On the Responsibilities of Software Architects and Software Engineers in an Agile Environment: Who Should Do What?

Antony Tang  
VU University  
Amsterdam  
the Netherlands  
atang@cs.vu.nl

Ton Gerrits, Peter Nacken  
Océ Technologies  
the Netherlands  
[ton.gerrits,peter.nacken]@oce.com

Hans van Vliet  
VU University  
Amsterdam  
the Netherlands  
hans@cs.vu.nl

ABSTRACT
As there is considerable freedom for software workers to decide what to do in an agile environment, there is a need to be explicit about social aspects such as task ownership between roles and the agility in owning those tasks. We have conducted interviews and a survey at a large organization to explore these issues. The results indicate that there are areas where architects and software engineers have different views on task ownership. Some of the less exciting tasks such as documentation and issue clarification then easily become a hot potato nobody wants.

Categories and Subject Descriptors
D.2.1 [Requirements / Specifications], D.2.2 [Design Tools and Techniques].

General Terms
Design, Human Factors.

Keywords
Agile development, design responsibilities, task ownership.

1. INTRODUCTION
Software architects have many responsibilities in a software team. They gather and understand the requirements from users and stakeholders, analyze the technical feasibility of the design and implementation, ensure that the implementation fits the project scope, budget and schedule [1]. Most important of all, software architects translate and communicate relevant information between different groups of stakeholders such as business users, managers, other architects and technical specialists and software engineers.

With the wide variety of functions that architects have to perform, their responsibilities can depend on the project situation and are not easily defined. Architecture frameworks and standards such as TOGAF [2] and ISO/IEC 42010 [3] only provide general guidelines. It is up to individual organizations and their architects to interpret those guidelines to define work practices and responsibilities, so architects’ responsibilities are subject to interpretation and individual preferences.

When an architect adjusts his/her responsibilities to suit the project situation, their responsibilities may not be made explicit to the software development team. Furthermore, in an agile environment software architects and developers often have overlapping responsibilities, the division of responsibilities cannot be sharply defined. For instance, both roles may do analysis, design, documenting, coding and even testing. The division of responsibilities is further complicated when there are different levels of software architect in a project.

If the mutual understanding of some responsibilities and tasks is not sharply defined and the workers in a software team do not clearly communicate what each person would do, a number of issues can occur. Firstly, architects may be internally focused, working in an ivory tower or working on their technical design in isolation, resulting in undesirable products [4], and the same can be the case for software developers [5]. Secondly, a misunderstanding of responsibilities may create delivery gaps because each person assumes the others would do the work. Thirdly, there may be unrecognized issues in requirements and design as no one has complete knowledge of the situation.

We studied these collaboration aspects within Océ Technologies, a large organization that has applied an agile development methodology in the past five years. The organization is flat and workers are encouraged to be proactive in owning up to work responsibilities. Océ Technologies has deliberately opted for a flat structure, as opposed to a hierarchical structure, to foster cooperation, innovation and entrepreneurship. In this paper, we explore the mutual expectations of architects and software engineers with regards to each other’s responsibilities and task ownership. The research involved over 50 architects and software engineers at Océ Technologies. We use two research methods in this study: interviews, followed by a survey.

In the interviews, architects and software engineers have indicated that documentation is an area that needs improvement. From the survey, respondents generally agree that documented knowledge is important. They also agree on the general responsibilities of architects and software engineers. But when it comes to
pinpointing who should perform certain tasks and what the extent of those tasks is, disagreements between the roles surface. Some of the less exciting activities such as issue clarification and documentation become a hot potato.

2. BACKGROUND

2.1 Related Work

There are many processes, methods and best practices to guide the practice of enterprise, system and software architecture. These processes and methods typically define the outcomes and the steps to achieve those outcomes. However, the implementation of a process or method needs to be tailored for each organization. The responsibilities and tasks need to be defined according to the people and the type of software being produced. Let us consider two extremes. Organizations that produce safety critical software can be rigid in defining the responsibilities and tasks that software architects need to perform to ensure that all the work steps are followed. On the other hand, some organizations practice agile development in order to receive early user feedback, and the responsibilities of an architect can be more flexible.

In the latter case, the key job responsibilities of an architect in terms of design, documentation and communication [6] can be defined at a high-level without a clear articulation of what tasks and what steps are required, and who would have ownership. The details are left to the architects and the software engineers to work out. There may be explicit agreements on who should do what or the agreements may be vague when the responsibilities are shared and more than one party is capable of performing the tasks. Workers in such a team have a tacit understanding of each other’s responsibilities through empowerment and self-motivation [7].

Hsieh et al. proposes an AxE model [8] to represent the differences in what is anticipated (A) to be delivered by a supplier of some information and what is expected (E) by a consumer of that information. For instance, an architect produces a requirement specification or an architecture design specification which the software engineers use as a basis for implementation. A mismatch in conveying information between the two parties can happen with the content, the quality or the timing of the delivery of the information, preconditioned by what information needs to be conveyed. In an agile development environment where there is liberty to adapt the work process to fit work situations, the chance for an expectation mismatch can be high if communication between co-workers is not clear. Turk et al. [9] suggest that some people practicing agile development assume that software engineers have the experience needed to define and adapt their processes appropriately; and some assume that documentation and software models do not play central roles in software development. These assumptions may create expectation mismatches amongst co-workers. Coplien and Harrison have named different roles in agile development organizations but the responsibilities of these roles are vaguely defined, they suggest that stakeholders can communicate with each other to gain an understanding [10].

Most studies of agile development focus on ‘pure’ agile approaches; see for instance the overview provided in [11]. More recently, one has started to realize that both traditional approaches, with their emphasis on upfront planning, and agile approaches, have their merits. Hence the attention to reconciling agility and architecture (see for instance the special issue of IEEE Software [12]). Especially in agile software product lines, there is an inherent tension between the level of the product line and that of individual teams realizing components thereof. At the level of the product line, one wants to plan and lay down the architecture of the product line. At the level of individual components, an agile team wants to be agile, and not bother about documentation and other long-range issues [13]. Baskerville et al. gives an interesting longitudinal study of the evolution of this tension [14]. In our literature search, we have found no empirical studies with a specific focus on mutual expectations of job responsibilities and task ownership between software workers in an agile development environment.

2.2 The Agile Software Organization

Our study was conducted in Océ Technologies. Océ Technologies produces high-end printers for the business markets in high-volume printing, wide-format printing and office printing. Printer software is one of the main components in a printer. The software interacts with users, renders images and controls the print engine. Development of a printer product is executed by a project team, part of which is responsible for software development. Each software project has a lead architect. There are many sub-teams in a software project and each sub-team consists of typically 2-7 people. Each sub-team has a unit architect and several software engineers; they are responsible for sub-system development. All teams conduct daily scrum meetings. Team sprints are typically every two weeks, and a release is typically every eight weeks. Software engineers are highly motivated and knowledgeable. Documentation is kept to a minimal on purpose. The culture can be described as innovative where the best people are assigned to the problems, and job rotation is common.

Océ Technologies has employed agile development methodology in the past five years. Their development environment is a mixture of plan-driven design and agile development. This mixed approach of planning and agile is similar to the one described in [15]. Using approved roadmaps and product features as a basis, architects create a high-level architecture design. A high-level architecture design must be approved by management and the architecture council, and is developed in consultation with unit architects from the sub-teams. The approved architecture design is then distributed to the sub-teams for implementation. This is a plan-driven process. Each sub-team would plan their implementation internally and in consultation with other sub-teams. Together they create a delivery plan for each sprint and release. At the sub-team level, decisions about the execution of the plan and the process of design and implementation are agile.

The agile development process at Océ Technologies is documented in wiki pages and some process documents. Additionally, there are specific job descriptions for lead architects, unit architects and software engineers.

In a development team that has three main roles: lead architects, unit architects and software engineers, the core responsibilities of each role are clear but there are responsibilities that overlap between roles (Figure 1). The lead architect is responsible for all major software product requirements and architectural design. A significant amount of his efforts go into architecture analysis and planning. The unit architect in a sub-team is responsible for the architecture design of the sub-system and the detailed sub-
system requirements. There are some overlaps of the responsibilities of the roles, and their typical responsibilities are:

• Lead Architect and unit architects – requirement and architecture design specifications of a system are the responsibility of the lead architect; detailed sub-system specifications are the responsibility of the unit architects. They hold regular project and review meetings, and communicate verbally and through emails.

• Unit architects and software engineers – unit architects are responsible for detailed sub-system specifications. They use scrum meetings, reviews and verbal communication to discuss the design. Software engineers implement the sub-system design. Unit architects from different teams also work together to design and define interfaces between their sub-systems.

The major responsibilities of a lead architect are architecture planning and high-level architecture design. He produces architecture design specification up-front that is used by all the development sub-teams. On the other hand, the major responsibilities of unit architects and software engineers are implementation. They are agile-driven. In agile development, tasks are created through a planning and negotiation process, called the Iteration Planning Meeting. Both a design and its associated tasks are decided on-the-go and can change as the design develops.

![Figure 1. Responsibility Overlap in an Agile Team](image)

A high-level architecture design is implemented by many sub-teams and interpreted by many unit architects. The interfaces and behavior of a system overlap in different sub-systems, so there are on-going design conformance issues between them. Discussions and decisions on design issues, clarifications, changes and documentation that arise in both high-level architecture design and implementation need to be dealt with. The overlap of these responsibilities and tasks can create misunderstanding, and such is the motivation of this study.

### 3. RESEARCH OBJECTIVES

The agile development process at Océ Technologies provides an environment that encourages people to communicate and to perform. This environment encourages software workers to decide what responsibilities and tasks should be taken up to achieve the results. It has worked very well as people are highly motivated and proactive. Océ Technologies treasures this agile culture to “let all flowers bloom”. Software development is a cooperative process, where each role relies on the others and so there are mutual expectations of what each team member would do. A lead architect may expect from unit architects to carry out certain tasks, whereas unit architects may expect the lead architect to do that work. There may be differences in expectation as to who has the responsibility to deal with a certain situation. When people are familiar with each other’s work practices, there is a tacit understanding of each other’s responsibility. But when new people join a team or people move to another project, this tacit understanding must be reestablished. There appear to be potential responsibility gaps between a lead architect, unit architects and software developers. The potential gaps do not appear to be addressed by the documented development process. As such, we want to understand the following:

• Is there any expectation mismatch of who should perform certain tasks between lead architects, unit architects and software developers?

• If so, where do the mismatches occur?

### 4. RESEARCH METHODS

We planned this research in two parts. Firstly, we carried out exploratory interviews to investigate if there are any perceived differences of responsibilities between lead architects, unit architects and software developers. If any differences exist, what are they and how do they affect software quality? Secondly, we conducted a survey to understand the perceived responsibilities and tasks ownership between lead architects, unit architects and software developers under different work scenarios.

#### 4.1 Exploratory Interviews

In order to understand the differences in responsibility expectations between lead architects, unit architects and software engineers, we interviewed 9 people from 3 projects. These 3 projects were picked because certain architecture-related issues had occurred causing delays or reduction of functionality. In each project, a lead architect, a unit architect and a software developer were interviewed. Interviews were recorded and the interviewees were assured of the confidentiality and anonymity. The length of the interviews was between one and two hours each. The interviews were semi-structured. The interviewees were asked to describe the issues that were encountered. The interviewees were also asked to discuss what happened in the project, what they thought about the development process, why something went wrong and who was responsible. The interviewees were asked to discuss the responsibilities each role should take and what they expected from each other. Interviewees could freely comment and discuss the projects, the issues and their job responsibilities. The interviewers could ask follow-up questions deemed interesting to this research.

#### 4.2 Online Survey

The analysis from the interviews indicates that there are misalignments in three general areas between architects and software engineers: (a) perceived responsibilities of job roles; (b) accessing documented knowledge; (c) task ownership. To understand these misalignments, we designed an online survey to gather information from architects and software engineers at Océ Technologies. One hundred and thirteen (113) invitations were sent to two software development business units. Both units are located in two sites. The survey was open for three weeks and participation was optional. Sixty six (66) responses were received.
There were 57 questions in the questionnaire. The questions were categorized into five groups: (a) general information about the respondents such as job role and experience; (b) perceived responsibilities of a job role; (c) accessing documented knowledge; (d) task ownership by roles; (e) current working methods. Apart from the two open questions in group (e) in which the respondents were asked to write their opinion, all responses to group (b), (c), (d) and (e) questions are in 5-point Likert scale where one (1) indicates strong disagreement and five (5) indicates strong agreement. A pilot test was conducted with four participants involved. Wordings in the questions that were ambiguous to the pilot participants were corrected before the actual survey.

4.3 Statistical Analysis Method

Two statistical methods are used in the analysis: (a) descriptive statistics to report the opinions of each group in general; (b) statistical tests to analyze the difference in opinion by comparing the means of each group. The comparison of means is performed by ANOVA. ANOVA is used for a 3-way comparison instead of pairwise t-test to avoid experimentwise Type I errors. ANOVA compares the systematic variance in the data to the unsystematic variance. It shows whether the means of the three groups are different or not (H0: \( \bar{x}_1 = \bar{x}_2 = \bar{x}_3 \)). In this test, the null hypothesis is that there is no significant difference between the means of the three groups (lead architects, unit architects, and software engineers). The alternative hypothesis H1 is to reject the null hypothesis and accept that there is a difference in opinion. If ANOVA shows that there is a significant difference between the means, i.e. rejecting H0, it doesn’t indicate which pair of groups is significantly different. To identify the difference pairwise, a further test of pairwise comparison of different groups needs to be conducted using the Tukey test. This test makes adjustments for the experimentwise error rate.

5. RESULTS

5.1 Exploratory Interviews

The recording of the interviews was transcribed and summarized by two researchers. We aim to understand three aspects: (a) an account of the key development process issues in the project; (b) an account of the tasks that were performed by each role; (c) a retrospective account of who should have done certain tasks in the project. The interviewers’ views on the development process issues and their expectations are summarized in Table 1. The interviews were analyzed to see what architects and software engineers thought the issues were, and if any responsibilities had been omitted during the design and development process due to misalignment of expectations.

From the interviews, we have some observations. Firstly, the views on project issues and where they happened are different between roles. Lead architect in P1 thinks that his project is fine whereas the unit architect and the software engineer are concerned about the comprehensiveness of requirements and design. Secondly, a major expectation by all three roles is the use of documentation to communicate requirement and design knowledge. Some interviewees mentioned that the documentation is insufficient. The interviewees suggested that the usefulness of documentation lies in a number of areas: (a) the recording of specific system interactions and behavior, e.g. the interviewees

5.2 Survey Results

The responses to the survey come from three groups: lead architects, unit architects and software engineers, belonging to two different business units, referred to as Unit A and Unit B. There are 48 Unit A respondents from two geographic locations and 18 Unit B respondents from two locations. Although Unit A is a bigger than Unit B, Unit B response rate, especially that of the software engineers, is proportionally much lower than Unit A. We noticed we got no responses from any software engineer from one Unit B site. The invitation emails did not reach the Unit B software engineers due to a technical difficulty. Thus, Unit B respondents by role are not proportional to the Unit B population, this would create a bias in the result and so we do not consider the Unit B responses in this article.

The distribution of the Unit A respondents by role is listed in Table 2. The experience level of the respondents in Océ and their current role is considered (see Table 3) because it is an indicator of whether the respondents know the development process and the working methods well. Most Unit A respondents (62%) have spent over 5 years in software development in Océ. Moreover, 48% of Unit A respondents have been in the same role for over 5 years. The software workers have job stability in the company and their job roles. This is an indication to believe that the responses reflect their true opinions.

<table>
<thead>
<tr>
<th>Table 2. Unit A Respondents’ Roles</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unit A</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 3. Unit A Respondents’ Experience</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unit A (Océ)</strong></td>
</tr>
<tr>
<td>------------------</td>
</tr>
<tr>
<td><strong>Unit A (current role)</strong></td>
</tr>
</tbody>
</table>

suggested that different modeling techniques may be used to communicate design ideas clearly; (b) communicating high-level system design, especially to people who are new to the system; (c) retaining decisions and design rationale that are tacit. Unit architects and software engineers in P2 and P3, and the P3 lead architect have noted this issue. However, there is no consensus as to who should own these tasks.

Thirdly, all roles require information from each other, but sometimes they do not know where to get it. The lead architects need detailed design information from the unit architects and sometimes implementation details from the software engineers, as noted in P2 and P3. The unit architect needs implementation details from the software engineers and the requirements and interface design from the lead architect. The software engineers need requirements and design specifications from both types of architect. However, this need for information is unknown to the people who can provide it, and so the information flow does not happen. Overall, there is no explicit understanding as to who should perform certain tasks such as documentation and clarification of design or requirement ambiguities.
Table 1. Summary of Interview Responses

<table>
<thead>
<tr>
<th>Project/Role</th>
<th>Issues Encountered by Interviewees</th>
<th>Expectations of Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1/Lead Architect</td>
<td>- Lead architect was not aware of any responsibility or task issues in his project.</td>
<td></td>
</tr>
<tr>
<td>P1/Unit Architect</td>
<td>- Architecture design not comprehensible</td>
<td>- Lead architect should document designs clearly and put them in a repository where workers can find them.</td>
</tr>
<tr>
<td></td>
<td>- Information about the architecture design is dispersed and sometimes undocumented</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Design is incomplete</td>
<td></td>
</tr>
<tr>
<td>P1/Software engineer</td>
<td>- Requirements are fuzzy and architect is unavailable to help</td>
<td>- All architects should clarify and explain specification</td>
</tr>
<tr>
<td></td>
<td>- Not knowing what is the latest requirement</td>
<td>- Lead architects should provide use cases and sequence diagrams</td>
</tr>
<tr>
<td></td>
<td>- Insufficient documentation</td>
<td>- Lead architects should document high-level system features</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Unit architects should document sub-system component functions</td>
</tr>
<tr>
<td>P2/Lead Architect</td>
<td>- Agreed requirements cannot be achieved through existing design</td>
<td>- Unit architect should provide full information in stand-up meeting</td>
</tr>
<tr>
<td></td>
<td>- Critical use cases not investigated before committing to plan</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Unit architect did not have sufficient knowledge of sub-system</td>
<td></td>
</tr>
<tr>
<td>P2/Unit Architect</td>
<td>- System behavior of requirements is unclear</td>
<td>- Should have high level documentation of system</td>
</tr>
<tr>
<td></td>
<td>- Insufficient documentation to show what was implemented</td>
<td>- Management should provide clear assignment of responsibilities</td>
</tr>
<tr>
<td>P2/Software engineer</td>
<td>- Design decisions were made via many emails</td>
<td>- Architects should better evaluate requirements</td>
</tr>
<tr>
<td></td>
<td>- Finding design knowledge by knowing whom to ask</td>
<td>- Should involve software engineers at a technical level</td>
</tr>
<tr>
<td></td>
<td>- Changing requirements and personnel make it difficult to retain knowledge</td>
<td>- Impact analysis should be done more thoroughly by understanding both the requirements and architecture design</td>
</tr>
<tr>
<td>P3/Lead Architect</td>
<td>- Requirements not properly documented</td>
<td>- Unit architects should contribute to architecture design document</td>
</tr>
<tr>
<td></td>
<td>- System behavior, especially exceptions, were not documented</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Some software engineers ignore what architects decide</td>
<td></td>
</tr>
<tr>
<td>P3/Unit Architect</td>
<td>- New requirements are uncovered during design because of incomplete analysis</td>
<td>- Should use interface specifications to communicate design</td>
</tr>
<tr>
<td></td>
<td>- Design constraints are undocumented and not communicated</td>
<td>- Should document design rationale</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Unit architect should actively participate to check fitness of an architecture design</td>
</tr>
<tr>
<td>P3/Software engineer</td>
<td>- No high-level architecture specification</td>
<td>- Should have vertical documentation at all levels</td>
</tr>
<tr>
<td></td>
<td>- Design specification is written during implementation</td>
<td>- Should keep documentation up-to-date</td>
</tr>
<tr>
<td></td>
<td>- Meeting delivery targets takes priority over good documentation</td>
<td>- Need in-depth implementation knowledge in design</td>
</tr>
</tbody>
</table>

5.2.1. Documented Knowledge. The interviewees have indicated the need for improved documentation and communication of architecture knowledge. In the survey, we pose a set of twelve (12) questions to the respondents about communicating and storing design knowledge. All three roles strongly agree that documentation should be kept systematically and in an organized manner. All three roles agree that the use of minutes and documentation of changes is not the proper way to communicate design knowledge. When the respondents are asked if it is sufficiently clear where and what information is stored (Q51), the general response from lead architects, unit architects and software engineers is in disagreement (X̄=2.00, X̄=2.55, and X̄=2.61 respectively).

5.2.2. Perceived Responsibilities. In this section, we pose questions to measure the perception of responsibilities is aligned between roles. We want to understand if the three different roles have the same ideas about their respective responsibilities in certain aspects of software development. We have used 18 questions to ask about four topics. Firstly, we ask questions related to the responsibilities of the three roles in terms of who should be responsible for cross-project issues (Q7), e.g., the design and development of software that can be shared with other project teams. The purpose is to understand who, amongst the architects and software engineers, should discuss these issues with another project team. The general consensus is that this is the responsibility of the lead architect.

Secondly, we ask about the expectations of the responsibilities of different roles. For instance, should software engineers discuss requirements with architects before implementation (Q18); should lead architects involve unit architects up-front in the design phase. There is a general consensus that unit architects should be involved up-front, and software engineers may be involved but to a lesser extent. Thirdly, we ask whether software engineers and unit architects should understand high-level requirements (Q17); and if the lead architect should communicate high-level product architecture design. There is a consensus that lead architects should communicate high-level architecture design to unit architects and software engineers, and the software engineers and unit architects should understand the high-level design.

Fourthly, we ask who should take proactive steps, i.e., if one sees a potential issue then one should take steps to address the issue. In general, most of the respondents agree that unit architects and engineers should help to address issues and complete the requirements and design documents. When we ask specifically if unit architects should be involved in searching for potential technical issues (Q15) during architecture design review process, we get a statistically significant difference (F=3.304, p=0.046) between the different roles. Lead architects are only slightly in agreement about unit architects’ involvement (X̄=3.83), but both unit architects and software engineers strongly agree in unit
architects’ involvement ($\bar{x}=4.64$ and $\bar{x}=4.58$, respectively, Figure 2).

![Figure 2. Unit Architecture’s Role in Resolving Potential Technical Issues](image)

**Figure 2. Unit Architecture’s Role in Resolving Potential Technical Issues**

On the issue of whether a unit architect, when acting as a reviewer, should ensure that his/her team of software engineers understand the requirements and design ($Q16$) in order to carry out impact analysis properly, there is a statistically significant difference ($F=6.54, p=0.003$) between unit architects ($\bar{x}=3.82$) and lead architects/software engineers ($\bar{x}=4.83$ and $\bar{x}=4.68$, respectively). Unit architects are slightly positive that s/he should make sure that his/her team members understand the requirements and the design, whereas the other two roles strongly think unit architects should make sure that software engineers understand the proposed requirements and design.

5.2.3. Task ownership. The interviews have indicated that documentation is a key responsibility but it was unclear from the interviews to what extent each role should contribute to this task. Each role produces some documentation and each expects the others to do so as well. The interviewees have also noted that the requirements and design are sometimes fuzzy but there is no clear indication as to who should clarify them. We have a set of fifteen questions (15) to identify who should be assigned the task to document and who should be responsible for design activities. We ask whether the lead architects should do the documentation planning and delegate most of the writing to the unit architects ($Q24$). Lead architects strongly agree to this statement ($\bar{x}=4.17$) but the other two roles are neutral ($\bar{x}=3.18$ and $\bar{x}=3.19$, respectively). This seems to indicate that lead architects are more inclined to delegate the tasks of documentation than to document themselves.

When we suggest that the unit architects should write most of the software requirement document ($Q22$), the difference in opinion becomes clear. Lead architects agree that unit architects should write most of the requirement document ($\bar{x}=3.83$) but unit architects and software engineers disagree ($\bar{x}=2.45$ and $\bar{x}=2.81$, respectively). The difference in opinion is statistically significant ($F=4.95, p=0.011$). Lead architects strongly agree that unit architects should write most of the architecture design document ($Q23$) ($\bar{x}=4.67$) but unit architects and software engineers are neutral ($\bar{x}=3.27$ and $\bar{x}=3.26$, respectively). The difference is statistically significant ($F=9.94, p=0.000$).

Furthermore, when we ask if certain specific tasks should be performed by a role, there are major disagreements:

**Task 1 – Clarify Requirements:** In case of missing requirements, the lead architect should lead requirement clarifications ($Q46$). Lead architects strongly disagree with this ($\bar{x}=1.83$) whilst unit architects and engineers strongly agree ($\bar{x}=4.09$ and $\bar{x}=3.90$, respectively). There is a statistically significant difference between the lead architects and the others ($F=12.65, p=0.000$).

**Task 2 – Clarify Missing Sub-system Requirements & Design:** In case of missing sub-system requirements and architecture design, software engineers should demand these from the unit architect ($Q47$) since unit architect is responsible for the sub-system design. Lead architects disagree ($\bar{x}=2.67$) whilst unit architects and engineers strongly agree ($\bar{x}=4.18$ and $\bar{x}=4.52$, respectively). There is a statistically significant difference mainly between the lead architects and the others ($F=9.06, p=0.000$).

**Task 3 – Clarify Missing System Requirements & Design:** In case of missing system requirements or architecture design, software engineers should demand these from lead architects ($Q48$). Lead architects strongly disagree with this ($\bar{x}=1.83$) whilst unit architects and engineers agree ($\bar{x}=3.00$ and $\bar{x}=3.68$, respectively). There is a statistically significant difference between the lead architects and the others ($F=7.982, p=0.001$).

**Task 4 – Correct System Requirements & Design:** In case of incomplete or incorrect system requirements or architecture design, unit architects should demand updates from the lead architects ($Q49$). Lead architects strongly disagree with this ($\bar{x}=1.50$) whilst unit architects and engineers slightly agree ($\bar{x}=3.73$ and $\bar{x}=3.71$, respectively). There is a statistically significant difference between the lead architects and the others ($F=9.643, p=0.000$).

6. MAIN FINDINGS

Océ Technologies practices an agile development methodology and architecture planning. Although they have a documented software development process and the workers frequently communicate with each other and are willing to exchange design knowledge people, we have found three main areas of responsibility mismatches between architects and software engineers.

**1. Documentation.** In an agile product line environment, a personalization strategy [16] using personal communication plays an important role. However, this study shows that a codification strategy with documentation is important as well, but who is responsible for producing certain part of the documentation is unclear.

On using documentation, unit architects and software engineers have indicated that they require more documented information from the lead architects. However, they generally do not know where the documented knowledge is ($Q51$), and this result agrees with the results from the interviews. Moreover, everyone agrees that the architects should provide information to help the software engineers gain knowledge about the architecture of the released software. This can be achieved by (a) a verbal explanation on architecture design must be supplemented by documentation ($Q43$); (b) documentation must be communicated verbally by architects to ensure its contents are understood ($Q44$); (c) all requirements and architecture design documentation must have clear status and versioning ($Q45$).

On producing documentation, there are disagreements as to who should write the specifications in certain scenarios. When we ask if lead architects would do the documentation planning, and the
other two roles would do the writing (Q22, Q23, Q24), the lead architects are more enthusiastic about this idea than the other two roles. When we ask if unit architects should write most of the requirement documents and architecture design document, the lead architects agree and the other two roles disagree. It shows that lead architects want to plan documentation instead of writing them, but unit architects and software engineers think that both the planning and writing tasks belong to a lead architect.

Additionally, when the respondents are asked if software engineers should write (a) requirement documents (Q13) and (b) architecture design documents (Q14) when they are incomplete, the lead architects are more supportive of these ideas than the unit architects or the software engineers. It may be that lead architects want to have additional design details in the documentation which they do not possess or do not wish to write, and they expect that this knowledge to be documented by the software engineers.

2. Architecture Requirements and Design Responsibility. When we ask about cross project responsibilities (Q7), there are no disagreements that the lead architects should be responsible for this activity. When we ask if software engineers (Q18) and unit architects (Q17) should understand requirements before implementation, there are no disagreements between the respondents. Respondents generally agree that lead architects should be responsible for the architecture design, with the help from unit architects and developers (Q25). There are also no disagreements of software engineers’ involvement in helping with requirements and design up-front.

There are, however, disagreements in requirements and design clarifications. When we ask if the unit architect should ensure the development team understands the requirements and the design (Q16). The lead architects and the software engineers strongly agree that this is the unit architects’ responsibility, but the unit architects, who are the ones to act, are neutral. This may be a reflection of lead architects and software engineers’ expectations that unit architects act as the middle-person to communicate and translate design ideas between the two roles. It is expected that the unit architects should translate and enhance high-level requirements and design to implementable design that the software engineers could use. Unit architects expect the lead architects to take up that responsibility as evident by their answers to Task 1, or they expect software engineers to clarify the requirements and the design by themselves.

Our results have shown that the responsibility for dealing with architectural issues should be made clear.

We ask if lead architects should lead the efforts to clarify architectural requirements. Lead architects disagree but the other two roles agree. Results of the interviews have indicated that there is an expectation of the lead architects to confirm the requirements. As such, the unit architects and software engineers expect the lead architects to lead the clarification of requirements (Q46). The lead architect expects the unit architects to translate requirements to the software engineers (Q16). The lead architects may think that the unit architects are capable of doing this task.

We ask if unit architects should search for potential issues during architecture reviews (Q15), there is a statistically significant difference between lead architects and the other two roles. Lead architects are neutral whilst the other two roles strongly think that unit architects’ involvement is necessary. This may be because unit architects and software engineers think that the development of the design details is important at this stage. For instance, the software engineer in project P3 has mentioned this. The lead architects, on the other hand, think that potential issues are solvable without input from unit architects. On the other hand, lead architects may be confident that the high-level architecture design is good enough and as such they are not as keen for unit architects to perform the task to search for potential technical issues as it may delay the architecture planning process. Such a difference may be attributed to the tension between plan-driven architecture process and an agile development process.

When we ask if software engineers should demand missing subsystem requirements and design from unit architects (Q47), lead architects disagree. This is consistent with the answers to (Q46) where lead architects do not think that it is their responsibility to lead requirements clarification.

Lead architects also do not think that software engineers should demand missing requirements (Q48) from them, nor do lead architects think that the unit architects should demand updates to incorrect requirements and design from them (Q49). From these questions, we observe that the person whom should provide the knowledge and communicate that knowledge to the others have disagreed with that responsibility, whilst the other two roles agree.

3. Task Ownership. The task ownership results (task 1 to task 4) show disagreements when tasks are pinpointed to a specific role. The disagreements are about who should write certain documentation, who should lead requirement clarifications, and who can demand what information and from whom. Even though there is a general notion of mutual responsibilities, our results show that the interpretation of what that responsibility entails is unclear at the task ownership level.

When responsibilities are not defined in detail in an agile, each person can interpret his/her own responsibility in terms of tasks and actions, personal bias may come to play, depending on what helps the person to achieve the immediate goals, such as meeting a design or development milestone. For instance, the effort required by a person to do documentation can be spent on achieving a sprint. So although a person may agree to a responsibility in principle, s/he may not practice it when there are personal efforts involved. The extent of responsibility (Figure 1) cannot be defined in absolute terms for each role. In an environment where both agile development and plan-driven methods are used, the responsibility extent and the tasks that come with it need to be flexible and agile as well.

7. THREATS TO VALIDITY

In this work, we have used two different instruments, i.e. interviews and survey, to gather information. The multiple sources of information have provided construct validity to allow us to identify the key discrepancies in responsibility perception. Architects and software engineers in both the interviews and the surveys are concerned with documentation and they also point out similar issues in documenting knowledge. This has provided some evidence to support internal validity of this research.

This research is conducted in a single company from one business unit in two separate development locations. This case study is specific and therefore the results cannot be generalized beyond
this study. However, it provides some insights on the working relationships between architects and software engineers in a real-life situation. Our results also support the observations made by current literature in the areas of agile software development and software architecture [9, 17].

8. CONCLUSION
In this study at Océ Technologies, we investigated the responsibility and tasks of lead architects, unit architects and software engineers. Océ Technologies uses an agile development methodology. The development culture is innovative, personal initiative and responsibility is encouraged. The agile development culture is about creativity and efficiency, and documentation has a lower priority. There is considerable latitude to interpret job responsibility and task ownership. Job responsibility can be tacit and its interpretation can be different between teams and people.

From this study, we have observed that there are expectation mismatches on responsibilities and task ownership between the three development roles. The results show ambiguities of responsibilities in a number of areas: (a) who should produce certain documentation; (b) who should ensure the understanding of requirements and design by the software engineers; (c) when requirements and architectural design issues arise, who should be dealing with them.

These collaboration gaps between software architects and engineers are not explicitly defined in the software development process. The frequent communication advocated by the agile development process has not helped. It appears that these tacit responsibilities need to be made explicit between co-workers. The findings of this study provide some insights to help people working in these roles to be cognizant of the issues, and allow them to be aware of the needs to flexing the boundaries of their perceived responsibilities.

9. ACKNOWLEDGMENTS
This research has been partially sponsored by the Dutch "Regeling Kenniswerkers", project KWR09164, Stephenson: Architecture knowledge sharing practices in software product lines for print systems.

10. REFERENCES