Making the Right Decision: Supporting Architects with Design Decision Data

Jan Salvador van der Ven[1] and Jan Bosch[2]

[1] Factlink, Groningen, the Netherlands
[2] Chalmers University of Technology Gothenborg, Sweden
mail@jansalvador.nl, jan.bosch@chalmers.se

Abstract. Software architects are often forced to make design decisions based on limited information. In this paper, we present an approach that allows software architects to study information about design decisions made by hundreds or more software architects by automatically analyzing the version management data of large open-source repositories. The contribution is, first, that it develops a conceptual model to reason about the automatic derivation of specifically medium level architectural design decisions. Second, we show that it is indeed possible to derive these design decisions automatically from open source projects. This provides a basis for statistical and quantitative reasoning about software architecture design decisions that allows software architects to make better-informed decisions.

Keywords: Architecture, Design Decisions, Architectural Knowledge, Components, Open Source Projects.

Introduction

Architects are lonely [1] because they often are the only ones with a system-wide overview and have no peers within the organization. These architects are responsible for making design decisions concerning the system or systems that they are responsible for. A significant portion of these design decisions involves the selection of 3rd party open source or commercial components. In our experience with architects at dozens of companies, this selection is done based on descriptions on websites, anecdotal experiences or sometimes proof of concept implementations [2]. Consequently, despite the best intentions and efforts of the software architect, the design decisions often are guesses based on circumstantial evidence that are only validated once the system has been built or changed and it is, once again, in operation. The vast majority of decisions that software architects are faced with have been made earlier software architects in other organizations working on similar systems. Wouldn't it be great if software architects could get access to the decisions made by other architects, that would allow them to determine what selections were made from a set of alternatives and with what frequency? That would give software architects hard, quantified data to base their own decisions on. The question of course is: how we can access these decisions? Interestingly, over the last decade or more, several open-source software repositories have achieved broad adoption and host thousands

of projects in virtually any programming language and application domain imaginable. Examples include SourceForge[1] and GitHub[2] with millions of repositories millions of developers. As many of the projects in these repositories are public, there is a large amount of data available about the structure of these projects as well as the evolution of these structures over time. In order to provide the lone software architects with objective, quantified and statistical information about the design decisions that other architects have made, version management systems provide an excellent source of data. However, considering the sheer volume of data, this requires an automated, rather than manual, approach to derive the information that software architects require. In order to achieve that, the first question, which is the research question of this paper, is whether it is feasible to automatically identify design decisions from commit data.

The contribution of this paper is twofold. First, it develops a conceptual model to reason about the automatic derivation of architectural decisions. Second, it shows that it is indeed possible to derive these design decisions automatically. This paper is organized as follows. First, the concept of architectural decision is introduced. Then, our hypotheses are presented. Sequentially, a description of how we acquired and processed the data is given, followed by the analysis or our results. This paper ends with the discussion and future work, related work and some concluding words.

Architectural Design Decisions

In research about architectural design decisions [3, 4], typically four aspects of decisions are considered: the *decision topic*, the *choice*, the *alternatives* that are considered and the *rationale* (sometimes formalized as ranking) of the decision. We use these four aspects of architectural decisions to identify decisions in repository data of open source projects. There are different abstraction levels of architectural decisions. As described by de Boer et al. [3], decisions are often related to each other, and this relationship typically forms a tree structure down from more abstract to more concrete (decisions cause new decision topics). Fig. 1. symbolically visualizes such a graph. Generally speaking three levels of decisions can be distinguished:

- **High-level decisions.** High-level architecture decisions affect the whole product, altough they are not necessarily always the decisions that are debated or thought through the most. Often, people that are not involved in the realization of the project (e.g. management or enterprise architects) heavily affect these decisions. Typical examples of high-level decisions are the choice to adopt an architectural style, use a programming language, application server, or specific large (COTS) components. Changing these decisions typically has a huge impact.
- **Medium level decisions.** Medium level decisions involve the selection of specific components or frameworks, or describe how specific components map to each other according to specific architectural patterns. These decisions are often debated in the architecture and development teams and are evaluated, changed and discarded as

[1] http://sourceforge.net/

[2] https://github.com/

needed. They have a high impact on the (nonfunctional) properties of the product and are relatively expensive to change.

- **Realization level decisions.** Realization level decisions involve the structure of the code, the location of specific responsibilities (e.g. design patterns), or the usage of specific APIs. These decisions are relatively easy to change, and have relative low impact on the properties of the system.

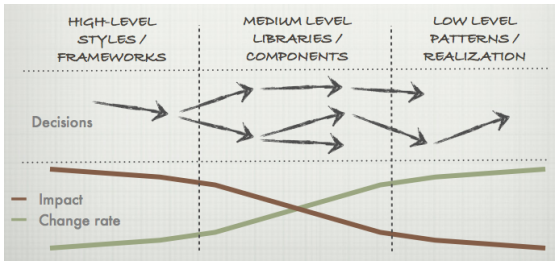

Fig. 1. Relationships between Design Decisions.

As we have experienced in our industrial cases [5], the architectural decisions that are hardest to make are the medium level decisions, for the following reasons: a) these decisions have a *high impact* on the functional and non-functional properties of the system; b) they *change constantly*, especially compared to high-level decisions that only change when remaking the system; c) they are *costly to change* because of the impact on the system; d) because new components and version are created constantly, it is *hard to stay knowledgeable* about all relevant alternatives, and; e) they have *unpredictable results* until they are implemented in the system.

The focus of this paper is on medium level design decisions that change during development or maintenance. These decisions express themselves through changes in the version management system, i.e. commits of new and changed code. All of the previously mentioned aspects of a design decision have a reflection in the version history or implementation of the system. The decision topic and the choice have a reflection in the (architecturally relevant) commits. The rationale for the decision can be reflected in the commit message, and the author of the commit can be contacted for additional rationale. Alternatives have reflection in the history of the architecturally relevant commits.

Hypotheses

The research approach we utilize consists of the following steps. First, we formed hypotheses about how design decisions are potentially represented in the version history of projects. Second, we selected a set of projects to test our hypotheses on. From these projects, we generated the data that potentially contained architectural design decisions, rationale and information about alternatives. This data was used to validate our hypotheses in a quantitative and qualitative way. The following three hypotheses are used in the remainder of this paper:

- *Hypothesis 1: Medium level design decisions can be identified in the version history of projects.*
- *Hypothesis 2: Commits in version management systems contain rationale of the made architectural decisions.*
- *Hypothesis 3: Alternatives can be found in the structure of commits in version management systems.*

Mining Git Repositories

This section describes how the data from the git repositories was processed to usable data in the Gitminer tool. We have looked at projects that contained a Gemfile, that were used by the community (>1 watcher, >1 fork), were active (change in last month) and were of moderate size (between 0,3 and 10Mb). From Google's BigQuery API[3] we have searched project urls that satisfied these criteria, resulting in 710 projects.

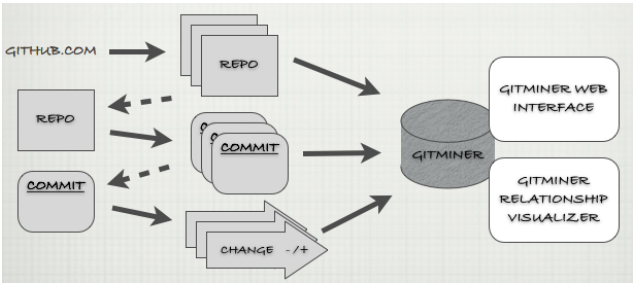

Fig. 2. Processing of Repos to Gitminer.

For the processing of the data from the repos, we used git[4] tooling. We only looked at the history of the Gemfile, as this file contained information about used components. In Fig. 2. this processing is visualized. First, a set of repos is selected and cloned to a local computer. Then, every commit on the Gemfile is taken, and every line that changed in the Gemfile within the commit is processed to a database insert query. To do this, we have automatically processed the output from the *git log* command, which outputs the history of a file. At last we inserted the queries to the database.

Table 1. Acquired Repo Data.

Parameter	Total #
Projects involved	710
Total commits	12600
Total commit messages > 30 characters	7527
Total changed lines	40464

[3] https://developers.google.com/bigquery/
[4] http://git-scm.com/

In our processing, we removed lines that were added and removed in the same commit (typically a copy-paste of lines to a different location in the same file) and lines that did not concern gems (but, for example comments). A summary of the resulting amounts after the above-mentioned steps is presented in **Table 1**.

Analyses Tool: Gitminer

In order to proof or disproof the hypotheses posed in this paper, we created two different 'views' on the data in the Gitminer database. The *Gitminer Web Interface* is a web interface that enables users to browse thought commits. This tool shows the commits that *removed* the component you are looking for, as this indicates that a specific decision was made to remove (or replace) a component. The view includes the commit messages, the date of the commit and the contact details of the authors. In this view you can see for every component: A) what replacements of this component are often used, and B) what this component was often replaced for. As a second view, the *Gitminer Relationship Visualization* provides a visual way to identify what components are related. A state machine displays a relationship when a component was removed in the same commit as where another component was added. A number representing the amount of projects accompanies the arrows in the diagram. In this view it is possible to see patterns of relationships between components.

Results

For our *quantitative* results, we have presented 100 different commits to six subject matter experts. We randomly picked 100 commits from the Gitminer database that had commit messages of more then 30 characters (therefore, had a solid chance of containing rationale). We distributed the commits among our subject matter experts. The participants that conducted the research were experienced Ruby software developers, experienced software architects, and researchers with software engineering background and experience. We asked them to answer if the presented commit involved a design decision, rationale for a decision and relevant information about alternatives for a decision.

Table 2. Quantitative Results.

%	Decision?	Rationale?	Alternatives?
% Yes	61,75 %	25,50 %	4,75 %
% No	38,00 %	68,75 %	84,00 %
% Empty	0,25 %	5,75 %	11,25 %

During our analysis of the data collected with the Gitminer tool, we found additional *qualitative* results in addition to the quantitative results. We identified different aspects related to design decisions, that we used as expert validation:

- There were commit messages that indicated changes of components and rationale about them. E.g. "use mysql2 instead of mysql because of shit encoding".

- Commit messages where a decision is made, but the rationale was clearly missing: "Changed to jeweler2", or "remove thin"
- Many messages described configuration issues: "Unfortunately, we can't put ruby-debug in the gemfile because it breaks 1.9.2 compatibility …".

Analysis

As shown in **Table 2**, roughly 60% of the commits on Gemfiles were considered as concerning a design decision. For our whole dataset, this would mean that 60% of the 7527 commit messages contains decisions (~4500). Of course, the other commit messages (with < 30 characters) could also contain decision information, so this number could very well be higher. Calculated in the same way, about 1900 commit messages contain rationale about made decisions. When relating this to the number of projects, on average every open source project we used contained ~ 6 decisions in commits and ~3 commit messages with relevant rationale. Following, we will discuss our hypotheses and related results.

Hypothesis 1: Medium level design decisions can be identified in the version history of projects. The subject matter experts have identified architecture design decisions in the commit messages. This is a clear quantitative indication that decisions exist in the commit messages of open source projects. Qualitatively, the researchers found several interesting design decisions. This qualitatively strengthens the validity of this hypothesis. Concluding, hypothesis 1 is confirmed by our data.

Hypothesis 2: Commits in version management systems contain rationale of the made architectural decisions. Rationale was found in 25,5% of the commits that were inspected by the subject matter experts. Qualitatively the researchers found rationale in many cases. This qualitatively strengthens the validity of this hypothesis. Concluding, hypothesis 2 is also confirmed by our data.

Hypothesis 3: Alternatives can be found in the structure of commits in version management systems. Our subject matter experts have not found many alternatives in the commit messages from the version management system (< 5%). So, quantitatively we have no evidence that alternatives can be found. So, hypothesis 3 is *not* confirmed by our data. However, alternatives were clearly identified by the researchers in the Relationship Visualization. So, based on the qualitative data we still think we could be able to find information about alternatives, but not solely in the commit messages.

Discussion and Future Work

In order to be able to discuss the architectural design decisions we discovered in this paper, we had to scope the definition of these decisions. We selected medium level design decisions that concerned component selection. For the validation we have taken only the commit messages that contained more then 30 characters. Based just on this research, it is tough to generalize to other kinds of architectural decisions.

We assumed that every added or removed line in the Gemfile was a potential decision. However, in Gemfiles there are non-gem lines too. For example, there can

be conditional lines or groups that are only called in specific situations. We have chosen to remove those lines. Hence, dependent on how often this happens, it could be that some of our found decisions are conditional.

As a reflection on our research, we considered the commit messages to be very useful in understanding what happened in a project. However, the messages were sometimes cryptic and short. In that case, data from multiple projects needed to be used for making similar decisions. The component relationships were interesting from a research point of view as an indication for dependencies and alternatives.

We are investigating ways to increase the number of repositories in the database, to be able to base the advice on a larger data set. In addition to this, we are planning to experiment with our approach on other programming languages. For example, the pom file of Maven (Java) projects could be used similar to Gemfiles. Also, we are working on making the results accessible to the public. As noted by several people that studied our results, the results could be used to statistically advice people about their architecture.

Related Work

There has been much attention to documenting software architectures [6], as well as documentation templates [7] and computational modeling [8]. A topic that is being discussed heavily is the role of the architect [1] and the role of 'the architecture document' in the design process [5]. Here, often the architect is responsible for creating and maintaining the architecture documentation. However, the architect is never supported in making these decisions in any way.

In the architecture design decision research hierarchical structures are used to model architectural knowledge [3] or design decisions [9, 10]. This research often emphasizes the recording of decisions, and the extraction of decisions later in the development process. However, we have not find any work where statistical data is used to help architects make better decisions. Dagenais and Robillard [11] investigated open source development for finding decisions based on surveys and documentation. Another mining initiative involves searching open source java frameworks [12], that focuses on code fragments instead of architectural decisions.

On the web, there are several initiatives that provide statistical data about software projects. For example, there are tools that help developers increase code quality by providing statistics about the code [13]. However, to the best of our knowledge no research or practical solution exists that actually searches for design decisions in the version history of software projects.

Conclusions

In this paper, we have given architects the first step to a wider knowledgebase for relying their architecture decisions on. We have shown that it is possible to extract architectural design decision information from the version management of open

source projects, by automating the process of extracting the decisions, and validated that architectural decisions can be derived from commits on the system.

Architects that are facing problems related to selecting components can benefit from this, by seeing what happened in similar situations in other software projects. The information presented in this research is based on real world projects, which are actually used, build and maintained around the world. By using this information, architects can be supported statistically for making their decisions.

Acknowledgements

We would like to thank the subject matter experts for helping us with the research. Also, we would like to thank the people of Factlink for the help during the development of our theory and tooling.

References

[1] Farenhorst, R., Hoorn, J. F., Lago, P., & van Vliet, H. (2011) The Lonesome Architect. J. Syst. Softw. **84** (9): pp. 1424-1435.
[2] Gorton, I., Liu, A., & Brebner, P. (2003) Rigorous Evaluation of COTS Middleware Technology. IEEE Computer **36** (3): pp. 50-55.
[3] de Boer, R. C., Farenhorst, R., Lago, P., van Vliet, H., Clerc, V., & Jansen, A. (2007) Architectural knowledge: getting to the core. In: Proceedings of the Quality of software architectures 3rd international conference, 2007.
[4] Tyree, J. & Akerman, A. (2005) Architecture Decisions: Demystifying Architecture. IEEE Softw. **22** (2): pp. 19-27.
[5] van der Ven, J. S. & Bosch, J. (2013) Architecture Decisions: Who, How and When?. In: To be published in: ASA, Agile Software Architectures.
[6] Clements, P., Garlan, D., Bass, L., Stafford, J., Nord, R., Ivers, J., & Little, R. (2002) Documenting Software Architectures: Views and Beyond, Pearson Education.
[7] Kruchten, P. (2003) The Rational Unified Process: An Introduction, Addison-Wesley.
[8] OMG (2012) UML Specification, Version 2.0. Online at: http://www.omg.org/spec/UML/.
[9] Jansen, A. G. J. & Bosch, J. (2005) Software Architecture as a Set of Architectural Design Decisions. In: Proceedings of the 5th IEEE/IFIP Working Conference on Software Architecture (WICSA 2005).
[10] van der Ven, J. S., Jansen, A., Nijhuis, J., & Bosch, J. (2006) Design Decisions: The Bridge between Rationale and Architecture. Rationale Management in Software Engineering, Springer: pp. 329-348.
[11] Dagenais, B. & Robillard, M. P. (2010) Creating and evolving developer documentation: understanding the decisions of open source contributors. In: Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering.
[12] Thummalapenta, S. (2008) SpotWeb: Detecting Framework Hotspots via Mining Open Source Repositories on the Web. In: in Proceedings of the 2008 International Workshop on Mining Software Repositories (MSR).
[13] Bluebox (2013) Code Climate. Online at: https://codeclimate.com/.