

# The Boomeranged Software Architect

Rahul Premraj  
VU University Amsterdam  
rpremraj@cs.vu.nl

Gaco Nauta  
Océ-Technologies  
gaco.nauta@oce.com

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

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

**Abstract**—In an agile environment, where the architect’s role is not crisply defined, the architect may have to deal with a number of issues that arise during development and are considered architecture-related. Such issues range from feature requests and enhancements through to defects found during testing. Architect specifications that turn into issues *boomerang* the architect to resolve them. In a case study we looked at issues entered into the issue tracking system that were assigned to architects to understand the root causes for their inception. We found that many of the avoidable issues can be circumvented by better communication between the architect and other stakeholders, and by small adaptations in the development process.

**Keywords**—architectural issues; empirical study; agile development

## I. INTRODUCTION

The role of software architects in a project is beyond just the architecture of the software. Kruchten argues that architects should play a key role in coordinating between the design, implementation, and integration teams, in project planning activities, assessing technical risks and devising solutions to mitigate them, and maintaining the architectural integrity of the system [1]. Besides, architecting the software system in itself is non-trivial — it involves understanding the user requirements and translating them into product properties, identifying those that are relevant to architecture, making choices between alternative solutions, prototyping and validating them, etc. As a result, software architects deal with many aspects in development, from requirements to design details. As a result, the boundaries of their roles and responsibilities are not always sharply defined. This is the more so in agile environments where the architecture is not crisply defined early on [2], [3].

In a case study conducted in a large company in the Netherlands (Section II-A), we noted that architectural issues are assigned to software architects to be resolved. These issues are the tickets filed in the project’s issue tracking system that are assigned to the architects. They relate to various subject matters including feature requests, enhancements, and defects found during development and testing.

Resolving the assigned issues can take up a lot of the architect’s time that could be potentially spent elsewhere. Hence, the focus in this study is to perform a retrospective root cause analysis into the issues assigned to the software architect to understand why they arise, what are their types,

and how can their occurrences be reduced in the future through improvements in the development process [4]. For instance, errors may systematically arise from certain practices and filed repeatedly as issues — once such systematism is uncovered, they can be eradicated by the use of tools or methods that counter them [5]. Root cause analysis into defects has been previously applied in industrial research and led to key directions for improvements in the future [6]–[8]. Jalote et al. [9] conducted a study at Microsoft to investigate in which phase are most defects filed, who finds them, and what tools or methods revealed them. A study by Yu [10] at Lucent Technologies analysed the most frequent types of errors made in a project’s code to understand whether they could be prevented in the future.

In contrast to the above cited research that have looked at defects, our study comprises a broader set of issues that includes defects, but also feature requests, enhancements, and any other issues filed in the issue tracking system. Additionally, our focus is on issues that were specifically assigned to architects to understand the root causes for their inception. We conjecture that many of the issues may be related to the specifications as delivered by the architect. In a way, some of these specifications turn into issues and *boomerang* the architect to resolve them. To the best of our knowledge, no previous work has exclusively focused on the analysis of issues assigned to architects.

In our analysis, we have found that 30% of the architectural issues are debated between different roles in the development process, and the lead architect was involved in resolving those issues. Twenty eight percent of the architectural issues are design omissions that are due to different reasons such as unanticipated complexity, improved design or indifference in detailed design. Twenty seven percent are misunderstanding of design requirements due to various reasons. The rest of the architectural issues are due to misspecification or changes in requirements. This analysis leads us to conclude that some of the architectural issues can be avoided with improvements in the development process and communication. Of course, some of the issues cannot be anticipated but an awareness of these issues would help architects, developers and testers to deal with them more effectively.

The remainder of our paper is organized as follows: Section II elaborates on the case study environment by

providing information about the company, the nature of the projects under study, and typical team structure and software processes in place to develop software. In Sections IV and V we discuss the various classifications of the root causes of the different types of issues assigned to software architects. We discuss our findings in Section VI, threats to validity in Section VII and conclude our paper in Section VIII.

## II. STUDY ENVIRONMENT

### A. About the company

Océ ([www.oca.nl](http://www.oca.nl)), part of the Canon group, produces printers and copiers to serve the business markets for high-volume printing, wide-format printing, and office printing. With distribution spanning across 100 countries, Océ employs over 20,000 people worldwide. Approximately 1,550 of these people are employed in ten research and development sites spread across nine countries. Printer software is one of the many components in a printer. The software is responsible for accepting requests, controlling print jobs, rendering images, and controlling devices such as the print engine, scanner, finishers, and local and remote user interfaces. All software projects are managed in an agile fashion, but with each iteration planned and implemented using the V-model.

### B. Projects under study

Our study was conducted using two projects at Océ running in the years 2009 and 2010. The reason to choose these projects was that a lead architect (our key contact in the company for this research) was involved in both projects simultaneously and that some software units were common to them (a software unit is a set of specifications that are functionally or logically closely related). Both projects were at a relatively mature stage at the start of 2009. The high-level architecture was defined and verified at that time, so the data we collected concern the development stage. Both projects dealt with the wide-format printing and scanning and each of them resulted in an independent product. Printers and copiers of this category are used for printing on large format media, up to 36" and wider. These projects reused components and interfaces from previous products, as well as developed new components and interfaces.

### C. Project team structure

A typical team structure of project teams at Océ is depicted in Figure 1. A project manager heads the project and is responsible for its planning, realization, and successful completion. The project manager also agrees upon the high-level specifications of the project with upper management and marketing personnel. Requirements and specifications are compiled into product properties by the lead architect in the team. The specifications written by the lead architect start from a user-centric view, i.e., scenarios on how the end-users will interact with the product. For instance, the

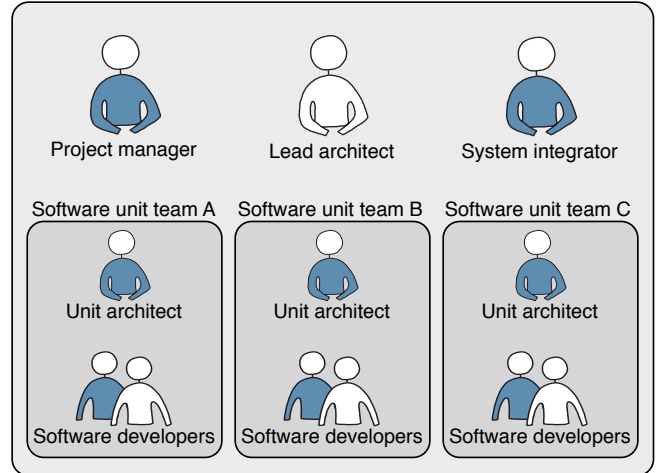


Figure 1. Typical project team structure at Océ.

architect is responsible to decide what happens in case of an interrupt request from the user of the printer as in how should it function, which software units should be triggered and how should they function, define the interfaces between these units and the like. In addition to the high-level design and specifications, this architect is also responsible for parameterizations (xml files with configuration parameters that align components to communicate with each other) and some user interface related aspects such as labels' text. Teams also comprise a system integrator who integrates the different software units to build the software system. Additionally, the integrator reports issues encountered during integration and assigns them to the appropriate team or person to be addressed. All three – the project manager, the lead architect, and the integrator – are assigned to a project for its entire duration until the product has been released.

Project teams also comprise one or more software unit teams that implement the software units. Each unit has a unit leader and a unit architect analogous to the project manager and lead architect at the project level. The unit leader is responsible for planning and organizing. The unit architects transform the high-level specifications received from the lead architect into detailed technical specifications and pass them to the software engineers who implement the code and test it. The unit architects are coordinated by the lead architect, who is often the only team member with the overall view of where (or in which units) do the different functionalities of the product reside. Most software units are not developed for a single product but their deliverable is tuned and integrated into several products. A software team can develop a software unit for four, or even more, projects at the same time. This challenges system behavior as well as architecture.

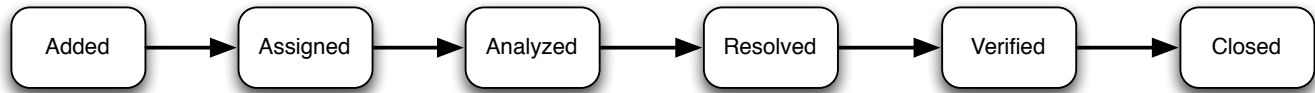


Figure 2. Ideal life cycle of an issue at Océ.

#### D. Issue report life cycle at Océ

At Océ, everyone involved in the project participates in testing. Everyone is allowed to file an issue in the tracking system (TestTrack<sup>1</sup>). For instance, besides software architects, developers and test engineers, people who can file issues include mechanical and chemical engineers, marketing people, technicians, and managers. About 90% of the issues is filed by the developers and test engineers.

The ideal life cycle of an issue is summarized in Figure 2. Upon being newly filed, an issue is given the status of *added*. Some senior team members (typically system integrators) in the project have the necessary permissions in the system to triage issues, i.e., determine in which unit the issue originates and assign it to the correct software unit to resolve the issue. The assignee team, who can also be the lead architect, then analyzes the issue and takes necessary steps to resolve it. Next, the resolution is verified by one or more senior team members and the issue is closed if the resolution seems satisfactory. Of course, several deviations may occur in this life cycle such as an issue may be repeatedly reassigned until it reaches the right team member to be fixed, there may be duplicates that must be identified, and verification of the resolution may conclude that the changes are incomplete. However, for the purpose of our study, the simple life cycle illustrated in Figure 2 suffices.

### III. METHODOLOGY OF STUDY

A lead architect and his projects are the focus of our case study. We consider a broad range of issues that have been assigned to the lead architect. These issues concern the entire product rather than specific software units.

Our methodology consists of four steps. Firstly, we imported all issues assigned to the lead architect on the selected projects into a relational database for categorization and analysis. Secondly, together with the lead architect, we examined each issue to understand it in detail (the issue descriptions can be vague to non-Océ employees), why the issue may have arisen, and whether it can be traced back to the product properties as outlined by the lead architect. Thirdly, we categorize the issue according to the reasons why it may have occurred. For instance, is an issue a result of misspecification. Fourthly, some architectural issues are complex involving a number of architects, testers and developers with many events occurring during the process. We traced these people and events to determine the root

causes of these issues. We refine our categorization system and the categorization of our issues accordingly.

In the following two sections, we describe the broad categories assigned to the issues that reflect on their types and the underlying reasons for their inception.

### IV. ARCHITECTURAL ISSUES

We begin with studying the issues that could be clearly traced back to the product properties that were outlined by the lead architect. As opposed to these types of issues, we also found several non-architectural issues (Section V) that were assigned to him because of this larger role in the project.

Architectural issues found were categorized into the following five categories:

- 1) **Debated:** These issues were filed when someone in the team has a different opinion on how a user interaction should be allowed on the product or how a feature should be implemented. These issues lead to a debate on the specifications, which may then either be revised or not.
- 2) **Omission:** Some issues filed were traced back to missing information from the specifications. Several causes as to why the information was initially missing were noted. In most cases, the lead architect filled in the gaps in the specifications to resolve the issues.
- 3) **Misunderstandings:** Misunderstandings occurred when someone testing a feature may have not comprehended the specifications correctly and thus have different expectations from a feature as opposed to what has been implemented.
- 4) **Wrong specification:** The specifications may have incorrectly outlined a feature. When this is realized during the development or testing phase, an issue is filed to revisit the specification and revise it as necessary.
- 5) **New insights:** These issues typically originate from the marketing and service teams who suggest new ideas to be implemented in the product that could potentially improve its standing in the market or ease maintenance.

In the following sections, we explain each category of issues in detail and provide examples to exemplify them.

#### A. Debated

The largest percentage of architectural issues (30%) assigned to the lead architect were categorized as *debated*. These

<sup>1</sup><http://www.seapine.com/ttpro.html>

	Architectural Issues (%)	Non-architectural Issues (%)	All Issues (%)
All issues	60	40	-
Debated	30	-	19
Omissions	28	-	17
Misinterpretation	27	-	16
Wrong specifications	11	-	7
New insights	4	-	3
Parameterization issues	-	40	16
Assignment errors	-	34	13
Farfetched feature requests	-	9	4
Misunderstood planning	-	7	3
Incorrect system usage	-	6	2
Incidental	-	4	1

Table I  
CATEGORIES OF ISSUES

issues stem from strong differences in opinion within the team (typically a developer or tester) on how features related to users' interaction and experience (both closely tied to the product's architecture) are deployed, while some others concerned the core architecture of the product. In rare cases, issues also concerned debating the decision about in which software unit a feature should be implemented.

Generally, resolving debated issues costs time because several people can get involved in the discussions and it could take a while to reach a consensus. 38% of the debated issues led to a change in the specification (Issue 1 and 2 below). The remaining issues were closed for reasons such as:

- The amendment is user-interaction related and the filer is not an expert on usability needs of customers,
- The amendment is system-behavior related and the observed behavior is designed to be consistent with other products,
- In principle, the amendment is agreed upon, but it cannot be realized due to architectural constraints. They can be fixed in a new product when the architecture is revised (Issue 3).

Below are three issues that exemplify the category of debated issues:

**Issue 1:** The scanner in the product holds on to the original after completing the scan rather than dropping it on the floor. In order to release the original, the specifications initially specified that a user must press the red button on the product. A team member filed an issue arguing that pressing the red colored button is counter-intuitive because the color is often associated with stopping or indicating that an error has occurred. As an alternative, the member recommended pressing the green button on the product to release the original. To this, another team member argued that the color green suggests the initiation of a job and is

hence also unintuitive because the release of the original marks the end of the scanning process and a debate ensued. Eventually, the issue was assigned to the lead architect and together the project team devised a solution and revised the specifications.

**Issue 2:** An example of an issue that dealt with the core architecture of the product relates to the interaction of its software architecture and power architecture. The product is switched on by plugging the power cable into the electric socket and pressing the on/off button on the printer. However, the display screen on the printer was powered by a USB interface, which directly draws power from the controller PC connected to the power cable. As a result, as soon as the power cable is plugged in, a splash screen appears on the display even though the printer has not yet been powered on using the on/off button. Also, the screen continued to display the splash screen even after the printer was turned off because it continued to get power from the controller PC. During the testing of the product, an issue was filed citing this behavior as potentially confusing to a user and eventually it got assigned to the lead software architect. After some debate, there emerged an agreement that the behavior could indeed confuse users and it is important to resolve the problem. Changes were thereafter incorporated in both the power and software architecture of the product to fix the issue in that the display was turned on only when the printer was actually turned on using on/off switch.

**Issue 3:** Another issue dealt with automatic scaling to a certain paper size. The user interface offers scaling to, e.g., A0, which in fact means scaling to a roll width from which an A0 in portrait can be cut. When an A1-sized original is scanned in landscape (as opposed to portrait), the copy will indeed be made on the A0 roll, but the resulting effective size will be A1. Since this behavior – resolving auto-scale, auto-rotate and auto-roll – is part of an existing component

whose architecture was not to be touched in this product, the behavior was not changed. The issue was however taken along for new products where this part of the architecture is revised.

At Océ, debating specifications is considered healthy because it demonstrates a feeling of ownership of and involvement in the project. It is a good way for overall product improvement. The process of handling debated specifications can be improved, though:

- Make project members aware of common behavior of product family members. This means that the system behavior documentation should clearly indicate what is generic, and what is product-specific.
- Make project members aware of which features are offered by 'old' components that cannot be touched.
- Make project members aware of generic as well as system-specific decisions taken by user interaction designers.

### B. Omission

Many issues assigned to the lead architect could be traced back to specifications with seemingly omitted information. These issues amounted to 28% of all architectural issues. Missing information creates unclarity and ambiguity in the team, eventually causing problems during development and testing and being filed as issues (typically) by the software engineers or system integrator. We observed several reasons as to why information in the specifications may have been missing or considered so and present them below:

- *Unanticipated complexities.* The largest proportion of issues with omissions were found to arise because situations emerged during the development or testing phase that were not anticipated by the lead architect at the time of writing the requirements. We noted that these issues often pertained to cases which may indeed be unforeseen and only observed during the implementation or testing phase. Issue 4 illustrates this category.
- *Retrospective improvements.* Another large subcategory included cases where information in the specifications on handling certain specific cases may have been revised or added in retrospect to allow better handling. Issues 5 and 6 illustrate this category.
- *Indifference.* These were issues that originated due to apparent lack of detail in the specifications, but were intentionally written by the architect as such to show that he is unconcerned about the implementation details of the specific feature as long as the feature 'just works'. Most printers have a 'media bypass' where users can manually insert a sheet to print upon instead of printing on the loaded media rolls. This feature is not used a lot yet it is crucial that it is developed. The lead architect's position in this matter is that the efforts expended on implementing the feature should be

minimal and that he is indifferent to how it is implemented as long as it works. Several issues were hence filed citing lack of adequate detail in the specifications regarding how certain aspects of the manual feed are to be implemented — in these cases, the lead architect either responded with the absolutely necessary details or simply reiterated his standpoint that the engineers are free to choose how to resolve the problem.

Other issues that were included in this category were those that were rooted in parts of the product where the architect had no control such as controlling the windows web server to prioritize handling of jobs differently to allow faster printing. Such issues, which formulated a very small percentage, were dismissed because no in-house solutions could be devised to address the problem.

- *Obvious.* In order to efficiently use his work time, the lead architect sometimes leaves out seemingly obvious details on handling specific cases from the specifications. But these missing details were noted to have come back to him as issues to resolve. Issue 7 is an example where an issue was filed due to seemingly obvious information omitted from the specifications.

**Issue 4:** An issue was filed related to being unable to restore the product's settings from a back-up by the end user after performing a software upgrade. A typical software upgrade involves the backing-up of the user's existing settings, performing of the upgrade of the software, and thereafter restoring the backed-up settings. In principle this is considered straightforward and trivial, and has previously been implemented in other projects successfully. But the project in question comprised a mix of new and old software components and interfaces that resulted in a relatively complex architecture. This intermix prevented some of the settings to be backed-up — a problem that was only observed when the upgrade feature was tested for the first time. Initially assigned to the software unit that implemented the backup and restore feature, the issue was soon realised to be a result of the overall system architecture and hence rightly escalated to the lead architect. The lead architect then consulted the concerned software units to arrive at a solution to address the problem. A key lesson drawn by the architect from this specific issue is that when new enhancements are made, an assessment of the behavior of old features and interaction with new ones must be done by the architect to prove the viability of a change. Future projects are likely to benefit from this episode.

**Issue 5:** Error messages were displayed to a user if the product malfunctioned as opposed to displaying an error code. The rationale behind this choice is that error messages are more user-friendly and informative so that the user can understand the nature of the problem. However, upon testing

the product, the service team explicitly asked for error codes to be reinstated in the error messages because they believed error codes reliably tell them what the problem is and are quicker to process. Whilst the users' interests are primary, the lead architect identified the service team as an important stakeholder and decided that both error codes and messages were to be displayed on the product, provided it does not degrade user experience. This issue was easily resolved by adding a new interface and modifying an existing one — these are changes that have to be initiated at the lead architect's level.

**Issue 6:** We noted another issue that was initially suspected to be a failure of the scan to USB feature. An issue was filed again stating that no file was copied to the USB stick despite there been no indications on the user interface to suggest there has been a problem. When the problem was eventually escalated to the lead architect, it was discovered that the particular test involved scanning the image in black & white to a JPEG file. This graphics format supports color and grayscale images, not black & white. Hence no image was produced from the scan and no file was written to the stick. So the specifications were revised to ensure that the user interface allow selection of only valid combinations of scan file type (PDF, JPEG, TIFF, etc.) and the color scale to prevent confusing the users and the issue was resolved.

**Issue 7:** Many printers from the company are sold along with a license file that enables certain features in the printer. In an incidental case, someone uploaded a license file on a printer, which was not meant for that model. The license file changed the behavior of the printer in unintentional ways and an issue was filed reporting the same. The lead architect believes that a check to ensure the license file belongs to the printer model is obvious and therefore the issue could immediately have been assigned to the software unit that handles licenses and should not have ended up on the lead architect's desk.

A majority of the omission-type issues are easily resolved by a clarification from the lead architect. Issues caused by unanticipated complexities and retrospective improvements are likely to remain. Issues that the architect is indifferent about or seem obvious may occur less frequently if the architects's view on such issues is made more explicit and clear to the team through the teams communication channels and the specifications.

When discussing the class of 'obvious' issues with the architect, we learnt that there maybe an implicit assumption on his part that people on the team are familiar with many details of the project from their past experience with related projects. For this reason, not all details are included in the specifications. However, this assumption may lead to problems when new employees are on board or some team members may be new to the family of products.

The lead architect may need to assess and/or check with the developers to ensure that s/he can accurately assume the level of knowledge of the developers. We observe that striking the right balance between preserving all relevant information in the specifications and omitting only the most obvious ones is challenging in large and complex projects.

### *C. Misunderstanding*

Some 27% of all architectural issues were attributed to misunderstandings on part of the software developer or tester. Misunderstandings may arise for several reasons including having interpreted the specifications differently from what the lead architect had in mind or not aware of all detailed specifications, especially project members who incidentally test the product. For example, a developer may misunderstand the requirement specifications and produce code accordingly. When this code is tested by the test engineer, a mismatch may be caught and filed as a defect. On the other hand, the test engineer may misinterpret the requirement specifications and on testing the code, may find a mismatch and file a defect. Interestingly, in some cases we learnt that the specifications may have been only skimmed over or even not read at all by the tester and the feature is tested using personal expectations or experience from previous products.

The following two issues illustrate this category of defects:

**Issue 8:** The product was noted to not permit scanning an image when the printer was out of toner. A message was displayed on the user interface stating that toner must be refilled. The user interface could not be used to do a scan-to-file. An issue was filed to question behavior because technically speaking, the process of scanning an image is independent of the printer toner and is hence possible. The lead architect then explained that this was done so as to preserve the integrity of the product as a single unit, keep the user interface simple, and to avoid further user interaction that may lead to problems. For instance, the user may wish to test the scanned image by printing a copy, which would not be possible when no toner is available. In this case, a simple clarification from the lead architect sufficed and behavior was accepted and the issue was closed realizing there was a misunderstanding.

**Issue 9:** In order to use a new product, a user has to first initialize and tune it with the help of an installation wizard. The first screen of the wizard lists all the languages in their native names and scripts so as to allow the user to choose the preferred language using a scroll bar to continue the installation process and use to operate the product. And unlike the following screens, no title or instructions to select the language were displayed on this screen. A team member filed an issue questioning why no instructions asking the user to select a language are displayed and the issue was assigned

to the lead architect. He clarified that this is not possible given the numerous languages available to set up printer, and that selecting one language to display the instruction would fail its purpose. This explanation was quickly agreed on and the issue was closed. Note that in this case, the same explanation was already present in the specifications which suggests that occasionally, specifications may be quickly skimmed over by the team and lead to misunderstandings.

We observed that misunderstanding related issues are often directly assigned to the lead architect. This is perhaps because issues are phrased such that they appear as if the specifications are incorrect for which the lead architect is responsible. They are typically resolved quickly with a clarification from the lead architect's end to bring everyone involved to a common understanding of the specification.

The issues belonging to this category of defects may be very finely distinguished from the debated issues. Misunderstanding defects are different in that they are often very quick to resolve (pending a clarification from the lead architect) and typically only the lead architect and the person who filed the defect are involved in the discussion as opposed to debated issues where many people share their opinions.

#### D. Wrong specification

These issues, amounting to 11% of the architectural issues assigned to the architect, are raised when there is admittedly an error in the specifications. For instance, consider the following two issues:

**Issue 10:** When making a copy, the printer scans the original to generate a bitmap image and prints the image on the media. After the job has finished, the printer displays a "Ready" message indicating that it is ready to accept new jobs. It was noted during testing that the message was displayed even while the printer was still busy making the copies. An issue to report this behavior was filed and assigned to the lead architect. Resolving this issue required changes in the architecture including revising the user interface to build up the system state by collecting the state information from other parts of the printer.

**Issue 11:** An issue was filed against a feature that allowed the user to print on any available media (different paper size) when the exact media size for the print job was unavailable. In case the available media size was larger, the file was printed in its original form. On the other hand, if the available media size was smaller, the file was scaled down and printed. In the latter case however, the printer persistently threw an error stating that the exact media size is unavailable and the print job cannot be completed. Later, it turned out that this option to scale down the image to fit a smaller media size was very difficult to realize and was eventually removed from the specifications. In this case, the

specification was changed to align it to what was actually implemented.

#### E. New insights

The smallest category of architectural issues (only 4%) assigned to the lead architect is *new insights* that are filed in retrospect with new ideas on how certain features should be added or improved in the user environment. The reason that the number of issues in this category is low, is that in principle issues can only be filed on agreed functionality. New insights, leading to new or updated features should then first get an updated spec, and then issues can be filed. Some new insights however are not truly new features but rather 'sharpening' existing features. These issues were all filed by the marketing or service departments who experience or learn new user scenarios that were not anticipated during the product's development. These new features needed to be realized quickly, and an issue was filed 'to update the spec' to reflect the new requests and assigned to the lead architect. The following issue illustrates this category:

**Issue 12:** The printer introduced a new feature Print-from-USB, by which the user could directly plug his USB-stick into the printer, select a file from the USB-stick, and press the print-button. The importance of this feature increased rapidly once it had been developed and demonstrated, and an immediate request was made to allow multiple files to be selected with a single print-action. It was filed as an issue, and assigned to the architect to update the specification for the software team developing the Print-from-USB feature.

## V. NON-ARCHITECTURAL ISSUES

Some 40% of the issues assigned to the lead architect were identified as non-architecture specific. These types of issues have been assigned to the architect given his additional responsibilities that may be specific to the job profile at Océ. Given that the focus of our research has been on analyzing architectural issues, we present non-architectural issues in the following sections in brief.

#### A. Parameterization issues

The lead architect is responsible for setting over 1,000 parameters (typically in an XML file) to configure how different components behave in a product. Parameterization allows software teams to build generic software components and all product specific features in the components are handled outside the team. The generated software components can thus be reused in multiple products without the team members needing knowledge on many product specific details. The component is configured to the product using the parameterization files; each parameter for a component accepts a different range of values for different products. Making the lead architect responsible for setting the parameters maybe specific to Océ. The issues related to the

parameters together comprise 40% of all non-architectural issues and consume a lot of his time.

Note that when parameterization issues are discovered, it is not always straightforward to ascertain whether the cause of the issue lies in the relevant components or in the parameterization files. When such ambiguity exists, the issue is more likely to be assigned to the lead architect, who then has to allocate time to analyze it and take next steps, i.e., resolve the issue himself or re-assign it to the right software unit. If the analysis reveals that the problem resides in the parameterization files, then the issue is usually quick to be resolved.

### *B. Issue assignment errors*

These issues were noted to have been incorrectly assigned to the lead architect to be resolved. Incorrect assignment can occur due to a clerical error, oversight, or ambiguity over who should resolve the issue. In the case of the lead architect, 35% of non-architectural issues were incorrectly assigned to him — in all cases, the architect spent time to analyze the issues and re-assign them to the correct software unit responsible to address them. Incorrect assignments, in many cases, suggested that the lead architect was also seen as the ‘lead analyzer’ who must determine the cause of the issues and assign them to the correct software team.

We discussed possible solutions with the lead architect to reduce the occurrence of such assignment errors in future projects — one solution was to improve communication within the team to bring about better understanding and alignment of the project’s goals. Another aspect to improve is to increase awareness within the team about the roles and responsibilities of each other, perhaps using a tool similar to Codebook [11].

### *C. Farfetched feature requests*

A small category of issues (9% of non-architectural issues) included feature requests made by team members that were considered very valuable but beyond the scope of the project. Again, like debated issues, such requests suggest a sense of ownership of the project within team members and their determination to deliver a high quality product. It is not unlikely that some such feature requests may become key selling points in future products of the company. However, for the purpose of the on-going projects, they are explicitly tagged as feature requests to be considered in the planning future iterations of the product.

### *D. Mis-understood planning*

A small proportion of issues filed pertained to features that were not planned for the current version of the product, but for a future version. These issues were either rejected or deferred to be addressed in the next development iteration of the product. They accounted for 9% of all non-architectural issues.

### *E. Incorrect system usage*

Some issues (6% of non-architectural issues) are filed because the product was tested or used in an unintended manner and resulted in an unexpected behavior. A vast majority of these issues were not even software related. For instance, in a specific case, explicit instructions on the local user interface on how to install a new print head in an inkjet printer were ignored and consequently, a printer cover was broken. An issue was filed and assigned to the lead architect, who simply rejected it citing the instructions displayed on the interface. It is surprising that such issues are assigned to the lead architect at all. Their frequency can be reduced by a more thorough analysis before assignment.

### *F. Incidental*

A small number of issues (4% of non-architectural issues) could not be reproduced when being analysed and were hence regarded incidental. No further action could be undertaken to resolve such issues; they were simply closed.

## VI. DISCUSSION

Our study at Océ has led us to uncover several categories of issues that are assigned to the lead architect, many of which could be traced back to the specifications and hence, they boomeranged the architect. Ideally, such issues should be few in number so that the architect can spend more of his time on the core aspects of the product’s architecture. However, it is unreasonable to expect that these issues can be entirely eliminated.

Of all the issues assigned to the lead architect, and supposedly architectural in nature, we have found that 40% of all issues are not architecture related. Some of these issues, such as parameterization issues, are assigned because of the larger role that lead architects play at Océ. Other issues seem to arise due to a lack of knowledge about the issue or uncertainty as to who is the right person to resolve them. Our discussions with the architect suggest that many issues could be resolved earlier, without them being assigned to the architect, if communication of product knowledge is improved between the architect and other team members. Examples of these issues include those filed due to misunderstanding the product’s plans, incidental issues, incorrect usage of the product, and assignment errors.

The remaining 60% issues that we classify as genuinely architecture related were classified into five categories: (a) debated; (b) omission; (c) misunderstanding; (d) wrong specification; and (e) new insights. Some of these issues are unavoidable in that it would be very difficult to avoid them. For instance, some of the issues (such as issue 4) caused by unanticipated complexities may not be avoided easily, causing omissions in the specification. Improving features in retrospect are also difficult to systematically eliminate because some architectural design issues cannot

be anticipated until everyone concerned looks at the design in detail, or perhaps not until they implement the design.

Then there are a set of issues that are implicitly valuable to the project and the team environment even though they are time consuming to resolve. Issues 1, 2, and 3 classified as debated are good examples where we see team members sharing their opinion on how a feature should be implemented in the product. Such collective spirit of ownership of the project in the team is positive and inspiring. In fact, some of the debated issues also led to changes in the original specifications because the suggestions made the product better. Issues that indicated mistakes in the specifications (e.g., issues 10 and 11) are also desirable so as to bring about necessary fixes in the specifications and lead to the developing the product as intended. Retrospective improvements made to the product, such as Issues 5 and 6, are also welcomed because of their positive effect on the product. Other issues including those categorizes as new insights (Issue 12) and feature requests are valuable to the product in the long-term since some of them may be incorporated as features into, occasionally the current, or the next iteration of the product.

A number of other architectural issues can perhaps be avoided with process improvements. For instance, omissions that account for 28% of all architectural issues may be reduced in number by better communication within the team. For instance, issues related to indispensable yet less important features can be analyzed in accordance with the nature and severity of the issue. Also, specifications can be written keeping in mind the team composition (senior and junior members) so as to strike the right balance of detail in them with respect to common knowledge in the team. If the architect anticipates that the intended readers of the specifications may not have the right background to understand the specification sufficiently, more detail may need to be written. Such a practice can avoid issues similar to Issue 7.

Up to 27% of the issues were caused by misunderstanding on the part of the developers or testers. It indicates that the communication of knowledge between the architects and the team can be further improved. Issue 8 and 9 suggest these situations are caused by team members who are in a learning curve or are technical specialist in a certain aspect of the product. This can be improved by more intensified communication and easy access to specifications.

Some 11% of all issues arise from misspecification (Issues 10 and 11). Some of these issues may be avoided if architects communicate and investigate more thoroughly. However, there are circumstances in development that are less than ideal that constrain the architects. An example is tight schedule to produce a specification in time and start development. As a result, architects may misspecify.

Overall, many of the avoidable issues can be reduced in number by improving communication and the software

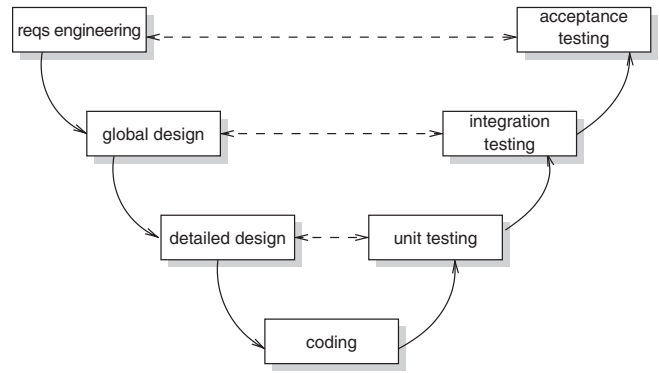


Figure 3. V-model of software development

development process. Two improvements which we believe are likely to yield good results are:

- Improve communication within the team to make everyone aware of the goals of the project and key specifications so as to reduce certain types of defects such as omissions and misunderstandings. As noted before, and common for product line development as done by Océ, products generally use components (with certain architectural and behavioral constraints) realized in other products. The lead architect may then be inclined to assume other project members have the same knowledge he has, and only specify the  $\delta$  with respect to the previous product. More generally, knowing what other team members know deserves attention.
- Encourage communication in the team horizontally across the v-model too (see dashed arrows in Fig. 3). In the present situation, information and communication flows (by and large) from the lead architect to the unit architect and then to developers (so down the left leg of the v-model), and from there via the unit testers to the integration testers (so up the right arm of the v-model). This is very much the same direction of the work flow in the v-model too. Improving communication horizontally in the v-model between the architect and the test team will help bring about better alignment of the project's goals between the two. A shared understanding of the intention of the specifications and the underlying architecture may lead to reduction in several types of issues such as those that are invalid, beyond the project's scope, or debated, can perhaps be resolved without the involvement of the architect. A promising tactic to realize horizontal communication is to involve the system integrator into the project earlier in the project rather than waiting until there are units to be integrated. In the current situation, the integrator is expected to quickly familiarize himself with the architecture, integrate the units, and file any issues that are found. But to be effective at doing the latter two tasks, good familiarity with the architecture

is crucial and any compromise in this direction may impact on the quality of the other tasks. We expect that early involvement of the integrator will lead to a fall in several categories of issues including misunderstandings, incorrect assignments, misunderstood planning, and wrong usage of the product.

## VII. THREATS TO VALIDITY

Like any other empirical study of this nature, ours too has some threats to validity. Both subject projects started using the issue tracking system extensively when the high level architecture and specifications were clear. Both projects also used components in their architecture that were in one of three stages of their architectural lifetime: mature units, units that were further developed in the project, and completely new units. This may have an impact on the types and number of architectural issues filed. Besides, we only considered issues assigned to the lead architect in this study and not unit architects. Thus, architectural issues that were directly assigned to unit architects have not been investigated. Having said that, we may have observed more high level architectural issues if more units were developed new.

Some subjectivity may have been involved in categorizing the issues assigned to the lead architect. However, utmost care has been taken to classify the issues by leveraging the information provided in their description and comment fields. Also, considering that the study was conducted in a single company on two projects, caution must be exerted when generalizing the results by adapting the results to the context.

## VIII. CONCLUSIONS AND CONSEQUENCES

In this study at Océ we identified several categories of issues that are assigned to the lead architect, many of which could be traced back to the specifications and hence, they boomeranged the architect. We found that 40% of the issues assigned to the lead architect were not architecture related, while 60% was. The latter were classified into five categories: (a) debated; (b) omission; (c) misunderstanding; (d) wrong specification; and (e) new insights. A number of those issues are valuable to the project and environment even though their resolution takes a lot of time. Issues 1 – 3 cited above are good examples hereof. They are proof of a collective spirit of ownership in the team, which is positive and inspiring. A number of issues can perhaps be prevented to occur in the future with process improvements. Two improvements we identified are:

- Improve communication between the architect and other team to make everyone aware of the goals and key specifications. Lead architects take their knowledge from earlier projects they have been involved in, and

may be inclined to assume other project members have the same prior knowledge. It is important to know what other team members know.

- Encourage horizontal communication in the team, as depicted by the dashed arrows in Fig. 3. Currently, the information flow by and large follows the workflow, as depicted by the solid arrows in Fig. 3. In particular, it is expected that early involvement of the system integrator will have a positive impact.

*Acknowledgements.* 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.

## REFERENCES

- [1] P. Kruchten, "Controversy corner: What do software architects really do?" *Journal of Systems and Software*, vol. 81, pp. 2413–2416, December 2008.
- [2] H. Erdogmus, "Architecture meets agility," *IEEE Software*, vol. 26, pp. 2–4, Sept/Oct 2009.
- [3] J. Madison, "Agile architecture interactions," *IEEE Software*, vol. 27, pp. 41–48, March 2010.
- [4] D. N. Card, "Myths and strategies of defect causal analysis," in *Proceedings of the Pacific Northwest Software Quality Conference*, October 2006.
- [5] —, "Learning from our mistakes with defect causal analysis," *IEEE Software*, Jan-Feb 1998.
- [6] M. Kalinowski, G. Travassos, and D. Card, "Towards a defect prevention based process improvement approach," in *SEAA '08: Procs. of the Euromicro Conference Software Engineering and Advanced Applications*, 2008, pp. 199–206.
- [7] M. Leszak, D. E. Perry, and D. Stoll, "Classification and evaluation of defects in a project retrospective," *Journal of Systems and Software*, vol. 61, no. 3, pp. 173–187, 2002.
- [8] —, "A case study in root cause defect analysis," in *ICSE '00: Proceedings of the International Conference on Software Engineering*. New York: ACM, 2000, pp. 428–437.
- [9] P. Jalote, R. Munshi, and T. Probsting, "The when-who-how analysis of defects for improving the quality control process," *Journal of Systems and Software*, vol. 80, no. 4, pp. 584 – 589, 2007.
- [10] W. D. Yu, "A software fault prevention approach in coding and root cause analysis," *Bell Labs Technical Journal*, April-June 1998.
- [11] A. Begel, K. Y. Phang, and T. Zimmermann, "Codebook: Discovering and exploiting relationships in software repositories," in *ICSE 2010: Procs. of the International Conference on Software Engineering*, May 2010.