8-5-2011

Investigating Software Maintenance Challenges in Small Organizations

Raza Hasan
Towson University, rhasan@towson.edu

Suranjan Chakraborty
Towson University, schakraborty@towson.edu

Investigating Software Maintenance Challenges in Small Organizations

Raza Hasan
Towson University
rhasan@towson.edu

Suranjan Chakraborty
Towson University
schakraborty@towson.edu

ABSTRACT

Software Maintenance (SM) constitutes a critical function that enables organizations to continually leverage their IT capabilities. However a majority of the SM methodologies are geared toward large organizations. This is detrimental to small organizations (SO), for which SM remains an important function. This research contributes to the discipline by a) evaluating the appropriateness of existing SM methodologies for small organizations b) investigating the explicit needs of a small organization through a case based empirical study. The preliminary findings of the research indicate a lack of alignment of existing SM processes to the overall needs of the small organization. Also, the empirical investigation suggests that SM processes in small organizations are ad-hoc, resulting in a reliance on key actors for project success. Consequently, a taxonomy of the key actors is suggested and initial insights are provided about its implications on project success and important functions such as communication, collaboration and coordination.

Keywords

Small organizations, software maintenance, case study.

INTRODUCTION

It is well established in published research and industry practices that software maintenance (SM) constitutes a significant portion of work carried out by the software industry (Takang and Grubb, 1996). While large institutions have resources that can be dedicated towards software maintenance, small-sized organizations find themselves deficient in resources and thus become inefficient in management of SM Processes. SM is extremely important for organizations of all sizes; 60–80 percent of organizational resources are spent on maintenance as opposed to 20-40 percent on software development (Takang and Grubb, 1996). Researchers realize that for software to be useful, it must change and adapt to new requirements. If it doesn't, it dies (Lehman, 1969).

Despite the over arching importance of software maintenance and small organizations, little research has been carried out to identify the key challenges small organizations face in small organizations (Brodman and Johnson, 1994). The proposed research attempts to fill this void by answering these questions: How is Software Maintenance(SM) carried out in Small Organizations (SO)? Why is it so challenging for small organizations to manage software maintenance?

The rest of the document is structured as follows. In the next section, we discuss the difficulties faced by SOs in adopting a process oriented approach. Then we list current SM processes in place and how they are related to SOs. Next, we propose a methodology to explore SM in a small organization.

SMALL ORGANIZATIONS

In the current economic hard times, SOs play a vital role on the world stage. Economic growth of many countries including U.S., Brazil, Canada, China and European countries rely heavily on small businesses (Software Industry Statistics 1991-2005). In Europe, 93% of all businesses and 85% of software companies are small. In Latin America, 94% of companies developing software are small. In the U.S., 56% of all businesses are small (Pino, Pardo, García and Piattini, 2010). According to Process Maturity Profile those organizations that have 25 or fewer employees allocated to software development are considered small (Software CMM, 2005)

Recognizing the importance of SOs and the criticality of their software functions, a number of process improvement efforts such as the Software Process Improvement in Regions of Europe (SPIRE) and Software Process Improvement Network in the Satakunta region (SATASPIN) have been undertaken (Hofer, 2002). SOs realize the need for improving their processes. There are indications that such efforts can have tangible benefits. Tosun and Bener (2009) studied the effects of applying a process improvement effort. Their results suggest that: a) time allocation for requirements, coding and testing steps improved from (4%, 31%, 65%) to (47%, 20%, 33%), b) defect rates decreased from 11% to 6.5%, c) estimated testing effort also decreased from 47% to 30% (Tosun and Bener, 2009). Another benefit of applying "process thinking" in an organization is the establishment of a common language. In a SO, although people communicate and interact with each other more frequently it is rarely that such a common language is established (Rautiainen, 2002). It has been observed that different people use different terms for the product parts, even within the product development team.

Despite the benefits that can be reaped from applying process improvement methodologies, the reality of SO is different. SOs are reluctant to adopt process improvement methodologies such as CMMI because they think it is not feasible for them, their size is small and they don't have enough time for such undertakings (Staples, Niazi, Jeffery, Abrahams, Byatt, and Murphy, 2007). Studies that have examined the relationship between the size component of organizations and their reluctance to adopt software process improvements, suggest that SOs face certain challenges in using and benefiting from CMM (Paulk, 1998). A number of these challenges are perhaps related to the unique characteristics of SOs.

Typically in a very small company, a single person enacts multiple roles (Rautiainen, 2002). Given the lack of resources, same developers may be working on improving the product platform, developing new features to an existing product, installing the product at the customer's site, maintaining the product (fixing defects), or developing an entirely new product. Furthermore, SOs are customer-driven and therefore strive to respond to customer requirements quickly. Another unique characteristic of SO is they tend to have development staff consisting of more or less 5 people who are highly experienced and qualified (Harris and Clause, 2007). Since SOs employ highly qualified and experienced people, their unit cost of operation is quite high and it becomes an important threat to their growth. Therefore, one would imagine that process improvement and management models, such as CMMI, should be seen as an opportunity for small companies to align their business objectives with practice (Garcia, 2005). Unfortunately, SOs face many challenges that preclude them from such efforts. They heavily rely on engineers rather than processes. They work with tight budgets and resources, resulting in a number of communication related challenges, particularly related to customers. Consequently communication issues between customer and developer emerge as one of the main causes for delays or failure of software projects (Hofer, 2002). Management of resources is another key area of challenges faced by SOs. Such challenges include staffing, budget and timing, and the need for a proper and regular training of employees especially to newer aspects in software engineering. The issue of methods and techniques in SO bring its set of challenges: lack of tools and unique processes lead to ill defined method of conducting business in which things are often ad hoc, reactive and unplanned.

SOFTWARE MAINTENANCE

SM is defined in IEEE Standard 1219 as "the modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment" (IEEE, 1993). According to Lientz and Swanson, SM is all about "change." Based on their landmark study of 487 organizations, they suggested four types of Software Maintenance. (Lientz and Swanson, 1980):1) Adaptive: Changes made to adapt to software environment. 2) Perfective: Changes made per user requirements. 3) Corrective: Changes made to fix errors/bugs. 4) Preventive: Changes made to prevent problems in the future. A vast majority of SM concerns "enhancements" and is therefore adaptive. Corrective Changes (fixing defects) comprise only 20% of SM of activities, whereas almost 80% of maintenance efforts are allocated to Non-Corrective maintenance (enhancements). Preventive maintenance typically constitutes less than 5% of the total mix (Pigoski, 1997)

As shown in Figure 1 below, main challenges faced by a small organization, in SM are lack of resources - personnel, time, and funds (Pigoski 1997). Lack of resources in-turn results in insufficient processes, methodologies, guidelines, tools and

documentation needed for SM. These reasons (deficiencies) lead to major problems: SM becomes difficult, complex, expensive, inefficient and unmanageable (Anquetil, De Oliveira, De Sousa and Batista, 2007).

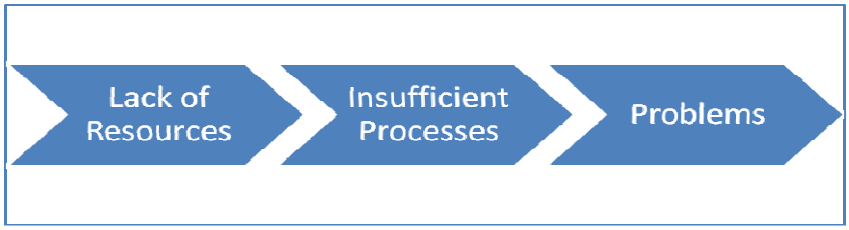

Figure 1. Software Maintenance Challenges in Small Organizations

EXISTING PROCESS MODELS OF SOFTWARE MAINTENANCE

Our previous sections indicate that a significant issue related to SM for small organizations is the lack of appropriate processes. In this section we evaluate existing SM process methodologies and examine their appropriateness for SOs. Our evaluation is based on three dimensions. First, as lack of resources (see Figure 1) is a critical bottleneck for SOs, we examine the extent to which the methodology is resource intensive. The second criteria relates to the extent to which the SM solution is easy to implement. Further, a review of literature shows that there are two types of methods/process in place for software maintenance. One type covers a portion of SM and the other, attempts to, covers all aspects. We feel that for the process to be useful, the latter category of processes are perhaps more useful to meet the SM needs of small organizations. We call this second category of processes "holistic." These "holistic" methods however are geared towards large organizations.

Non-Holistic Approaches

The most popular SM method used by SOs is Quick-Fix Model. It is based on a firefighting (ad-hoc) approach. It does not offer specific guidelines; problems are fixed as they appear (Grubb and Takang, 2003). The main problem with it is it does not consider ripple effects. This approach is not resource intensive and perhaps the easiest to implement, however it is not by its very nature holistic. Another approach is Boehm's Model. In this approach management makes SM decisions based on pure economic reasons (Boehm, 1983). Each decision goes through three stages: Investment of resources, high payoffs, and diminishing returns. This model is relevant to SOs as finances are extremely important for them, however, due to its limited scope, this model in isolation cannot be used. It also lacks in clear prescriptions, making it not easy to implement. Another non-holistic approach is Iterative Enhancement. This model states that SM basically consists of an iterative process. It goes through the stages of Analysis, Characterization of proposed modifications and Redesign & implementation in an iterative fashion (Grubb and Takang, 2003). This approach is very much applicable to SOs as it does not have too much complexity. It is not resource hungry and relatively easy to implement, however, while it allows for application in adaptive, corrective and perfective maintenance, it cannot be considered holistic as it does not offer prescriptions for preventive maintenance. Reuse-Oriented Model is yet another non-holistic method. This method states that maintenance is a re-use of existing components (Basili, 1990). There are four phases in this model: 1) Identify parts of the old system that can be re-used. 2) Understand these parts. 3) Modify these parts. 4) Integrate them into the new system. While this is a good approach to identify changes, which constitutes a major portion of software maintenance, it does not cover all aspects of software maintenance.

Holistic Approaches

We were able to identify four methods which cover all aspects of SM and can be categorized as Holistic Approaches. They are, however, out of the reach of SOs due to their costliness. They are Osborne, Staged, ISO/IEC 12207, and SM Capability Maturity Model (SC^{cmm}). Osborne's Model is based on software life-cycle with provision for maintenance at each stage. The idea is to build-in maintainability from the beginning. Osborne notes that many maintenance problems are due to shortcomings in management communication and control (Osborne, 1987). While this is perhaps one of the more comprehensive SM approaches, it is inherently resource hungry and difficult to implement for small organizations. Staged Model suggests five stages in the life cycle of software maintenance (Bennett and Rajlick, 2000). They are Initial Development, Evolution, Servicing, Phase-out, and Close-down. This is an excellent model that can be adopted by SOs to better understand maintenance stages and to ascertain specific stages of software maintenance. It is easy to implement, but suffers from being resource intensive. ISO/IEC 12207 Model is perhaps the most comprehensive model. It aims to be the standard that defines all tasks required for developing and maintaining software. It defines five primary processes for

software development (Acquisition, Supply, Development, Operation and Maintenance) and then further details the process of Maintenance (Paulk et al, 1993). Maintenance is broken into six activities of Process Implementation, Problem and Modification Analysis, Modification Implementation, Maintenance Review/Acceptance, Migration and Software Retirement. This is an extremely detailed approach but is not suitable for smaller institutions due to its complexity and large resource requirements. The final model that we evaluated is the Software Maintenance Capability Maturity Model (CMMi) from Software Engineering Institute (SEI). The purpose is to recommend a model that improves upon the existing standards (such as ISO/IEC 12207) and that incorporates daily SM activities. Based on the process domains of Capability Maturity Model (SC^{cmm}), four process domains are recommended. They are Process Management, Maintenance Request Management, Evolution Engineering and Support to Evolution Engineering. This model also requires huge implementation and does not fare well in terms of demands on resources and ease of implementation for SOs.

Conclusion about Process Models

Table 1 below provides a snapshot summary of the evaluation of the SM process models as related to SOs. Our conclusion from this investigation is that most of these models fail to provide a comprehensive solution for SM problems. There is clearly lack of specific and adaptable process improvement models (April, Hayes, Abran, and Dumke, 2005). They are not holistic for SOs and those that are, fall out of the reach of SOs due to their prohibitive resource costs and their lack of ease of implementation. Some of the problems associated with them are documentation overload, unrelated management structure, inappropriate scope of reviews, high resource requirements, high training costs, lack of need guidance and unrelated practices. This therefore reemphasizes the need for understanding better the challenges faced by SOs and identifying a process approach that fits optimally to their needs. In the remaining section of this paper we briefly describe our empirical investigation in understanding the nature of processes within SOs and report our preliminary findings

Process Model	Easy to Implement	Resource Intensive	Holistic
Quick-Fix	Yes	No	No
Boehm	No	No	No
Osborne	No	Yes	Yes
Iterative Enhancement	Yes	No	No
Re-use Oriented	No	Yes	No
ISO/IEC 12207	No	Yes	Yes
Staged	Yes	Yes	Yes
(SC^{cmm}).	No	Yes	Yes

Table 1: Evaluation the appropriateness of SM Process Models for SO

METHODOLOGY

We adopted case study approach in multiple projects in a single site to meet our research objectives. Case study method is a widely used technique of carrying out qualitative research in information systems (Orlikowski and Barooudi, 1991). It is deemed as an appropriate method for a situation where "a how or why questions are asked about a contemporary set of events over which an investigator has little or no control" (Yin, 1994).

For this study, a single site, an Information Systems department of a University was selected because it fits the description of a small organization. This site has a total staff of 16 people including 7 developers, 5 analysts, 3 managers and a director. Its main focus is to meet the software development and SM requirements of the university. It is a SO but its customer base is quite large with over 18,000 students and more than 3000 employees. To carry out its functions, this organization relies on separate technical groups such as Network Group, Database Support Team and Help Desk Operation.

To get a good mix of data and get a picture closer to reality, we propose to undertake study of multiple projects of varying sizes (large, medium and small projects). Small projects require less than 80 hours efforts, medium ones entail more than 80 hours but less than three months and large ones span over 3 months efforts. We have completed our initial data collection

for a large SM project and report our initial findings based on the analysis of this data, in this paper. This project, known as the Portal Project, was implemented over a span of 14 months and entailed enhancing the existing faculty and staff web portal with four major enhancements: Removing disjointed web sites that had appeared over the years, improving navigation to allow quicker access, allowing single sign-on to multiple applications, and providing personalization to users.

Data collection is also planned for a medium and a small project. The medium size project is titled Financial Aid. This project was undertaken to fulfill legal requirements and provide new functionalities needed by users. The smaller project involved refining of faculty security roles within the system to meet new requirements from the Registrar's and Human Resource Offices. The aim of this project was to make faculty access to student information more convenient. Enhancements related to all of these projects can be categorized as Adaptive or Perfective Software Maintenance.

Qualitative data in this Case Study were collected from two groups: the staff of IS organization and users. Five interviews were conducted with the first group. It included two developers, two analysts and one manager. They were interviewed in-depth regarding their understanding of the term software maintenance, SM processes, resources and tools, and enablers and inhibitors of software maintenance. These interviews were semi-structured and ranged from 40 – 60 minutes. In addition, nine interviews were conducted with users (faculty and staff) regarding their understanding of the enhancements needed in the Portal Project. Interviews with development team members were all tape recorded and transcribed. Interviews with the users were all written down instead of recorded due to their reluctance. Detailed notes of these user interview notes and transcribed notes of IS team were analyzed using an interpretive approach following the recommendation of Walsham (Walsham, 2006).

PRELIMINARY FINDINGS

Our initial analysis of interviewee data suggests that none of the holistic approaches to SM (mentioned above) are implemented in the selected site. For large projects, such as the Portal, we found that there is a recommended process in place. Similar to Osborne's, this process includes Project Initiation, Requirements Gathering, Development, Testing and Implementation. However, due to lack of resources, such process is often not adopted. Instead, ad-hoc processes, informal (individual and organizational) heuristics are followed which rely heavily on human actors. Such reliance brings along its set of complexities which makes it very challenging to implement SM projects.

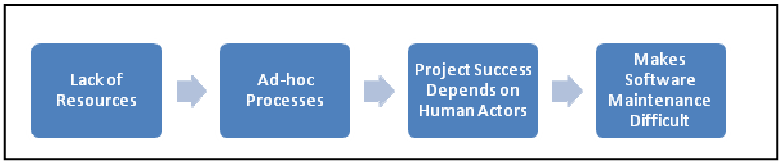

Figure 2. Why is SM so Challenging?

As shown in Figure 2, a consequence of the ad hoc process is an extreme reliance on key individuals for the success of the project. A preliminary finding of our investigation was the identification of taxonomy of the major actors – users, analysts, developers and project managers. A detailed description of the taxonomy of the actor types and their implications for SM projects in SOs is provided in Table 2 below.

Actors	Type and Description	Enablers/Inhibitors of Ad-hoc Projects	Recommendations for Small Organization
Users	**Power-Active:** Uses Portal for at least an hour daily. These users are highly motivated, possesses advanced knowledge of the application, and are highly interested in enhancements	Enablers. Their willingness, knowledge and availability greatly assist the project	This group should be identified early on in the project and brought on-board ASAP
	Power-Passive: Uses Portal for at least an hour daily and posses similar levels of application knowledge as the above category. However despite their worth, they are apathetic to the project. Interestingly while not interested in giving time to the project they demand to be kept well informed	Inhibitor. They are source of resource drain, as keeping them informed creates overheads	Inform via emails & meetings. Try to convert this group to Power-Active users
	Occasional-Active: Uses Portal occasionally. This group is interested in give time to the project, however, its members may not possess advanced knowledge	Enablers. Their willingness and availability greatly assist the project	Pair members of this group with Power-Active users and involve them. Alternatively the project can actively attempt to enhance their application knowledge
	Occasional-Passive: Uses Portal occasionally and is apathetic to the enhancements project. Not interested in giving time. Vast majority is found in this group	They provide not worth to the project but may not be inhibitors.	Should be identified early and so that there is minimum effort wastage in trying to involve them
Developers	**Active:** As they are involved in multiple projects, possess vast experience. Want to fully understand the business reasons behind the project enhancements. Have several ideas on how to solve the problem. May cause strife in project by interfering with Analyst role	Enablers if they provide expertise in the right place. Inhibitors if they interfere with the analysts and work in cross purpose with them.	The project might consider moving them to Analyst roles and also as mentors to novice developers. They might also be paired with novice analysts
	Passive: Will start coding from the specs given to them. They are not interested in the reasons behind problems and specifications	While not enablers or inhibitors in true sense, they provided worth by performing their role and not interfering with the analysts.	Such developers should be provided with clear accurate specs, as they may increase efficiency when specs are good.
Analysts	**Technical:** Want to jump in coding. They can cause strife by interfering with developers work	Enablers when their technological knowledge is harnessed properly. Inhibitors when they interfere with developers' work	Clarify roles and responsibilities. Their technical knowledge can make them a good resource when paired with novice developers
	Non-technical: Not interested in coding. Will focus on providing specifications only	Enablers. They are helpful to projects as they tend to be good at developing good specifications	Encourage to keep abreast of technological changes s their technical knowledge may get rusty. Could be paired with active developers to enhance their technological knowledge
Project Managers	**Process-oriented:** Want to follow a Waterfall type phased approach for managing projects	Enablers. They tend to rely on processes and templates to get project moving in an organized manner	Encourage on achieving objectives as opposed to focusing on means, as small organizations don't have enough resources
	Process-averse: Do not follow a process. Manage projects on an ad-hoc basis	Enablers. They rely on experience and intuitions	Encourage to learn project management methodologies.

Table 2. Taxonomy of SM project actors

The taxonomy in Table 2 highlights the need for understanding different types of actors and utilizing that knowledge to better mold certain actors towards greater involvement in project's success.

Our interview data gives indication that certain combinations of the different actor subtypes identified in table 2 could help optimize the SM process. For example a combination involving power-active users, passive developers, non-technical analyst and either type of PMs could prove to be very effective. In such a combination, actors can complement and supplement each other's skill sets. For example keeping a passive developer and a non technical analyst in the team would be helpful because this could reduce role interference and friction and increase accountability as both these actors prefer to operate within the boundary of their roles. In fact an analyst mentions whilst talking about active developers

> *"often when the developer is involved in analysis, we (analysts) then tend to not do our job as we think the coder is here and he will capture all the specs."*

Similarly our respondents indicate that technical analysts were often found to be lacking in providing specifications because of their tendency to be more involved with the coding function. In other words, while the multi-dimensional skills of the

technical analyst may be a useful asset in SOs, they do not automatically lead to project success. From our data, we found indications that they have a tendency to shying away from documenting requirements and jumping in code to fix things, thus becoming a liability. In comparison, the non-technical analysts play an important role of capturing requirements, communicating between different users and developers and ensuring change management. We feel that the above taxonomy while still at its preliminary stage provides interesting directions towards getting a better insight of the SM processes in SOs.

In addition, from the interviews data, we have learned that due to the ad-hoc nature of projects in small organization, three elements become very important in SM projects. They are Collaboration, Communication and Co-ordination (3 Cs). These elements are closely linked to the human actors identified above.

Collaboration

Collaboration was noted as a significant requirement for the successful completion of projects. One of the developers noted

> *"We depend on each other. There are so few of us, we can't know everything. So we share and collaborate to compensate for lack of knowledge."*

One recommendation is to steer collaboration by organizing a kick-off meeting at the beginning of the project. To this meeting, power-active and occasional active users should be invited. They are most inclined to collaboration. Also, it may make sense to include a developer who is not distracted by other projects and an analyst (non-technical) who would be focused on analysis rather than developing. The PM as a lead actor needs to ensure that collaboration continues after this meeting. One developer aptly put it: *"It's all up to the PM to keep the ball rolling"*.

Communication

Closely tied to collaboration is communication. Our respondents felt that this function is an important responsibility of the PM. An analyst mentioned:

> *"It falls on the PM to keep the channels of communication flowing. S/he needs to utilize various methods to keep all informed of the upcoming tasks, project status and deadlines."*

An important aspect of successful communication seems to be the capability of the PM in understanding technological verbiage such as patches, ports, firewalls, bundles and the ability to translate them into language functional users can understand. One analyst noted that *"having a PM who is technical is very important. Such a person can talk to both worlds (technical and functional)."* Also, different media of communication can be used. The first three types of users from Table 2 should be communicated to via meetings and emails whereas the last one should be informed via web-bulletins. Both PM and Non-technical analyst can split responsibilities regarding communication as it can become a huge job for a single person. Also, as noted in Table 2, certain types of actors (active developers and technical analysts) need to be reminded about their job boundaries and responsibilities. To avoid confusion and ensure accountability, the PM needs to communicate this to them.

Coordination

Another role the PM needs to play is to be constantly reminding all of the scheduled deadlines and coordinating activities of different actors. One of the respondents, a developer, stated:

> *"If the PM doesn't remind me about my assignment, things will get delayed. S/he needs to remind me constantly through emails, sending me project plans, and even verbally – as I may not have time to read project plans."*

Since SOs mostly operate under functional management style, the positions of PMs are created for each project and mainly the role is given to analysts. In other words, analysts play the role of PMs. Their titles are still however that of analysts. Without proper title and authority, conflicts and confusion arise. Hence, due to lack of authority, such PMs find it very difficult to get team members to commit to deadlines. Two strategies are employed by PMs to overcome this hurdle: Rely on functional managers to get things moving and hold regular status meetings. One analyst stated,

> *"status meetings are good reminders for us to get tasks done. As people know they have to give status to the whole team, it motivates them to show results by the meeting time."*

Moreover, coordinating appropriately with the right type of actors is very important. For instance, Power-Active users are few in a number and have great demands on their time. Dealing with this group requires efforts to minimize time wastage, it's best to give them advanced notice and send them agenda for what is needed.

CONTRIBUTION AND FUTURE RESEARCH PLANS

Small organizations (SO) are gaining importance in national economies and software maintenance (SM) occupies a huge chunk of software their software development efforts, however few scholarly papers are available that cover these areas. In this paper we attempt to fill this gap by specifically investigating the SM process as it relates to SOs. This research while still in its initial stages, contributes to the knowledge of the discipline in different ways. First it examines existing SM process methodologies and attempts to ascertain their appropriateness to small organization. Our evaluation reveals that while some of existing processes are compatible to SOs because of their ease of implementation and relatively low demand for resources, they do not in their entirety meet the needs of such organizations. Therefore there seems to be an imperative need for developing SM process methodologies explicitly tailored to suit the needs of SOs. Our second contribution relates to the preliminary findings of our empirical investigations. Our investigations into the SM processes of a SO indicates that these processes are ad hoc in nature leading to an over reliance on key personnel. In addition the analysis of our data enabled us to propose a taxonomy of the major actors. This taxonomy identifies different types of users, analysts, developers and PMs and also provides initial insight into their roles in enabling or inhibiting the SM process. The nature of the taxonomy suggests that certain combinations of these actors would be more beneficial to SM initiatives than others. We intend as a part of our future research to complete our empirical investigations to further develop the taxonomy and explain the ad-hoc processes and in the long run develop a SM process methodology which takes into account the criticality of key actors and is explicitly tailored to the needs of SOs.

REFERENCES

1. Anquetil, N., De Oliveira, K. M., De Sousa, K. D., and Batista Dias, M. G. (2007) Software maintenance seen as a knowledge management issue, *Information & Software Technology*, 49(5), 515-529.
2. April, A., Hayes, J. H., Abran, A., and Dumke, R. (2005) Software maintenance maturity model (SM^{mm}): The software maintenance process model, *Journal of Software Maintenance & Evolution: Research & Practice*, 17(3), 197-223.
3. Basili, V. R. (1990) Viewing software maintenance as reuse-oriented software development, *IEEE Software*, 7: 19-25.
4. Bennett, Keith H, Rajlich, Václav T. (2000) Software maintenance and evolution: A roadmap, *Proceedings of the Conference on the Future of Software Engineering*, p.73-87, June 04-11, 2000, Limerick, Ireland.
5. Brodman, J.G., Johnson, D.L.(1994) What small businesses and small organizations say about the CMM, *Proceedings of 16th International Conference on Software Engineering* (ICSE 1994). IEEE Computer Society Press, New York, pp. 331–340.
6. Darke, P., Shanks, G., and Broadbent, M. (1998) Successfully completing case study research: combining rigor, relevance and pragmatism, *Information Systems Journal* 8.4 (1998): 273-289.
7. Eisenhardt, K. M. (1989) Building theories from case study research. *Academy of Management Review*, 14: 532–550.
8. Garcia, S. (2005) Thoughts on applying CMMI in small settings, *Presented in Carnegie Mellon Software Engineering Institute,2005*, http://www.sei.cmu.edu/cmmi/adoption/pdf/garciathoughts.pdf.
9. Grubb P., Takang, A.A. (2003) Software maintenance concepts and practice, 2nd Ed. World Scientific Publishing, Singapore.
10. Harris, M., Aebischer, K., and Claus, T (2007) The whitewater process: Software product development in small businesses", ACM Communications Magazine, vol. 20, no. 5, 2007, pp 89-93
11. Hofer, C. (2002) Software development in Austria: Results of an empirical study among small and very small enterprises, *Proceedings Euromicro Conference, 2002*
12. IEEE, Std. 1044 (1993) IEEE standard classification for software anomalies, IEEE.
13. Lehman, M.M. (1969) The programming process. IBM Res. Rep. RC 2722, IBM Research Center, Yorktown Heights, NY 10594.
14. Lientz B. P., Swanson E. B. (1980) Software Maintenance Management, Addison Wesley, Reading, MA
15. Orlikowski, W. and Baroudi, J.(1991) Studying information technology in organizations: Research approaches and assumptions, *Information Systems Research* 2.1 (1991): 1-28.
16. Osborne, W.M., (1987) Building and Sustaining Software Maintainability, *Proceedings of Conference on Software Maintenance*, pages 13-23.

17. Paulk M. et al., (1993) Capability Maturity Model for Software, Version 1.1, tech. report CMU/SEI-93, TR-24, Software Eng. Inst.
18. Paulk, M. (1998) Using the Software CMM in Small Organizations, *Proceedings Joint 16th Pacific Northwest Software Quality Conf. and 8th Int'l Conf. Software Quality*, 1998, pp. 350–360; www.pnsqc.org/proceedings.
19. Paulk, M.C. (1998) Using the software CMM in small organizations, Joint Proceedings of the Pacific Northwest Software Quality Conference and the Eighth International Conference on Software Quality, pp.250–361.
20. Pigoski TM. (1997) Practical software maintenance: Best practice for managing your software investment, Wiley, New York NY, 29-36, 117-138.
21. Pino, F., Pardo, C., García, F., and Piattini, M.(2010) Assessment methodology for software process improvement in small organizations, *Information & Software Technology* 52.10 - 1044-1061.
22. Rautiainen, K., Lassenius, C., Vähäniitty, J., Pyhäjärvi, M., and Vanhanen, J., (2002) A tentative framework for managing software product development in small companies, *Proceedings of the 35th Hawaii International Conference on System Sciences-2002.*
23. Software CMM (2005) Mid-Year Update by Software Engineering Institute, www.sei.cmu.edu/cmmi/casestudies/profiles/pdfs/upload/2005sepSwCMM.pdf).
24. Software Industry Statistics for 1991-2005, Enterprise Ireland, 2006, www.nsd.ie/htm/ssii/stat.htm.
25. Staples, M, Niazi, M., Jeffery, R., Abrahams, A., Byatt, P., Murphy, R. (2007) An exploratory study of why organizations do not adopt CMMI, *Journal of Systems & Software*, Jun2007, Vol. 80, Issue 6, p883-895, 13p.
26. Takang, A.A. and Grubb, P.A. (1996) Software maintenance concepts and practice, Thompson Computer Press London, UK.
27. Tosun, A., Bener, A., Turhan, B., and Burak Turhan (2009) Implementation of a Software Quality Improvement Project in an SME: A Before and After Comparison,2009, *35th Euromicro Conference on Software Engineering and Advanced Applications.*
28. Walsham, G. (2006). Doing Interpretive Research. *European Journal of Information Systems 15*, 320 -330.
29. Yin, R.(1994) Case study research design and methods, 2nd Ed., Sage Publications, 9.