Fundamental Concepts for Practical Software Architecture

Alexander Ran
Nokia Research Center,
alexander.ran@nokia.com

ABSTRACT

Architecture of software is a collection of design decisions that are expensive to change. How to identify which design decisions are expensive to change? What are architecture views and which views are needed to adequately describe the architecture of a specific system? How to create and manage software architecture for a product family? This tutorial offers answers to these and other questions that arise in the context of complex software development.

We introduce a system of concepts useful in order to understand, design, and evaluate architecture of software intensive systems and system families. Our approach utilizes different software structures in order to control important system qualities related to its development, performance, and evolution.

We draw our experience primarily from software embedded in voice and data communication systems. However the same principles can be applied to software architecture in other domains. This tutorial should be useful to engineers and technical managers involved in construction or evaluation of complex software.

1. INTRODUCTION

This is a brief overview of the ideas and concepts presented in our tutorial. Some of this material was described in more detail in [1].

The initial interest in architecture emerged when software built by companies crossed a certain threshold of complexity. This happened across industries sometime between mid seventies and mid eighties. The general situation could be characterized as a loss of intellectual control over the software developed in industry. Therefore a useful understanding of architecture should offer a tool for dealing with the complexity of software development and maintaining intellectual control over design, construction, and evolution of software-intensive systems and system families.

Abstraction is one of the most effective ways to deal with complexity. Effective abstraction preserves the essence of what it represents. Therefore software architecture must be an abstraction, representing most essential decisions made in the design of a system. Which design decisions should be considered most essential? We make a simple proposition: **essentials of software design are decisions that are expensive to change.** Of course, the most-expensive-to-change decisions are those on which most other design decisions depend. This is why architecture is often associated with the early phases of system design. In practice, architecture management continues through the lifecycle of the product monitoring that later design decisions do not violate the initial architecture and also evolving the architecture as necessary in correspondence with new information, requirements, and technology.

We identify four categories of essential design decisions: concepts, focus, structure, and texture. Other conceptual frameworks for software architecture concentrate mainly on the structure of software and overlook or underplay the importance of the other three categories. Let us explain what we mean with each category and why this category of design decisions is expensive to revise and thus should be considered architecturally significant.

ESEC/FSE 2001, Vienna, Austria

2. CONCEPTS

From the perspective that considers software architecture to be an approach to dealing with complexity, probably the most important architectural decision is selection of concepts used to design the system. A design of a telecommunication system depends in many ways on whether "connection" is a primary concept in the system or is a relationship that may be established between system users. In the same way operating systems may or may not use the concepts of tasks, processes, monitors, queues, etc. Costs of revising these choices at a later stage of design or development may well exceed the costs of developing a new product. Therefore a major part of software architecture and its description should be selection and representation of the concepts used in system design.

3. FOCUS

Each system has a small set of properties that make it different from all other systems, make it attractive to users, feasible to construct, and competitive on the market. This is the focus of the system. The focus may be described with the essential use cases, important qualities of operation or construction. A flight reservation system may be able to charge a user for reserved flights, to provide a user with weather forecast, currency conversion, time zone and other relevant information. This however is not a focus of a flight reservation system. The focus of a flight reservation system is first of all the capability to

access and update the database of available flights on behalf of the user. The focus may also be on accessibility of the system over the web, security and privacy of user information, performance or flexibility of queries understood by the system. The focus may determine selected technologies as well as many other tradeoffs in system design. The focus determines the major concerns that need to be addressed by the proper software architecture. These concerns are not necessarily evident from system requirement documents or any other system descriptions. Identifying and documenting architectural concerns and their owners, also called stakeholders, is an important part of software architecture. Architectural concerns should be refined into architecturally significant requirements (ASR) that are specific in terms of desired system properties and how achieving these properties influences or constrains the architecture.

4. STRUCTURE

On each level of abstraction a system is decomposed into a number of interdependent components. The interdependency implies that if one of the components is removed or changed in an essential way, or the functionality is repartitioned between the components, or a new component needs to be introduced, such a change may require extensive revision of all other components – as a rule, a very costly task. Therefore definition of system structure falls in the category of essential decisions and thus belongs to software architecture. Notice that there are multiple structures that partition software as a whole into different kinds of components. Some of the most common are source or object code components, executable components, executing components. There are relationships, but there is no direct correspondence between these different kinds of components.

5. TEXTURE

Certain design decisions that are only visible within relatively fine-grained components are nevertheless very expensive to revise. This happens when the implementation of the decision cannot be localized, but must be replicated consistently creating recurring uniform microstructure, or texture. The texture of software is created by recurring uniform microstructure of its components. Decisions that affect texture of software have significant impact on the system and they are as hard to revise as decisions regarding the structure. Consistency of the texture is very often a problem, since the decisions appear to be local to a component. It is not easy to identify the common concerns present in the implementation of different components without concentrating on the texture on the system level.

6. SEGMENTATION OF CONCERNS

In addition to abstraction, a common approach to dealing with complexity is separation of concerns. A proper separation of concerns must identify the concerns that can be addressed independently from each other. We use separate segments of software transformation cycle as guides for separable concerns. Software always goes through a transformation cycle from source code modules to object modules to executable units to threads and objects. In each segment software consists of a different kind of elements. During the design or development segment the software is essentially source code. During the build segment the software is essentially object files and library archives. During the start-up segment the software is system state and groups of executable entities with their dependency structure. During the operation segment software is threads and objects. Each kind of components forms its own component domain. The structure and texture of software in each component domain address different concerns. Performance requirements are addressed by partitioning software into execution threads of varying priority, specifying thread scheduling policies, regulating use of shared resources, etc. Change and reuse requirements are addressed by partitioning software into modules having well-defined boundaries, predictable interaction with the environment, and minimal, well-specified dependencies on other modules. Requirements for independent re-start are addressed by partitioning the software into a set of separately loadable and executable processes.

Architecturally significant requirements must be grouped so that requirements in different groups may be satisfied independently, while requirements within each group may interact and even conflict. This can be achieved if we group the requirements by the segment of software life cycle. For example requirements that address software development and change can be satisfied almost independently from requirements that address run time behavior, or for example software upgrade.

7. WHAT IS SOFTWARE ARCHITECTURE

We define software architecture as **a set of concepts and design decisions about structure and texture of software that must be made prior to concurrent engineering to enable effective satisfaction of architecturally significant, explicit functional and quality requirements, and implicit requirements of the problem and the solution domains**.

According to this model the purpose of architecture is to enable satisfaction of architecturally significant requirements (ASR). The content of architecture is a set of concepts and design decisions about structure and texture of the software – the architecturally significant decisions (ASD). ASD must be made prior to concurrent engineering because they influence many design decisions in every component.

8. REFERENCES

[1] Ran, A. “ARES Conceptual Framework for Software Architecture” in M. Jazayeri, A. Ran, F. van der Linden (eds.), “Software Architecture for Product Families Principles and Practice”, Addison Wesley, 2000