Governance of innovation project management : necessary and neglected

Stettina, C.J.

9

Routines and Boundary Objects in Achieving a Sustainable Practice

In this chapter we use two experiments to highlight the importance of routines their interplay with artifacts and their impact on the sustainability of a practice. Moreover, the importance of routines is presented by two experiments.

We address the research question RQ2 ("*What are the components of governance necessary to understand knowledge worker project organizations?*"). In order to make RQ2 suitable for our field research, we formulated the following subquestions RQ2f: *"How do externalization formalisms influence documentation practices in agile teams?"* and RQ2g: *"What are the implications of iteratively updated internal documentation on the quality of artifacts, perceived amount of work and project satisfaction?"*

This chapter is based on the following publication[1]:

Stettina, C. J., Heijstek, W., & Fægri, T. E. (2012). **Documentation work in agile teams: the role of documentation formalism in achieving a sustainable practice.** In Agile Conference (AGILE), 2012 (pp. 31-40). IEEE. [1]

[1]The author would like to thank IEEE and his co-author for permission to reuse relevant parts of the article in this thesis.

9.1 Documentation: Required or Requested?

When compared to document driven traditional software development, agile software development practices advocate an emphasis of direct communication (Abrahamsson et al., 2003; Dybå & Dingsøyr, 2008a; Clear, 2003). This focus is embedded in reoccurring practices such as iterative development, frequent customer involvement, daily stand-up meetings or team-based effort estimation (Fægri, 2010). During the course of a development project such routines make work more predictable, support the transfer of knowledge and ensure communication within the team. While knowledge sharing practices based on socialization of team members enable agility and common understanding at the team level (Melnik & Maurer, 2004) they can also result in problems such as gaps of undocumented knowledge and lacking architectural integrity between projects. This is particularly problematic if people leave the team or organization (Abrahamsson et al., 2003; Ramesh et al., 2007; Stettina & Heijstek, 2011b). In this sense, agile development may make project management and program level management more difficult.

Efficient teamwork in agile development relies fundamentally on shared mental models (Moe et al., 2010). Artifacts in agile development lose much of their effectiveness if participants lack a shared, overall understanding of project objectives. Sharp et al. (2009) point out that the artifacts largely lack detailed information about the application under development. Non-functional requirements, for example, are difficult to link to user stories and tend to be ignored or ill-defined (Ramesh et al., 2007). Code comments are generally accessible to technical staff only. Contrarily, in highly regulated and contract-driven environments formal documentation is a must. In European research projects, for example, project partners are encouraged to use documentation deliverables to disseminate their knowledge and to stimulate collaboration among the participating projects.

Software practitioners tend to perceive writing documentation as a burdensome side-task and literature suggests that this results in deliverables usually compiled and submitted at the end of a project (Clear, 2003). Software projects are executed under strict constraints of time and budget. Team members are often needed for new projects and committed to these projects early on. There is rarely enough time to document all experiences and knowledge after the project delivery. Small teams might perceive codification of their knowledge less important, according to earlier findings (Stettina & Heijstek, 2011b) and according to our own experience this can lead to gaps in knowledge. As knowledge is not transferred seamlessly these gaps can hinder the agility of an organization as a whole.

To improve our understanding of knowledge codification strategies in software development we designed and executed a quasi-experiment in which students were required to iteratively develop a small-to-medium size application in teams. The 28 students were divided into 8 teams and two groups: SAD and UML. Group SAD (Teams A-D) was to update and deliver their high-level software architecture in form of a textual description defined by RUP

templates. Group UML (Teams E-H) was instructed to update and deliver their low-level software design in form of UML models. In this chapter, we report on this study into the team's practices to keep documentation up-to-date.

9.2 Related Work

In their seminal theory of knowledge creation, Nonaka and Takeuchi (1995) explain knowledge creation as resting in conversions between explicit and tacit forms - and between the ontological levels of individuals and groups. Nonaka and Takeuchi identify four modes of conversion in which new knowledge emerges; socialization, externalization, combination and internalization. A key point in their argument is that innovation (a result of knowledge creation) must be enabled by encouraging and facilitating the occurrence of all conversion modes in daily practices within the organization. Because software development is fundamentally driven by the creation of new knowledge we believe that Nonaka and Takeuchi's model is a relevant conceptual model to gain understanding of knowledge processes in agile methods.

Agile methods derive much of their agility by giving preference to social interaction among people rather than documentation. The socialization mode of Nonaka and Takeuchi is therefore more prominent. Agile methods encourage knowledge creation through practices of frequent interaction among the the team members (Melnik & Maurer, 2004). Consequently, however, agile methods may demand a more 'social attitude' among team members compared to plan-driven methods (Nerur, Mahapatra & Mangalaraj, 2005). Furthermore, socialization incurs inherent scalability limitations. Socialization works well for small teams or teams that can be synchronized via frequent meetings (Laanti, 2008) but for teams whose members are spatially separated or knowledge must be passed on beyond the team members, the organization is often forced to rely on explicit forms of knowledge (see Figure 9.1). In this chapter we refer to the knowledge recipient at the program level - in contrast to the project level, which refers to the project team.

Few empirical studies have examined practices of inter-project knowledge sharing and externalization in agile development, and most work is focused on requirements engineering (Ramesh et al., 2007) and software process improvement (Dingsøyr & Hanssen, 2002). Laanti (2008) proposes that knowledge sharing in Scrum can be up-scaled to the program level by means of two nested control loops consisting of a team level backlog and a program level backlog. This way, next to the common Scrum daily and sprint meetings, the program level would include a *program Scrum* once or twice a week to synchronize the program level with the team level. Postmortem reviews have been successfully applied in agile projects to externalize and transfer project experiences enabling software process improvement (Dingsøyr & Hanssen, 2002). Sharp et al. (2009) point out that agile artifacts such as user stories and project wall work well supporting the project process but largely lack detailed information about the application under development.

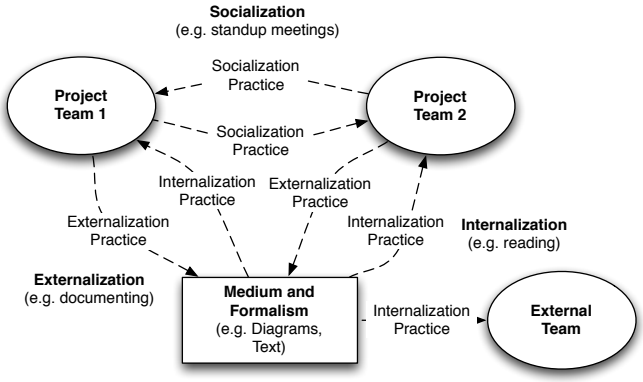

Figure 9.1: Problem domain: Model of knowledge transfer through agile practices

Current studies on the use of documentation during software development are few and present mixed results, studies on documentation in agile settings are even fewer (Dybå & Dingsøyr, 2008a). Stettina and Heijstek (2011b) found that agile practitioners in their data set perceive documentation important or even very important but that too little documentation is available in their projects, and indicate that knowledge is lost for older projects. Clear (2003) points at the behavior of students and observes documentation being seen as a burdening by-product and being hurriedly pieced together at project end. Past studies suggest that documentation is frequently out of date and should be rather seen as a communication medium than an accurate or up-to-date artifact (Forward & Lethbridge, 2002).

Based in the present state of the art we find it appropriate to explore in more depth how documentation work is carried out in agile teams, and in particular the relationship between documentation formalism and developer practice. Our guiding hypothesis is that documentation formalism influences agile practices (compare Figure 9.1). We thus pose the following two sub research questions:

1. *RQ2f: How do externalization formalisms influence documentation practices in agile teams?*

2. *RQ2g: What are the implications of iteratively updated internal documentation on the quality of artifacts, perceived amount of work and project satisfaction?*

9.3 Method

We applied a mixed methods approach to collect data (Miles & Huberman, 1994). Qualitative process descriptions were compared and contrasted with quantitative questionnaires and

the artifacts developed in course of the project. Ecouraged by agile methods' emphasis on self-managing teams and the integration of individuals into the software development process (Stettina & Heijstek, 2011a), we sampled individual team members' perceptions using anonymized questionaires. To create a deeper picture supporting our research we further used informal interviews and conducted a document inspection on the created artifacts.

Collecting practices of writing documentation as human behavior requires both qualitative data and observations in a real world context. Following Langley's (Langley, 1999) framework for building theory from process data we have selected visual mapping which requires at least five cases in moderate level of detail to begin a pattern identification. Using graphical forms allows the presentation of large amount of information in little space and is a useful tool to develop and verify ideas in theory development (Langley, 1999).

9.4 Research Design

To address the research questions we embedded the study within a software engineering project course at Leiden University with a class of 28 students divided into 8 groups. This provides a suitable environment allowing a comparison of several teams working on the same project.

The project was designed as a fourteen week quasi-experiment including seven development iterations with the aim to develop a program being able to recognize and extract UML class diagrams stored in bitmap images. Guided by Richards (2009) we devised project-based courses that encouraged students to engage in practical work, form their own groups and stimulated students' individual reflections on ongoing accomplishments. While we had given recommendations, the students were given free choice of development platform and libraries. The coaching took place after the weekly software engineering lectures in nine sessions. In each session we discussed the progress of the teams, the challenges and the next steps. For instruction we used handouts in class explaining the course of the project and actions to be taken.

Students were instructed to document their development efforts focusing on architectural aspects (Lethbridge, 2000). As we wanted to study the implications of documentation formalisms we have chosen two of the most common artifacts to be updated: (1) high-level software architecture in form of a textual description and (2) low-level software design in form of UML models. We explicitly did not incorporate requirement descriptions in our study as those are already addressed by agile artifacts (Sharp et al., 2009).

Due to the lectures being originally aligned to the Rational Unified Process, we decided to use RUP templates for the textual deliverables: Software Development Plan, Software Requirements Specification, Configuration Management Plan and the Software Architecture Document. One reason to choose RUP templates was that in many established software companies such are seen as standard and are expected to be in use even while applying agile methods. We emphasized throughout the project that the formal templates should be seen

Table 9.1: Project course

Project Planning and Initial Design
08-09-2011: (Session 1) Introduction
15-09-2011: (Session 2) Requirements Elicitation Session
22-09-2011: (Session 3) Requirements Elicitation Session 2
29-09-2011: (Session 4) Requirements Review Presentation
06-10-2011: (Session 5) Architecture and Design Presentation
Development
13-10-2011: Sprint 1 - User Interface
20-10-2011: Sprint 2 - Integrate shape recognition and OCR libraries
27-10-2011: Sprint 3 - Recognize outer UML shapes (Session 6)
03-11-2011: Sprint 4 - Integrate shape and character recognition
10-11-2011: Sprint 5 - Recognize relationships
17-11-2011: Sprint 6 - Optimization
24-11-2011: Sprint 7 - Additional Sprint (Session 7)
Delivery and Postmortem
01-12-2011: (Session 8) System Demonstration and Handover
07-12-2011: (Session 9) Project Closing: Discussion & Postmortem Review

as guidelines rather than as stiff documents to be filled. The UML models to be delivered consisted of a class diagram, a sequence diagram, an activity diagram and a state chart diagram.

After the definition of an initial architecture in session five the student teams were divided into two groups: Teams A-D were required to update and deliver the low-level software design models with every sprint. Teams E-H were required to update and deliver the software architecture document with every sprint. From now on we will refer to those teams as the UML and the SAD groups. The students were graded in the following schema: 25% Requirements Engineering, 25% Software Design, 25% Implementation and 25% overall quality. The updated artifacts have been made an explicit "customer requirement" the students would be graded for.

In order to test how the students perceive the importance of documentation artifacts after project delivery we chose a setup similar to the one used by Smith, Mann and Buissink-Smith (2001) where at the end of the development phase the student groups would swap their project source code repositories and documentation to simulate a maintenance environment. The project is hosted at Google Code and is freely accessible[2].

9.4.1 Qualitative Data Sources

To address the research question we first have to clarify our research lens on human action. As patterns of action we understand those as defined by Cohen et al. (1996) as (i) Recurring (they can take place at different times, or involve different actors), (ii) selectable (meaning that there are forces that make them more or less likely to happen) and (iii) set in an organizational

[2] `http://code.google.com/p/image-to-uml/`

context (the actions are not those of an isolated individual).

In order to collect patterns of action after system delivery we asked the student groups to discuss and draw the process of documenting their software as UML activity diagrams on a paper and to present them one after another at the board during the closing meeting (Session 9). We chose to use activity diagrams as we assumed those to be most accessible to software engineers. At the end of the session we took pictures of the models on the board and collected those drawn on paper earlier. We also kept an audio recording of the session for reference. We conducted direct observations only during the sessions in class. However, as the artifacts were mainly produced outside the classroom we conducted informal interviews and asked the students for a better level of detail or for clarification regarding certain choices or team member roles (*"How did your team assign the work related to documentation?"*, *"Why did you choose to work on documentation at that particular stage?"*). During the final session we also conducted a postmortem review (Dingsøyr & Hanssen, 2002) of the project to identify good and bad experiences of the development teams. This final session lasted for about three hours. At the end of the project the created documents were collected from the repository and assessed according to their size in word count and quality. We analyzed the content of the deliverables for each group according to the structure of the templates and readability.

9.4.2 Quantitative Data Sources

We had access to the following quantitative data sources: (1) the longitudinal survey answers collected each week in class, (2) longer questionnaires conducted in each development phase and the (3) online repository. First, a short questionnaire for a longitudinal collection of the individual team member's satisfaction with the project, the amount of work and the documentation was used. This part of the questionnaire was collected every week during the project. The objective was to measure how team member perception on work environment and documentation changes during course of the challenging project, similarly to the project satisfaction graph as earlier presented by Moe et al. (2010). The questions were: *"How satisfied are you with the project?"*, *"How satisfied are you with the amount of work?"*, *"How satisfied are you with the teamwork in your team?"* and *"How do you feel about documentation in your project?"*. Second, longer questionnaires were applied in the course of the development phase, at project delivery and after project handover to maintenance. The longer questionnaires were used to collect perceptions on the usefulness of specific artifacts, the distribution of roles within the groups and gave students the opportunity to anonymously comment on their projects. Sample questions were: *"What was your predominant role in the project?"*, *"How useful did you find documentation for your project?"*, *"How useful did you find updating and delivering documentation with every milestone?"*, *"What influence did the requirement of updating documentation have on your project satisfaction?"* and *"If you would know that after development the project will be maintained by another team, which documentation artifacts*

would you choose to keep up-to-date?".

9.5 Results

In this section we will proceed to address the two research questions. The project started on September 8, 2011 with an introductory session and proceeded as outlined in Table 9.1.

9.5.1 Patterns of Team Practice

Figure 9.3 visualizes each team's practice to create and update their internal documentation as emerged from observations, the final session in class, the informal interviews and the online repository. Actions marked dark gray were assigned to a single member of the team throughout the whole project, while actions marked light gray were rotated.

The project was somewhat challenging in terms of programming skills for the second year students as some of them expressed (e.g. *"Teamwork would be so much better if everyone was a skillful programmer..."*) and the majority of teams established specialization among their team members. While the initial version of the artifacts has been developed commonly, five of the eight teams (teams A,D,F,G, and H) defined a single team member to do the updates. because *"it was most convenient"* i.e. because that particular member was least good in coding or his own preference to work on documentation. As we can see two teams established a review process. In three of the teams the appointed member used SVN logs to update documentation.An exception here were team B and E as, we can recognize in Figure 9.3. In team B every developer was in charge of bringing the documentation up-to-date with his changes in code. The team had a skillful programmer leading the project and employed pair-programming. Team E kept track of hours spent on the project (keeping track of hours was a recommendation given to the students during the sessions), the person who had the least effort spent before end of the iteration had to update the documents. For team C updating documentation was not really an issue. After creating the initial documents the team was struggling to get the implementation to work, thus changes were not necessary. As emerged from the survey, Figure 9.2 represents the team member roles of the participants as specified. Multiple answers were possible and out of the three categories, "Code", "Documentation" and "Administration" the majority of the participants (63%) specified their predominant role as "Code" only, 11% as Documentation and 0% specified their role as purely administrative. When clustering the categories in Figure 9.2 we can see that 78% of the participants were involved in coding (coding, coding & documentation and administration & coding) and 30% were involved in documentation. Interestingly, as we can see in Figure 9.2 is that only 8% of the participants defined to be involved in coding and documentation.

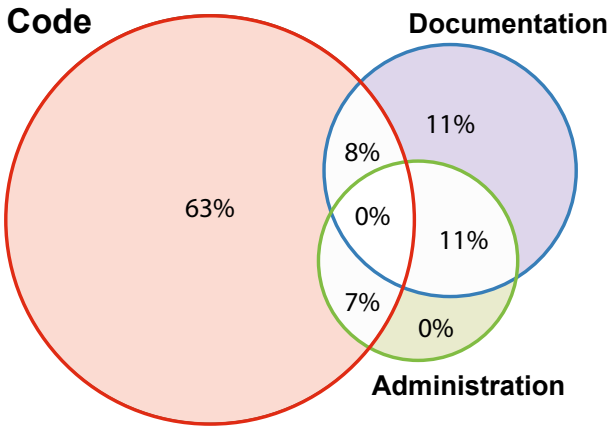

Figure 9.2: Distribution of team roles. Total number of participants involved in coding: 78%, in documenting: 30% and in administration: 18%

Table 9.2: Summary of team practices to write and maintain documentation

Team A	All members worked on initial versions, thereafter one team member was in charge of maintaining documents, "according to repository and team member comments"
Team B	Every developer was in charge of updating documentation with changes in code. Difficult parts of the program were developed in pair programming.
Team C	Updating models was not really an issue. Most of the architecture was done in UML models, and as the team was struggling to implement the initial design there was not really much to change.
Team D	All team members but one wrote the initial documents, the remaining team member reviewed them now and then.
Team E	Kept track of hours, the person "who had time" updated the artifacts. After asking the team how many were involved, a team member answered "basically all".
Team F	Initial documents were created by all team members, updates were done by team member TM1.
Team G	While one team member was in charge of updating the textual SAD documentation two others were in charge of UML models
Team H	Initial by all, updates were committed by a single team member because he was the "least good in coding"

9.5.2 Project and Teamwork Satisfaction

The students generally welcomed the project while finding it challenging with comments varying from *"It will be tough getting everything to work on the deadline"* to *"Very nice*

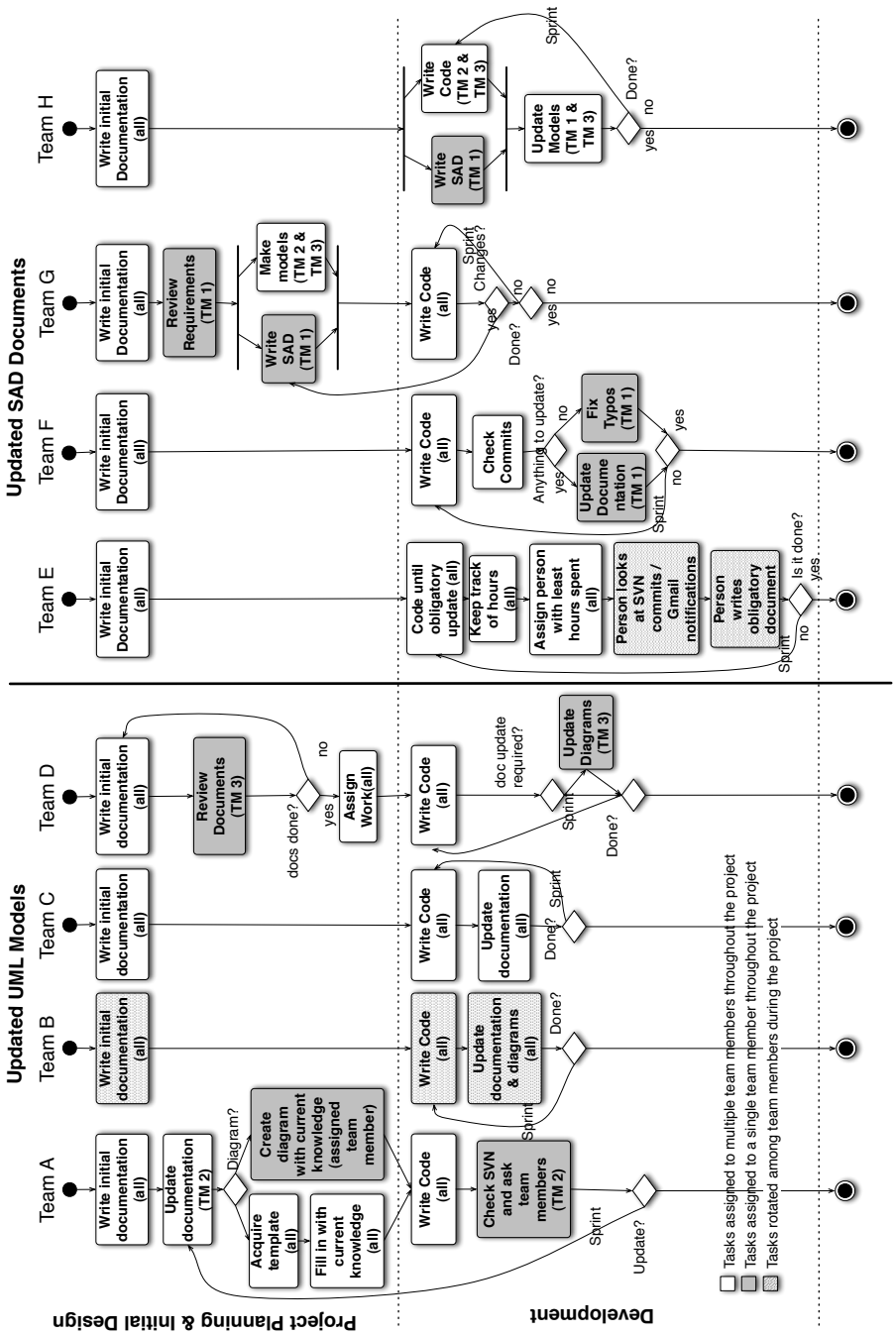

Figure 9.3: Team practices to update documentation

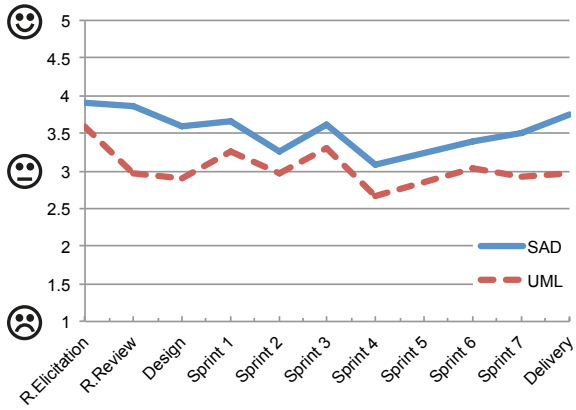

Figure 9.4: Project satisfaction. [Scale: 1=Not at all satisfied, 2=Slightly satisfied, 3=Moderately satisfied, 4=Very satisfied, 5=Extremely satisfied]

project. Challenging, yet very interesting.". During the requirements review session there was quite some uncertainty due to the free choice of implementation platform. During the implementation phase the project remained challenging, and at the end of the first iteration a student commented: *"Using the internet, we have made good progress. However, because the code is from mixed sources and adapted to our situation, the code is a mess. The documentation isn't up to date either, though I do think we are all on the same level."* During the feedback session one student remarks that after integrating the OCR and shape recognition libraries many discovered how much work it really was.

Figures 9.4 and 9.5 illustrate the satisfaction with the project and the amount of work for both customer groups during the course of the project. In Figure 9.4 we can observe that the curves for satisfaction with the project run parallel, with the UML groups being generally less satisfied with. Figure 9.5 illustrates the satisfaction with the amount of work for the SAD and UML groups. The teams perceived updating and delivering documentation as not very useful. When asked on how much influence documentation work would have on their project satisfaction, they said it would have only slight influence (compare Fig. 9.7).

Teams perceived documentation as a burden, as work 'that needs to be done'. Especially for the SAD group we can observe an increase in perceived amount of documentation in the period between Sprint 3 and 7 of the development phase. This increase of perceived documentation amount as visualized in Figure 9.6 happens at the same time as the team members perceive an increased amount of work (compare Figure 9.5). Interesting is to note that when asked which documents the teams would keep up to date if the project is to be handed over, majority of the participants chose software architecture document, software design models and software requirements specification out of the five deliverables.

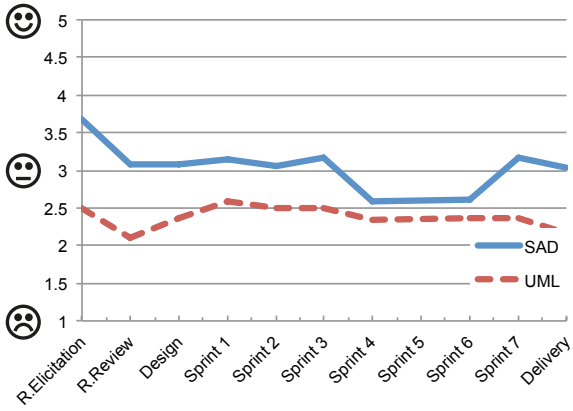

Figure 9.5: Satisfaction with the amount of work. [Scale: 1=Not at all satisfied, 2=Slightly satisfied, 3=Moderately satisfied, 4=Very satisfied, 5=Extremely satisfied]

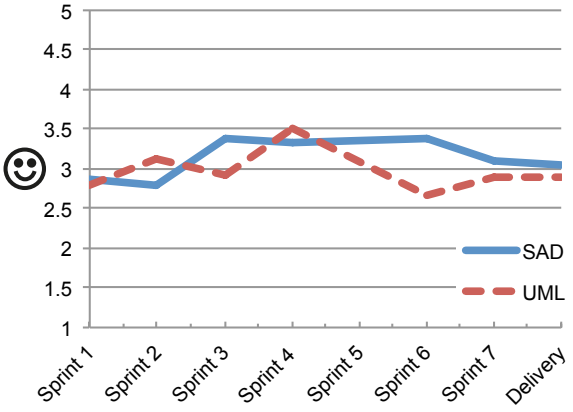

Figure 9.6: Perceptions on internal documentation in course of the implementation phase. [Scale: 1=Way too little, 2=Slightly too little, 3=Just about right, 4=Slightly too much, 5=Way too much]

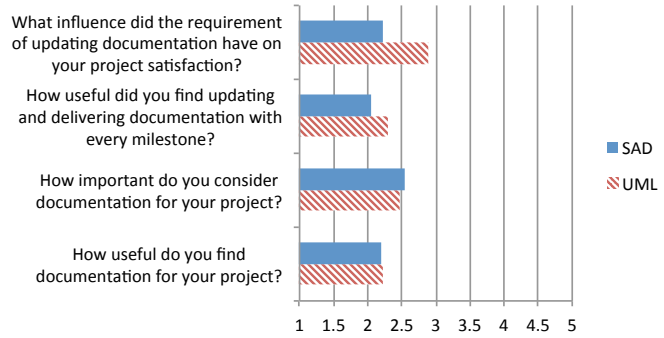

Figure 9.7: Perceptions after project delivery. [Scale: 1=Not at all, 2=Slightly, 3=Moderately, 4=Very, 5=Extremely]

9.5.3 Quality of Documentation

After completion of the project on December 7 we downloaded the latest versions of the deliverables and conducted a document analysis. Table 9.3 illustrates the results of our analysis. As we can see in the table the teams instructed to update their architecture document with every milestone generally produced larger and more elaborate documents. The quality of the artifacts delivered by the SAD groups was significantly better. Teams that were instructed to update their UML models did so, however as one could expect, the changes within the models were less visible. As they did not had to update other artifacts than the models, the quality of the documents delivered was lower significantly, for example the documents contained template explanations still. Table 9.3 also contains the assessment of the implementation, which indicates that there is no significant correlation of the updated documentation on the quality of the implementation. The software development plan was largely omitted by the teams which made it the documentation artifacts with the lowest quality delivered.

9.6 Discussion

In this section, we discuss the emergent patterns of action, the team members' expressed satisfaction and the quality of documentation deliverables. We will first consider the role of documentation as 'product', e.g. as tangible project deliverables. Second, we will consider the role of documentation as a medium of knowledge exchange and how documentation supports sharing and leveraging project knowledge at the organization's program level.

Table 9.3: Descriptive variables and team results (lowest & **highest** values)

		Updated UML Models				Updated SAD Documents			
	project team	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**
	persons in team	4	4	3	4	4	3	4	3
	environment	C++	C++/PHP	C++	C++	C++	C#	C++	C#
	implem. quality	●○○○	●●●●	●●○○	●●○○	●○○○	●●●●	●●○○	●●●●
wordcount	*SAD*	1051	670	508	670	1601	1036	1123	**2296**
	SRS	1238	850	1347	326	1116	2167	1757	**2393**
	CMP	757	377	700	578	1143	717	-	**1747**
	SDP	-	**1692**	648	602	-	912	-	-
quality	*SAD*	●○○○	●○○○	●○○○	●○○○	●●○○	●○○○	●●○○	●●●○
	SRS	●●○○	●○○○	●●○○	●○○○	●○○○	●●●○	●●●○	●●●●
	CMP	●○○○	●○○○	●○○○	●○○○	●●●○	●●○○	-	●●●○
	SDP	-	●●○○	-	●○○○	-	●○○○	-	-
	class diagram coupling	low	mid-high	low	low	mid-low	mid-high	mid-low	mid-high
	activity diagram complexity	mid	low	mid	mid	high	mid	mid	mid-high
	documentation participation	single	all, on changes	single	single	all, on effort	single	single	single

9.6.1 Documentation as a Product: Iterations Produce Better Textual Documentation

The qualitative comparison of the deliverables in Table 9.3 illustrates that textual documentation benefits from being build up iteratively. The benefits of iterative practices on diagrammatic documentation are less visible in our data. While the updates of UML models, as a more formal medium, were less visible, the updates of the teams assigned to update the less formal textual architecture documents produced more extensive artifacts.

The visualization of actual patterns of actions within the different teams (see Figure 9.3) shows how feedback resulting from multiple forms of knowledge conversion (Nonaka & Takeuchi, 1995) is embedded into the documentation. Hence, emergence of textual documentation artifacts can be clearly linked to the iterative development process. This may help to improve transparency towards the quality management of documentation deliverables (or 'documentation products'.)

However, one has to consider another aspect: In our designed experiment, team members did not perceive documentation as contributing value to their projects. Team members referred to it as *"a task that needs to be done."* Motivation to participate in an activity increases with the perception of importance and influence of the activity's results (March & Simon, 1993). Therefore, we find it reasonable to suggest that writing documentation becomes more attractive if someone, and preferrably an acknowledged stakeholder, has expressed an interest in the results. A comment of a Dutch Scrum coach in a recently conducted interview pinpoints the

issue: *'I always try to explain, "You define what amount of documentation you want, I am happy to provide you with all kinds of documentation, but I need someone who is happy with what I deliver"'*. Clients should acknowledge for the team that documentation is work of equal importance as other tasks, and their requirements towards documentation should be as clear as other requirements. In short, documentation should be considered a valuable product.

9.6.2 Documentation as a Medium: Creating Artifacts While Gambling With Collaboration

SAD teams expressed an increase in the amount of work on documentation tasks lasting throughout the development phase. This corresponds with their perception of an increasing amount of documentation being available. Among the UML teams the perception of the amount of work stayed more constant during the project and the UML teams perceived only a short peak in the amount of documentation.

One can argue that the SAD teams perceived a surplus of available knowledge during development as the team members work tightly together. This would be consistent with our earlier findings where a team member commented: *"During project development any needed documentation is generally available. However, finding documentation for older projects is not always easy, and sometimes this documentation is missing."* (Stettina & Heijstek, 2011b). However, one can also interpret this increase as a distraction that the teams perceived from the documentation task. This interpretation is supported by the perception of increasing amount of work. It suggests that updating UML models was considered less intrusive, and less demanding than updating textual documentation.

Our data shows that the majority of teams implemented strict roles dividing coding and documentation work. Most teams appointed documentation work to a single team member, most frequently the least qualified programmer. While all in all 78% of the participants were involved in coding, 30% worked on documentation and only 8% answered to have worked on both (see Figure 9.2). This is very far from the 'agile ideal' where team members are generalists and avoid specialized roles. The circles would then have much greater overlap. Agile team members do not have specific roles, instead each team member has a variety of roles. This encourages socialization within the team and is an important enabler for knowledge creation (Nonaka & Takeuchi, 1995).

An interesting aspect we discovered in the data was a strong tendency towards increasing role enforcement throughout the duration of the project. In the initial phases all team members were involved in writing documentation but gradually the 'documentation role' became the province of a single team member. We speculate that time pressure is an important explanation for this tendency. Time pressure has been found to have an impact on the choice and maintenance of team routines. Even if an inadequacy has been indicated beforehand, time pressure increases the likelihood to choose an established routine (Becker, 2004). Although perceived

as highly important, time pressure and cognitive demands of documentation tasks impede sharing of documentation work. Explicit rotation of roles and more effective documentation tools may create the necessary stimulus for sharing the documentation tasks (Fægri et al., 2010).

The analysis of the team practices in Fig. 9.3 and the distribution of roles in Fig. 9.2 illustrates that the joint effort of producing documentation faces other obstacles than the availability of documentation templates and stated documentation requirements. As we can see in Fig. 9.7, both groups similarly perceive the usefulness and importance of internal documentation. As one team member comments: *"Documentation was written because it was required, but because of the program's modularity, no documentation other than the original class diagram (and of course OpenCV, Tesseract [library] reference manuals) was necessary contributing to the project."*. While looking at the teams' documentation practices in Figure 9.3 one can recognize how the developed patterns of action, also called organizational routines (Becker, 2004), enable the feedback of team members and source code to be embedded in documentation. However, the circular knowledge creation processes are missing (Nonaka & Takeuchi, 1995). This suggests that increasing insights - stemming from documentation work - was not incorporated back into the source code.

Hence, we can see that the unbalance in documentation practices distracts from coding activities while not sufficiently adding value to the project. This led the team to perceive documentation as an intrusive task. As opposed to the incremental nature of development from agile practice, neither the formal documentation templates nor the UML models alone were very suited to support collaboration within the project teams. These observations contradict the studies of Sharp et al. (Sharp et al., 2009) which demonstrate how the notational attributes and social processes of agile artifacts such as user stories and the project wall applied in practice are mutually supportive for an agile team. For better externalization of development knowledge these reoccurring patterns of knowledge creation activity should be more balanced.

9.6.3 Agile Program and Portfolio Management: Usability and Importance of Knowledge Sharing

"There is one thing we did not implement, we did not use RUP", said team member of team B jokingly at the beginning of the system demonstration while summarizing which requirements have been met (Using RUP was not explicitly a requirement, but was widely covered in the lectures). In the short maintenance phase of the project (see project plan in Table 9.1) the teams reported not to have taken any substantial advantage of the produced documentation artifacts other than a short handover description that was to be prepared. However, team members do perceive that documentation is important for future development. When asked which documents the teams would keep up to date when the project was to be handed over, out of the five artifacts vast majority of the participants chose for software architecture document

(75%), software design models (71%) and software requirements specification (54%) to be kept up-to-date. This is also visible through the difference on perceived usefulness and importance as depicted in the two questions in Fig. 9.7. While the team members perceive it less useful within their teams now, they perceive that documentation is more important, thus indicating a tension between team and external stakeholder's interests.

As we can see in Fig. 9.7, the teams perceived updating UML models more useful and perceived it to have a bigger influence on their project satisfaction than updating SAD documents. In organizational science Carlile (2002) discussed documentation artifacts as types of boundary objects helping to bridge functional and organizational boundaries. Participants involved in cross-boundary knowledge exchange need to be aware of the personal costs and the necessity to transform own as well as influence knowledge in other domains. Effective boundary objects stimulating collaboration are a prerequisite to that. Taking our findings one can argue that the UML models are more effective boundary objects according to Carlile (2002) due to their visual representation and less effort necessary for alternation. This, however, is only true if they are created in a format easily adaptable to all participants. Software and file formats available to a single functional group only can hinder feedback of participants from other functional groups.

To conclude, for management of agile project programs and portfolios it is important to note that since team members perceive inter-project knowledge transfer as important, it is just that they oppose how the applied codification practices were implemented in their particular projects.

9.6.4 Recommendations for Research and Practice

In this chapter we have employed a new longitudinal approach to measure team and task satisfaction, based upon the project satisfaction graph as applied by Moe et al. (2010). The notation can help measuring the implications of events occurring during development on team satisfaction. Such events can be the dependency of team satisfaction on external factors. The two project satisfaction graphs (Fig. 9.4) running similarly indicate that both groups perceive a similar satisfaction in course of the project while the "humps" indicate the uncertainties encountered. We further discuss documentation as an intrusive task during development, we also present an approach how to quantify it, but the question to be solved yet is what non-intrusive documentation should look like.

We recommend that for *documentation as a product* (e.g., in regulated and contract-driven environments) it is recommended to (1) develop and deliver textual documentation iteratively by the use of easily accessible and adjustable artifacts as this will most likely improve the quality of deliverables and contribute to collaboration between people in different functional domains. In any case, however, (2) documentation needs to be communicated and accepted by the team as a proper product, product owners should motivate it as such and should be clear

on their requirements. Furthermore, (3) we argue that *documentation as a medium* (e.g. for sharing and leveraging of internal development knowledge) should be a non-intrusive task.

9.6.5 Validity Considerations

Although industrial teams are expected to have more experience than the chosen software engineering students, we argue that the challenging project environment with strict time pressure is very likely to provide an environment with similar implications on perceptions on documentation and project satisfaction. The extent to which graduate, undergraduate and doctoral students are representative of professional software developers and the threat of interaction of selection and treatment respectively, have been addressed in various other studies (e.g. (Briand, Labiche, Penta & Yan-Bondoc, 2005; Höst, Regnell & Wohlin, 2000)).

Considering the ease of use and the findings that scales with two, three, or four response categories yield least reliable scores (Preston, 2000), we have decided on using a 5-point, unipolar scale. To reduce bias we encouraged the respondents to provide their honest opinions by emphasizing the anonymous treatment of data. We re-iterated that the data collection had no implication on grading.

9.7 Chapter Conclusions

In this chapter, we presented an in-depth analysis of documentation practices in small software teams. Our combination of quantitative and qualitative data analysis allowed us to shed light on longitudinal and process-centric aspects of the documentation work carried out in the teams - covering a period of 14 weeks.

As an answer to RQ2f: *"How do externalization formalisms influence documentation practices in agile teams?"*, we may conclude that routines and boundary objects have a large impact on the sustainability and effectiveness of a routine. As an answer to RQ2g: *"What are the implications of iteratively updated internal documentation on the quality of artifacts, perceived amount of work and project satisfaction?"*, we may also conclude that iterations improve especially textual artifacts. Following our observations, as a partial answer to RQ2: "*What are the components of governance necessary to understand knowledge worker project organizations?*", we may conclude that (1) routines, and (2) boundary objects, as well as the concrete formalisms of these artifacts have an impact on the sustainability and effectiveness of a routine.

Dividing our analysis into the perspectives of *documentation as a product* and *documentation as a medium*, we found that primarily textual artifacts benefited from iterative documentation practice. Iterative practices thus not only help improving source code but can also improve the quality of documentation deliverables. We found that teams perceive documentation created for internal knowledge sharing as a distraction which causes them to choose

the least qualified programmer to update documentation. This perception of documentation as what we can call an *intrusive task* is a reason why developers in small teams don't like to write documentation - it distracts them from doing what they find most important: writing code. When developed for internal purposes one can conclude that the formal templates are able to capture the development knowledge. However, they seem not well suited to support collaboration within the team members as the majority of the teams chose a single team member to update the documents. This specialization is hampering collaboration in self-managing teams.

With respect to management practices for agile project programs and portfolios we conclude that the formal documentation was perceived as a burden by the teams as it was not contributing to the development of the software, however, knowledge sharing for future projects and maintenance was still perceived as important. Team members noted that if the project was to be handed over that they would keep software architecture document, design models and requirements specification up-to-date. This implies that the teams perceive knowledge sharing as important, however, they are unsatisfied how design knowledge has been documented.