IJSETT
International Journal

"International Journal for Science and Emerging Technologies with Latest Trends"
3(1): 36-45 (2012)

Software Architecture and its Various Tools

Varinder Pabbi
Asst. Prof.
Ramgarhia Institute of Engineering & Technology,
Phagwara,India

Abstract

The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both. The term also refers to documentation of a system's "software architecture." Documenting software architecture facilitates communication between stakeholders, documents early decisions about high-level design, and allows reuse of design components and patterns between projects. The software architecture discipline is centered on the idea of reducing complexity through abstraction and separation of concerns. To date there is still no agreement on the precise definition of the term "software architecture". However, this does not mean that individuals do not have their own definition of what software architecture is. This leads to problems because many people are using the same terms to describe differing ideas.

Keywords;- The goal of architecture, various tools to evaluate the software architecture; ATAM, SAAM,PASA.

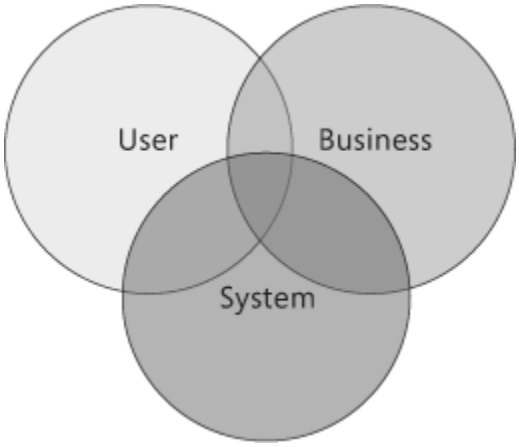
Introduction

The software architecture of a program or computing system is a depiction of the system that aids in the understanding of how the system will behave. Software architecture serves as the blueprint for both the system and the project developing it, defining the work assignments that must be carried out by design and implementation teams. The architecture is the primary carrier of system qualities such as performance, modifiability, and security, none of which can be achieved without a unifying architectural vision. Architecture is an artifact for early analysis to make sure that a design approach will yield an acceptable system. Software application architecture is the process of defining a structured solution that meets all of the technical and operational requirements, while optimizing common quality attributes such as performance, security, and manageability. It involves a series of decisions based on a wide range of factors, and each of these decisions can have considerable impact on the quality, performance, maintainability, and overall success of the application. Software architecture encompasses the set of significant decisions about the organization of a software system including the selection of the structural elements and their interfaces by which the system is composed behavior as specified in collaboration among those elements composition of these structural and behavioral elements into larger subsystems and an architectural style that guides this organization. Software architecture also involves functionality, usability, resilience, performance, reuse, comprehensibility, economic and technology constraints, tradeoffs and aesthetic concerns.

Why is Architecture Important?

Like any other complex structure, software must be built on a solid foundation. Failing to consider key scenarios, failing to design for common problems, or failing to appreciate the long term consequences of key decisions can put your application at risk. Modern tools and platforms help to simplify the task of building applications, but they do not replace the need to design your application carefully, based on your specific scenarios and requirements. The risks exposed by poor architecture include software that is unstable, is unable to support existing or future business requirements, or is difficult to deploy or manage in a production environment.

Systems should be designed with consideration for the user, the system, and the business goals. For each of these areas, we should outline key scenarios and identify important quality attributes (for example, reliability or scalability) and key areas of satisfaction and dissatisfaction. Where possible, develop and consider metrics that measure success in each of these areas.



Architecture focuses on how the major elements and components within an application are used by, or interact with, other major elements and components within the application. The selection of data structures and algorithms or the implementation details of individual components are design concerns. Architecture and design concerns very often overlap. Rather than use hard and fast rules to distinguish between architecture and design, it makes sense to combine these two areas. In some cases, decisions are clearly more architectural in nature. In other cases, the decisions are more about design, and how they help us to realize that architecture.

By following the processes described in this guide, and using the information it contains, we will be able to construct architectural solutions that address all of the relevant concerns, can be deployed on our chosen infrastructure, and provide results that meet the original aims and objectives.

Consider the following high level concerns when thinking about software architecture:

- How will the users be using the application?
- How will the application be deployed into production and managed?
- What are the quality attribute requirements for the application, such as security, performance, concurrency, internationalization, and configuration?
- How can the application be designed to be flexible and maintainable over time?
- What are the architectural trends that might impact your application now or after it has been deployed?

The Goals of Architecture

Application architecture seeks to build a bridge between business requirements and technical requirements by understanding use cases, and then finding ways to implement those use cases in the software. The goal of architecture is to identify the requirements that affect the structure of the application. Good architecture reduces the business risks associated with building a technical solution. A good design is sufficiently flexible to be able to handle the natural drift that will occur over time in hardware and software technology, as well as in user scenarios and requirements. An architect must consider the overall effect of design decisions, the inherent tradeoffs between quality attributes (such as performance and security), and the tradeoffs required to address user, system, and business requirements.
Keep in mind that the architecture should:

- Expose the structure of the system but hide the implementation details.
- Realize all of the use cases and scenarios.
- Try to address the requirements of various stakeholders.
- Handle both functional and quality requirements.

What Software Architecture Is Not

Software architecture must be distinguished from lower-level design (e.g., design of component internals and algorithms) and implementation, on the one hand, and other kinds of related architectures, on the other. For instance, software architecture is not the information model, though it uses the information model to get type information for method signatures on interfaces, for example. It is also not the architecture of the physical system, including processors, networks, and the like, on which the software will run. However, it uses this information in evaluating the impact of architectural choices on system qualities such as performance and reliability. More obviously, perhaps, it is also not the hardware architecture of a product to be manufactured. While each of these other architectures typically have their own specialists leading their design, these architectures impact and are impacted by the software architecture, and where possible, should not be designed in isolation from one another. This is the domain of *system* architecting.

Why Is Evaluation Necessary

Evaluation is the last chance where hidden requirements are discussed and complemented into the design. In short of communication perfectly and understanding of software project, a great many stakeholders do not know what they want exactly. In the requirement gathering phase, they may list several demands, the most crucial ones they believe in. But after evaluation, their opinions may entirely change, during which they start to be aware of some points they originally specified are not so important, while some other concerns begin to draw their attention. They are often surprised by the social power and get excited when they feel the positive improvement taken by their participating. And architects, during this activity, accept stakeholders various ideas, some of which are not mentioned in the requirement specification, and take off by adjusting the initial architecture design. This is also the good opportunity for him or her to deepen the insight of the to-be-built system. Shortly, architecture evaluation clears the barriers

among stakeholders, and empowers them with open communication channels. The direct result is the achievement of a commonly satisfactory system blueprint, which means a more than half success of a project.

Architecture is the center of development process, deciding the team structure, work division, configuration repository, documentation organizations, and management strategies and, of course, the development scheduling, An unsuitable architecture will draw a mass of mess when it must be modified to fit for the new concerns or those defects not uncovered in the early phase. The consequence of excessive cost spent on this alternation was accessed above. What's more, the whole team will face the terrible status that the project is out of control: More bugs are introduced after original bugs are fixed; demoded work breaks team structure which further disturbs lucid development; old plans and budgets are thrown away but the new ones cannot be created in time; all the guys, including customers, managers and programmers, expect vexed for the end of this nightmare, but no one gets the exactly the idea of the due date. Software architecture is destined to be evaluated, if it wants to be applied in practice. In fact, numerous architecture models are created specifically as the input of evaluation processes. Maybe experts who are concentrating on well-formed representation of architecture do not care about this very much. All in all, we need architecture evaluation.

Various Tools for Software Architecture

1)Architecture Tradeoff Analysis Method

The Architecture Tradeoff Analysis Method (ATAM) is a method for evaluating software architectures relative to quality attribute goals. ATAM evaluations expose architectural risks that potentially inhibit the achievement of an organization's business goals. The ATAM gets its name because it not only reveals how well an architecture satisfies particular quality goals, but it also provides insight into how those quality goals interact with each other, how they trade off against each other. The ATAM is the leading method in the area of software architecture evaluation. An evaluation using the ATAM typically takes three to four days and gathers together a trained evaluation team, architects, and representatives of the architecture's various stakeholders

Challenges

Most complex software systems are required to be modifiable and have good performance. They may also need to be secure, interoperable, portable, and reliable. But for any particular system

- What precisely do these qualities attributes such as modifiability, security, performance, and reliability mean?
- Can a system be analyzed to determine these desired qualities?
- How soon can such an analysis occur?
- How do you know if software architecture for a system is suitable without having to build the system first?

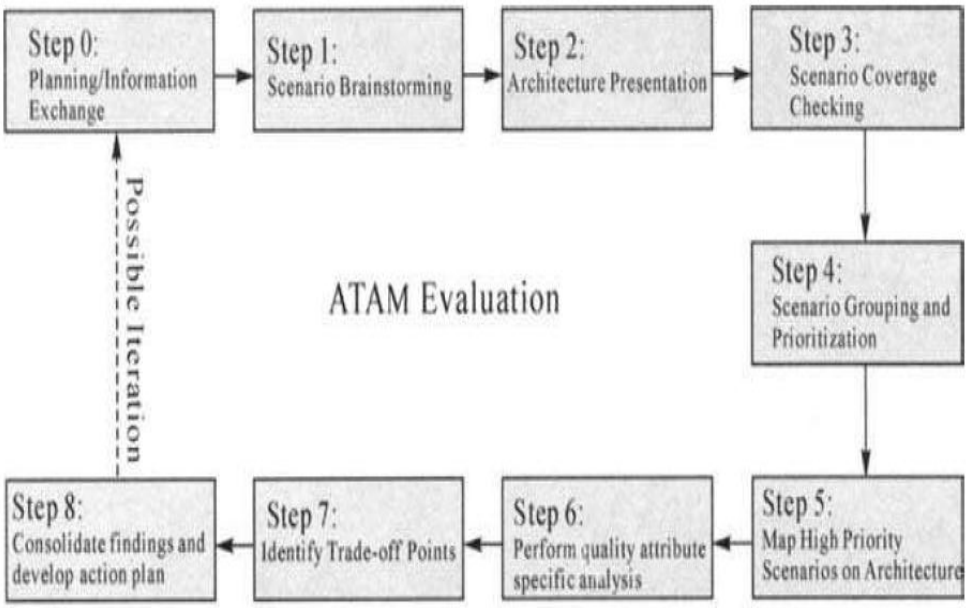
The ATAM consists of nine steps:

1. **Present the ATAM**. The evaluation leader describes the evaluation method to the assembled participants, tries to set their expectations, and answers questions they may have.
2. **Present business drivers**. A project spokesperson (ideally the project manager or system customer) describes what business goals are motivating the development effort and hence what will be the primary architectural drivers
3. **Present architecture**. The architect will describe the architecture, focusing on how it addresses the business drivers.
4. **Identify architectural approaches**. Architectural approaches are identified by the architect, but are not analyzed.
5. **Generate quality attribute utility tree**. The quality factors that comprise system "utility" (performance, availability, security, modifiability, usability, etc.) are elicited, specified down to the level of scenarios, annotated with stimuli and responses, and prioritized.
6. **Analyze architectural approaches**. Based on the high-priority factors identified in Step 5, the architectural approaches that address those factors are elicited and analyzed during these step architectural risks, sensitivity points, and tradeoff points are identified.
7. **Brainstorm and prioritize scenarios**. A larger set of scenarios is elicited from the entire group of stakeholders. This set of scenarios is prioritized via a voting process involving the entire stakeholder group.
8. **Analyze architectural approaches**. This step reiterates the activities of Step 6, but using the highly ranked scenarios from Step 7. Those scenarios are considered to be test cases to confirm the analysis performed thus far. This analysis may uncover additional architectural approaches, risks, sensitivity points, and tradeoff points, which are then documented.

9. **Present results**. Based on the information collected in the ATAM (approaches, scenarios, attribute-specific questions, the utility tree, risks, non-risks, sensitivity points, tradeoffs), the ATAM team presents the findings to the assembled stakeholders.

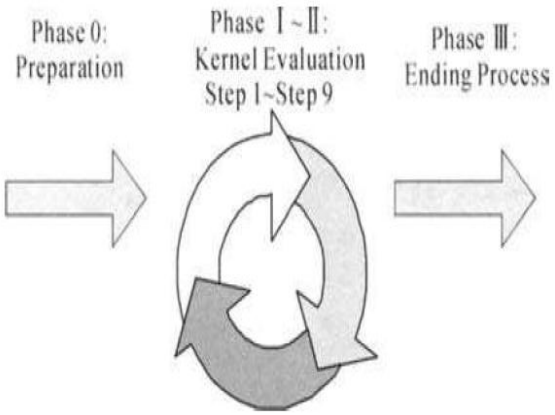
Benefits

- clarified quality attribute requirements
- improved architecture documentation
- documented basis for architectural decisions
- identified risks early in the life cycle
- increased communication among stakeholders



1)General Process of ATAM

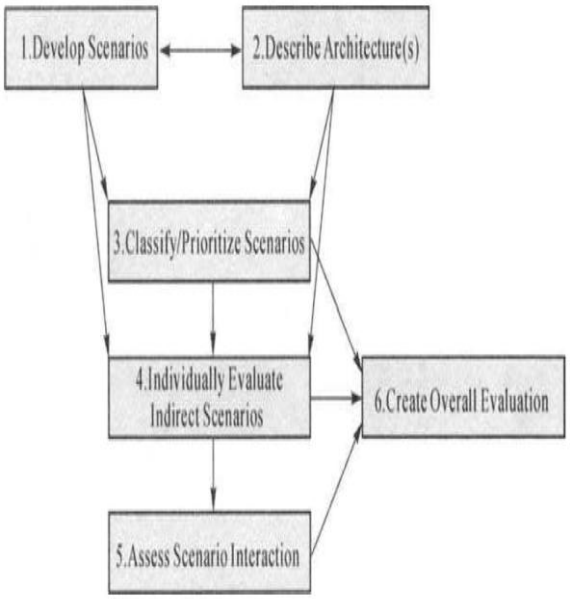
The complete process of ATAM currently contains four phases and nine main steps. Here, steps still do not mean that each of them has to be executed in a linear manner. In practice, evaluation leaders should make decisions to carry out which steps before to complement something, or jump to a step that should have been performed in several steps later. It depends on the situation. Steps indicate only the order of generation of intermediate evaluation products. Steps defined in the later always need products got in the former as inputs. Therefore, if evaluation team has own information that should be generated in a certain step or those information is useless for evaluation, that step can be omitted.



General process of ATAM

1) Software Architecture Analysis Method

Software architecture analysis method (SAAM) is a method used in software architecture to evaluate system architecture. It was the first documented software architecture analysis method, and was developed in the mid 1990s to analyze a system for modifiability, but it is useful for testing any non-functional aspect. SAAM was a precursor to the architecture tradeoff analysis method. SAAM is an intuitive method trying to measure the software's quality through scenarios, rather than the general and inaccurate quality attributes description. SAAM is simple, caring about only the relationship between scenarios and architecture structures, by taking not too many steps and specific techniques. Therefore, it is the ideal start point that beginners of architecture evaluation take. Initially, SAAM is designed to deal with modifiability of architecture. But after evolution and practice for several years, SAAM has shown its power in many other common quality attributes, and becomes the basis of some other evaluation methods, such as ATAM. It can detect the possible risks of evaluated architecture and take comparison among several architecture candidates with respect to meet predefined scenarios.



Activities and dependencies in a SAAM analysis

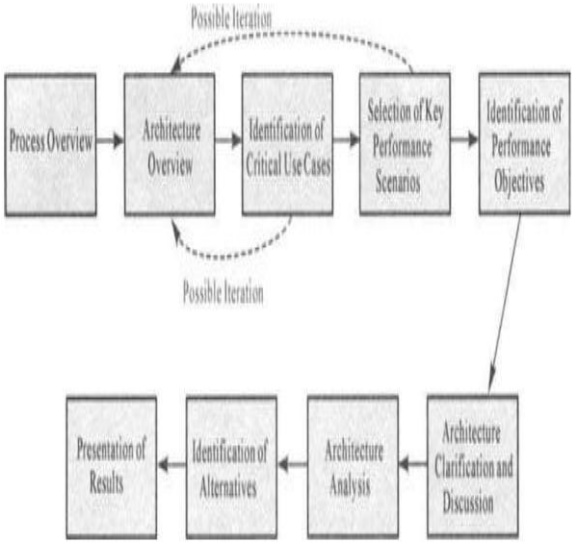
The main output normally is given in the style of evaluation report, by which SAAM shows the defects that current design cannot reach the quality requirement, in the single architecture evaluation case; or indicates which candidate meets the scenarios best, in the multiple cases. It also has the capability of figuring out the potential unsuitable design due to ugly decomposition or excessive complication. At last, SAAM provides the estimation of cost and range incurred by modification, avoiding the blind construction.

2) Performance Assessment of Software Architecture

"Software's performance cannot be retrofitted. It must be designed into software from the beginning, the make it run, make it run right, make it run fast' are dangerous." This is the basic motivation of PASA, a method coming from Williams and Smith's work. PASA is performed through 9 steps.

1) **Process Overview**: This is a presentation that introduces the general steps of PASA, motivation of assessment, and the whole process' outcome. It helps improve participants' familiarity with what they should do, which actions are suitable and benefit to the assessment, and thus avoid doing something nonsensical.
2) **Architecture Overview**: As the basis for the subsequent activities, architects have to describe current architecture designs and explain those critical structure or behavior specifications that the assessment needs.
3) **Identification of Critical Use Cases**: Try to find the external visible use cases that reflect the important system behaviors, especially those tightly relevant to responsibility and scalability.

4) **Selection of Key Performance Scenarios**: From the critical use cases generated in the previous step, the important performance-related scenarios are identified.
5) **Identification of Performance Objectives**: Each scenario involved in assessment should be measurable; therefore, assessment participants have to define the performance objectives against each key performance scenario.
6) **Architecture Clarification and Discussion**: It is the time to inspect the architecture elements in more deep detail that influence the realization of the scenarios above. This is achieved by participants' further discussion about system's architecture, through which the potential problem areas will be exposed.
7) **Architecture Analysis**: Against each key performance scenario, analysis of architecture is conducted to figure out that whether current design can support the corresponding performance objectives.
8) **Identification of Alternatives**: If some problems exist (in fact, in most cases it is), ori/~nal architecture should be fine-tuned in local area by alternating structures which are capacity to meet the objectives. Sometimes, the whole architecture style is replaced to repair its performance problems.
9) **Presentation of Results**: This is the final presentation of assessment's conclusion, including the found problems, plan of architecture modification, estimated work and cost on modification.



PASA evaluation process

Summary

SAAM and ATAM expose the impression of common features of most scenario based evaluation methods. They get the input of scenarios and architecture description, evaluate and judge whether current architecture (or several architecture candidates) is capable of meeting desired quality requirements. Potential defects and risks are identified, which then become the motivation of modification. Finally, raw evaluation results are collected and prepared for the following use, such as hints of future development or historical data accumulated for reuse. Reality is flexible, thus it needs flexible solution. Just remember the principles why evaluation is so useful.

References

1. Software Architecture By Zheng Qin Jiankuan Xing Xiang Zheng.

2. Software Architecture Analysis Tool By Johan Muskens.