An Experimental Study of A Design-driven, Tool-based Development Approach

Quentin Enard, Christine Louberry, Charles Consel, Xavier Blanc

An Experimental Study of A Design-driven, Tool-based Development Approach

Quentin Enard, Christine Louberry, Charles Consel
INRIA / University of Bordeaux, France
first.last@inria.fr

Xavier Blanc
LaBRI / University of Bordeaux, France
first.last@labri.fr

***Abstract*—Design-driven software development approaches have long been praised for their many benefits on the development process and the resulting software system. This paper discusses a step towards assessing these benefits by proposing an experimental study that involves a design-driven, tool-based development approach. This study raises various questions including whether a design-driven approach improves software quality and whether the tool-based approach improves productivity. In examining these questions, we explore specific issues such as the approaches that should be involved in the comparison, the metrics that should be used, and the experimental framework that is required.**

I. Introduction

Design-driven software development approaches have been extensively studied in the past decades, as shown by the literature (*e.g.,* [1]–[3]). From the requirements specification, the design of a software system is described at a high level; it guides the various stages of the development process. There exists a variety of design-driven approaches, ranging from Architecture Description Languages (ADLs) (*e.g.,* [4]) to UML-based developments (*e.g.,* [5], [6]). These approaches rely on a notion of design language that provides syntax (whether or not textual) and abstractions allowing one to express the design of a software system.

Designing a software system prior to programming it is said to have a number of key benefits, in particular, regarding software quality [3]. Although there are few case studies analyzing software architectures [7], [8], there is no experimental study to demonstrate that a design-driven development approach improves software quality [9].

In this paper, we propose to conduct an experimental study to measure this assumption in the context of design artifacts used in a productive way. This study uses an existing design-driven approach, namely DiaSuite [10], [11]. DiaSuite is a tool-based approach ensuring the conformance between the design and the implementation of a software system. The goal of this study is to address the following research questions:

- **Q1**: Does a design-driven software development approach improve software quality?
- **Q2**: Does such an approach improve productivity when it is tool-based?

The reminder of this paper is structured as follows. Section II presents DiaSuite, a tool-based, design-driven software development approach. Section III describes our experimental study: (1) we propose a framework for our study and a data collection process; (2) we discuss how to analyze the collected data to address our research questions. Finally, Section IV summarizes our work.

II. DiaSuite

A key aspect of software design is the software architecture. It is aimed at decomposing an application into components, characterizing them with properties, and defining relations among them. This process is said to improve software quality in general but this benefit can widely vary depending on the effectiveness of the design in guiding the implementation. A design artifact can play a variety of roles in the development process, ranging from contemplative to productive. When a design language is used in a contemplative way, its syntax and semantics may remain informal, making challenging, if not impossible, the process of rigorously tracing the design in the implementation code. As such, contemplative approaches to design-driven software development are difficult to assess.

To promote design traceability, we need a design language with a syntax and semantics that are precisely defined, if not formally, at least in the form of a compiler. To match these requirements, our study relies on a suite of tools, named DiaSuite, that supports the programming of a software system from its design artifact written in a design language, named DiaSpec. This language is specific to a design paradigm in that it offers dedicated abstractions and semantics for designing applications. A compiler automatically generates programming support from a DiaSpec artifact, guiding the development process. The generated programming support ensures the conformance between the design and the implementation, enabling traceability.

A. A paradigm-based approach

DiaSpec is dedicated to the Sense/Compute/Control (SCC) paradigm and provides a conceptual framework for the design of applications. An SCC application is one that interacts with an environment, whether or not physical. The SCC paradigm targets a large spectrum of applications, from avionics and automotive to assisted living and healthcare. This generality prompted us to consider this approach as a paradigm-based approach rather than a domain-specific one.

As depicted in Figure 1, our SCC paradigm distinguishes three layers of components. The first layer consists of *entities*, whether hardware or software, that interact with an environment. For example, a temperature sensor and a heater actuator are *entities* of an application that regulates the temperature of a room. The second layer consists of *contexts* that process raw data produced by the *entities* and compute high-level information. The designer of the temperature regulation application can specify a context for the computation of the average temperature. The third layer consists of *controllers* that use context information to compute orders for the control of the actuating *entities*. An example of *controller* is one that uses the average temperature of a room to compute the heating level and the duration required to control a heater in a room.

To enable the designer to refine the description of a software system and provide more support for the implementation, DiaSpec makes explicit the data and the control flow of the components of an application in the form of interaction contracts [10]. For example, the designer can specify that an action on the heater may be triggered whenever the temperature sensor produces a new value.

A DiaSpec design is processed by DiaSuite that comprises a number of tools to support testing, programming, and deployment.

B. A tool-based approach

DiaSuite provides support for each development stage as depicted in Figure 2. A designer describes an application using the DiaSpec language (stage ①). From this description, the DiaSpec compiler generates a Java programming framework (stage ②) that guides the developers to implement the application (stage ③) and ensures the conformance with the design. Testing support is also provided (stage ④). The generated framework abstracts over communication technologies (*e.g.,* Web Services, OSGi) that can be selected for the deployment (stage ⑤).

From a design, the generated programming framework consists of one abstract class per DiaSpec declaration. Implementing a DiaSpec declaration amounts to extending the corresponding abstract class. The generated abstract classes make the programming framework both

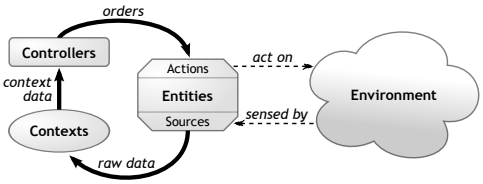

Figure 1: The Sense/Compute/Control paradigm

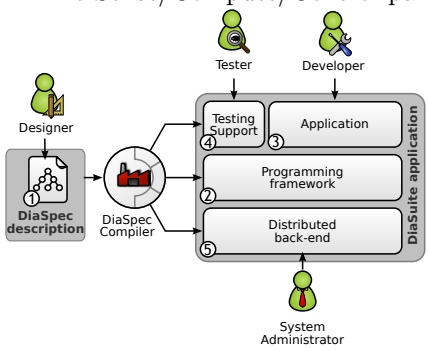

Figure 2: The DiaSuite tool-based development process

prescriptive and proscriptive: they provide high-level methods to interact with other building blocks while constraining developers to implement abstract methods that ensure the conformance with the design.

Presumably, a tool-based approach to design-driven development should improve the productivity of developers because of the generated programming support. However, the paradigm-specific nature of the DiaSuite approach may raise a new question.

- **Q3**: Does a paradigm-specific approach to designing applications have an impact on software quality?

III. Experimental Study

In this section, we present the experimental study of DiaSuite. We first propose a framework of this experimental study. Then we present a process to collect data from the experiment. Finally, we discuss how to analyze these data to address our research questions.

A. A framework for our experimental study

As a first step, we plan to compare DiaSuite to only one other approach. As DiaSuite is dedicated to the SCC paradigm and is tool-based, a key question is:

> What makes an approach pertinent to be used for a point of comparison with DiaSuite?

Another question concerns a baseline for comparison:

> What would make an adequate baseline for our comparative study?

Furthermore, since DiaSuite is based on Java,

> Should we consider that developing a software system in Java is the baseline for our comparative study?

The objective is to evaluate the benefits of DiaSuite on both software quality and productivity. For software quality, we plan to evaluate it via the evolutivity dimension. For software productivity, we will require the participants of the study to develop an SCC application using either DiaSuite or a programming-only approach (*i.e.,* Java); this development will be monitored. The characteristics of the application used for the experimental study has a key role. For example, using a design-driven approach for the development of a simple application may require unnecessary efforts from the developer. Alternatively, the development of a large application requires the participants to master the application domain. Besides, it is difficult to recruit participants for a long period of time.

Our proposed application is a simple avionics error monitor. This application is large enough to make the design stage relevant but it is simple enough to be developed in less than half a day.

To evaluate the evolutivity of the application, we propose to enrich it with additional features to be developed by the participants once the initial development is completed. An alternative would be to have participants extend each other's initial development.

B. Data collection process

To gather data about the participants and the application they develop, we propose to use a questionnaire and a versioning system.

1) Questionnaire: The questionnaire requires the participants to report on their level of expertise in Java. Indeed, the participants should be proficient in Java so that the language will not be an obstacle to the development.

To evaluate their productivity, we gather the participants' feedback about their productivity and any difficulty they might have had evolving the application.

2) Versioning system: To maximize our control on the experimental environment and minimize the overhead for the participants, the experimental infrastructure is running on a remote server. The versioning system tracks the evolution of the application developments: a script running on the server invokes a commit of the participant's workspace every minute. Our proposed setting allows to evaluate the participant's productivity by calculating the time he effectively spent on the application files. The effective time spent on the development is calculated by adding the time intervals between successive commits. Intervals that last more than a given time (*e.g.,* 15 minutes) are discarded to take into account the breaks during the development.

This versioning system also allows to estimate the effort that is required for the participants to make the application evolve. This estimation is based on the time spent on the evolution and the number of modifications the participants had to apply.

C. Data analysis

To answer the questions Q1, Q2 and Q3, we propose to analyze the following data gathered by our data collection process:

- The information from the questionnaire about the participants and their development
- The log of the versioning system
- The applications code.

Analyzing these data with respect to our questions requires to select pertinent metrics. For each question, we discuss the metrics that could be used.

1) Question Q1: To evaluate whether the DiaSuite approach improves software quality, we propose to compare the evolution of applications developed using DiaSuite to applications developed using Java only. The comparison relies on the following metrics: the time spent making the application evolve and the number of modifications required. Combining these two metrics should provide an accurate measure of whether software quality depends on the approach. We are still investigating other metrics to assess the evolutivity of applications. For example, others techniques, such as the use of artifacts' shared content [12], could be used. Beyond the evolutivity, other aspects of software quality (*e.g.,* modularity) could be measured by analyzing the application code with metrics such as complexity, cohesion and coupling [13], [14].

2) Question Q2: To evaluate whether our tool-based approach increases the productivity of developers, we propose to compare the time spent by the participants to develop their application using either DiaSuite or Java only. As for the analysis of the evolutivity, we use both the data from the questionnaire and the data from the versioning system. For a finer grained comparison, the time spent on the design and the time spent on the implementation by the DiaSuite participants could be distinguished. Another interesting metric could be the adequacy of the application to the specification. However, this metric is rather subjective.

3) Question Q3: Since we compare DiaSuite to a programming-only development approach, it could be difficult to distinguish between the impact of the tool-based approach and the paradigm on software quality. One step towards addressing this issue could be to evaluate to what extent our design language guides the participants for the design of their applications. A possible metric is the similarity between application design artifacts: if DiaSuite applications are more similar than the Java ones, then DiaSpec effectively guides the participants in their design. This similarity can be calculated by measuring the distance between the applications designs. Dedicated tools such as EMFCompare [15],

SiDiff [16] or DSMDiff [17] can be used to measure this distance. Nevertheless, to further investigate this question, additional studies are required. For example, we could require other participants to develop the same application, using another tool-based approach that is not based on the SCC paradigm. Considering Java as a baseline, the results of this new study could be compared to the existing results of the participants who used Java only.

IV. Summary

In this paper, we have presented a proposal for a study to assess the benefits of a design-driven software development approach, in terms of software quality and productivity. To conduct this study, we have chosen to compare the development of applications by participants using either Java or DiaSuite, a design-driven, tool-based software develop approach. Experimental data are gathered by questionnaires filled out by the participants and a versioning system that monitors the participants' activities. Finally, we discuss the relevance of metrics to be used to analyze the collected data.

The undergoing work consists of conducting the proposed study and analyzing the results. To address Question Q3, we aim to consider general-purpose approaches that are design-driven and tool-based, such as UML environments, to further our experimental study. We also plan to conduct other studies that target non-functional properties supported by DiaSuite such as quality of service and error handling [18], [19].

References

[1] R. N. Taylor, N. Medvidovic, and E. M. Dashofy, *Software Architecture: Foundations, Theory, and Practice.* Wiley, 2009.

[2] I. Sommerville, *Software Engineering (6).* Pearson Studium, 2001.

[3] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[4] M. Abi-Antoun, J. Aldrich, D. Garlan, B. Schmerl, N. Nahas, and T. Tseng, "Modeling and Implementing Software Architecture with ACME and ArchJava," in *Proceedings of the 27th international conference on Software engineering*, ser. ICSE '05. New York, NY, USA: ACM, 2005, pp. 676–677.

[5] Q. Sheng and B. Benatallah, "ContextUML: A UML-based Modeling Language for Model-driven Development of Context-aware Web Services," in *Mobile Business, 2005. ICMB 2005. International Conference on.* IEEE, 2005, pp. 206–212.

[6] T. Lodderstedt, D. Basin, and J. Doser, "SecureUML: A UML-based Modeling Language for Model-driven Security," *«UML» 2002—The Unified Modeling Language*, pp. 426–441, 2002.

[7] M. Babar and B. Kitchenham, "Assessment of a framework for comparing software architecture analysis methods," in *Proc. 11th International Conference on Evaluation and Assessment in Software Engineering, Keele, England.* Citeseer, 2007.

[8] M. Wermelinger, Y. Yu, A. Lozano, and A. Capiluppi, "Assessing Architectural Evolution: a Case Study," *Empirical Software Engineering*, vol. 16, pp. 623–666, 2011.

[9] D. Falessi, M. Babar, G. Cantone, and P. Kruchten, "Applying Empirical Software Engineering to Software Architecture: Challenges and Lessons Learned," *Empirical Software Engineering*, vol. 15, pp. 250–276, 2010.

[10] D. Cassou, E. Balland, C. Consel, and J. Lawall, "Leveraging Software Architectures to Guide and Verify the Development of Sense/Compute/Control Applications," in *ICSE'11: Proceedings of the 33rd International Conference on Software Engineering.* Honolulu United States: ACM, 2011.

[11] D. Cassou, J. Bruneau, C. Consel, and E. Balland, "Towards a Tool-based Development Methodology for Pervasive Computing Applications," *IEEE Transactions on Software Engineering*, Oct. 2011.

[12] T. Arbuckle, "Studying Software Evolution using Artefacts' shared Information Content," *Science of Computer Programming*, 2010.

[13] S. Chidamber and C. Kemerer, "A Metrics Suite for Object-oriented Design," *Software Engineering, IEEE Transactions on*, vol. 20, no. 6, pp. 476–493, 1994.

[14] F. Brito e Abreu and W. Melo, "Evaluating the Impact of Object-oriented Design on Software Quality," in *Software Metrics Symposium, 1996., Proceedings of the 3rd International.* IEEE, 1996, pp. 90–99.

[15] A. Toulmé and I. Inc, "Presentation of EMF Compare Utility," in *Eclipse Modeling Symposium*, 2006, p. 2009.

[16] C. Treude, S. Berlik, S. Wenzel, and U. Kelter, "Difference Computation of Large Models," in *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering.* ACM, 2007, pp. 295–304.

[17] Y. Lin, J. Gray, and F. Jouault, "DSMDiff: A Differentiation Tool for Domain-specific Models," *European Journal of Information Systems*, vol. 16, no. 4, pp. 349–361, 2007.

[18] J. Mercadal, Q. Enard, C. Consel, and N. Loriant, "A Domain-specific Approach to Architecturing Error Handling in Pervasive Computing," in *OOPSLA'10: Proceedings of the Conference on Object Oriented Programming Systems Languages and Applications*, Reno United States, 10 2010.

[19] S. Gatti, E. Balland, and C. Consel, "A Step-wise Approach for Integrating QoS throughout Software Development," in *FASE'11: Proceedings of the 14th European Conference on Fundamental Approaches to Software Engineering*, Sarrebruck Germany, 03 2011.