SOFTWARE ARCHITECTURE'S ROLE IN BRIDGING BETWEEN REQUIREMENTS AND IMPLEMENTATION

Sagar Autade[1], Rahul Takawale[2], Hema Gaikwad[3]
[1,2]MBA-IT, SICSR, [2]Assoc. Professor
Affiliated to Symbiosis International University (SIU), Pune, Maharashtra, India

Abstract: A model-based system development cycle involves two semantically distinct aspects: the requirements specification and the implementation model. Due to the conceptual and semantic differences between these two major system lifecycle stages, the transition from requirements to implementation is inherently a non-coherent process. The processes of requirement engineering and software design take place during a software development. Business processes, goals, organizational models of company can be considered for specification of software requirements. In iterative software development, which is more popular nowadays, requirement engineering and software design processes occur on different stages during the project increasing the complexity of the projects. Thus, software design is constantly refined and minor improvements are made in it. Software architecture is a promising approach to controlling software complexity. Software architecture research is directed at reducing costs of developing applications and increasing the potential for commonality between different members of a closely related product family.

I. INTRODUCTION

Software Architectures enable developers to centre on the big picture in developing a system and to adopt a component-based development philosophy instead of always building a system from scratch. Architectures do this by making a software system's structure precise, isolating the computation of components from their interactions in a system, and providing a high-level model of a system that can be managed and evaluated before any changes are effected in an actual implementation. In most cases, today, there's no documented software architecture to support the requirements and help in the implementation. The requirements gathered at the inception of the project are mostly general descriptions of what the customers expect from the software. These 'general descriptions' of the requirements are difficult to understand and implement by the development team due to lack of structure to the requirements. In order to provide a sound structure and meaning to the requirements, software architecture diagrams are designed and modelled to give the requirements some practical value.

II. WHAT MAKES A GOOD ARCHITECTURE?

There is no inherently "good" or "bad" software architecture. A good architecture is one that meets the requirements set out for it. The characteristics that make for a good architecture are:

- Highly modular
- Avoids duplication
- Well described
- Runs and passes all tests and acceptance criteria.

Need of good architecture

Software architecture is a "strategic model" - the generic rules of building and organizing your system, or its philosophy. Architecture is something you cannot "re-factor", and you'll end up having to rewrite the system from scratch if it's bad. A strong software architecture can reduce the cost of development in the long run, ensuring your system won't crack at the joints when you add the next portion of the requirements. In short, "architecture" is thinking in advance. Development of "good" software architecture takes time and can be costly. But it pays off in the long run

The architectural requirements of the system are such that they -

- Allow understanding of and reasoning about a system at a level of abstraction above the source code and closer to the stakeholder's mental models of the system.
- Narrow the gap between system requirements, which exist in the "problem" space, and software designs, which are in the "solution" space.
- Support reuse and families of applications as averse to custom and "one of a kind" solutions.
- Enable codification of successful design and evolution properties from traditional projects.
- Allow upstream analysis to correct errors early and reduce costs associated with those errors.
- Allow reformability of software both before and during runtime.
- Allow components of differing granularities, enforces in different programming languages.
- Support distributed and heterogeneous environments with numerous address spaces, strings of authority, and operating system processes.

III. ARCHITECTURE'S ROLE IN REQUIREMENTS

Proper software architecture is the best solution that simultaneously satisfies business requirements, business constraints (including cost), and technical requirements. To achieve the right harmony between these contesting concerns, the architect must have a good perception of 4 key areas:

1. The business – The company's mission statement, philosophy, values, capabilities, business strategies, and activities.
2. Processes and existing computer systems - The way in which end-users accomplish their work and the hardware/software that they are currently using.
3. Project constraints - What the business expects the software to accomplish, hardware/software that the business is locked into using, timeline, and budget of the project.
4. Technology Suitability, cost, and compatibility with existing systems

A well thought-out architecture must consider these vital principles:

- Build to change instead of build to last
- Understand the end user needs and the domain prior to designing the components
- Identify sub-systems in your product and study the layers and components to abstract them and determine the crucial interfaces
- Use an incremental and iterative approach to designing the architecture
- Learn from history, document your decisions and identify and reduce major risks
- Do not under-invest in architecture

IV. CHALLENGES TO PROPER ARCHITECTURE

In pursuing this hybrid discipline that melds business, technical, and people skills, the software architect is often called upon to perform a delicate high wire act. Here are some of the challenges that the architect must overcome to protect the integrity of the architecture and produce great software:

Politics: The software architect is a key interface between business and IT, and therefore must balance the storms that brew along that front. The architect must act as a mediator, helping to reach a consensus on what needs to be accomplished and how it should be done. Some compromises, while necessary, are problematic from an architectural standpoint.

Out-of-date standards: Existing software components might conform to old standards, making it difficult for an architect to integrate them into a new system.

Exclusive focus on technology: If an architect compromises business needs to achieve a specific technical outcome, the resulting architecture will be inappropriate and ineffective.

Poor understanding of the business goals or domain: The architecture cannot succeed unless the architect really understands the business that must be served by the software.

Lack of understanding by the business: A business that does not understand the value of proper software architecture will push for shortcuts at the planning stage, which will lead to "accidental" architecture and the 'shabby' software that goes with it.

Poor implementation: The most brilliant design is useless without the ability to act on it. The architect must have the skills, support, and resources to shepherd a software project from requirements to successful release.

V. ARCHITECTURE'S ROLE IN IMPLEMENTATION

Implementation is the one aspect of software engineering that cannot be considered as optional. Architecture-based development provides a unique twist on the typical problem- it becomes, in a large measure, a mapping activity.

Maintaining and mapping means ensuring that our architectural intent is reflected in our constructed systems.

Components and Connectors

- Partitions of application calculation and communication service.
- Modules, packages, libraries, classes, explicit components or connectors in middleware.
- Interfaces.
- Programming-language level interfaces are common.
- State machines or protocols are more difficult to map.

Configurations:

- Interconnections, references, or dependencies between functional segregation
- May be implicit in the implementation
- May be externally determined through a MIL and set up through middleware
- They may involve use of reflection

Design rationale:

- Often does not appear directly in application
- Retained in remarks and other documentation
- Architectures inevitably change after implementation begins

For maintenance objectives

- Because of time burdens
- Because of new information
- Implementations can lead to the rise of new information
- We learn more about the feasibility of our designs when we implement
- We also learn how to optimize them
- Keeping the two in sync is a difficult technical and managerial problem
- Places where strong mappings do not exist are often the first to diverge
- One-way mappings are easier
- Must be able to realize the impact on implementation for an architectural design selection or modification
- Two-way mappings require more intuition
- Must understand how a change in the implementation affects architecture-level design decisions

One strategy: limit changes

- If all system modifications must be done to the

architecture first, only one-way mappings are required
- Works very well if a myriad of generative technologies in use
- Usually difficult to regulate in practice; introduces process delays and limits implementation freedom

Alternatives:
- Allow modifications in either architecture or implementation
- Depends on round-trip mappings and maintenance strategies
- Can be serviced (to an extent) with automated tools

VI. CONCLUSION

With the advent of new technologies, the user interface portion of interactive systems is becoming increasingly large and complex. Software architecture and architectural modelling is, therefore, becoming a central problem for large complex systems. In addition, software architecture is the activity at the turning point between two worlds: software requirements and software implementation. Because of its interlinking location, software architecture must take into account the properties of both these worlds. In this paper, we have discussed the role of software architecture in the requirements and implementation phases of software development life cycle.

REFERENCES

[1] Software Architecture and Component Technologies: Bridging the Gap Information and Computer Science University of California, Irvine Irvine, CA 92697-3425 USA {peymano,neno,taylor,dsr}@ics.uci.edu http://www.ics.uci.edu/pub/arch/ Peyman Oreizy Nenad Medvidovic Richard N. Taylor David S. Rosenblum

[2] McGovern, James, Scott W. Ambler, Michael E. Stevens, James Linn, Vikas Sharan, and Elias K. Jo. A Practical Guide to Enterprise Architecture, (Upper Saddle River, New Jersey: Pearson Education, 2004), 37.

[3] Jalote, Pankaj. An Integrated Approach to Software Engineering (Third Edition), (New York: Springer Science + Business Media, 2005), 162-163.

[4] Ambler, Scott W. The Object Primer: Agile Model-driven Development with UML 2.0 (Third Edition), (New York: Cambridge University Press, 2004), 278.

[5] Pfleeger, Shari Lawrence, and Joanne M. Atlee. Software Engineering: Theory and Practice (Fourth Edition), (Upper Saddle River, New Jersey: Pearson Higher Education, 2010), 236.

