Chapter 9

Conclusions

This dissertation addresses how software architecture and design is to be represented, disseminated and coordinated in the context of global software development. To this end, the role of software architecture (as a process as well as an artifact) was empirically assessed in various industrial contexts. In addition, the special case of model-driven software development was addressed. This chapter contains a summary and integration of the findings that were presented in the previous chapters as well as an outline of future work.

9.1 Summary of Findings

In this section, the empirical evidence collected throughout this dissertation is summarized and used to address the research questions central to this dissertation (Section 1.3).

9.1.1 RQ1: How is Software Architecture Represented, Disseminated and Coordinated in the Context of Global Software Development?

This dissertation set out outlining how organizations tailor software development process descriptions for the challenges that GSD introduces, from a process perspective (Chapter 2). Investigating how software development process descriptions are tailored to accommodate for GSD, we found that the process approach to GSD is dependent on organization size, maturity, intended use of the description and the expertise and experience of the process engineers.

Subsequently, we presented and demonstrated our method to visualize GSD processes consistent with an iconic process visualization (Chapter 3). These visualizations uncovered aberrant distribution of analysis and design effort which were the result of

unclarities in the processes of communication and coordination of software architecture.

We then explicitly analyzed the role of software architecture design in the context in global software development by means of three case studies (Chapter 4). We found that some problems relating to software architecture dissemination and coordination processes led to poor architectural compliance. This, in turn, led to project overruns. The dissemination of software architecture as well as the role of the software architect are not formalized even though this might very well have benefited the development process. An important benefit of the application of MDD tools and techniques is that most of the software architecture is generated. As a result, architecture compliance improves. While this mitigates the problems associated with disseminating software architecture design, it introduces the problem of teaching developers to work with a Domain-Specific Language (DSL), associated tooling and the MDD approach in general.

Finally, we validated the findings from the case studies by means of a series of interviews with experts (Chapter 5). We then integrated the factors that influence how software architecture design is coordinated and disseminated and identified three main drivers to explain these factors:

1. First, the strong implementation focus of software development project management prematurely forces projects into the construction phase.

2. Second, a knowledge gap exists between the onshore and offshore location regarding software architecture and its role during the software development life cycle.

3. Third, cost reduction forces a move of responsibilities towards the offshore software development location. This compounds the "knowledge gap" problems as less resources are available for knowledge improvement (training) and more work is required of less experienced team members. In addition, the added value of activities related to implementation is more tangible than that of design-related activities. As a result, the "'implementation focus" problem is aggravated.

9.1.2 RQ2: How can we design software architecture documentation so that it is understood well by developers in the context of global software development?

We designed and executed an experiment in which we evaluated how software developers comprehend software architecture representations from the perspective of diagram-dominant versus text-dominant representations (Chapter 6). We found that neither diagrams nor textual descriptions are significantly more efficient in terms of communicating software architecture design. In addition, we found that diagrams were not able to alleviate the difficulties participants with a native language other

than English had in extracting information from the documentation. However, while diagrams were not superior regarding media effectiveness they still seemed to perform a special role. Participants were more likely to use diagrams as their first source. They were more likely to look at the diagram at the very moment when they provided answers to questions of a topological nature.

Finally, we identified developer characteristics that can be used as developer performance predictors: linguistic distance, media preference, experience and self-rated modeling skill. The participants who performed best had a native language close to English, looked at text more than at diagrams, were more experienced and rated their own modeling skills to be relatively high. We conclude that, contrary to current industrial practice, architecture documentation should be specifically tailored for its audience in terms of the developer's experience and native language and the general readability of the text.

9.1.3 RQ3: How does the application of model-driven development tools and techniques relate to the problems associated with global software development?

We analyzed how the characteristics of a large scale, industrial model-driven development project in the context of global software development compare to non-MDD projects (Chapter 7). In MDD, models instead of code are the central development artifact. We found that the same logic applied to code cannot be applied to models: First, the majority of development effort was spend on developing the models. That is significantly more than the time spent on code development in classical software development. Second, most model elements were already present at one third of the development process. The remaining development time was spent on altering the models. In addition, 40 percent less defects were found in the MDD case when compared to projects of similar size. Models and code are fundamentally different and therefore not easily compared as we found absent, for example, a positive relation between model size and model complexity on the one hand and model defects on the other - relations that have often been observed in source code. Also, larger diagrams were changed more often and worked on longer but did not necessarily contain more defects.

We then analyzed how the application of MDD tools and techniques specifically impact the problems associated with Global Software Development (Chapter 8). We found that the use of models as a common language mitigated some of the problems associated with socio-cultural distance and also resulted in fewer traveling back and forth between the offshore and onshore locations. In addition, MDD techniques in general and shared model ownership in particular forces more frequent interaction between more team members.

Finally, an important implication of the use of code generation is that the software

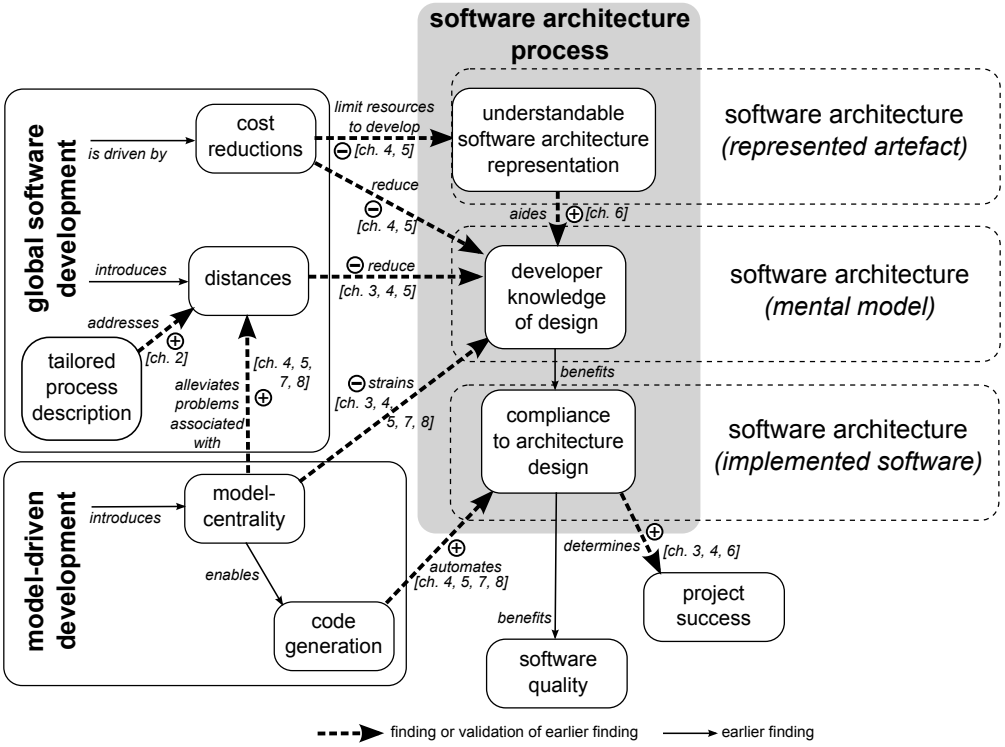

Figure 9.1: *Cause-effect diagram integrating the main findings of the study*

architecture is generated. Software architecture compliance is therefore automated. Nevertheless, the application of MDD required more formal artifacts in terms of e.g. more extensive and detailed design documentation and models that strictly adhere to modeling guidelines. The architect played a central role in the continuous additional training that team members required.

9.2 Contributions

The main contributions of this study and their interrelations are visualized in Figure 9.1. First, this dissertation establishes that *the software architecture process is significantly different in the context of GSD* when compared to co-located development. Second, we have demonstrated a clear link between architecture compliance and project success in terms of limiting rework. Third, we have presented evidence that the *cost reductions inherent to GSD limit the resources available for software architecture design and representation*. Fourth, we have found how developer knowledge of software architecture design:

1. can be positively influenced by improving diagrams and text in software archi-

tecture representation and training developers;

2. is limited by the socio-cultural, geographical and temporal distances in that GSD introduces;
3. is hampered by cost reductions which limit the resources available for coordination of software architecture;
4. is less important in the context of MDD because the architecture is a stable framework that is already implemented to enable code generation. Instead, however, developers need to learn to work with a new development paradigm, new tooling and a DSL.

Fifth, we found evidence for the principle that code generation increases the extent to which a software implementation complies to its intended architecture as much of the architecture is generated.

An important contribution that is not explicitly modeled in Figure 9.1 is that the use of models as a central development artifact is fundamentally different from using code. The logic or intuitions regarding effort and amount of defects that we have when it comes to source code cannot be applied to models. However, the implications of application of MDD tools and techniques are not yet clearly understood and therefore warrant further study.

9.3 Recommendations to Industry

Following from the findings of this dissertation, we formulate five recommendations:

1. *The problems associated with GSD should be specifically addressed in software development process descriptions*
   Such an addition might be as informal as a (concrete) list of best practices. Problems that were encountered and solved in a specific project by means of technology or a (set of) best practice(s) should find their way to the process description so that other projects may benefit in the sense that similar mistakes are not repeated. Given that the majority of problems in GSD is thought to benefit from intra-team member communication-related practices, the intended audience of such an augmented process description should be all team members, rather than project management.
2. *The processes of dissemination and coordination of software architecture must be explicitly formalized*

   - Preferably, one or more experienced offshore developers should be involved in the development of the architecture under guidance of an experienced onshore architect.

- The most fundamental aspects of the architecture design should be developed before starting offshore construction.
- If budget allows, the development of an architecture POC in which new, complex or otherwise unknown functionality is addressed, is recommended.
- The most knowledgeable architect should travel to the offshore location at least once, preferably when the construction phase commences.
- When disseminating software design, validate that the receiving party made the correct interpretation.
- The availability of the principal (onshore) architect should only be limited to the extent that the knowledge and experience of the principal offshore developer (the "technical lead") allows this. Any cost savings from limited association of the "expensive" onshore architect are unlikely to offset the costs incurred by rework as a result of architectural noncompliance.
- Offshore developers should be able to directly contact the software architect — preferably in a group so that effective use can be made of the architect's time. These sessions should be planned regularly to avoid developers having to batch their questions.
- Developers should be coached to understand that knowledge of the role of "their" component in relation to other components matters to the extent that it determines the quality of their work.

3. *Increase allocation of resources in architecture design*
   Upfront investment in architecture design is likely to lower budget overrun due to having rework a faulty architecture implementation.

4. *Create unambiguous and concise software architecture documentation that is specifically tailored to its intended audience*
   Pay particular attention to the use of both text and diagrams, even for topological information and annotating each (non-UML) diagram with a description of how to read it. Additionally, investments in UML training for developers benefits developer understanding of architecture representation.

5. *Consider application of MDD tools and techniques to reduce the negative impact of communication-related GSD problems*
   The use of models as a common language eases communication between on- and offshore teams and enables a larger group of stakeholders to participate in implementation-related discussions.

9.4 Future Work

As with all good research, the value of our findings in great part lies in the impetus they provide to perform further research. We describe relevant directions for future

work in the next sections.

9.4.1 In-dept Evaluation of the Role of Documentation in GSD

We argued that software documentation plays a more central role in GSD than in co-located software development. In industrial practice, however, documentation is often regarded as a by-product and information is preferably shared directly and informally between people — hence the popularity of documentation-light Agile approaches. We take the point of view that not more or less documentation must be created, but *better* documentation. Nevertheless, advanced methods of documentation generation, an increase in the level of technical maturity of clients and the advent of team wikis do question the role of documentation in the software development process. Future work in this direction should aim to quantify the usefulness of documentation by, for instance, mapping the match between information need and availability and evaluating the extent to which parts of existing documentation is perceived as adding value. The outcome of such a study would be more lean documentation templates, a knowledge support system and/or best practices for knowledge codification practices. The action research paradigm is likely to be a suitable research method to address this topic.

9.4.2 Quantifying the Relation Between Developer Architecture Design Understanding and Software Quality

This dissertation provides evidence which supports recommending upfront investments in architecture development and representation to ensure developer architecture design understanding. While we take the stance that understanding of software architecture design is beneficial for architecture compliance, we are unsure about the mechanism underlying the nature of this relation. For example, how much should a developer know about an architecture to ensure his software is compliant? All of it? That seems uneconomical. Is it perhaps enough to understand the relation between "his" components and neighboring parts of the system? To summarize: How much better would developers implement if they know more about architecture? Such a topic could be addressed by means of a quasi-experiment in the sense that a researcher could invest in educating a selected group of developers with regards to the software architecture.

9.4.3 Facilitating industrial application of MDD

This dissertation contains evidence for the significant impact of shifting from code to models as central development artifacts. We also know that industrial application of MDD is slow and that limited evidence exists for MDD's potential benefits. Various reasons have been offered to why MDD seems used so little and so often to no avail. Reasons include unrealistic expectations, the problematic offering of MDD tooling and

a general lack of understanding of the concept of MDD. Future work should preferably investigate industrial cases of MDD to collect factors that contribute to successful industrial application.