Software Architecture: The Next Step

Jan Bosch

University of Groningen, Department of Computing Science
PO Box 800, 9700 AV Groningen, The Netherlands
Jan.Bosch@cs.rug.nl
http://segroup.cs.rug.nl

Abstract. This position paper makes the following claims that, in our opinion, are worthwhile to discuss at the workshop. 1) The first phase of software architecture research, where the key concepts are components and connectors, has matured the technology to a level where industry adoption is wide-spread and few fundamental issues remain. 2) The traditional view on software architecture suffers from a number of key problems that cannot be solved without changing our perspective on the notion of software architecture. These problems include the lack of first-class representation of design decisions, the fact that these design decisions are cross-cutting and intertwined, that these problems lead to high maintenance cost, because of which design rules and constraints are easily violated and obsolete design decisions are not removed. 3) As a community, we need to take the next step and adopt the perspective that a software architecture is, fundamentally, a composition of architectural design decisions. These design decisions should be represented as first-class entities in the software architecture and it should, at least before system deployment, be possible to add, remove and change architectural design decisions against limited effort.

Introduction

For those that have followed the emergence of the notion of software architecture over the last decade or more, the field must have represented a classical example of technology innovation. Although references to the concept of software architecture were made earlier, the paper of Perry and Wolf [4] formed the starting point for an evolving community that actively studied the notion and practical application of software architecture. In the years to follow, software architecture was broadly adopted in industry as well as in the software engineering research community. Today, virtually all conferences in the field of software engineering mention software architecture as a topic of interest and in industry is the role of software architect well established.

The definition of software architecture, although for a long time an issue of lively debate, has been concluded (for now) with the adoption of the IEEE 1471 standard that defines software architecture as the fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution [2]. With this definition, the component and the connector are reinforced as the central concepts of software architecture.

The research in the area of software architecture, including workshops, e.g. the ISAW series, conferences, e.g. the WICSA series, special issues of journals and books, has focused on the careful design, description and assessment of software architecture. Although some attention has been paid to evolution of software architecture, the key challenge of the software architecture community has been that software architectures need to be designed carefully because changing the software architecture of a system after its initial design is typically very costly. Many publications refer to changes with architectural impact as the main consequence to avoid. Interestingly, software architecture research, almost exclusively, focuses on this aspect of the problem. Very little research addresses the flip side of the problem, i.e. how can we design, represent and manage software architectures in such a way that the effort required for changes to the software architecture of existing systems can be substantially reduced. As we know that software architectures will change, independent of how carefully we design them, this aspect of the problem is of particular interest to the community.

To reduce the effort in changing the architecture of existing software systems it is necessary to understand why this is so difficult. Our studies into design erosion and analysis of this problem, e.g. [1] and [3], have led us to believe that the key problem is knowledge vaporization. Virtually all knowledge and information concerning the results of domain analysis, architectural styles used in the system, selected design patterns and all other design decisions taken during the architectural design of the system are embedded and implicitly present in the resulting software architecture, but lack a first-class representation. The design decisions are cross-cutting and intertwining at the level at which we currently describe software architectures, i.e. components and connectors. The consequence is twofold. First, the knowledge of the design decisions that lead to the architecture is quickly lost. Second, changes to the software architecture during system evolution easily cause violation of earlier design decisions, causing increased design erosion.

The message of this position paper is twofold. First, we claim that the traditional software architecture research addressing the components and connectors has matured and disseminated to industry to an extent that diminishes the relevance of further research in this domain. The first phase of software architecture research has, in this sense, ended. Second, this does not mean that research in software architecture is finalized. Instead, we need to fundamentally change our view on software architecture. Rather than components and connectors, we need to model and represent a software architecture as the composition a set of architectural design decisions, concerning, among others, the domain models, architectural solutions, variation points, features and usage scenarios that are needed to satisfy the requirements. Once we are able to represent software architectures, in several phases of the lifecycle, in terms of the aforementioned concepts, changing and evolving software architectures will be considerably simplified.

The remainder of the position paper is organized as follows. In the next section, we discuss the problems of the current perspective on software architecture. Subsequently, we present what in our opinion should be the future perspective on software architecture. Then, we present the concept of an architectural design decision and architectural fragment. Finally, the paper is concluded with a short summary of our position and statements.

Problems of Software Architecture

Software architecture has become a generally accepted concept in research as well as industry. The importance of stressing the components and their connectors of a software system is generally recognized and has lead to better control over the design, development and evolution of large and increasingly dynamic software systems.

Although the achievements of the software architecture research community are admirable, we should not let ourselves believe that no issues remain in this area. In this section, we present a set of problems that we feel are of particular importance to be addressed in the next phase of software architecture research. The discussion of these problems is organized around the concept of an architecture design decision. We define an architecture design decision as consisting of a restructuring effect on the components and connectors that make up the software architecture, design rules imposed on the architecture (and resulting system) as a consequence of the design decision, design constraints imposed on the architecture and a rationale explaining the reasoning behind the decision. In our definition, the restructuring effect includes the splitting, merging and reorganization of components, but also additional interfaces and required functionality that is demanded from components. For instance, when a design decision is taken to use an object-oriented database, all components and objects that require persistence need to support the interface demanded by the database management system. Architecture design decisions may be concerned with the application domain of the system, the architectural styles and patterns used in the system, COTS components and other infrastructure selections as well as other aspects needed to satisfy all requirements.

The problems that are inherently present in the current definition of software architecture and insufficiently addressed by the research community are the following:

- **Lack of first-class representation**: Architecture design decisions lack a first class representation in the software architecture. Once a number of design decisions is taken, the effect of individual decisions is implicitly present, but almost impossible to identify in the resulting architecture. Consequently, the knowledge about the "what and how" of the software architecture is quickly lost. Some architecture design methods stress the importance of documenting architecture design decisions, but experience shows that this documentation often is difficult to interpret and use by individuals not involved in the initial design of the system.
- **Design decisions cross-cutting and intertwined**: Architecture design decisions typically are cross-cutting the architecture, i.e. affect multiple components and connectors, and often become intimately intertwined with other design decisions.
- **High cost of change**: A consequent problem is that a software architecture, once implemented in the software system, is, sometimes prohibitively, expensive to change. Due to the lack of first-class representation and the intertwining with other design decisions, changing or removing existing design decisions is very difficult and affects many places in the system.
- **Design rules and constraints violated**: During the evolution of software systems, designers, and even architects, may easily violate the design rules and constraints imposed on the architecture by earlier design decisions.
- **Obsolete design decisions not removed**: Removing obsolete architecture design decisions from an implemented architecture is typically avoided, or performed only

partially, because of the (1) effort required, (2) perceived lack of benefit and (3) concerns about the consequences, due to the lack of knowledge about the design decisions. The consequence, however, is the rapid erosion of the software system, resulting in high maintenance cost and, ultimately, the early retirement of the system.

Software Architecture: The Next Step

In the first two sections, our aim has been to convince the reader of two things. First, the first phase of software architecture research, where the key concepts are components and connectors, has matured the technology to a level where industry adoption is wide-spread and few fundamental issues remain. Second, the traditional view on software architecture suffers from a number of key problems that cannot be solved without changing our perspective on the notion of software architecture.

In our research, we have evolved to an approach to software architecture that addresses the problems discussed in the previous sections. The key difference from traditional approaches is that we do not view a *software architecture* as a set of components and connectors, but rather as *the composition of a set of architectural design decisions*.

An architectural design decision consists of a solution part and a requirements part. The solution part is the first-class representation of a logically coherent structure that is imposed on the design decisions that have already been taken. The design decision may (1) add components to the architecture, (2) impose functionality on existing components, (3) add requirements on the expected behaviour of components and (4) add constraints or rules on part or all of the software architecture. A design decision may not (1) remove components from the software architecture, although components may be isolated and (2) split or merge components, although part of the functionality of a component may be ignored.

The requirements part of a design decision is present because of the fact that no design decision can be a complete architectural design, except for perhaps toy examples. Each design decision addresses some of the requirements of the system, but leaves others to be resolved. In addition, a design decision may introduce new requirements on components in the system.

An architectural design decision may represent a number of solution structures, including a reusable domain architecture, an architectural style or pattern, a design pattern or a component selection, e.g. a COTS component, a reusable component from a system family or a system specific component. In the next section, we define the concept of an architecture design decision in more detail.

Architecture Design Decisions

Designing a software architecture can be viewed as a decision process, i.e. a sequence of easy and difficult design decisions. The software architect, in the process of translating requirements into a design, basically takes, potentially many, design decisions that, combined, give the software architect its final shape.

We, as a software design community, have a tendency to freely speak about design decisions, but few have addressed the notion of design decisions in detail. In our experience, one can identify four relevant aspects of a design decision:

- **Restructuring effect**: The most visible effect of a design decision is the effect on the structure of the software architecture. As a consequence of the design decision, components may be added, split, merged or removed. Although this effect is very clear at the time the decision is taken, the accumulated restructuring in response to a series of design decisions is not at all so easy to understand any more. Current notations and languages do not provide support to describe design decisions individually.
- **Design rules**: A second aspect of a design decision is that it may define one or more design rules that have to be followed for some or all components in the system. A rule typically specifies a particular way of performing a certain task.
- **Design constraints**: In addition to design rules, a design decision may also impose design constraints on a subset of the components. A design constraint defines what the system, or parts of it, may not do.
- **Rationale**: Finally, each design decision is taken in response to some functional or quality requirement(s). The software architect reasons about the best way to fulfill the requirement(s) and then decides. The rationale, including possible new principles, guidelines and other relevant information about the design of the system, should be documented.

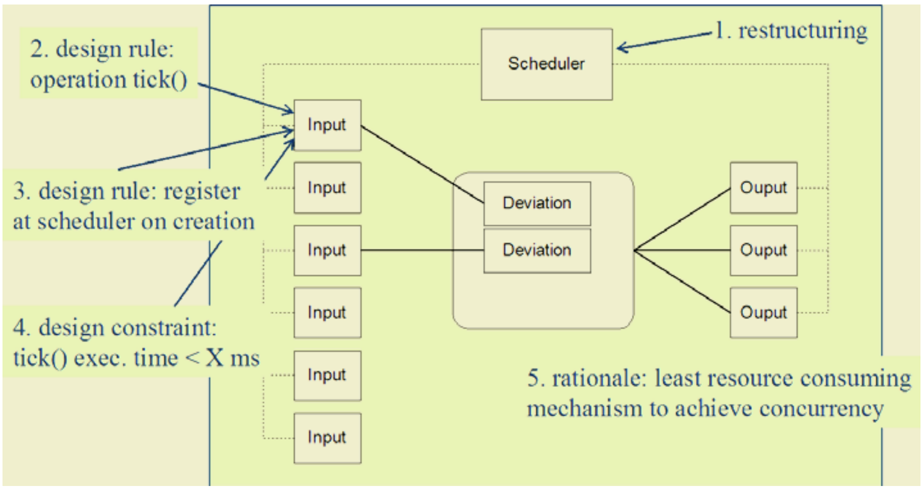
1. restructuring
2. design rule: operation tick()
3. design rule: register at scheduler on creation
4. design constraint: tick() exec. time < X ms
5. rationale: least resource consuming mechanism to achieve concurrency
Scheduler
Input
Input
Input
Input
Input
Input
Deviation
Deviation
Ouput
Ouput
Ouput

Fig. 1. Application-level scheduler for small, embedded system.

In figure 1, the different aspects of design decisions are illustrated. In the example, for a small embedded system, the decision is taken to use an application-level scheduler to achieve concurrency in the system. As shown, this decision results in one added component (Scheduler), two design rules, one design constraint and a described rationale.

In the traditional perspective on software architecture, most of the information above is lost during the design process. Consequently, when, during the evolution of the system, some component is added that requires concurrency, there is a considerable likelihood that the software engineer violates some rules or constraints, resulting in either errors in the behaviour of the component or the failure of the system as a whole.

Conclusion

This position paper makes the following claims that, in our opinion, are worthwhile to discuss at the workshop:

- The first phase of software architecture research, where the key concepts are components and connectors, has matured the technology to a level where industry adoption is wide-spread and few fundamental issues remain.
- The traditional view on software architecture suffers from a number of key problems that cannot be solved without changing our perspective on the notion of software architecture. These problems include the lack of first-class representation of design decisions, the fact that these design decisions are cross-cutting and intertwined, that these problems lead to high maintenance cost, because of which design rules and constraints are easily violated and obsolete design decisions are not removed.
- As a community, we need to take the next step and adopt the perspective that a software architecture is, fundamentally, a composition of architectural design decisions. These design decisions should be represented as first-class entities in the software architecture and it should, at least before system deployment, be possible to add, remove and change architectural design decisions against limited effort.

References

1. Jilles van Gurp, Jan Bosch, 'Design Erosion: Problems & Causes', Journal of Systems and Software, 61(2), pp. 105-119, Elsevier, March 2002.
2. IEEE Recommended Practice for Architecture Description, IEEE Std 1471, 2000.
3. Anton Jansen, Jilles van Gurp, Jan Bosch, 'The recovery of architectural design decisions', submitted, 2004.
4. D.E. Perry, A.L. Wolf, 'Foundations for the Study of Software Architecture,' Software Engineering Notes, Vol. 17, No. 4, pp. 40-52, October 1992.