3 On the Relation of Blueprints and Code Anomalies: A Study of Evolving Software Systems

Before answering several of the research questions posed in Section 1.5, we needed to better understand the relationship of architecture blueprints and code anomalies. Then, we initially conducted an empirical study based on structural blueprints representing the software descriptive architecture. The empirical evaluation relies on assessing to what extent architecture blueprints might help to reveal architectural problems related with the presence of critical code anomalies. In this sense, we preliminarily investigated how the prescriptive architecture, represented by architecture blueprints, might be impacted during the evolution of two software systems. In addition, this investigation will foster discussions on how to properly address our first research question **RQ$_1$**. This research question aims at evaluating whether the use of architectural information would help developers on revealing problems in the descriptive architecture observable in the source code. Furthermore, we discuss how the evolution of the actual implementation of a software system might be associated with inconsistencies in the descriptive architecture (see Section 3.1.3.2).

For doing so, we analyzed different evolution "scenarios" involving the blueprints and the source code. In these scenarios, we observed the documented changes required in the descriptive architecture in order to properly evolve the system implementation. These analyses are important to understand the direct or indirect role of architecture blueprints in the emergence of architectural problems in evolving systems. These also served us to check whether it is possible to explore architecture problems to improve the detection of critical code anomalies, i.e. those related to architectural problems. The focus of our study was the analysis of blueprints where the software modularity was a key priority since the design outset. In fact, Allen *et al.* (2001) claim that conventional modularity properties, such as cohesion and coupling, consistently play an important role in software modeling tasks. When the architectural design is well modularized, the software descriptive architecture is easier to be evolved. Thus, a fewer number of inconsistencies is likely to be observed between the

PUC-Rio - Certificação Digital Nº 1012688/CA

descriptive architecture and the actual implementation of a software system. In addition, software developers and architects can use different modeling techniques to produce well modularized software systems.

Our first pilot study, reported elsewhere (Guimarães *et al.*, 2010), showed that architecture blueprints using the aspect-oriented software development paradigm (Section 3.1) tend to lead to fewer inconsistencies between the blueprints and the produced source code if compared to the descriptive architecture using the object-oriented paradigm. Moreover, aspect-orientation is an extension of object-orientation and, as a result, the aspect-oriented design is decomposed in terms of aspects and classes. Anomalies in both types of modules can be manifested, making the study of aspect-oriented software systems more interesting as a preliminary study. Software systems with a superior modularity are less prone to manifest inconsistencies in the descriptive architecture with respect to the produced source code. The reason is that well-designed software architecture can be more easily evolved, and hence, it requires less effort from developers in the software maintenance and evolution. However, the presence of code anomalies on the system's actual implementation might be related with problems in the architecture design.

Given these arguments, we decided to focus on the analysis of architecture blueprints following an aspect-oriented decomposition in this first study. Aspect-oriented modeling (AOM) can be used by software developers, for instance, as means to achieve a superior separation of concerns and, therefore, it is expected to achieve better modularity. The main reason why AOM (Section 3.2) contribute to a better separation of concerns is because this paradigm holds two specific properties, namely *obliviousness* and *quantification* (see Section 3.1.2). The results of this chapter were published at: (i) 13th International Workshop on Aspect-Oriented Modeling (AOM) held in conjunction with MODELS; and (ii) 29th Symposium on Applied Computing (SAC).

3.1. Aspect-Oriented Software Development

This section motivates and describes aspect-orientation (Kickzales *et al.*, 1997) as it was the approach used to architect the systems analyzed in the study of this

PUC-Rio - Certificação Digital Nº 1012688/CA

chapter. Aspect-oriented software development (AOSD) is a viable alternative for the modularization and composition of *crosscutting concern* (Kickzales *et al*., 1997) during the development process. AOSD is an extension of object-oriented software development by offering new abstractions – e.g. aspects – and composition mechanisms – e.g. pointcuts – in addition to classes and objects. These extensions yielded by AOSD demonstrate the constant evolution and emergence of new paradigms, which has a direct impact on how software engineers architect their systems.

Object-oriented software development (Rentsch, 1982) emerged with the goal of encapsulating data and behavior under the abstraction of *objects*. Thus, interactions between objects occur by means of operations available on their interfaces. With the advent of the object-oriented programming (OOP), many researchers and developers claimed the development of software systems became more reusable and flexible (Meyer, 1997). They argue object-orientation (OO) facilitated the maintenance and development of modules as it allowed to explicitly encapsulate related data and behavior in modules of a software system.

However, the use of the OO paradigm usually leads to crosscutting concerns (Kickzales *et al*., 1997), which are related with the source code of a concern scattered and tangled in many places of the core system. The scattering and tangling of crosscutting concerns might hinder the comprehensibility and maintenance of a software system (Kickzales *et al*., 1997). Through the separation of otherwise crosscutting concerns into aspects, it is possible to achieve better modularization and avoid inconsistencies in the prescriptive architecture in relation to the actual system implementation. Typical examples of crosscutting concerns are error handling, persistence, distribution, security, logging, monitoring, and profiling (Kiczales *et al*., 1997). In order to support a better modularization of the crosscutting concerns, a new modularization abstraction called *aspect* and new composition mechanisms (e.g. *joint points, pointcut, advice and intertype declarations*) have been proposed. Each of these concepts are described in the following:

Aspects can be defined as modular units, designated to encapsulate crosscutting concerns, which are responsible for defining where, when and how they affect the core system (Filman *et al*., 2005). Thus, the *aspect* helps to improve the system modularity and to reduce the scattering and tangling of crosscutting concerns in the

PUC-Rio - Certificação Digital Nº 1012688/CA

system implementation, which might positively impact on the maintenance of the descriptive architecture.

Join Points can be defined as places in the program structure or execution flow, where additional behavior can be inserted (Filman *et al*., 2005).

Pointcuts gather a set of *join points* defining the concept of quantification, where unitary and separate statements can affect many non-local places in the modules of the descriptive architecture (Filman *et al*., 2005). In other words, *pointcuts* indicate a set of *join points* that will be affect for one or more crosscutting concerns encapsulated by a given *aspect*.

Advice represents a set of operations in a program that are executed when specific *join points* in a pointcut are reached in the system implementation. Usually, each *advice* has on or more pointcuts associated with it. Such *pointcut* determines the set of *joint points* over which the *advice* will be executed. In addition, advice operations can be organized in three types: (i) before – the advice is executed when the *joint point* is reached; (ii) after – the *advice* is executed after the control flow is returned in to the *join point*; and (iii) around – the *advice* is executed when the *pointcut* is reached and the control flow is explicitly delegated to the aspect.

Intertype Declaration is a mechanism used by the *aspect* to introduce new elements (e.g. attributes, methods, interfaces, inheritance) to the core system, providing reflection and modularization. That is, it allows new elements to be introduced in the actual implementation without the need of directly modifying the program specification (Filman and Friedman, 2001).

All the concepts of AOSD, mentioned above, can be used since early software design, i.e. when architects are determining how to organize the prescriptive architecture (Navasa *et al*., 2002)(France *et al*., 2004). For instance, they can, since the design outset, decide which components play the role of aspects (realize crosscutting concerns), and which components realize non-crosscutting concerns. Then, architecture blueprints can be used to represent these architectural decisions.

PUC-Rio - Certificação Digital Nº 1012688/CA

3.2.
Aspect-Oriented Modeling

In this chapter, we analyze systems based on aspect-oriented modeling to represent the descriptive architecture blueprints for each version of the target application under assessment. We decided to use the aspect-oriented (AO) paradigm since the descriptive architecture, represented in the architecture blueprint, is though to be better modularized. Then, it is likely to be more challenging to detect critical code anomalies, even when exploring blueprints in addition to the source code.

Figure 4 depicts an example of the Aspect-Oriented Modeling (AOM) notation used to provide the architecture blueprint of a software system. Currently there are several AOM languages for modeling the system in many levels of abstractions, ranging from use cases to detailed software design (Stein *et al*., 2002)(France *et al*., 2004)(Clarke and Baniassad, 2005)(Elrad *et al*., 2005). We have chosen the AOM language illustrated in **Figure 4** due to two main reasons: (i) we selected architecture blueprints as our focus to the availability of existing descriptive architecture specifications and the history of the evolution in both target applications – the AOM language was the one used to represent the architecture in those systems; and (ii) this language has been widely adopted for representing AO architectures in other contexts, as well as it provides a number of modeling features related with the AO properties (e.g. *obliviousness* and *quantification*). Moreover, the language is an extension of UML's component diagram, and supports the visual symmetric representation of AO architectures (e.g. aspectual and base components are both represented in the same way). Furthermore, the language provides a notation for expressing different forms of collaborations between aspects and architectural components.

PUC-Rio - Certificação Digital Nº 1012688/CA

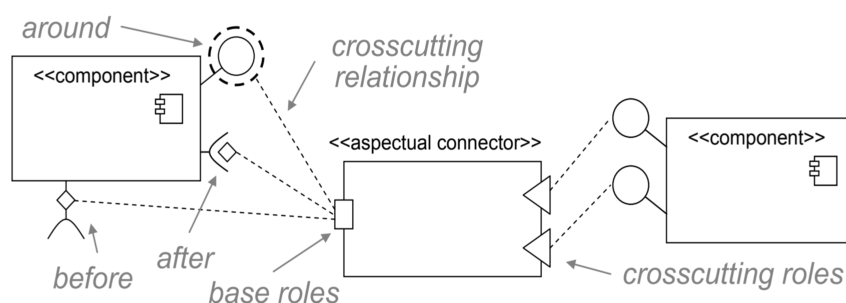

Figure 4 – AOM notation for architecture blueprints

Aspectual connectors represent collaborations between aspectual and base components. An *aspectual connector* is represented by rectangles (see **Figure 4**) and it defines components' interfaces, components and operations, which are affected by the aspect. Moreover, *aspectual connectors* are associated with the crosscutting relationships represented by dashed arrows. The notation also provides means to represent *advices* of aspects introducing a diamond on the interfaces (before and after advices), or even a dashed circle, in the case of *around* advice. The language also supports the visual modeling of specific *pointcut designators* (e.g. advising all the provided interfaces) and sequencing *operators* (after, before, and around). For the sake of simplicity, our studies have only considered the representation of aspectual connectors and crosscutting relationships in the architecture blueprints. In this way, it makes easier to understand how the notation works; all the other visual details have been omitted. As previously mentioned, we analysed also other factors might impact in the number inconsistencies observed in the descriptive architecture in relation to the actual system implementation. Therefore, we also evaluated the impact of two AO properties, *obliviousness and quantification*, when considering the number of inconsistencies in the descriptive architecture represented in the blueprint. Those properties have been proposed, in particular, as the core characteristics of the AOSD paradigm, and they are often used to define whether a language is aspect-oriented or not (Filman and Friedman, 2001).

On the one hand, *obliviousness* states that the design places where these quantifications were applied did not have to be specifically prepared to receive these enhancements (Filman and Friedman, 2001). For this study, we are measuring obliviousness by considering the number of change operations performed on the base components in order to accommodate the interaction between the aspects and base components. This measure helps us to indicate the obliviousness degree, as well as it makes possible to assess how the model elements are unaware regarding the existence of aspects. On the other hand, *quantification* can be defined as the idea one can write unitary and separate statements that have effect in many non-local places in design modules (Filman and Friedman, 2001). When the quantification property holds, it follows that aspects may crosscut an arbitrary number of component interfaces simultaneously.

PUC-Rio - Certificação Digital Nº 1012688/CA

Our explicit study of these two properties of AOM – obliviousness and quantification – was made, as they are key properties of aspect-oriented software architecture. They are explicitly represented in a blueprint of an aspect-oriented architecture design, independently from the level of details available in a blueprint. It might be certain high-level blueprints do not bring specific information about the specific type of advice used (before, after or around), but it will always: (i) determine which components are affected by an aspect component (quantification) – it can be inferred from the directed relationships from aspectual components to components, and (ii) determine some or all the elements exposed from a component to an aspect component (obliviousness).

3.3. Study Settings

PUC-Rio - Certificação Digital Nº 1012688/CA

There is a lack of empirical investigation on how architecture blueprints, representing the system`s descriptive architecture, might help to reveal the critical code anomalies. In this sense, there is a need for investigating whether and how the code anomalies might influence in inconsistencies on the architecture decomposition for evolving software systems. Our goal was to check whether modularity properties might be associated with inconsistencies in the descriptive architecture, represented by means of blueprints. In this sense, we derived two auxiliary questions in order to address **RQ**^{1} defined as:

ARQ^{1} - *Can modularity properties represented in the architecture blueprints help to void inconsistencies in the architecture decomposition?*

ARQ^{2} - *Is there a correlation between the presence of anomalies in the source code and inconsistencies observed in the descriptive architecture?*

The **ARQ**^{1} aims at investigating to what extent the modularity properties – observable in an architecture blueprint - may impact on the quality of the descriptive architecture during the system evolution. The hypotheses H_1 and H_2 derived from the **ARQ**^{1} are intended to evaluate the impact of modularity properties on the quality of the AO descriptive architecture represented through architecture blueprints. Our assumption is that aspects with higher quantification might lead to a higher number of inconsistencies observed in the descriptive architecture represented in the blueprint

($H_{1.1}$). In other words, the more the number of aspects affecting the base components, higher is the number of relationships with the base components - and more information are likely to be exposed in the descriptive architecture. On the other hand, the *obliviousness* might also impact on the number of inconsistencies, and as consequence, in the overall quality of the descriptive architecture. Therefore, our expectation is that higher obliviousness might lead to a lower number of inconsistencies between the descriptive architecture and the actual system implementation ($H_{2.1}$). A high obliviousness means that the architectural base elements are more unaware regarding the presence of aspectual components in the architecture blueprint. Finally, the **ARQ^2** aims at evaluate the correlation between the presence of anomalies in the source code and inconsistencies in the descriptive architecture investigated through the mapping of artifacts in both abstraction levels. Our assumption is that the presence of anomalies in the source code, mainly those related with AO properties, obliviousness and quantification might be related with the misuse of mechanisms provided by the AO (design and programming) languages. In this sense, the hypothesis $H_{3.0}$ states that there is no correlation between the presence of anomalies in the source code, and the number of inconsistencies observed in the architecture blueprints.

PUC-Rio - Certificação Digital Nº 1012688/CA

3.3.1. Target Systems

Systems with different characteristics were selected in order to evaluate our study hypotheses. The first target application is the Mobile Media system, which is a software product line which purpose is to provide support for manipulation of photos, music and videos on mobile devices (Figueiredo *et al.*, 2008). Our second target application is the Health Watcher system (Soares *et al.*, 2002), which is a framework that supports the registration and management of complaints to the public health system. The common reasons for selecting those target applications in this study are: (i) the architecture blueprints are the artifacts used to reason about change requests and derive new products; and (ii) the original developers produced the architecture blueprints without any architecture recovery tool (Garcia *et al.*, 2012)(Maqbool and

Babri, 2007) or a model composition techniques (Fleurey *et al*., 2008)(Reedy and France, 2004).

Moreover, the Mobile Media system was selected as target applications for two specific reasons: (i) we had available a total of seven fully documented evolution scenarios; (ii) different types of change were performed in each release, including refinements of the architecture style employed; and (iii) the evolution scenarios of Mobile Media range from changes in heterogeneous mobile platforms and additions of many alternatives and optional features. On the other hand, during the evolution of the Health Watcher system many maintenance tasks have been performed. Most of those tasks are from adaptive and perfective nature (Greenwood, 2007). The use of aspect-oriented paradigm to express the architectural design allows us to investigate its impact on software modularity, and hence, in the perfective and adaptive changes performed during the system evolution. In addition, Health Watcher was selected as target application for two specific reasons: (i) all the evolutions scenarios and architecture descriptions are available; and (ii) many changes were performed during the system evolution (e.g. insertion of design patterns to improve the system modularity).

PUC-Rio - Certificação Digital Nº 1012688/CA

3.3.2. Quantifying Modularity Properties and Inconsistencies

To measure the number of inconsistencies (see Section 2.4.3) observed in the architecture blueprints representing the prescriptive architecture, we calculate the *inconsistency rate* (IR). The *inconsistency* rate is measure by the ratio of the number of inconsistencies in the descriptive architecture in relation to the actual implementation of a software system, and the total number of architectural elements represented in the architecture blueprints. In order to measure some those inconsistencies, we have to perform the mapping between the descriptive architecture represented in the blueprint and the source code.

We also quantified the modularity properties in the AO descriptive architecture represented in the blueprints. First, we evaluated the quantification in the AO architecture blueprints by using the metric *set of join points* (SJP). The SJP metric considers the set of *join points* in the base elements captured by the aspectual

components in the descriptive architecture represented by the blueprint. For each aspectual component, we considered the number of join points presented in each *pointcut expression* specified in the architecture blueprints. Besides considering the explicit *join points* declared in the *pointcut expressions*, we are also considering all the *join point* references. Thus, when aspects make use of the *wildcard* mechanism (see Section 3.1), we are also counting all the references to join points in the base elements that are affected by the aspectual element.

Second, we evaluate the *degree of obliviousness* of the base elements regarding the presence of aspectual components. We considered the number of operations required to accommodate the aspectual components being added in the architectural design considering different changes required to evolve the systems' descriptive architecture. The idea is that obliviousness should be quantified as the amount of preparatory actions performed on the base classes (or other aspects) to enable their interaction with aspects. The set of preparatory modifications performed in the base components indicates how they are (un) aware of the presence of aspects, as well as the changes required to implement them. The higher the number of preparatory modifications being implemented, the lower is the degree of obliviousness. It is also important to mention we are considering changes in terms of class inheritance, interfaces, methods and modifications in the methods parameters list. The collected measures allow us to compare whether models with a higher (or lower) obliviousness tended to present lower inconsistencies in the descriptive architecture in relation to the actual implementation of the software system.

PUC-Rio - Certificação Digital Nº 1012688/CA

3.3.3. Study Phases and Assessment Procedures

In the section, we describe each of the main activities performed in order to conduct our empirical investigation. Those different activities are related with the procedures for assessing the modularity properties, as well as the inconsistencies observed in the architecture blueprints. As previously mentioned, the architecture blueprints in this study represents the structural view of the descriptive architecture for each target application.

Architecture Blueprints and Evolution Scenarios. Different evolution scenarios were considered for both target applications. The evolution scenarios define what changes are required from one version to another in order to evolve the system's descriptive architecture. Thus, we considered 5 evolution scenarios of Mobile Media using the AO paradigm for representing the descriptive architecture. Similarly, for Health Watcher system we considered 8 evolution scenarios.

Deriving AO Architecture Blueprints. We initially specified the evolution scenarios of each target application using the AO notation, before applying the composition techniques in order to evolve the system architecture. Thus, to evolve the architecture blueprints representing the descriptive architecture we have used model composition techniques to semi-automate the process. The main reason for using model composition techniques is due to the fact both industry and academy recognize its importance in evolving the architecture design models. In addition, composition techniques have already been employed in different domains (e.g. software product lines, UML models). Moreover, several studies have investigated the use of model composition techniques for evolving architecture design models (Guimaraes *et al.*, 2010)(Farias *et al.*, 2011)(Farias *et al.*, 2012). Additionally, changes performed during the system evolution are visible in the descriptive architecture represented in the blueprints. After AO architecture blueprints have been derived and the evolution scenarios specified, we assessed the inconsistencies in the descriptive architecture in relation to the actual implementation of a software system

Model Refactoring. It is an essential activity to refine the information about the system's descriptive architecture, represented in the blueprints, in order to express the changes required for evolution scenarios. The evolution scenarios represent all the key design decisions required to evolve the software system. In this sense, the architecture blueprint of both target applications were refactored as means to specify the *delta model* itself, which comprises all the changes for each evolution scenario. To derive the evolution scenarios, we considered the evolution description provided by original developers in previous studies (Figueiredo *et al.*, 2008)(Soares *et al.*, 2002).

Assessing Inconsistencies and Modularity Properties. To support a detailed data analysis, the assessment phase was further decomposed in two main stages aiming to: (i) identify the *inconsistency rate* in the architecture blueprints after the composition techniques have been applied for each evolution scenario; and (ii)

PUC-Rio - Certificação Digital Nº 1012688/CA

compute the data regarding the quantification and obliviousness in the AO architecture blueprints to evaluate their impact on the inconsistencies observed for each target application (see Section 3.3.2).

3.4. Hypotheses Testing and Initial Research Findings

In this section we evaluate our study hypotheses based on the data collected after the composition techniques have been applied to evolve the architecture blueprints of both target applications. We followed the traditional steps of applying a statistical model to data of a software engineering experiment (Wohlin *et al.*, 2000). For instance, we firstly tested if all the data follow a normal distribution by applying the Shapiro's test (Wohlin *et al.*, 2000). The main trends were also calculated and the Wilcoxon signed rank test (Wohlin *et al.*, 2000) was used to validate our hypotheses. In addition, the Pearson's correlation test (Wohlin *et al.*, 2000) was applied to analyze to what extent the modularity properties are related with the emergence of inconsistencies in the descriptive architecture represented in the blueprints.

Our first hypothesis ($H_{1.0}$) evaluates whether the degree of obliviousness impact on emergence of inconsistencies observed in the architecture blueprints generated by using composition techniques. As previously discussed, our hypothesis assumes that the higher the number of modifications required to solve inconsistencies in the architecture blueprints, the lower is the obliviousness of the architectural elements. The architectural elements considered are those providing *join points* to aspectual components defined in the architecture design. In this sense, our analysis evaluated whether there is a positive correlation between obliviousness degree and the inconsistency rate observed in the architecture blueprints generated by using composition techniques is positive. When testing the hypothesis, a Pearson's correlation test was performed to measure the strength of the linear relationship between degree of obliviousness and inconsistencies in the architecture blueprints in relation to the *prescriptive architecture*. **Table 2** summarizes the results of applying the Pearson's correlation test. Assuming a sample size (SS) = 26 and p-value = 0.05, the correlation test presented a calculated p-value = 0.009654 and a correlation coefficient = 0.497829. The calculated correlation coefficient indicates that there is a

PUC-Rio - Certificação Digital Nº 1012688/CA

moderate relationship between the obliviousness degree and inconsistencies observed in the architecture blueprints. In other words, our correlation analysis suggests that the higher the number of modifications in the architecture elements to accommodate changes related with aspectual components, the higher the inconsistencies in the architecture blueprints generated by using the two composition techniques. A higher number of modifications implies in a lower degree of obliviousness. In this sense, the alternative hypothesis $H_{1.1}$ is confirmed and we can say that, in general, AO architecture blueprints with lower obliviousness degree tend significantly to present higher inconsistencies in relation to the architecture initially intended by the system architect.

Table 2 - Correlation between Inconsistencies and AO Modularity Properties

Variable	Median	Mean	S.D.	Correlation
Obliviousness	3	7.4	8.8	0.5
Quantification	9	15.2	16.7	0.4
Inconsistencies	0.105	0.49	0.375	

Similarly to the first hypothesis, the results regarding the emergence of inconsistencies in the architecture blueprints remain consistent in both target systems. Considering all the evolution scenarios, we found the number of *join points* in the Health Watcher is higher than in the Mobile Media. A higher number of join joints, which implies in a higher quantification, might yield to higher inconsistencies in the architecture blueprint. The quantification can also be affected by the level of details of the information exposed in the architecture blueprints. For example, when analyzing the architecture blueprints of Mobile Media, a low number of *join points* can be observed if compared with the Health Watcher models. It is caused due to the high abstraction of Mobile Media models, which implies in less information being exposed to the software developers. The smaller the amount of information that can be observed regarding the *join points* affected by the aspects, the lower are the quantification measures collected; and (ii) the inconsistencies in the architecture blueprints in Health Watcher is smaller than in Mobile Media, since in the latter most part of the *pointcuts* specified in the aspects affect only 1 or 2 *join points*. In turn, most part of the aspects in the Health Watcher has *pointcuts* affecting more than 3 *joint points*.

PUC-Rio - Certificação Digital Nº 1012688/CA

Moreover, we also observed that although the architecture blueprints in Health Watcher have a lower level of abstraction, the larger number of architectural elements does not necessarily generate inconsistencies. Our correlation analysis is aimed at examining whether the inconsistencies in the architecture blueprints are directly related with the quantification degree. In order to examine the strength of relationship between the inconsistencies and the quantification degree, we have applied the Spearman' correlation. **Table 2** shows correlation test between quantification and inconsistencies observed in the AO architecture blueprints in relation to the prescriptive architecture. Assuming the sample size (SS) = 26 and p-value = 0.05, the Pearson's correlation test presented A correlation coefficient = 0.470866 and calculated *p-value* = 0.02. The correlation coefficient presented a positive value, which indicates a moderate correlation. Therefore, the results suggest that when the quantification in the AO architecture blueprints increases, the inconsistencies with the *prescriptive architecture* increase as well. As the calculated *p-value* is lower than 0.05, the correlation result is statistically significant, and hence, the alternative hypothesis $H_{2.1}$ is confirmed.

3.5. Code Anomalies and Inconsistencies in the Descriptive Architecture

So far, we have analyzed systems using AOM for architecture blueprints representing the descriptive architecture. In particular, we have evaluated the impact of modularity properties, obliviousness and quantification, in the inconsistencies observed in the prescriptive architecture in relation to the actual implementation of each target application. Our initial findings showed the AO modularity have a strong relationship with the number of inconsistencies in the architecture blueprints. We also observed the inconsistencies in architecture blueprints regarding the descriptive architecture is likely to be reduced when: (i) classes are oblivious to the presence of the aspects in each evolution scenario; and (ii) the quantification of aspects is low. As a consequence, we started investigating which specific design practices in AOM might either reduce or increase the inconsistencies in AO architecture blueprints. We figured out that many inconsistencies observed in the descriptive architecture were

PUC-Rio - Certificação Digital Nº 1012688/CA

often associated with the misuse of the AO mechanisms - which tend to either harmfully reduce obliviousness or increase quantification in an undesirable way.

In this sense, we can now focus on the main research question of our study, which aims at evaluating, as a preliminary investigation, how code anomalies can be associated with inconsistencies in the descriptive architecture. We have initially observed the presence of anomalies is often caused by the misuse of AO mechanisms (i.e. pointcuts, advices). Those anomalies seemed to be critical to the emergence of inconsistencies in the architecture blueprints. We decided then to focus on the anomalies related with the harmful obliviousness reduction and the misuse of quantification mechanism. The reason for selecting a specific set of anomalies is due to the fact that obliviousness and quantification might impact inconsistencies observed in the architecture blueprints regarding the *descriptive architecture*.

Our goal was to check whether the presence of those anomalies in the architecture blueprints would be correlated with inconsistencies with the *descriptive architecture*. The anomalies were independently detected for each version of the target applications and documented in previous studies (Macia *et al*., 2011). Aiming to investigate the impact of anomalies with the presence of inconsistencies in the architecture blueprints, eight types of anomalies are considered (see **Table 3**). All those anomalies are related either to harmful obliviousness decrease or misuse of quantification mechanisms, and have been documented in previous work and further information can be found in (Svirisut and Muenchaisri, 2007)(Macia *et al*., 2011).

Most of the selected anomalies are directly related with the misuse of pointcuts. For instance, aspects exhibiting the anomaly *Anonymous Pointcut* might have high quantification and can possibly generate more inconsistencies in the architecture blueprints. The problem is that *Anonymous Pointcut* has no signature and all the information is exposed in a *pointcut expression*. Therefore, we decided to represent the different join points in this manner, instead of using *wildcards*. Thus, for example, when a pointcut affecting 3 *join points* is represented by an expression, each of these join points are represented through a different relationship between aspectual component and each base element. So, the higher the number of information exposed in a *pointcut expression*, higher is probability of inconsistencies could be observed in the architecture blueprints – it implies in inconsistencies on the relationships between components' interfaces and connectors.

PUC-Rio - Certificação Digital Nº 1012688/CA

Table 3 – Anomalies in AO Target Applications

Anomaly Name	**Description**
God Aspect	It occurs whenever an aspect is realizing more than one system concern. For those cases, the aspect can be organized in many aspects as the number of concerns realized (Macia *et al*, 2011.
Lazy Aspect	It represents an aspect that has either none or only fragments responsibility (Piveta *et al*., 2006).
Forced Join Point	It is associated with code elements (e.g. methods and attributes) in the base code only exposed to be used by aspects. For example, some methods in the source code in which the implementation details can be exposed so that the aspects can access them (Macia *et al*, 2011)
Duplicated Pointcut	It occurs whenever different pointcut definitions collect the same set of join points in the base code. (Piveta *et al*., 2006)
Anonymous Pointcut	It occurs wherever a pointcut is directly defined on the advice.
God Pointcut	It occurs when a pointcut has either a complex expression involving many keywords or affect many scattered join points. This anomaly can also occur when the respective advice of a given pointcut has a very complex implementation (Macia *et al*, 2011).
Idle Pointcut	It is associated with pointcuts, which do not match any joint point. This code anomaly might occur due to several reasons: (i) mistakes in the pointcut expression, which may lead the wrong joint point to be affected; (ii) pointcuts are not referred by any advice, therefore no action is performed in the join point affected (Macia *et al*, 2011).
Redundant Pointcut	It occurs when pointcuts can be reused or combined by logical operations in order to define new-composed pointcuts. This code anomaly is associated with partial pointcut expressions equivalent to others already defined (Macia *et al*, 2011).

Another example of anomaly is the *Forced Join Point*, which can negatively influence the obliviousness of the base components regarding the presence of aspectual components. Its definition states that the operations/services in the base element are artificially created only for exposing extra information via their signature, so that one or more aspects can have access to them. As a consequence, the presence of this anomaly in an evolution scenario decreases the obliviousness of the base components. The artificially created operations/services are also potentially sources of propagation in inconsistencies observed in the architecture blueprints given their strong coupling with the pointcuts using the exposed information.

Code Anomalies vs. Inconsistencies in Architectures Blueprints. We also investigated how the presence of specific anomalies could impact on the inconsistencies observed in the descriptive architecture represented in the blueprints. The results showed the presence of anomalies is often caused by the misuse of AO mechanisms, such as pointcuts and advice. Thus, our focus was on the anomalies related with misuse of quantification mechanism and the harmful obliviousness reduction. The architecture blueprints used for each evolution scenario were analyzed

PUC-Rio - Certificação Digital Nº 1012688/CA

in both target systems to check whether the presence of modularity anomalies would impact on inconsistencies in the *descriptive architecture*. Therefore, we collected the data regarding the instances of each anomaly and computed the inconsistencies observed in the architecture blueprints for the evolution scenarios defined of each application. **Table 4** summarizes the data used for the correlation of anomalies detected for each version, as well as the number of inconsistencies observed in he descriptive architecture. The *inconsistencies are* computed for each version of the target applications when one of the two composition techniques *merge* (IR-M) and *override* (IR-O) is applied. When analyzing the Mobile Media, only in the evolution scenario related with version **V3** (N – represents the number of the system version under analysis) the presence of anomalies are not associated with inconsistencies in the architecture blueprints. Although 6 instances of anomalies were identified, they have not produced any inconsistencies. In other evolution scenarios of the Mobile Media system, we observed that the presence of anomalies is somehow associated with inconsistencies in the architecture blueprints.

PUC-Rio - Certificação Digital Nº 1012688/CA

Table 4 – Code Anomalies in Health Watcher and Mobile Media

	Mobile Media					**Health Watcher**							
Anomaly	**V1**	**V2**	**V3**	**V4**	**V5**	**V1**	**V2**	**V3**	**V4**	**V5**	**V6**	**V7**	**V8**
	1	4	8	9	10	20	24	47	49	49	48	40	40
IR-M	38	21	0	66	0	50	10	1	0	2	3	0	11
IR-O	38	36	0	73	6	41	4	1	0	0	0	0	34

On the other hand, a more interesting effect of the presence of anomalies could be observed in Health Watcher, because the number of anomalies is fairly high. However, taking as example the evolution scenarios from **V3** to **V7**, most of the anomalies are not related with inconsistencies in relation to the descriptive architecture. This can be explained by the fact that in those evolution scenarios none of the elements involved are related with anomalies. The low number of inconsistencies observed from release **V3** to **V7** in Health Watcher can be explained due the fact those evolution scenarios consists basically on applying design patterns for improving the system modularity. In version **V8**, the exception handling was improved and new elements were introduced to implement those enhancements. Since the system underwent a lot of refactoring operations, the number of inconsistencies in the architecture blueprints regarding the *descriptive architecture* for this evolution scenario is high – when compared to the previous versions.

A more critical scenario occurs when instances of anomalies are propagated from previous version. For instance, considering the evolution scenario from version **V1** to **V2**, there are some cases where *Anonymous Pointcut* and *Duplicated Pointcut* are related with inconsistencies in the architecture blueprints. However, when those anomalies are not properly solved, their relation with inconsistencies in the architecture blueprints will. Thus, we conclude that the occurrence of the anomaly was propagated from one version to another, as well as the inconsistencies in the architecture blueprints regarding the prescriptive architecture. For these evolution scenarios the number of anomalies is the same of the previous release. However, before draw any conclusions the Pearson's correlation test is performed in order to confirm or refute the study hypothesis $H_{3.0}$.

Table 5 shows data for the correlation analysis between anomalies and the presence of inconsistencies in the *descriptive architecture*. As we can observe, the correlation presented a positive value, which means that there is a true correlation. Our analysis considered a set of 26 evolution scenarios, of which 8 scenarios from Health Watcher and 5 scenarios for Mobile Media. The composition techniques were applied for evolving the architecture blueprints according to each evolution scenario defined for the target applications. Finally, the tests achieved a correlation coefficient equals to 0.2, which indicates a positive correlation but with a low statistical significance. Therefore, our study hypothesis $H_{3.1}$ s is confirmed.

Table 5 – Correlation between Code Anomalies and Inconsistencies

Variable	Median	Mean	S.D.	Correlation
Code Anomalies	49	36.6	20.4	0.2
Inconsistencies	0.105	0.5	0.74	

Frequent Code Anomalies. In the Mobile Media, the 3 most recurrent anomalies were *Duplicate Pointcut*, *God Aspect* and *Lazy Aspect*, considering the total of modularity anomalies of each release. They are responsible for around 95% of the anomalies presented in Mobile Media versions. On the other hand, for the Health Watcher system the 3 most recurrent anomalies were *Redundant Pointcut*, *Anonymous Pointcut* and *Idle Pointcut*. Those anomalies are responsible for more than 85 % of the total of anomalies in Health Watcher. Moreover, the 3 most recurrent anomalies of Mobile Media were related with 64% of inconsistencies observed in the architecture blueprints. From the total number of inconsistencies associated with the anomalies,

PUC-Rio - Certificação Digital Nº 1012688/CA

the percentage of inconsistencies specifically related with the anomalies *Duplicated Pointcut*, *God Aspect* and *Lazy Aspect* are 71.88%, 15.63% and 12.5%, respectively. On the other hand, the number of inconsistencies in the blueprints related with those three anomalies is even higher Health Watcher system, reaching 92%. Considering this high percentage of inconsistencies in the architecture blueprints, we found that around: 64.02% of inconsistencies are related with *Redundant Pointcut*, 20.11% are related with *Anonymous Pointcut*, and around 15.87% are related with *Idle Pointcut*. An analysis of the results in both systems shows that pointcut-related anomalies are the ones consistently associated with inconsistencies in the architecture blueprints when comparing to the *prescriptive architecture*.

3.6.
Summary

In this chapter we presented a preliminary study to assess the impact of AO architecture blueprints, representing the system's descriptive architecture, when evolving software systems. In order to mimic an automated process of evolving the system's architectural design, we have opted for using model composition techniques. As previously mentioned, this study was performed aiming to investigate how the use of architectural information would help to reveal architectural problems in the source code. In this sense, we evaluated how inconsistencies observed in the systems descriptive architecture would be related to the presence of anomalies when observing the actual implementation of the target applications. In order to address our first research question, we defined two auxiliary questions to investigate whether: (i) modularity properties represented in the architecture blueprints could help to void inconsistencies in the architecture decomposition; and (ii) there is a correlation between the presence of anomalies in the source code and inconsistencies observed in the descriptive architecture.

As expected, we also found that AO architecture blueprints have an improved modularization and therefore helped to better localize inconsistencies in the descriptive architecture in relation to the system's actual implementation. Thus, when analyzing the modularity properties *obliviousness* and *quantification*, we could also observe that: (i) when components in the architecture blueprints have a higher

PUC-Rio - Certificação Digital Nº 1012688/CA

obliviousness we observed a lower number o inconsistencies; and (ii) aspectual components with higher quantification are often related to inconsistencies with the system's actual implementation. The reason is that several elements in the source code are related to code anomalies associated to *pointcut* problem, which directly impacts on the quantification property.

In summary, we observed inconsistencies in the AO architecture blueprints might also be related to the presence of anomalies in the source-code. Moreover, most part of those anomalies are related to the misuse of AO mechanism, with are intended to provide means for a better modularization of software systems. After analyzing the architecture evolution of both target applications, we observed that developers should be careful when using aspect-oriented paradigm for building software systems, mainly in cases where: (i) the aspectual components have a high quantification and low obliviousness; (ii) some inconsistencies might be observed in the AO architectural design, mainly when an architectural components are implemented by anomalous code elements. Thus, developers must avoid the overuse of aspects with high quantification, particularly those pointcuts that are associated with the anomalies investigated in our preliminary study.

Once we observed that architecture blueprints could help to reveal how inconsistencies are associated with the presence of anomalies in the source-code, henceforward we were able to focus on a more in-depth analysis (presented in the next Chapters). For instance, we will investigate how the use of architecture information could be properly used as means to prioritize and rank the most critical code anomalies. In addition, we also will investigate what architectural information can be more effectively used in the prioritization and ranking problems. The reason for prioritizing and ranking code anomalies, as early as possible in the development of software systems, is to prevent inconsistencies between the system descriptive architecture and the system actual implementation. In this sense, we can help developers when avoiding more severe design problems that might lead to architecture degradation as the system evolves.

.

PUC-Rio - Certificação Digital Nº 1012688/CA