INTerview Guidelines for Analyzing Software Architectural Practices in Agile Projects

Hennah S. Sugumaran, Peter A. Horsfield, Paul J. Bruschi
Siemens Corporate Technology
755 College Road East
Princeton, New Jersey USA 08540
hennah.sugumaran@siemens.com
peter.horsfield@siemens.com
paul.bruschi@siemens.com

Abstract
During the development of complex systems using an agile development process there is an unavoidable friction between providing “not enough” and “too much” of a software architectural knowledge while agile development is proceeding. Excessive architecture gives the appearance of “slowing down” development in order to address architectural quality attributes that may not be immediately relevant. On the other hand, ignoring architectural quality attributes is risky.

In this paper we describe an interview process and detailed guidelines to aid in collecting data for this research.

Key Words

1. Introduction
1.1 Agile Methodology and Software Architecture
Agile development methodologies are increasingly popular in the software industry. They have gained popularity through a set of principles and practices. Agile development methodologies are people-oriented processes and light in weight [5] [6] [7] [10].

The Manifesto for Agile software development [9] states that agile software methodologies value:
- Individuals and interactions over process and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan.

Software Architecture is “concerned with capturing the structures of a system and the relationships among the elements...”

Software Architecture acts as a design plan and also as an abstract of the product to be implemented [34]. Software architectural practices are defined here as “the set of steps or actions applied to build an understanding of the big picture of a system amongst system stakeholders”. Such practices include multiple views of software architecture (4+1 views), domain analysis, and the application of architectural patterns.

1.2 Problem Statement and Objective
Despite the popularity of agile there is an increasing bafflement of software architecture in agile projects [35]. Architecture is a part of product quality [1]. So a decrease in architectural focus results in a decrease in product quality. The studies say that there is a lack of focus on software architecture in agile projects [10]. Certain architectural decisions regarding quality attributes need to be taken early in the project. Agile proponents see architectural practices as a Big Design Up Front [BDUF] stemming from traditional methodologies and having no value. BDUF leads to big documentation which is anti-agile [8] [35]. Good Design Up Front or “just enough architecture” is absolutely essential [29] [30], [45].

One of the Principles [9] of the Agile methodology says “The best architecture, requirements and design emerges from self organizing teams”. This may be true in some projects. But in the case of developing large complex systems, some practices must be followed by the teams in order to self-organize and allow themselves the capability to produce the best architecture, requirements and design. In our study we ask what agile teams do for self-organization and to get the best architecture, requirements, and design.

The objective of the ongoing research is to find:
- How much architecture is needed for an agile development project
- How to perpetuate the goal of Good Design Up Front rather than falling into the trap of Big Design Up Front
- How to handle the iterative nature of agile methodologies since system design tends to erode with the evolving requirements of the system and stakeholders.

DOI: 10.2316/P.2013.796-014 791
The objective of this paper is to provide the initial interview guidelines that we will use to conduct a series of interviews first step for our research.

In conducting these interviews with disparate teams, and understanding how those teams operate and the practices that they employ, we plan to identify potential best practices and explore the success of those practices amongst the software development community. Some best practices might be:

- Track architectural concerns within a risk backlog
- Understand the extent to which architectural quality attributes should be included within user stories
- Use of architectural sprints independent of development sprints

2. OUR APPROACH

The first step in our approach towards the research objective is to interview people involved in running projects within Siemens (e.g. development managers, architects, developers, process experts), and find out how architectural concerns are addressed in agile software development projects. Example Siemens projects are those that are software-intensive and have used an agile methodology over an extended period of time. We have planned to interview team members on these projects using guidelines that we have developed and collected in this paper. However, the guidelines are not specific to any particular project and are useful in general.

2.1 Conducting Interviews

Since agile development teams are usually busy, our interview approach uses the guidelines to be minimally invasive and efficient in both the use of the interviewee’s time, and in our ability to process the collected data.

Prior to each interview, the research team reviews the guidelines and chooses a set of example questions based on the role of the interviewee and the time allocated to the interview, typically 45 minutes to 1 hour. Additional questions from the guidelines can be added to the interview if needed or desirable.

The interview is conducted in a relaxed setting and while we do record audio of the interview, we assure our interviewees that the information will be kept internal to the research team and is simply used to allow us to transcribe and code the resulting conversation. Multiple interviews may be conducted within the same team but should be at different times to keep the data confidential and the opinions voiced honest.

Immediately after the interview the research team will conduct a debriefing to improve the questions for later interviews and to determine the major points discussed during the interview. This will help to improve the encoding of the conversation.

2.2 Processing Interview Results

By collecting a body of interview results and integrating those results as one, any particular individual’s contribution is not apparent. This allows us to discuss the results of our investigation with a wider audience without compromising the anonymity of our participants.

Furthermore, the agile software development community may use different terms to describe the same phenomena. For example, we have seen the terms ‘domain experts’ and ‘analysts’ used to identify the same role within one project.

Our approach involves coding the interview conversation from collected data in the form of transcripts, audio recordings and notes into a single integrated model. Each successive interview contributes to this model. A subject of our ongoing research is to refine this model into a grounded theory of architectural practices in agile projects.

3. INTERVIEW GUIDELINES

We have developed interview guidelines based on the following eight topics:

1. Development process and practices
2. Requirements
3. Quality process and practices
4. Risk management
5. Technical debt
6. Documentation
7. Architectural practices
8. Project management.

Interviews will be based on the example questions listed in section 5.

These questions probe the practices that development teams follow in actual agile environments to address the architectural concerns. We will collect details about individual projects through sessions with one or more team members.

3.1 Development Process and Practices

Agile development methodologies encourage iterative and customer focused development [3] [4] [5] [6] [7] [10]. Several implementations of agile methodologies exist, e.g. SCRUM [13], XP [12], and Crystal [14].

Regardless of the methodology, a typical development phase starts with prioritized items taken from a product backlog. The product backlog is a list of requirements (functional and non-functional) [11], and is a central part of day-to-day activities in the development process. Product backlog items can be features, functions, bug fixes, defects, requested enhancements and technology upgrades [7] [45].

There is a lack of scientific support with most studies saying that agile development does not meet architectural needs sufficiently and must extend the use of architectural practices to improve the agile projects [10].
In this topic, we will discuss with the agile development team their approaches and specific practices and try to determine how their day-to-day activities in agile development address architectural concerns.

3.2 Requirements

Requirements in agile methods are gathered in various ways such as the ‘Story Card’ method in XP [15] [37] and through maintaining the Product Backlog in SCRUM [7] [36].

User stories are defined as “One thing the customer wants the system to do. Stories should be estimated at between one to five ideal programming weeks and should be testable” [12].

While writing the user stories and the product backlog, stakeholders should consider non-functional requirements. Considering architectural quality attributes only at the implementation level might result in big issues [18] i.e. the failure of the system because developers did not consider architectural viewpoints at the product level.

The relevant activities involved with requirements engineering are:

- **Identification of requirements (Functional and Non-Functional):** The initial challenge in identifying the non-functional requirements is that customers might not be aware of the concept of architectural quality attributes, and will not be able to submit the complete set of non-functional requirements to the product manager in order to have it included in the product backlog. As a result, the product backlog contains only functional requirements. This gives the developers a way to ignore architectural quality attributes [16]. Customers may not realize the importance of architectural quality attributes; working software products can be tested at the close of the iterations, but some types of problems may only be recognized at the final release [17].

- **Prioritization of requirements:** Prioritization of the user stories is based on a stakeholder’s judgment or on expert opinion.

- **Requirements management:** Agile methods do not present any formal techniques to extract or manage non-functional requirements [17] [18] [19]. Agile welcomes changing requirements [9], however, such changes need to be managed and therefore requirements management should be in place to address the evolving needs of customer.

  “Requirements Management is a precondition of managing change based on customers’ needs (Requirements)” [18]

In this topic, we will discuss with the agile development team their practices with regard to prioritization and management of requirements. We will try to determine to what extent architectural quality attributes are considered in their requirements engineering process.

3.3 Quality Process and Practices

ISO 9000 defines quality as “the degree to which a set of characteristics fulfills requirements”, the characteristics being in this case being those of the software product under development, and the requirements are those both stated and implied for the software system. A first impression can only be made once, and having a high quality product fosters good will amongst stakeholders, elevates team morale, and helps the organization capture a greater market share.

The author’s perspective is that software architecture and design techniques must be employed to achieve the desired software quality attributes (e.g., in ISO/IEC-9126-1: functionality, reliability, usability, efficiency, maintainability, and portability).

The literature describes how quality standards like Six Sigma fit into an agile process [32], but it is not clear that agile teams in general will follow such a well defined quality assurance process.

In this topic, we will discuss with the agile development team their practices and approaches with regard to quality assurance in their specific agile environments, including how quality of the product is measured, who performs the assessment, and how the architectural quality attributes are considered in tuning the quality assurance process.

3.4 Risk Management

Risks are potential problems or perhaps opportunities that may occur during a project. Risk analysis and mitigation is an integral part in the development process [22] [45]. The fundamental characteristics of risks [39] are probability and impact. Probability is the possibility of risk event may occur and the impact is the effect the risk will have on the software development effort, and both are quantitatively or less ideally, qualitatively defined. A Risk Management plan is put in place which contains a set of actions that will be used to reduce the probability of the risk occurring or to reduce the impact if the risk event occurs. [38].

Agile projects rarely define a risk management strategy [22]. Agile teams tend to ignore the risk management discipline and instead rely on the core agile principles to inherently manage risk. Yet, it has been shown that these principles alone are not enough to handle risk [16] [33] [45].

Some agile methodologies introduce the concept of a “Risk backlog” that can be used for management of risks. The Risk backlog contains the list of risks while the activities associated with a risk backlog are mentioned in [23]. An alternative to an independent risk backlog is to combine the product and the risk backlogs together allowing risks to be considered and prioritized along with the product backlog tasks at the end of each iteration [23].
In this topic, we will discuss with the agile development team, the practices that they follow to identify and mitigate risks, and whether risks are traced to architectural quality attributes.

3.5 Technical Debt

‘Technical Debt’ has been defined as [2] "...doing things the quick and dirty way sets us up with a technical debt, which is similar to a financial debt. Like financial debt, the technical debt incurs interest payments, which come in the form of extra effort that we have to do in the future development because of the quick and dirty design choice..." - Martin Fowler. ‘Technical Debt refers to delayed technical work that is incurred when technical short-cuts are taken. Explicit decision making before taking on debt and more explicit tracking of debt are advised’ - Steve McConnell.

Development teams should take effort to pay down technical debt because unmanaged technical debt creates increased maintenance costs in the future [24].

Refactoring is a mechanism for paying down technical debt. However, refactoring at the component level may not be sufficient to support architectural change resulting in architectural-level technical debt [24] [43].

In this topic, we will discuss with the agile development team their practices and approaches that they follow to monitor and manage technical debt at the architectural level.

3.6 Documentation

Providing working software over comprehensive documentation is one of the core points in the agile manifesto [9]. This point addresses delays in product delivery due to excessive documentation in traditional software development methodologies. While this is a legitimate concern, agile methodologies compromise the benefits of documentation for the sake of producing a partially complete system.

The purpose of providing documentation is to inform those who are not familiar with the system [26]. Agile encourages tacit knowledge rather than knowledge captured in documentation [6]. Still, agile development needs "just enough" documentation [6] [31] to capture architectural design decisions. Finding "just enough" documentation may vary by project.

High level design with architectural specifications, design rationale, rejected design alternatives cannot be properly documented in code [26] [27]. While concise documentation is a noble goal, it still must be kept up to date each iteration [31] [40].

In this topic, we will discuss with the agile development team their practices and approaches in order to determine how architectural decisions are documented and tracked.

3.7 Architectural Practices

Software architecture is defined as “the set of structures needed to reason about the system, which comprises software elements, relations among them, and properties of both” [23].

Software architecture describes the big picture of the system and provides a common language for understanding among the stakeholders (e.g. customers, developers and testers). Software architecture focuses on quality attributes of the software system [23] [43].

The literature specifies a lack of scientific support concerning agile and architecture [10] and that care should be taken to blend architecture into agile without disturbing the principles of agile [43]. It is an open question to find if there are well defined architectural practices that can be applied on all agile projects.

Some agile teams believe that Test Driven Development (TDD) replaces the design [30]. It is our concern that TDD should not be considered a replacement for creating a proper architecture.

In this topic, we will discuss with the agile development team their architectural practices that they follow and to determine how those practices were integrated into their methodology.

3.8 Project Management

Project management is an integral part of any software development process which manages every aspect of the project. It covers people, process and technology [42].

Project Management in agile methodologies is not a matter of control but one of co-ordination. Development activities are managed and monitored to achieve the agile business objectives [45]. The relevant project management activities are project planning, risk management, quality management, hiring process, contract management, and procurement. In each of these activities, decisions made will affect how the project team manages the architecture during development of the software. For example, a team comprised of veteran developers may need a less rigorous architectural description than a less experienced team.

In this topic, we will discuss with the agile development team how the project was operated in the wider context of an organization, how the project manager manages balances the development activities versus the long term needs of the project.

4. CONCLUSION AND NEXT STEPS

In this paper, we have defined an interview guideline based on several topics to probe the balance between software architecture and agile development. For each topic we have shown the relevance to software architecture.
This interview guideline can be used by a project team to understand how architecture and architecture qualities are being managed on a given agile project.

We believe that this breakdown of topics is complete with regard to exploring software architecture best practices in agile development projects.

After conducting interviews and collecting a significant body of results, we will perform an analysis on the data to define a common vocabulary allowing us to contrast the approaches of the disparate software development teams. A follow-on paper will describe our analysis technique that will be based on an objective scientific approach.

5. APPENDIX – SAMPLE QUESTIONS

Here are some sample questions for each interview guideline topic.

5.1. Development Process and Practices
1. How did you choose the approach? Did you have any architectural consideration? Did you strictly follow the practices of the approach?
2. How did you choose the items on your product backlog and how did you prioritize it?
3. How did you plan the Sprint? Describe the typical activities in sprint?
4. How did you communicate the architectural decisions and how did you make sure the decisions are communicated to the development team?

5.2. Requirements
1. When user stories do not describe any architectural quality attributes, how do you deal with architectural quality attributes in the project?
2. Did you have any steps or tools to expand the requirements or user stories concerning architectural quality attributes? How did you prioritize the requirements list?
3. How did you negotiate the changes of the functional requirements? Do you think your customers are aware of the impact on non-functional requirements?

5.3. Quality Process and Practices
1. What quality practices did you employ to improve the quality in agile project? How did you track and measure the product quality?
2. How does your quality practice consider architectural quality attributes? How did you define the “definition of done” for the quality?
3. What techniques did you follow to improve your quality practices? Did you change your quality processes based upon architectural quality attributes?
4. Did you practice Continuous Integration in your project? When you perform the integration, how did you ensure the architectural quality attributes?

5.4. Risk Management Practices
1. Do you believe your chosen agile approach itself manages risk or did you follow any risk management practices?
2. What techniques or risk analysis did you follow to identify the risks before iteration starts?
3. If you practice to deal with the risks after you come across during the development phase, how did you handle it?
4. Did you have risk backlog? Did you consider having risks concerning architectural quality attributes in the risk backlog? How did you prioritize the risk backlog?

5.5. Technical Debt
1. What trade-offs did you make during design?
2. How did you monitor the architectural decisions throughout the project?
3. What techniques did you follow during architectural refactoring to fix or improve the architectural quality attributes of the system?

5.6. Documentation
1. What practices did you employ for the documentation at design level? How much time did you spend on documentation?
2. As architectural decisions are made throughout the project, how do you track/check the agreed upon changes are incorporated?
3. What techniques did you follow to create your chosen level of documentation?
4. How documentation is kept in sync with the project?

5.7. Architectural Practices
1. What are the architectural practices did you follow in your agile project and were those practices are sufficient? What did you consider as the key architectural decisions that needs to be taken early in the project?
2. Did you have architectural sprints? With the evolving requirements in the development, what kind of control did you have towards architecture?
3. As agile follows quick development, how did you make sure that the design standards are followed?
4. Did you use any architectural styles/patterns? Did you use any frameworks relating user stories and the quality attributes?

5.8. Project Management
1. What practices did you follow to incorporate organizational knowledge into your project approach during start up and close out of the project?
2. Did you hire candidates from outside based on the experience in the agile environment, architectural skills, technical lead experience, domain knowledge or internal candidates based on previous project performance?

3. In your estimation, did prior architectural experience assist in capturing architectural aspects or the definition of architectural quality attributes?

4. How did you manage contracts with the customer and what are the collaboration practices did you follow to maintain the customer relations?

5. In relation to procurement, how did you pass through the requirements? How did you co-ordinate the teams?

6. How did you examine the quality attributes of third party frameworks?

7. What practices did you employ to measure the project progress and was that sufficient?

REFERENCES

[27] T. Sauer, Using Design Rationale for Agile Documentation, IEEE International Workshops on...


