ISSN:0975-9646

Identification of Common Challenges and Key Factors in Software Architecture Using Grounded Theory

Vinay Krishna
Agile Transformation and Consultation
SolutionsIQ India Consulting Services Pvt Ltd
Bangalore, India

Dr. Anirban Basu
APS College of Engineering
Somanahalli, Kanakpura Road
Bangalore, India

Abstract— **Software Architecture plays vital role in developing a system by addressing quality related aspects such as performance, security, scalability etc. [1]. Architecture of a software system has to ensure that design is able to support all functional and non-functional requirements as well as able to incorporate any changes requested by the customer. We conducted a Grounded Theory study to investigate challenges and possible solutions in Software Architecture involving 34 technical experts from 28 different software companies in India, USA, Netherland and South Africa. We identify six key factors that helps to bring agility in Software Architecture.**

Keywords- Software Architecture; Plan driven; Agility; Grounded Theory

I. INTRODUCTION

Software architecture and design is an important factor [1, 2, 3] in success and failure of any software [The art and science of software architecture]. The architecture includes non-trivial design decisions like business, technical etc. Basically, Software architecture is not only concerned with structure and behavior, but also with usage, functionality, performance, resilience, reuse, comprehensibility, economic and technological constraints and tradeoffs, and aesthetics. In short software architecture focus on both structure and vision [3].

There are two school of thoughts. In one case architecture needs to be planned well in advance so that it meet all functional and non-functional requirements adequately. Eventually this requires big up-front design and architecture. The other one believes on emergent architecture and evolutionary design [4]. This requires clear distinction between essential and accidental complexity [5].

This raises a critical question: how do we determine the complexity required in architecture? We found the answer to this question through a Grounded Theory study that involved 34 technical experts which includes enterprise architects, software architects, technical leads, CTOs, COOs, and consultants from 28 different software companies in the India, USA, Netherland and South Africa. We found the six strategies that helps in determining complexity in architecture: 'Focus on immediate and visible business value', 'Collaboration', 'Communication', 'Continuous improvement', 'Art of simplicity: Keep it simple' and 'Feedback mechanism'.

II. RESEARCH METHOD

A. Grounded Theory

Grounded Theory (GT) is becoming a popular research method in Software Engineering [6]. Grounded Theory (GT) is a systematic research method that stresses the generation of theory derived from systematic and rigorous analysis of data. GT was developed by two sociologists, Barney Glaser and Anslem Strauss [7]. GT is a complete research method as it provides concrete procedures that cover all stages of research including sampling participants, data collection, data analysis, use of literature, and write-up [6]. Mainly it focuses on surfacing the main concerns expressed by majority of participants. Further it helps in generating the theory to explain how they go about resolving this main concern. The main concern could be any aspect of the field that the researcher is interested in exploring that is particularly important (and even problematic) for those involved.

We chose GT as our research method because it is suitable to be used in areas that are under-explored or where a new perspective might be valuable, and not much work is done in this area. The other reason is, GT allows researchers to study human and social aspects in the context of solving problems, and human interactions plays major role in software architecture [7, 8]. We used Glaser's approach and commenced our research with open ended discussion. As per GT we should not start a GT study with a specific research question. This restricts us from leading to preconceived ideas or hypotheses of the research phenomenon [9, 10]. As per Glaser both problems and its key concerns emerge in the initial stages of data analysis [9, 11].

B. Data Collection

Theoretical sampling is the process of data collection in GT. It is used for generating theory whereby the analysts jointly collects, codes and analyzes his data and decides further what data to collect and where to find them [7]. We interviewed technical experts such as enterprise architects, software architects, technical leads, CTOs, COOs, and consultants, from different organizations from various countries such as USA, Netherland, South Africa and India. We conducted face-to-face, Skype and telephonic, one-on-one interviews with our participants using open-ended

questions. Initially we prepared a set of questions for the initial interviews to have a smooth discussion with the participants. The interview questions focused on the challenges that participants faced in developing software architecture, and the strategies adopted to overcome them.

Interviews were conducted at a mutually agreed time and location. In most of the cases it lasted for at least an hour. We voice-recorded all the interviews with consent from the participants. The reason for voice-recording was: it helped us to just concentrate on the conversation and understand participant's main concerns as we don't need take a note of our conversation.

We analyzed the initial interviews and few key concerns emerged. We framed research questions based on the emergent key concerns and further conducted in-depth investigation on them and presented the initial findings in different papers [12]. In this paper we investigate in-depth how we have applied Grounded Theory.

C. Participant and Project Details

Table 1: Details of participants (Position: Enterprise Architect (EA), Chief Technical Officer (CTO), Software Architect (SA), Chief Information Officer (COO), Engineer Director (Eng Dir), Technical Consultant (Tech Cons) and Technical Lead (Tech Lead).

Participants	Position	Location	Group involved	Project Duration (In months)
P1	EA	South Africa	4	6
P2	CTO	US	8	8
P3	EA	India	5	12
P4	Eng Dir	India	5	16
P5	SA	India	4	19
P6	COO	Netherland	8	12
P7	SA	US	5	15
P8	EA	US	5	9
P9	Tech Lead	India	3	12
P10	Tech Cons	India	6	6
P11	SA	India	6	9
P12	EA	US	9	12
P13	EA	South Africa	7	16
P14	Tech Lead	India	5	9
P15	Tech Lead	India	3	5
P16	SA	India	5	6
P17	SA	India	4	8
P18	Eng Dir	US	8	12
P19	SA	India	5	6
P20	SA	India	4	7
P21	SA	India	5	8
P22	SA	India	7	9
P23	SA	India	5	6
P24	SA	India	4	11
P25	SA	India	6	15
P26	SA	India	4	9
P27	CTO	India	6	14
P28	Eng Dir	US	9	8
P29	Tech Cons	India	4	10
P30	Tech Cons	India	4	6
P31	SA	India	3	7
P32	SA	India	5	11
P33	SA	India	4	14
P34	SA	India	6	10

We interviewed 34 participants with various roles across the globe.

Table 1 depicts participant and other important details. Multiple groups were involved from 3 to 9, and project duration varied from 5 to 14 months though some projects were still ongoing when we interviewed the participants. Due to privacy and ethical consideration, we will only identify our participants using the codes P1 to P40

D. Data Analysis

We transcribed the interviews and analyzed it using open coding [10]. Open coding breaks down, examines, compares, conceptualizes and categorizes the data [13]. We apportioned a code or a summary phrase to each key point. Using Grounded Theory's constant comparison method [14], we constantly compared each code with the codes from same interview, and those from other interviews. The codes that are related to a common theme were grouped together to produce a second level of abstraction called a concept.

As we continuously compared codes, many fresh concepts emerged. These concepts were themselves analyzed using constant comparison method to produce a third level of abstraction called a category.

III. RESULTS

A. Challenges

Figure 3 shows the concepts 'Many stakeholders', 'Large amount of data', 'Overlap in requirements', 'Scalability', 'Match the pace of business growth' and 'Incidental complexity' that gave rise to the category Architectural Challenges in large/critical applications.

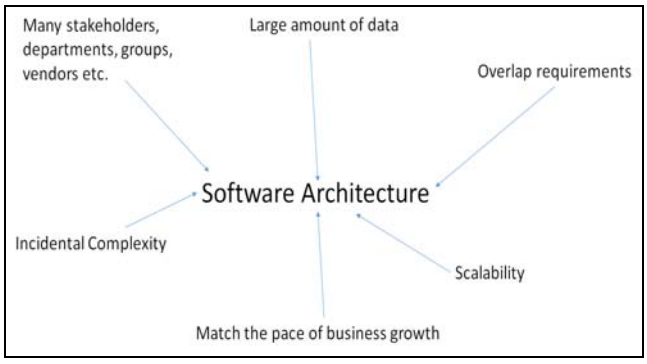

Figure 1. Challenges

A1. Many stakeholders

It's a very common problem in case of large/critical applications. It creates more chaos, confusion in terms of requirement. It has direct impact on the decision making process which causes unstable architecture, disagreement about approach (top-down, bottom-up) and large amounts of noise.

A2. Large amounts of information

Handling large amounts of information is important, since it grows very fast. It needs extra care as there is possibility of noise in information. Unclear information leads a wobbly architecture.

A3. Large amounts of information

Since multiple groups/departments existing, noise is likely to be present in requirement. Many cases redundant requirements are found. Eventually it's difficult to attain a stable architecture with noisy requirements.

A4. Scalability

Most of the time it needs to scales the application in terms of data and function throughout the application development. It's most challenging task. Participants explicitly discussed the scalability where application struggle to handle growing data and new need.

A5. Match the pace of business growth

The only factor which is constant in any enterprise is change. A common belief is architecture must be able to handle it gracefully which is really challenging.

A6. Incidental complexity

It is also known as "Accidental Complexity". It arises from choices made in terms of technology, hardware etc. to be used. Essentially anticipation introduces more incidental complexity. In large-scale software, though, removing accidental complexity while retaining the solution to the essential complexity is challenging.

B. Recommendations

Figure 4 shows the concepts 'Drive immediate business value', 'Collaboration', 'Communicate', 'Continuous improvement', 'Art of simplicity: Keep it simple' and 'Feedback mechanism' that gave rise to the category Architectural Improvement in large/critical applications.

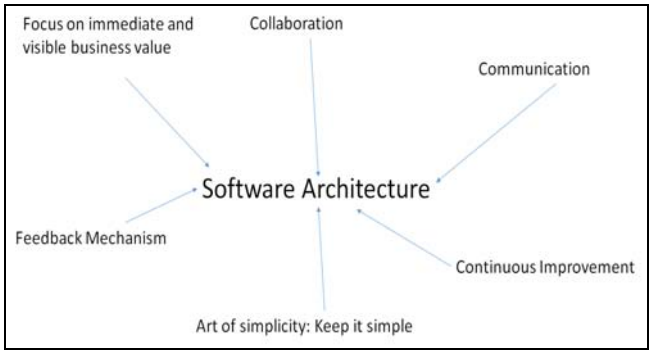

Figure 2. Recommendations

B1. Focus on immediate and visible business value

".. to address accidental complexity, need to just focus on visible and immediate business value…"– P30, Technical Consultant

It's important to deliver immediate business value. There is need to prioritize the requirement based on time to market and value. Based upon this participants suggested to do just enough architecture.

B2. Collaboration

"*We have defined periodic steering committee reviews to address any high level issues and adapt as required.*" – P11, Software Architect
"*We found that frequent communication fosters good understanding between project team and management*" – P3, Enterprise Architect

Participants encouraged collaboration with all stakeholders in order to reduce noise in requirements. They explicitly discussed the 'Collaboration' where all stakeholders strive to interact and perform as one team.

B3. Communicate

"*To build trust, understand correctly and remove ambiguity, it's necessary to speak in the language of business by technical team…*" – P6, COO.
"*.. the only way to understand business is - Communicate, communicate and communicate .*" – P8, Enterprise Architect

Participants felt that communication plays vital role to bridge the gap between business and technology. They emphasized to communicate in the language of business. This is good way to build strong trust.

B4. Continuous improvement

"*Always Measure, adapt and become the best*" – P2, CTO.

Develop mechanism to capture knowledge and share with others. There are various ways to capture and share but the ultimate aim is to keep improving.

B5. Art of Simplicity: Keep it simple

"*We identified essential aspects of the system and produced lean solutions. Not technology was used unless it was critical. For example we did not introduce a complicated rules system as the rules we had were simple*" – P3, Enterprise Architect.
"*We always ask to prove the need in case someone is going to choose any third part tool/new hardware, before deciding it*" – P13, Enterprise Architect

Similar voice was found in other participants views. This is one of the effective ways to keep away the incidental complexity from essential complexity.

B6. Feedback mechanism

"*Define mechanism to encourage early feedback from end user*" – P6, COO.

Feedback helps to gain confidence for any decision taken and it also helps to improve it in lots of other dimension that might have been ignored inadvertently. This process encourages adoption over anticipation and eventually paves way for evolutionary architecture.

CONCLUSION

We used Grounded Theory study to investigate common challenges and key factors in software architecture and involved 34 technical experts from 28 software companies in India, USA, Netherland and South Africa. Through our analysis, we found tech experts value a lot on six items: Focus on immediate and visible business value, Collaboration, Communication, Continuous Improvement, Art of simplicity – keep it simple and Feedback Mechanism.

REFERENCES

1. SEI, "Defining Software Architecture", [online] Available: http://www.sei.cmu.edu/architecture
2. M. Fowler, "Who needs an architect", September 2003, IEEE Software
3. S. Brown, "Is Software Architecture important" in "*Software Architecture for Developers*", Lean Publishing, 2015
4. P. Kruchten et al, "Agility and Architecture: Can They Coexist?", March/April 2010, IEEE Software
5. N. Ford, "Evolutionary architecture and emergent design: Investigating architecture and design", February 2009, IBM Developer Works. [online] Available: http://www.ibm.com/developerworks/library/j-eaed1/
6. R. Hoda, J. Noble & S. Marshall, 2011 "Grounded Theory for Geeks", School of Engineering and Computer Science. Victoria University of Wellington, New Zealand. [online] Available: http://www.hillside.net/plop/2011/papers/E-13-Hoda.pdf
7. B.G. Glaser, A. L.Strauss, "The Discovery of Grounded Theory: Strategies for Qualitative Research". Sociology Press, Aldine, Chicago (1967)
8. K. Charmaz, "Constructing Grounded Theory", Sage Publications, 2006
9. B. Glaser, "Doing Grounded Theory: Issues and Discussions", Sociology Press, Mill Valley, CA (1998)
10. C. Urquhart, H. Lehmann, M. D. Myers, "Putting the 'theory' back into grounded theory: guidelines for grounded theory studies in information systems", Information Systems Journal 20(4) (2010) 357–381
11. B. Glaser, "Basics of Grounded Theory Analysis: Emergence vs Forcing", Sociology Press, Mill Valley, CA (1992)
12. V. Krishna, A. Basu, "Software Architecture for large/critical applications", Software Engineering (CONSEG), 2012 CSI Sixth International Conference, Available: IEEE Xplore
13. A. Strauss, J. Corbin, "Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory", Sage Publications (1998)
14. B. Glaser, "The constant comparative method of qualitative analysis", Social Problems 12(4) (1965) 436–445