Why SPI Initiative Failed: Contextual Factors and Changing Software Development Environment

Pasi Virtanen
Tampere University of Technology, Finland
pasi.virtanen@tut.fi

Samuli Pekkola
Tampere University of Technology, Finland
samuli.pekkola@tut.fi

Tero Päivärinta
Luleå University of Technology, Sweden
tero.paivarinta@ltu.se

Abstract

For today's software business and its productivity, software process improvement (SPI) plays a significant role. Organizations that produce software face challenges with the productivity and effectiveness of their operation. The literature lists numerous methods to make the operation better. Critical success factors are defined in order to make the successful improvement procedures more certain. However, these methodologies need to be adjusted to match the organizational context. All organizations and their environments are different, and thus the solution that is the most suitable for individual needs must be modified or localized to fit the case-specific contextual demands. This paper studies the importance of these contextual demands in SPI. In the paper, a framework is presented through which the software improvement process can be better understood and studied. The framework offers a view to understanding the change process describing eight change paths that may be observed when software process improvement is regarded.

1. Introduction

Various systems and software development activities and practices take place in software organizations. The diversity of software development and its practices has resulted in the topic being widely discussed in the literature [17,30,33,37]. The field is, however, still far from being complete. For example, productivity and quality still present problems that need to be solved [15]. Fitzgerald [11] identified factors that have an impact on system and software development practices that range from political and organizational to personal and contextual. The diversity of the field creates several practice-oriented problems that are, for instance, directly related to the implementation of the development method [2], to the development project [19], to the understanding of the user [16], and to learning from earlier mistakes [22]. These problems made Goldfinch [13] suggest that in general a pessimistic attitude should be assumed rather than overt enthusiasm and optimism.

Attempts are made to improve software development and its practices by different means and activities that are often referred to using the concept of software process improvement (SPI) [37]**.** SPI practices are usually aimed at improving software quality, increasing customer satisfaction through better responses to changing needs, and reducing risk by the improved visibility and predictability of the project [14,26]. This is, however, not an easy task. Niazi et al. [26] identified 30 success factors that each have an impact on the instantiation of SPI practices. These factors are similar to the factors that affect the instantiation of information systems methods in practice (c.f. [11]).

Fundamentally, the aim of SPI is to standardize the development practices so that it is easier to assess the current and future state of the project [37]. Another question in SPI is whether the benefits are put into practice on a more permanent basis, i.e. are lessons really learned from the experiences gathered [22]. In short, SPI means that developers (ideally) use a pre-defined practice or method in a way that is intended by the method engineer. In our research, the case organization tried to change their software development practice towards componentization in order to fulfill the task of implementing SPI within the organization, and thus defining componentization as a means to implement SPI. However, the instantiation of this new practice was not successful because developers continued their own "build-from-scratch" approach instead of developing standardized components that could be used by others. Furthermore, they did not use the components made by others.

In this paper, an attempt to change to component-based software development was studied. The aim of the study was to exploit two frameworks that may also be described as theoretical lenses [27,28] to analyze how the change took place and why the implementation of the practice deviated from the intended results. The research question considered in this study was the following: *why did the SPI initiative fail?*

The following section summarizes our theoretical background. In section three, the theoretical lenses are introduced. Section four presents the case study description and the research settings, and section five presents the results of the case study. The discussion in section six binds the results of the study with the theory.

2 Theoretical backgrounds

The success factors for software process improvement have been widely studied. One of the most cited studies was carried out by Niazi et al. [26]. In the study, 30 critical SPI success factors were identified from the literature and through empirical study. The most important success factors identified are top management support, training and the allocation of resources, staff involvement, staff experience, and well-defined SPI implementation methodologies. Of lesser importance, but still influencing the final outcome are communication, project management, tailoring improvement initiatives, company culture, and creating process action teams or external agents. From these 30 success factors, they "suggest that organizations should focus on [the 6 most] common CSFs to successfully implement SPI programs, because […] a factor […] have an impact on SPI implementation if it is critical in both [literature and empirical study]." However, even though Niazi et al. [26] pragmatically argue for a limited focus in SPI initiatives, the other factors cannot be left totally aside as different organizations may have different issues that are caused by their development context, content, and culture (c.f. [8]). There are similar studies in the literature that further confirm these findings, although they may observe the phenomenon from a slightly different angle of organizational aspect [9] or they study more the action itself [29]. The actual expansion of SPI has been studied less than the individual parts of the phenomenon [24].

These SPI success factors are also quite similar to Fitzgerald's [11] framework for the information system development (ISD) process. He argues that the use of an ISD method is shaped by the political roles of methodology such as the comfort factor, the legitimacy factor, the audit trail, the confidence factor, and the power of individual departments/actors. This is in addition to the intellectual roles of methodology such as project management, reduction to variety and complexity, economics, and communication facilitation, as well as the profile of the development environment that comprises the number of developers, project duration, responsible autonomy, productivity/rigor trade-off, and by the developers and their personal factors such as skills, domain knowledge, commitment, motivation, and trust.

However, none of these factors and frameworks emphasizes the role of individual actors in these SPI activities or in the success of new system development methods. Both Niazi et al. [26] and Fitzgerald [11] approach the topic at an organizational level, largely omitting the roles of individuals. Their findings do, however, provide a basis for contextual factors that effect on individuals. Individuals and their roles are usually seen through management commitment [1] or motivators [3,4]. Teamwork has been studied to some extent including various aspects such as multi-locational teams [6] and their productivity [5]. The importance of individual agents in these SPI endeavors or in ISD is rarely studied. The exceptions include, for example, Myers [25] who described briefly the difference between IS professionals and non-IS professionals and Choudrie & Selamat [7] who studied individuals in continuous IS development from the viewpoint of the organizational learning process. They assembled a framework that may be used to manage and monitor knowledge sharing. The latter study contributes more to knowledge management discourse than IS research.

In this paper, we will analyze an attempt to standardize software development methods by using component and componentization as a means of change. Componentization, i.e. the use of ready-made components, is often seen as a way to cut costs and as a base for mass-customized products [21,35]. Component-based software engineering (CBSE) and production is also seen as a way to increase the effectiveness of software development in several technically oriented studies (e.g. [23]). Traditional, "built-from-scratch" software development often contains a lot of unnecessary work [31]. As Fitzgerald [11] argued, the use of methods introduces discipline into the production process. If an organization moves from a "build-from-scratch" approach to a rigorous method-based development approach, the development process becomes more efficient because redundancies between components and their development are reduced, and the working practices are standardized by formalizing the development method. Consequently, at least in theory, the software development process can be improved.

Component-based software development may be further divided into centralized and de-centralized production [12,18]. In centralized component-based production, component creation and component use are separated from one another: component creators and their users are not the same [12,18]. There is a dedicated unit responsible for the creation and production of the components delivered for the actual production teams to use in their projects. In decentralized component-based production, anyone can be a creator or user of a component. The latter production method requires a repository of some kind for the components as well as a method for searching for and locating the required components. In addition, a compromise between the two is possible

[18]. The effective use of resources and intra-organizational communication concerning the development and use of reusable software is an important factor in enabling the componentization in any chosen model [32]. In the next section, the theoretical lenses are used to observe the case and to illustrate the theoretical settings as well as the way the findings were analyzed.

3 Analysis frameworks for understanding change in software development practices

We used two theoretical lenses to analyze how the target organization changed its software practice towards componentization, and why the implementation of the practice deviated from the intended results. Firstly, we adopted the theoretical lens of eight potential change paths that have an impact on the actual implementation of any software development practice in the organization [20,27,28]. Secondly, we developed some local theories that affected this implementation by using a framework for theorizing from practice descriptions, as suggested by Päivärinta et al. [28].

Figure 1 integrates the above-mentioned models by forming a theoretical framework for our case analysis. In the following, the framework is briefly explained.

It is assumed that descriptions of software development methods and practices on a general-level actually represent some kind of attempt to theorize [28]. Consequently, any particular generalized idea of a practice that guides software development actions (such as agile or structured methods, particular project management principles, etc.) will involve assumptions that concern the development contexts in which it would be useful to adopt the practice, the rationale behind the practice, the actual description of the idealized set of practices in question, and the expected effects of implementing the practices [28].

However, as denoted by the plethora of systems development literature, the actual practices, when implemented in the development contexts of organizations, may deviate greatly from the idealized method descriptions and "best practices" [10,11,17,19,34]. The stakeholders working in a particular development context may study existing literature on methods and "best practices" and adopt them to their organizations. This necessitates that their contextual rationale for adoption matches "well-enough" with the "idealized" methods and best practices. Many issues in the actual development context may intervene in the realization of a particular practice – even if its adoption has been largely intended. For example, Larsen et al. [20] identify the many different pressures on a development organization to change existing practices in their multi-case study. The examples, which confirm the theoretical issues identified by Fitzgerald [11], include the experiences of developers, particular management goals and strategies of the organization, the resources available for development, and customer preferences, etc. However, Larsen et al. [20] takes this one step further and argues that pressures to change existing practices may cause eight change paths when we analyze how certain practice has evolved in the organization: initiation, implementation, emergence, formalization, recalcitrance, abandonment, informalization, and entropy.

For example, while an organization may have a rationale to initiate and implement a common practice for managing software projects, other issues such as customer preferences may first cause recalcitrance that

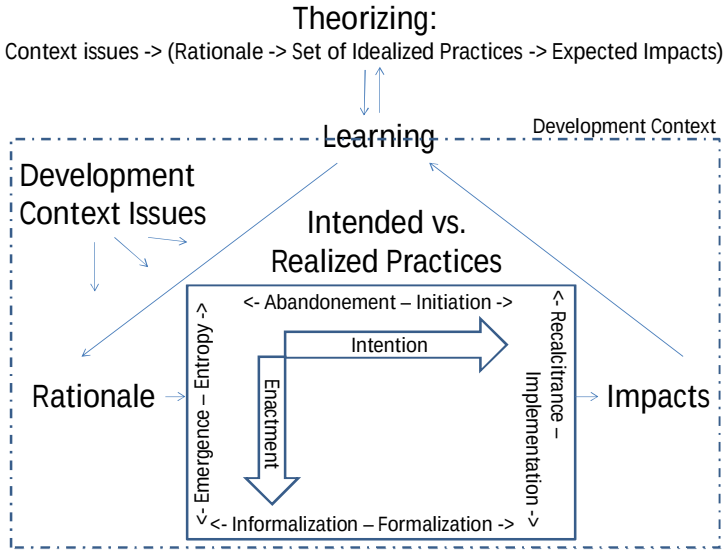

Figure 1. A Theoretical lens used to analyze and learn from practice implementation descriptions.

causes individual projects to ignore the practice, and ultimately the organization as a whole to abandon the initial idea. On the other hand, some practices may emerge implicitly through the learning-by-doing of individual developers or project managers and be formalized later on, after a project or even by the whole organization. Larsen et al [20] presents examples of the informalization and entropy of practices that may take place without managerial or developer intentions. From the viewpoint of the analysis framework, it is thus important to note several issues in order to understand a practice and its change history in context:

What was the rationale to implement a particular development practice?

What contextual issues had an impact on the rationale?

What contextual issues had an impact on the practice implementation, and how?

Finding answers to these questions helps us to form local theories concerning particular practices (and to make practice descriptions of systems and software development in organizations):

Contextual issues à Rationale for adoption/intention à Actual implementation of a practice à Realized impacts

Figure 2 below presents the framework with the findings observed in the case study. Such local theorizing gives us answers to the following types of research questions:

How do contextual issues affect rationale to adopt a practice?

How do contextual issues affect the actual implementation of a practice?

And finally – how do contextual issues affect the impact from the more or less successful implementation of a development practice in question?

The local practice descriptions [36] become interesting when they reflect the lessons learned from practice. The lessons learned are particularly interesting when they are brought back to the level of more general-level theorizing of the types of practices in question.

For example, we can now discuss about the differences between the assumed contextual issues in idealized descriptions versus the actual issues in the development context under analysis; the differences between generic versus local method adoption rationales; the differences between the ideal versus the instantiation of the practices in question; and the differences between the assumed versus realized impacts. Hence, the understanding of local practice descriptions becomes valuable from the research viewpoint when those lessons learned are analyzed in relation to more generic ideas and assumed ideals, often materialized in the form of method textbooks and "best practices" in a particular field of interest. The next section presents the case organization and features of the study.

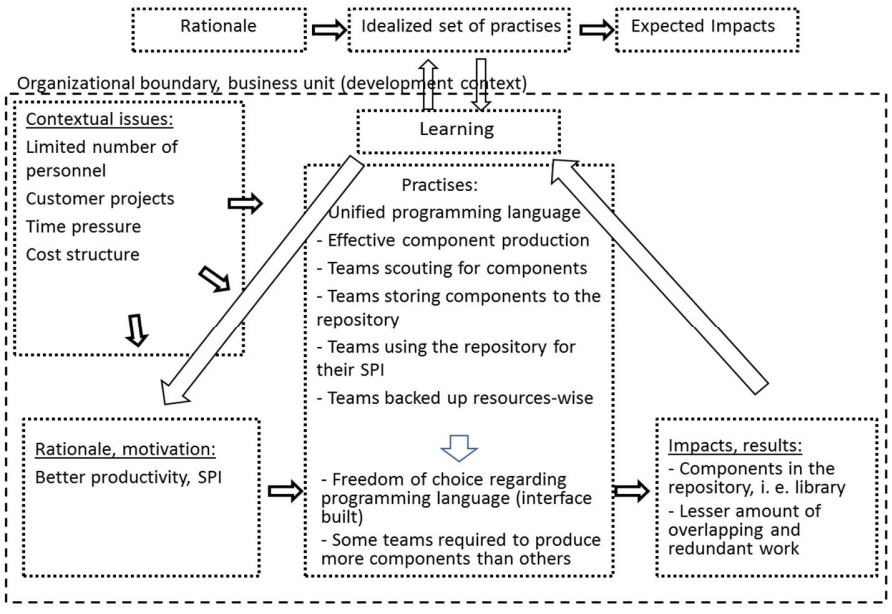

Figure 2. The theoretical lens with applied features from the case

4 The case organization and the research methods

The case organization is a business unit of a multinational software company with over 3200 employees that operates in business-to-business markets.

The organization provides large and complex ICT systems and solutions for its organizational customers. The company has grown rapidly in recent years, mostly through acquisitions and mergers. These acquisitions have made the company rather dispersed. Typically, the acquired companies have not been merged at the practical level. Instead, they have continued to work as separate teams, even in a company-like manner, inside the mother company. The mother company's original operations are also based on working in teams. Consequently, all the teams differ in many ways. They have different organizational backgrounds, technologies and tools in use, practices, products and customers, and have very different compositions. Each team has been responsible for its own software development, production, and sales. The teams are also geographically dispersed.

This study took place on eight sites. The geographical dispersion makes it difficult for the teams to know what the others in the organization are doing. Even the team leaders do not necessarily know what the other teams are working on. Due to this, the teams are usually, if not always, building software from their own premises, and often from scratch. This leads to situations where the teams do overlapping programming and software development work. Consequently, very similar features are being produced at different sites. This overlap in the software development creates extra costs and inefficiency for the company.

Among the interviewed personnel from various hierarchical levels in the case organization, there was a clear consensus that something should, and could be done to improve the organization's operation. There was little to be done with the business environment, the operational practices of personnel was the only thinkable target to look into. The competitive situation would not improve, so productivity came into focus.

Increasingly fierce competition has put the company under pressure to search for newer and more efficient ways of working in its software development and production. The aim was to eradicate redundancies, increase productivity, and improve innovativeness in order to create better and more effective solutions for their customers. The organization realized that they needed better use of knowledge. Improvements in knowledge flows and closer collaboration between the teams and individuals throughout the organization were thus perceived as essential.

The organization tried to tackle the aforementioned problems by changing their development approach to decentralized component-based software engineering where the components are distributed through a centralized component repository. By implementing this strategic decision, the organization aimed to exploit their knowledge base more effectively. With this, they aimed to release more resources from the development of standard components to the development of new, innovative ideas. This meant that the teams, in addition to doing their day-to-day tasks, had to identify potential components, i.e. products, sub-parts or features, for the possible use of the whole organization. After being approved as a common component, it was intended that the component would then be entered into the component repository to be available for the others in the organization.

However, the situation turned out to be tricky as the teams did not change their working practices. To study this phenomenon, 44 people were interviewed in two rounds of interviews. Table 1 below illustrates the interviewees and their distribution both geographically (by sites and teams) and professionally. A plus sign indicates in which round of interviews the person was interviewed.

Table 1. Summary of the Interviewees

Interviewees	Architects	Management	Teamleaders	Programmers	Other	Total
Site A		1+1	2+0		1+0	5
Site B	3+0	0+1	2+2	1+0	1+0	10
Site C			4+0	3+1		8
Site D		1+0	1+1			3
Site E	1+1	0+1	1+0	3+1	2+0	10
Site F	1+0		1+0			2
Site G	1+1		1+0		1+0	4
Site H			2+0			2
Total	8	5	17	9	5	44

The first group of interviewees comprised architects. They were change agents responsible for making the change from "build-from-scratch" to components happen. The architects were not software architects per se.

Their job was to monitor and manage the standardization process and try to ensure that the harmonization could and would happen. They were also distributed to and operating at different sites. The second group included the rest of the organization, as illustrated by different columns in Table 1. After the first round of interviews at the beginning of the organizational change, another round was conducted two to three months later, after the busiest change period was over. In the second round of interviews, there were fewer participants, as they were perceived both as a backup and as a supplement.

This division was performed in order to stress the special role of the architect team in this project. The team leaders were experienced IS professionals who were specialized in the business. Even though their level of participation in the actual software development varied, they were often as active as any other member of their team.

The interviews were anonymized, transcribed, and analyzed by thematization. The themes under which the transcribed data were classified were identified from the success factors listed in the literature [11,26]. The themes included phrases such as software development, project management, and resource allocation. After having found confirmation or falsification for the alleged success factors, the contents of the findings were assessed. Based on the assessments, interpretations of their meaning were made. These are presented in the following section.

5 The results

The interviewees were in agreement with the general-level rationale to implement the new componentization practice. They agreed on the intended benefits of componentization as a way to increase the effectiveness of software development. Theoretically, CBSE is meant to decrease the amount of overlapping work as features may be downloaded from a centralized repository. In the case organization, all stakeholders more or less agreed on the theoretical benefits as a rationale to change the effective practice. However, they also saw the need for actions to improve productivity and competitiveness, as their competitive situation was getting more difficult. They shared the unanimous opinion that their current practices and operations were far from optimal, as the amount of overlapping and redundant work in the organization had increased. This chaotic situation was acknowledged to be partly due to merged technologies and products and teams, as well as being partly due to the geographical distribution to multiple locations. Management made a strategic decision to focus their efforts on the improvement of the software development process operation through the componentization of software. Hence, in light of the analysis framework, the organization adapted the general-level ideas of componentization from the literature, taking lessons learned and best practices from the existing body of knowledge. The implementation phase was then launched.

The organization decided to adopt the decentralized approach to componentization. The architect team defined the components and their repository in addition to the component library. The architect team set the guidelines as to what was to be regarded as a component and how the component ought to be built while at the same time offering advice and consultancy to the teams. This feature was thus initiated by the architect team. As the teams were mostly developing software, they were supposed to evaluate whether various parts of their outputs could also be used as components by other teams and products.

However, in our case organization, this practice was never widely adopted. A team was formed to introduce and to help the implementation in other teams. This architect team was also supposed to promote the new way of doing software development, and to define both the concept of a component and the storage facility of the components for the organization. The first attempts to reach beneficial impacts failed. Some issues on this shortcoming emerged already during the implementation process, even though some remedial actions with positive forces for definition and adoption were taken. For example, the management declared that only one specific programming language was to be used. The architect team itself declared that they carried out their part, the perhaps sometimes-unpopular task of informing the teams about the change. According to the architect team, they had arranged meetings, built a website for the company intranet, and they had informed all the personnel on various occasions about the new ways of developing software. However, their customers, i.e. the software development teams were reluctant to change. In the words of a member of the architect team:

"Then we may talk about the communications media, that is Confluence [Intranet application, authors note] at this time. It withholds terms completely strange to some people […]"

The software developers, project managers and other stakeholders who were supposed to obey these instructions, disobeyed. There were several issues that caused recalcitrance and non-adoption of the intended practice. For example, it was said that the message was not clear enough, and that the frequency of these informative meetings and briefings was not sufficient.

"*I've been to these meetings regarding this theme [..] There it was discussed what it is that is meant by this componentization*." [team member, Team T]

"*There were discussions what it really means, this componentization, as it is a concept that can be perceived as one will*." [team leader, Team L]

This reluctance may have been caused by the fact that there were insufficient resources allocated to the change project. The teams were kept busy maintaining their level of production with ongoing customer projects. They had to concentrate on multiple things simultaneously.

"*Naturally everybody has had other things as well, apparently I've been the one who could disengage from other duties* [..] [team leader, Team T]

"*Well, I suppose that they've been able to allocate much less time than intended* [..]" [team member, Team L]

The least successful action was the attempt to standardize the programming language. The teams that were already using that language in their software development were logically more satisfied with this decision. Unsurprisingly, the teams unfamiliar with the new programming language did not welcome this decision at all. They were, however, promised support in the form of training and manpower. Some teams acknowledged the possible need for change and modernization, but still the actual change was too much for them. In addition, the training needed and promised was found to be too laborious to schedule as there were no extra resources allocated to the teams and the teams still had to keep up with their preordered software development work. Therefore, these contextual factors hindered the successful implementation of the componentization practice despite there being a match between the general rationale and between theory and the context.

6 Discussion

In the literature, the numerous factors that affect the success of SPI projects and processes have been identified. It is acknowledged that the support of top management is essential [9,26,29]. This was also found in the case organization. The announcement by company management was seen as a kick-off for the change project. A number of interviewees acknowledged that the CEO was behind this endeavor. Similarly, the architect team felt that they carried a mandate from top management, and thus they were empowered to do their job.

Training is another often mentioned feature of the success factors [26,29]. In the case organization, it was planned that training would be provided when needed. Since the first objective was to standardize the programming language, training was seen as important. However, the provision of training turned out to be poorly executed. This was because practically no one had the extra time to request, plan, arrange, or participate in the training sessions.

This final problem illustrates the importance of adequate resource allocation. Niazi et al. [26], Rainer & Hall [29] and Munk-Madsen & Nielsen [24] all list the adequate allocation of resources as a major factor in making SPI successful. The teams had to meet their everyday goals as well as to search for components with the manpower they had. There were simply no extra resources; neither manpower nor time. The deadlines of the customer projects were tight and a first priority. The teams in the case organization were promised the support they needed. However, this promise remained on an abstract level, as it was not clearly stated what this meant. It may be concluded that clear communication is also of the upmost importance as well as a real will to invest in the effort.

Moreover, the experience of personnel is a critical success factor in SPI projects [26,29]. Both architects and production team leaders were experienced professionals with track records. They continuously balanced the needs of their teams and the demands of both the project management and general management. The team leaders managed to start the production of components while simultaneously delivering their customer projects. This was not a small achievement. This, however, resulted in modifications to the initial process as predefined SPI practices had to fit in with the individual needs and context. This was contrary to the literature where the implementation of the predefined SPI methodology, and sticking with it, has been listed as one prerequisite for a successful SPI project [26,29]. The failure to follow this recommendation is one example of the "localized" implementation of the set guidelines and "idealized practices" as depicted in the theoretical framework earlier. In general, the know-how of the personnel involved is almost as critical as their commitment to the cause itself. For the management of the project, it is advisable to leave some room for contextual alterations with regard to the guidelines.

Another CSF for SPI implementation is staff involvement [26,29]. On this matter, there were diverging opinions among the architect team and the production team leaders and the team members. The architect team felt that they had carried out their tasks by informing those who were affected by the organizational change. The personnel saw their role, involvement, and commit-

ment rather differently. It became evident that those employees who had closer contacts with the architect team were better informed about the project than those without such contacts.

In general, there are a number of ways to improve and enhance the SPI process. Methodologies and methods can be found in abundance. However, special attention should be paid to selecting the appropriate measures to be put into practice, as not all measures are suitable for all situations. The same also applies to the presentation and implementation of these measures. As has been shown in this case study, there are a number of different factors that influence the implementation of SPI practices. This paper, therefore, has presented an analytical framework that also helps to form a picture of how the SPI process really proceeds. The framework helps to clarify lessons learned and to compare them to the theory and previous practices. The framework also gives an opportunity to assess organizational learning and the obstacles to its successful outcome.

In addition to social factors, the characteristics and psychological factors of individuals have a great impact on SPI. In the case organization, for example, the teams were not pleased when they were told that they would have to give up their learned practices. Doubts were also raised about the decisions taken to shift to the unification of the programming language – even though the benefits of such a shift were acknowledged in the interviews. The resistance to change may be more than just a negative phenomenon. It may initiate discussion and point out flaws in the plan.

As is often the case, no matter how active the communication about the change process is, the recipients still feel they would have preferred more and clearer information. The alleged shortcomings in intraorganizational communication effected the success of the reform significantly, i. e. the varying opinions of how much and what kind of information was communicated and how. A reluctance to change the ways of working was emphasized by the fact that employees felt they had been given too little information on the whys and the hows of the change. Eventually, management realized there was a problem and they revoked the decision regarding the programming language, as long as the interfaces enabled black-box thinking and the basic idea of the use of components remained. This satisfied the team members to some extent and the successful prototypes of the components and their use catered for the rest. This clearly shows that management should be aware of the general attitudes and happenings among their subordinates. In the case of concessions, the main principles must remain clear.

Retrospectively, in the case organization, some of the centric SPI requirements were met and some were not. When the case organization and the realization of the componentization are considered, it seems that some of the prerequisites, mentioned for example by Niazi [26], for a successful SPI project are case-specifically more important than others and few of them are entirely meaningless. This means that it is advisable to find the main features and ensure that they are properly addressed. This does not, however, justify the neglect of the others. Furthermore, based on the case organization, it appears that the contextual factors have a very significant role in implementing SPI. This would require more and deeper study. Also, from the analytical framework's point of view, it is evident that idealistic methods and practices are, and are required to be, altered when they are instantiated. The case study shows evidence that, as Larsen et al. [20] and Päivärinta et al. [28] theoretically argue, realized practices turn out to be quite different from those intended. Some intended practices and principles may even be abandoned because of recalcitrance, entropy, unproductiveness, or simply because no one has the resources to consider or implement them. Our case study showed this as management realized the problem and altered their approach. They still maintained the idea of a component-based operation that would improve productivity and the effectiveness of the software development process. Yet, it remains to be seen whether this intention lives long enough to become institutionalized practice.

7 Conclusions

Why did the SPI initiative fail? To put it bluntly, the initiative failed because top management prioritized ongoing projects over process improvement. This meant that resources such as time and money were not adequately allocated to teams to improve their development practices. The teams were left with little chance to succeed with the renewal of their modus-operandi as mundane project-related details overruled the contextual factors and larger-scale development needs and requirements.

This paper has contributed to research in multiple ways. Firstly, we have illustrated how and why realized development practices deviate from what was intended. Although these are not novel issues, their illustration and conceptualization in relation to the literation helps one to understand and ground the findings to contextual issues and the literature. Secondly, we have provided more evidence for the Larsen et al. [20] framework in how the methods evolve in organizations. Thirdly, we showed that the SPI factor prioritization by Niazi et al. [26] is highly contextual and focusing only on some success factors may not guarantee ultimate success. Instead of focusing on a subset of factors, one should take a much broader view and analyze the whole context. The Larsen et al. [20] framework may make this challenging

task easier. A practical contribution is the notion that contextual factors are far more important than were previously realized. There are managerial issues that may seem to belong more to the human resources department than to SPI. The latter stresses the need for management to conceive the whole picture rather than the separate and individual details. It becomes vital that management realizes that the adequate allocation of resources must be assured in order to achieve the expected results. These resources entail both temporal and financial leeway where customer projects are concerned.

The conclusions may be summarized on a practical and theoretical level. On the practical level, prior frameworks may provide a model to follow up on and that there are methodologies worth applying if implemented correctly: suitable under the circumstances and fitting to the context. There are so many methodologies that it implies that perhaps some are modifications to various contexts. There is no reason why things could not be done again in a similar way in another company. That is to say that the use of a methodology has generally no value as such, only as a means to an end. If there is a methodology found in the literature that suits the case, according to the experiments observed in this company, it may be implemented. Sometimes it is necessary to concede an unsuitable solution for a certain purpose. If need be, the methodology may be modified to fit the needs and the outcome can still be functional.

This paper is based on a qualitative study. The study aimed to increase understanding of the change process in software production and the factors that influence change adaptation. The authors acknowledge that the data consists of single case study organization and the opinions of certain personnel therein. This calls for consideration when making generalizations based on the data. Due to the uniqueness of each case study, more research in definitely needed.

References

[1] Abrahamsson, P. Commitment nets in software process improvement. *Annals of Software engineering 14*, 1 (2002), 407–438.

[2] Agerfalk, P.J., Fitzgerald, B., and Slaughter, S.A. Flexible and distributed information systems development: State of the art and research challenges. *Information Systems Research*, (2009).

[3] Baddoo, N. and Hall, T. Practitioner roles in software process improvement: an analysis using grid technique. *Software Process: Improvement and Practice 7*, 1 (2002), 17–31.

[4] Baddoo, N. and Hall, T. De-motivators for software process improvement: an analysis of practitioners' views. *Journal of Systems and Software 66*, 1 (2003), 23–33.

[5] Bosch-Sijtsema, P.M., Ruohomäki, V., and Vartiainen, M. Knowledge work productivity in distributed teams. *Journal of Knowledge Management 13*, 6 (2009), 533–546.

[6] Bosch-Sijtsema, P.M., Ruohomäki, V., and Vartiainen, M. Multi-locational knowledge workers in the office: navigation, disturbances and effectiveness. *New Technology, Work and Employment 25*, 3 (2010), 183–195.

[7] Choudrie, J. and Selamat, M.H. Managing organisational learning through continuous information systems development: tacit knowledge diffusion and meta-abilities perspectives. *International Journal of Knowledge and Learning 1*, 4 (2005), 342–356.

[8] Dorr, J., Adam, S., Eisenbarth, M., and Ehresmann, M. Implementing requirements engineering processes: using cooperative self-assessment and improvement. *Software, IEEE 25*, 3 (2008), 71–77.

[9] Dyba, T. An empirical investigation of the key factors for success in software process improvement. *Software Engineering, IEEE Transactions on 31*, 5 (2005), 410–424.

[10] Fitzgerald, B., Russo, N., and others. Information systems development: Methods in action. *Recherche 67*, (2002), 02.

[11] Fitzgerald, B. An empirically-grounded framework for the information systems development process. *Proceedings of the international conference on Information systems*, (1998), 103–114.

[12] Frakes, W.B. and Kang, K. Software reuse research: Status and future. *Software Engineering, IEEE Transactions on 31*, 7 (2005), 529–536.

[13] Goldfinch, S. Pessimism, computer failure, and information systems development in the public sector. *Public Administration Review 67*, 5 (2007), 917–929.

[14] Hall, T., Rainer, A., and Baddoo, N. Implementing software process improvement: an empirical study. *Software Process: Improvement and Practice 7*, 1 (2002), 3–15.

[15] Iivari, J. and Huisman, M. The relationship between organizational culture and the deployment of systems development methodologies. *Mis Quarterly 31*, 1 (2007), 35–58.

[16] Iivari, J., Isomäki, H., and Pekkola, S. The user–the great unknown of systems development: reasons, forms, challenges, experiences and intellectual contributions of user involvement. *Information systems journal 20*, 2 (2010), 109–117.

[17] Introna, L.D. and Whitley, E.A. Against method-ism: exploring the limits of method. *Logistics information management 10*, 5 (1997), 235–245.

[18] Jacobson, I., Griss, M., and Jonsson, P. *Software reuse: architecture, process and organization for business success*. acm Press, 1997.

[19] Kautz, K., Madsen, S., and Nørbjerg, J. Persistent problems and practices in information systems development. *Information Systems Journal 17*, 3 (2007), 217–239.

[20] Larsen, E., Päivärinta, T., and Smolander, K. A model for analyzing systems development practises and their evolution in organizations. *Journal of Information Technology Theory and Applications Forthcoming 2012*, .

[21] Li, Y., Yin, J., and Dong, J. A Component Management System for Mass Customization. *Computer and Computational Sciences, 2006. IMSCCS'06. First International Multi-Symposiums on*, (2006), 398–404.

[22] Lyytinen, K. and Robey, D. Learning failure in information systems development. *Information Systems Journal 9*, 2 (1999), 85–101.

[23] Meyers, B.C. and Oberndorf, P. Managing software acquisition: Open systems and COTS products. *Recherche 67*, (2001), 02.

[24] Munk-Madsen, A. and Nielsen, P.A. Success Factors and Motivators in SPI. *International Journal of Human Capital and Information Technology Professionals (IJHCITP) 2*, 4 (2011), 49–60.

[25] Myers, M.E. The IS profession and the IS professional: fit of mis-fit? *Proceedings of the 1992 ACM SIGCPR conference on Computer personnel research*, (1992), 350–351.

[26] Niazi, M., Wilson, D., and Zowghi, D. Critical success factors for software process improvement implementation: an empirical study. *Software Process: Improvement and Practice 11*, 2 (2006), 193–211.

[27] Päivärinta, T., Sein, M.K., and Peltola, T. From ideals towards practice: paradigmatic mismatches and drifts in method deployment. *Information Systems Journal 20*, 5 (2010), 481–516.

[28] Päivärinta, T., Smolander, K., and Larsen, E. \AA. Towards a Framework for Building Theory from ISD Practices. *Information Systems Development*, (2011), 611–622.

[29] Rainer, A. and Hall, T. Key success factors for implementing software process improvement: a maturity-based analysis. *Journal of Systems and Software 62*, 2 (2002), 71–84.

[30] Rico, D.F. *ROI of Software Process Improvement*. .

[31] Schuh, P. *Integrating agile development in the real world*. Cengage Learning, 2004.

[32] Sherif, K., Appan, R., and Lin, Z. Resources and incentives for the adoption of systematic software reuse. *International Journal of Information Management 26*, 1 (2006), 70–80.

[33] van Solingen, R. A follow-up reflection on software process improvement ROI. *Software, IEEE 26*, 5 (2009), 77–79.

[34] Stolterman, E. How system designers think about design and methods. *Scandinavian Journal of Information Systems 4*, (1992), 137–150.

[35] Womack, J.P. and Jones, D.T. Beyond Toyota: how to root out waste and pursue perfection. *Harvard business review 74*, (1996), 140–172.

[36] Wynekoop, J.L. and Russo, N.L. Studying system development methodologies: an examination of research methods. *Information Systems Journal 7*, 1 (1997), 47–65.

[37] Zahran, S. Software Process Improvement: Practical Guidelines for Business Success. *ISBN 0-201-17782-X*, (1998).