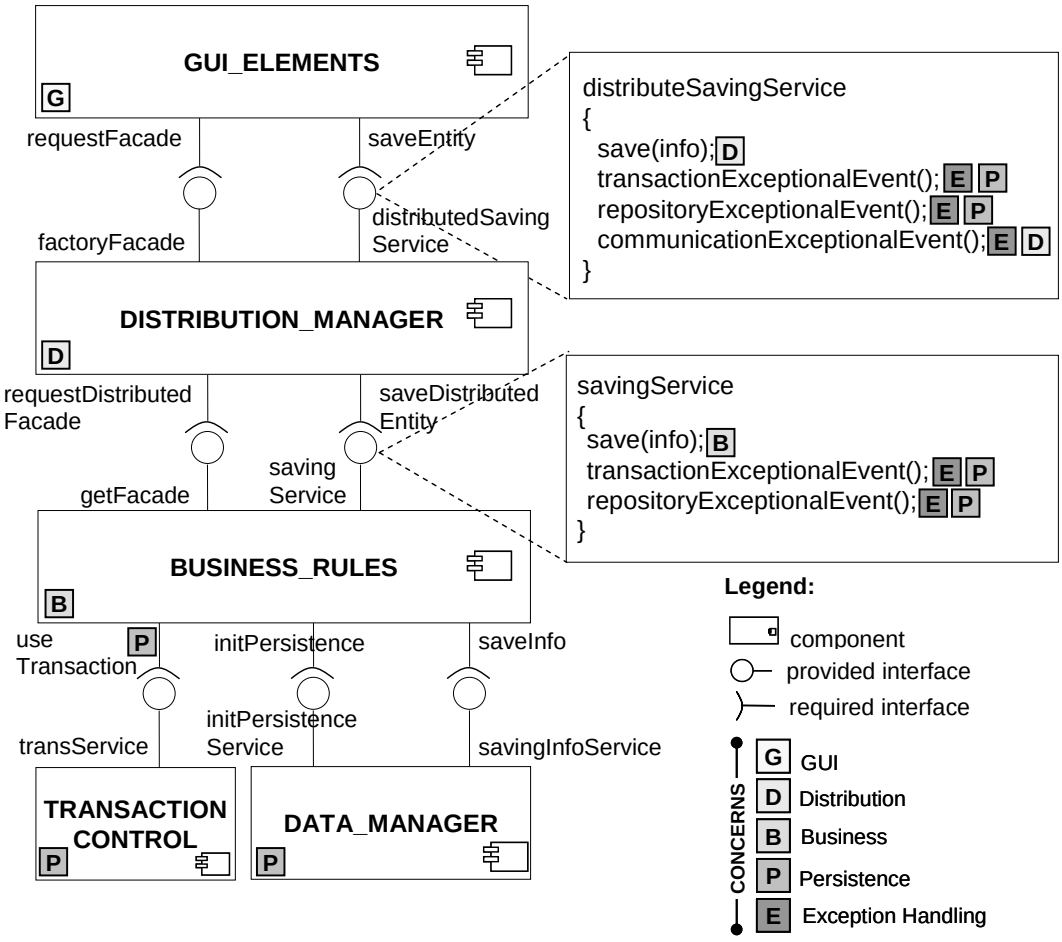

Figure 1: Architecture of the Health Watcher system

1.2.1. Inaccuracy on Identifying Non-localized Concerns

When choosing certain architecture abstractions, styles and mechanisms for decomposition, architects may not modularize some concerns. These concerns are not satisfactorily captured in separate modular units in the architecture description and, as a consequence, are not localized within components with well defined interfaces. The architecture presented in Figure 1 shows that the exception handling concern is partially addressed by abnormal events, such as transactionExceptionalEvent and repositoryExceptionalEvent, exposed in a

number of interfaces of the components. In this system, it is important to well modularize the exception handling strategies because they are similarly applied over several operations defined by the system architecture.

Conventional architecture metrics are not able to highlight that the exception handling concern has a wide impact on several interfaces. The main problem is that they do not rely on the identification of the architectural elements related to each concern, thereby causing a number of false negatives in the architecture assessment process. This is because typical architecture metrics are not able to explicitly capture this kind of modularity-related drawback as, for instance, exception handling is an architectural property typically diffused all over the architecture elements. As a result, existing metrics are often inaccurate to support the identification of non-localized architectural concerns.

1.2.2. Inaccuracy on Identifying Dependence between Concerns

The dependence between system concerns is a pivotal information for software architects in order to support design change management. Changes on a concern may impact concerns that depend on it. However, current coupling metrics are inaccurate to identify architectural inter-concern dependencies. Coupling architecture metrics quantify the dependence between components, thus only assess the dependence between primary concerns modularized within components.

Figure 1 shows that the Business_Rules component depends on two other components, namely Transaction_Control and Data_Manager. One component, the Distribution_Manager, depends on it. However, concerns which are not entirely modularized by the architecture abstractions do not have modular boundaries, in the sense that their boundaries are not well defined by component interfaces. Hence the dependence between such non-modularized concerns or even between non-modularized and modularized concerns cannot be measured by traditional measures. Therefore, these metrics cannot support the assessment of the impact of non-modularized concerns on other architectural concerns. Figure 1 shows that the distribution concern depends on the persistence concern because the Distribution_Manager component includes operations representing

persistence-related exceptional events, which are not modularized by any component. Moreover, the exception handling concern interacts with the persistence concern because the transactionExceptionEvent and repositoryExceptionEvent operations are related to both concerns.

1.2.3. Inaccuracy on Identifying Instabilities

Conventional metrics are also inaccurate to identify potential unstable architecture elements. A design element is typically unstable if it is influenced by a high number of concerns. This means that changes related to several concerns impact on that design element (Greenwood et al., 2007a; Figueiredo et al., 2008b; Eaddy et al., 2008a). Architecture stability is conventionally measured by the dependence between components (Martin, 1997). A component that depends upon no other component is considered to be stable, as changes in any other components are unlikely to be propagated to it and cause it to change. Therefore, current architecture metrics evaluate the stability of a component by measuring its coupling with other components (Martin, 1997).

Figure 1 shows that the Distribution_Manager component depends upon the Business_Rules component, thus changes in the latter can be propagated to the former. However, since some concerns are not totally modularized within components but cut across several components, the stability of a component has also to do with the number of concerns that affect it. The more concerns affect a component the more unstable that component is, because it can be changed due to changes related to those concerns. Figure 1 shows that the Distribution_Manager component is affected by the exception handling and persistence concerns, once its interface encompasses exceptional events related to persistence – the transactionExceptionEvent and repositoryExceptionEvent operations. This same reasoning is also valid for interface stability: the more concerns affect an interface the more unstable that interface is, because it can be modified due to changes related to those concerns. Since the current metrics do not take into account the number of concerns that affect the architecture elements, they are not able to capture this dimension of architecture stability.

1.2.4.
The Tyranny of Dominant Modularity Attributes

Software architecture measurement also suffers from what we call the tyranny of the dominant architectural modularity principles. The fact that the assessment of certain principles, such as low coupling and narrow interfaces, are overemphasized, and other equally important design principles, such as separation of concerns, have been neglected in architecture and detailed design measurement processes. This hampers modularity design assessment because it prevents the understanding of how the separation of certain concerns influence other modularity attributes. For instance, there are cases in which high coupling or large interfaces are caused by inadequate separation of concerns.

As shown in Figure 1, the exception handling concern affects the distributedSavingService and savingService interfaces, as they have to expose exceptional events. The operations representing these events contribute to increase the size of those interfaces. Even though the traditional metrics can provide information about the interface size, they do not support the architects to reason about the fact that the exception handling concern is the main contributor for that complexity. Hence, the identification of the impact of architectural concerns on traditional attributes is hindered.

1.3.
Proposed Solution

The goal of this work is to develop techniques that improve the quantitative assessment of software design modularity by promoting the concept of concern as a modularity measurement abstraction. In this context, the central focus of this work is to define a measurement approach targeted at complementing conventional modularity-related metrics by explicitly relying on concern-based metrics. A complementary goal of this work is to define such approach in a manner that makes it useful for assessing modularity of aspect-oriented software designs. This is because, although the aspect-oriented paradigm promises superior modularization of concerns, the inadequate use of its abstractions and mechanisms may hinder design modularity even more (Section 1.1).

In order to reach these goals we defined a concern-sensitive measurement approach for assessing design modularity. The proposed approach aims at supporting the software engineers to: (i) anticipate modularity problems caused by architecturally-relevant concerns, (ii) detect detailed design flaws caused by the inadequate modularization of key concerns, and (iii) compare aspect-oriented and conventional alternatives of design solutions with respect to their ability to modularize distinct sets of concerns.

Our approach relies on evaluating the modularization of a system's concerns from architectural to detailed design. Therefore, it includes two suites of metrics: one defined upon architectural design abstractions and the other defined upon detailed design abstractions. In order to cope with each of the conventional metrics' limitations depicted in Section 1.2, both suites comprise metrics for quantifying: (i) a concern's degree of scattering over design elements, (ii) dependence between non-localized concerns in terms of shared design elements, (iii) cohesion based on the amount of concerns addressed by a component, (iv) the contribution of a concern to the degree of coupling of a component, and (v) the contribution of a concern to the interface size of a component. In addition, since the interpretation of the detailed design metrics is fine-grained, we developed a suite of concern-driven heuristics rules. These rules combine the results of different metrics and enable the comparison of these results against configurable threshold values in order to support the identification of potential design flaws.

In order to consider concern as an abstraction in the measurement process, there is a need to explicitly document the concerns in the design. Therefore, our approach also includes a notation and a tool to support the architect with the documentation of the driving architectural concerns. Using this notation, the architect can assign every architecture element (components, interfaces, and operations) to one or more concerns. Also, our approach evaluates how a particular concern realization affects traditional attributes such as coupling between components and interface complexity. Hence it includes metrics for assessing these attributes.

In summary, the proposed concern-driven measurement approach is composed by: (i) a suite of concern-driven architectural metrics, (ii) a suite of concern-driven detailed design metrics, (iii) a suite of concern-driven design heuristic rules, (iv) the notion of concern templates as a notation for documenting

the concern-to-design mapping, and (v) a concern-oriented measurement tool, called COMET.

The *concern-driven architectural metrics* are defined upon the concern abstraction. These metrics allow the identification of architectural design flaws and degeneration caused by the poor modularization of architecturally-relevant concerns, and the comparison of alternatives of architecture design solutions in terms of how well architecturally-relevant concerns are modularized. They can be applied to the architecture description of systems, more specifically to component-and-connector views (Clements et al., 2003). They can be applied to all types of software architecture, including aspect-oriented software architectures.

The *concern-driven detailed design metrics* are, as the architectural metrics, uniformly defined upon the concern abstraction. They can be applied either to object-oriented or aspect-oriented design, or to compare both designs. A number of the metrics that constitute our detailed design metrics suite were proposed in previous studies (Sant'Anna et al, 2003, 2004; Garcia et al, 2006b, Cacho et al., 2006a). The contribution of this thesis is the extension of the existing set of metrics. This extension focuses on the definition of metrics for quantifying the contribution of certain concerns on the coupling and size of modules in detailed design, such as classes and aspects.

The *concern-driven design heuristic rules* are defined based on the proposed concern-driven detailed design metrics. A design heuristic rule is a composed logical condition based on metrics by which design fragments presenting specific problems can be detected. The proposed heuristics rules support the interpretation of the concern-driven detailed design metrics by pointing out design fragments that are negatively affected by the poor modularization of concerns.

The notion of *concern templates* is a notation for documenting architecture elements related to each concern considered in the measurement process. Concern templates, as well as concern metrics, are paradigm agnostic in the sense that they can be applied to designs structured according to different software decomposition paradigms. The *Concern-Oriented Measurement Tool (COMET)* is a tool that supports the concern-driven measurement at the architectural level. COMET supports: (i) the importation or definition of the architecture description of a system, (ii) the assignment of concerns to design elements, and (iii) the application of the concern-driven architectural metrics.

Each component of the proposed approach is viewed as an original contribution of this work. These contributions have been partially published in one journal paper (Sant'Anna et al., 2008), two conference papers (Sant'Anna et al., 2006, 2007b), and an international workshop paper (Sant'Anna et al., 2007a). In addition, some of the proposed metrics have been used by other research groups in the context of a controlled experiment (Eaddy et al., 2008a).

1.4. Empirical Evaluation

A series of empirical studies was undertaken in order to evaluate the proposed concern-driven measurement approach. The main goal of these studies was to evaluate the usefulness, applicability and effectiveness of our concern-driven metrics and heuristic rules on the assessment of modularity of software architecture and design, in particular aspect-oriented design. The studies aim at investigating how useful the approach is for: (i) comparing the modularity of aspect-oriented and conventional software design, (ii) assessing how architecture modularity is affected along the system evolution, and (iii) detecting specific design flaws.

Three studies were undertaken in order to evaluate the architectural metrics (Section 7.2, Section 7.3 and Section 7.4). Four different systems were involved in these studies. The concern-driven architectural metrics were used to perform modularity comparisons between conventional and aspect-oriented architecture of the systems. One study was undertaken to evaluate the applicability and accuracy of the design heuristic rules in order to detect flaws in both object-oriented and aspect-oriented designs. Six systems were used in this study (Section 8.1). Finally, a study was carried out to compare the effectiveness of conventional and concern-driven detailed design metrics on the identification of specific design flaws (Section 8.2). The empirical studies are also viewed as original contributions of this thesis.

1.5. Thesis Outline

The reminder of this thesis is organized as follows. Chapter 2 defines modularity and describes existing conventional metrics for assessing both high-level and detailed design modularity. Chapter 3 introduces the aspect-oriented software development paradigm as well as presents recent works on aspect-oriented architecture and aspect-oriented metrics. Chapter 4 defines the suite of concern-driven architectural metrics. In addition, it compares the proposed metrics with existing concern-oriented metrics in the light of a measurement framework specifically dedicated to concern-sensitive metrics (Figueiredo et al., 2008a).

Chapter 5 focuses on detailed design assessment. It defines the suite of detailed design metrics and heuristic rules. Chapter 6 describes both COMET, the tool that supports the concern-driven measurement at the architecture level, and the notion of concern templates. Chapter 7 describes and discusses the results of three empirical studies for evaluating the proposed architectural metrics. Chapter 8 discusses the results of studies for evaluating the detailed design metrics and heuristic rules. Chapter 9 draws the conclusions that tie together the claims and contributions of the thesis. Chapter 9 also discusses ongoing and future work.