Chapter 17
The Role of Compliance and Conformance in Software Engineering

José C. Delgado
Universidade de Lisboa, Portugal

ABSTRACT

One of the most fundamental aspects of software engineering is the ability of software artifacts, namely programs, to interact and to produce applications that are more complex. This is known as interoperability, but, in most cases, it is dealt with at the syntactic level only. This chapter analyzes the interoperability problem from the point of view of abstract software artifacts and proposes a multidimensional framework that not only structures the description of these artifacts but also provides insight into the details of the interaction between them. The framework has four dimensions (lifecycle, concreteness level, concerns, and version). To support and characterize the interaction between artifacts, this chapter uses the concepts of compliance and conformance, which can establish partial interoperability between the artifacts. This reduces coupling while still allowing the required interoperability, which increases adaptability and changeability according to metrics that are proposed and contributes to a sustainable interoperability.

INTRODUCTION

Software systems are neither monolithic nor self-contained, but rather composed of models, specifications and working modules that are interrelated and need to fit together, usually by design. Decomposing a complex problem into several simpler and smaller artifacts, in a divide & conquer approach, is a fundamental software engineering technique to deal with complexity and improve design characteristics such as reusability, agility, changeability, adaptability and reliability.

An artifact can be any entity related to software engineering such as a concept, a specification or a program. The relationships between artifacts are essential to accomplish the goals of the software system but, at the same time, they create dependencies and coupling between them that translate into constraints and partially hinder these characteristics.

Therefore, software engineering can be described as the application of engineering principles, methods and techniques to computer-based artifacts under *quality* and *sustainability*

constraints. Quality means that the problem needs to be decomposed into the right artifacts and with the right relationships (satisfying the problem's specifications with a good architecture). Sustainability (Jardim-Goncalves, Popplewell & Grilo, 2012) means that changes in the problem specification or in its context should translate to incremental changes in the artifacts and their relationships, implemented at a faster rate than the changes that motivated them.

Quality and sustainability are not exclusive of software engineering. A car, for example, is a system with several thousand components that need to fit together perfectly, under the same constraints. What distinguishes software engineering is the fact that, in most cases, artifacts are virtual (easy to create and to destroy), very flexible and exhibit a high variability rate. A computer program can be changed in minutes or even seconds, which is certainly not the case of physical products such as cars.

This chapter concentrates on the sustainability side of software engineering and specifically in the relationships between software artifacts, in an attempt to improve the characteristics mentioned above. The basic tenets that we will use are:

- If an artifact *A* has no relationship with an artifact *B* (does not depend on it), then *B* can change freely without impacting *A*. This is good for sustainability. Ideally, all artifacts should be completely independent (decoupled from all other artifacts);
- Artifacts that have no relationship cannot cooperate, which means that no value comes from decomposing a system into artifacts. Any system needs that artifacts establish relationships and cooperate, somehow. This implies some coupling between some artifacts.

These are conflicting goals. The fundamental problem that we are trying to solve is how to get the best compromise, or *how to minimize cou-*

pling as much as possible while still satisfying the problem's specifications.

Relationships between software artifacts can be established at various levels, such as:

- **Conceptual:** Involving concepts such as strategies, goals and architectures. For example, different artifacts may cooperate towards some common goal or complementary goals.
- **Documental:** Which pertains mainly to specifications. For example, a given artifact must use the features defined by some standard.
- **Design:** Entailing the way artifacts are used to build a composed system. For example, any software development method will include a decomposition of the problem's specification and a composition of already existing artifacts (such as a software library), trying to match both approaches.
- **Operational:** In which working artifacts (such as software modules) interact by sending messages. The receiver of a message must be able to understand the content of a message and the intention of the sender in sending that message.

This means that relationships between artifacts are not limited to message based interaction but can occur at any stage of the artifacts' lifecycle, right from their conception, even if the artifact never becomes active and able to interact, such as a concept or a document.

At the operational level, in particular, it is also important to check whether artifacts can be bound together in a single application or are distributed, most likely in different computers and probably implemented in different programming languages. Solutions to support the interaction between these artifacts are different in both cases. The concept underlying meaningful relationships between artifacts that exchange messages is designated *interoperability* (Wang, Tolk & Wang, 2009),

26 more pages are available in the full version of this document, which may be purchased using the "Add to Cart" button on the publisher's webpage: www.igi-global.com/chapter/the-role-of-compliance-and-conformance-in-software-engineering/125300

Related Content

Market Response to ISO 9000 Certification of Software Engineering Processes
G. Keith Fullerand Ilan Vertinsky (2008). *Standardization Research in Information Technology: New Perspectives (pp. 203-213).*
www.irma-international.org/chapter/market-response-iso-9000-certification/29689

How High-Technology Start-Up Firms May Overcome Direct and Indirect Network Externalities
Mark Pruett, Hun Lee, Ji-Ren Leeand Donald O'Neal (2006). *Advanced Topics in Information Technology Standards and Standardization Research, Volume 1 (pp. 306-320).*
www.irma-international.org/chapter/high-technology-start-firms-may/4669

Analysis and Validation of Learning Technology Models, Standards and Specifications: The Reference Model Analysis Grid (RMAG)
Jan M. Pawlowskiand Denis Kozlov (2013). *Innovations in Organizational IT Specification and Standards Development (pp. 223-240).*
www.irma-international.org/chapter/analysis-validation-learning-technology-models/70701

Factors Influencing the Lifetime of Telecommunication and Information Technology Standards: Results of an Explorative Analysis of the PERINORM Database
Knut Blind (2007). *International Journal of IT Standards and Standardization Research (pp. 1-24).*
www.irma-international.org/article/factors-influencing-lifetime-telecommunication-information/2580

Analysis of Standards, Certifications and Labels for Bio-based Products in the Context of Sustainable Bioeconomy
Stefania Bracco, Özgül Calicioglu, Alessandro Flammini, Marta Gomez San Juanand Anne Bogdanski (2019). *International Journal of Standardization Research (pp. 1-22).*
www.irma-international.org/article/analysis-of-standards-certifications-and-labels-for-bio-based-products-in-the-context-of-sustainable-bioeconomy/249239