

**TOWSON UNIVERSITY
OFFICE OF GRADUATE STUDIES**

**INVESTIGATING CHALLENGES TO SOFTWARE
MAINTENANCE IN SMALL ORGANIZATIONS: A GROUNDED
THEORETICAL APPROACH**

By

Raza Hasan

A dissertation

**Presented to the faculty of
Towson University
in partial fulfillment
of the requirements for the degree of**

DOCTOR OF SCIENCE

**Department of Computer and Information Sciences
Towson University
Towson, Maryland 21252
May 2, 2012**

© 2012 By Raza Hasan

All Rights Reserved

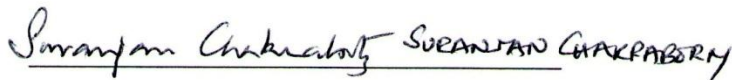
**TOWSON UNIVERSITY
OFFICE OF GRADUATE STUDIES**


DISSERTATION APPROVAL PAGE

This is to certify that the dissertation prepared by Raza Hasan

entitled Investigating Challenges to Software Maintenance in Small Organizations: A
Grounded Theoretical Approach

has been approved by the dissertation committee as satisfactorily completing the
dissertation requirements for the degree Doctor of Science

 SURANJANA CHAKRABORTY 05.02.2012
Chair, Dissertation Committee Date

 YEONG-TAE SONG 5/2/2012
Committee Member Date

 ROBERT J. ANNELLI 5/2/2012
Committee Member Date

 JOSH DELORENZO 5/2/2012
Committee Member Date

Dean of Graduate Studies

Date

Acknowledgements

Having come to the research field from industry, little did I know that being a researcher requires a totally different mindset. I learned that the field of research is an ocean by itself. I am much thankful to my adviser, Dr. Suranjan Chakraborty, who taught me how to swim in this ocean; mentored me toward becoming a researcher and helped me a great deal in completing this dissertation. I would also like to thank Dr. Josh Dehlinger and Dr. Robert Hammell, members of my dissertation committee, who have painstakingly read through my dissertation manuscripts and suggested enhancements. I am grateful to Dr. Yeong-tae Song, a member of my committee, for giving me apt advice on improving my dissertation and relating its content to established processes for large organizations. Furthermore, Alexander Peter and Atiya Afsana, my fellow doctoral students, have been great source of encouragement and support along the way.

I could not have completed the journey of my dissertation without the support of my family. My wife, Gulsanga, has encouraged and backed me momentously. While I pursued my doctoral education on a full-time basis in addition to holding a day job, it was she who held the fort at home and persevered in running a household with the arduous affairs of four children between the ages of four and eighteen. I truly appreciate the understanding of my children (i.e., Soleman, Iman, Sofia and Myra) who made room for me in their time to pursue this endeavor. My father, Mohammad Hassan, has been a source of great inspiration for me. My father started his scholarly activities after retirement and has since then successfully penned more than seven books. It was his example that got me started initially on the pursuit of a doctorate degree.

ABSTRACT

INVESTIGATING CHALLENGES TO SOFTWARE MAINTENANCE IN SMALL ORGANIZATIONS: A GROUNDED THEORETICAL APPROACH

Raza Hasan

(Chair of the Dissertation Committee: Dr. Suranjan Chakraborty)

Software Maintenance constitutes a critical function that enables organizations to continually leverage their information technology (IT) capabilities. Despite the growing importance of small organizations, a majority of the existing software maintenance guidelines are geared toward large organizations. To investigate the challenges and critical success factors in small organizations' software maintenance projects, Grounded Theory Method and case study method are used to conduct an empirical investigation.

Results from this investigation indicate a shortage of resources at the disposal of small organizations. Such shortage leads to a misalignment of existing software maintenance processes to the needs of a small organization. It is learned that software maintenance in small organizations gets achieved through heuristics undertaken by key actors of the organization. A taxonomy of key actors is provided and explicit details of heuristic development from individual's usage to organizational adoption are provided. Also presented are political processes that are utilized in small organizations to achieve software maintenance success which relies on the important functions of communication, collaboration and coordination. The two main contributions of this dissertation are: (i) it provides unique insights in the inner workings of small organizations' software maintenance projects; and, (ii) it presents key elements of software maintenance projects found in small organizations.

TABLE OF CONTENTS

LIST OF TABLES	vii
LIST OF FIGURES	viii
CHAPTER I	1
INTRODUCTION	1
CHAPTER II	4
LITERAURE REVIEW	4
Small Organizations	5
Software Maintenance	8
Critical Aspects of Software Maintenance	8
Summary of Literature Review	31
CHAPTER III	33
METHODOLOGY	33
Data Collection	34
Data Analysis	41
Summary of Methodology	53
CHAPTER IV	54
RESULTS	54
Actors	57
Individual Heuristics	66
Political Processes	86
Organizational Heuristics and Their Adoption	94
Summary of Results	104
CHAPTER V	105
Contributions	105
Limitations	108
Conclusion and Future Research	111
APPENDICES	114
Appendix A: Interview Questions	114
Appendix B: Institutional Review Board Exemption Letter	115
REFERENCES	116
CURRICULUM VITAE	125

LIST OF TABLES

Table 1: Evaluation of Software Maintenance Processes for Small Organizations.....	15
Table 2: Reverse Engineering in Software Maintenance.....	19
Table 3: An Example of Restructuring of Software Code	23
Table 4: Listing of Software Maintenance Tools	28
Table 5: Data Collection Sites	37
Table 6: Projects Data	38
Table 7: Actors in Successful and Failed Projects.....	40
Table 8: Open Coding (Labels related to Training).....	43
Table 9: Open Coding (Labels related to Politics).....	43
Table 10: Paradigm Approach	46
Table 11: Selective Coding Categories	48
Table 12: Taxonomy of Actors in Software Maintenance.....	59
Table 13: Implications of using Actors Taxonomy	64
Table 14: Individual Heuristics' application to Projects.....	68
Table 15: Humans and Computers Processing Information	75
Table 16: Political Strategies in Projects	90
Table 17: Successful Project.....	92
Table 18: Failed Project	93
Table 19: Organizational Heuristics	96

LIST OF FIGURES

Figure 1: Challenges of Small Organizations	2
Figure 2: Critical Aspects of Software Maintenance	9
Figure 3: Distribution of Software Maintenance Efforts (Pigoski, 1997)	17
Figure 4: Reengineering Model (Colbrook et al, 1990).....	24
Figure 5: Summary of Literature Review	32
Figure 6: Labels	44
Figure 7: Categorization of Concepts	45
Figure 8: Axial Coding - Category Linkages.....	47
Figure 9: Summary of Methodology.....	53
Figure 10: Theoretical Framework	55
Figure 11: Actors	57
Figure 12: Individual Heuristics	66
Figure 13: Individual Heuristics Formation Model	74
Figure 14: Political Processes	86
Figure 15: Political Styles in Both Projects	92
Figure 16: Organizational Heuristics	95

CHAPTER I

INTRODUCTION

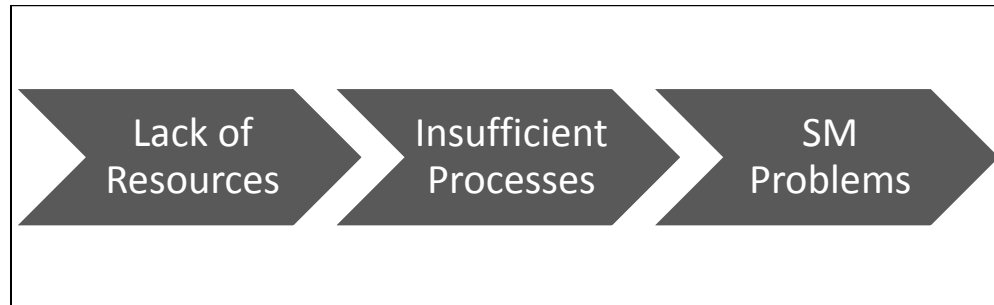
It is well established in published research and industry practices that Software Maintenance (SM) constitutes a significant portion of Software Development (Goldsmith and Siegel, 2010; Takang and Grubb, 1996). While large institutions have resources that can be dedicated towards SM, small-sized organizations find themselves deficient in resources and, thus, become inefficient in management of SM Processes. According to Process Maturity Profile those organizations that have 25 or fewer employees allocated to software development are considered small (Software CMMI, 2005).

Software Maintenance is defined in the 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).

The main challenges faced by a small organization (SO), for SM are lack of resources: personnel, time, and funds (Swanson and Dans, 2000; Benestad, Anda and Arisholm, 2009; Pigoski, 1997). As shown in Figure 1 below, lack of resources in-turn results in insufficient processes, methodologies, guidelines, tools and documentation needed for SM. These deficiencies lead to major problems: SM becomes difficult,

complex, expensive, inefficient and unmanageable (Antquetil, De Oliveira, De Sousa and Batista, 2007; Ko, Coblenz and Htet, 2006). Also, it requires shifting of resources from other projects, raising opportunity costs. Another problem is the morale of workers is affected as they are shifted to work on projects that are not interesting.

Figure 1: Challenges of Small Organizations



Areas of knowledge that can be tapped for a solution to such problems are process improvements approaches, namely Capability Maturity Model Integration (CMMI) and ISO 9000 (Guerrero and Eterovic, 2004). However, understanding and implementing CMMI and ISO 9000 based solutions are difficult, expensive and complex; therefore they fall out of the reach of SOs (Coleman, 2005).

The objective of this dissertation is to examine SM related challenges and obstacles faced by SOs. This dissertation proposes to attain this objective by investigating the following research questions:

- What are the specific obstacles and challenges faced in SM activities by SOs?
- What are the key factors that enable or inhibit successful completion of SM projects in SOs?
- How is SM carried out in SOs?

Keeping the above research questions in mind, this dissertation approaches the research investigation in the following three stages:

Stage 1: Undertake a review of related literature to identify critical aspects of SM and formulate a theoretical lens to investigate the nature of SM activities in a small organization.

Stage 2: Investigate and analyze in detail SM activities in SOs.

Stage 3: Propose an empirically driven conceptual framework that exhibits key elements that have a major impact on SM processes in SOs.

The rest of this dissertation is structured as follows. In Chapter II, a comprehensive review of literature is presented; in Chapter III, the adopted approach of qualitative research is detailed; in Chapter IV, results of the study are offered; and finally topics of contributions, limitations, conclusion and future research are covered.

CHAPTER II

LITERAURE REVIEW

Main Points

- Small Organizations (SOs) are important for the growth of national economies.
- For SOs, some process improvement approaches (CMMI, SPIRE, ISO 9000 and SATASPIN) have been undertaken; however, these efforts are resource-intensive and fall out of the reach of SOs.
- The Majority of small organizations' Information Technology resources gets allocated to Software Maintenance (SOs).
- SM involves change. Identification and implementation of change constitute SM tasks.
- None of the existing approaches are appropriate for SM in SOs.
- SOs face a shortage of resources needed for SM projects.

This chapter provides an introduction of SOs and SM. It provides detailed listings of elements that are critical to SM including a thorough review of existing methodologies and components of change. 'Change' is the essence of any SM project. In explaining

“software change,” the topics of identification, implementation, resources and tools are covered. In the discussion on “resources,” the role of “actors” in SM is emphasized, which prepares preliminary ground for one of the major findings of this dissertation, i.e., actors and their unique heuristics play an important role in the management of SM operations in SOs.

Small Organizations

In the current economic hard times, Small Organizations (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 CMMI, 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 carried out (Hofer, 2002; Aysolmaz, and Demirörs, 2011). This shows that SOs realize the need for improving their processes. There are indications that such efforts can have tangible benefits. Tosun et al. 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, Bener, Turhan and Turhan, 2009).

Another benefit of applying “process thinking” in an organization is the establishment of a common language. In a small organization, although people communicate and interact with each other more frequently, it is rare that 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 approaches the reality of SOs is different. SOs are reluctant to adopt process improvement approaches such as CMMI because they think it is not feasible for them, their size is small and they do not 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 related to the unique characteristics of SOs.

Typically in a very small company, a single person enacts multiple roles and strategic release management is done by as few as three to four people (Rautiainen, 2002). Given the lack of resources, the 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 (i.e., fixing defects), or developing an entirely new product. Furthermore, SOs are more customer-driven and their main goal is to respond customer requirements quickly. Another unique characteristic of SOs is they tend to have staff consisting of highly experienced and qualified people, and because of that their unit cost of operation is quite high and it becomes an important threat to their growth (Harris and Claus, 2007). Therefore, one would imagine that process improvement and management approaches, such as CMMI, should be seen as an opportunity for small companies to align their business objectives with practice (Garcia, 2005). However, SOs have unique characteristics as presented above and face many challenges that preclude them from the processes such as CMMI.

Challenges relate to staffing, budgeting, scheduling time, and training of employees especially to the newer aspects in software engineering. Because of these constraints, SOs rely heavily 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 (which, as a finding of this dissertation, is a major enabler of success in SM projects) 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. Lastly, the issue of methods and techniques in SOs brings its set of challenges: lack of tools and unique processes lead to ill-defined methods of conducting business in which things are often ad-hoc, reactive and unplanned.

Software Maintenance

Software Maintenance (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). Costs associated with SM can reach 90% of the total life-cycle of software cost (Midha and Bhattacharjee, 2012; Rashid et al., 2009). Researchers realize that for software to be useful, it must change and adapt to new requirements. If it does not, it dies (Lehman, 1966). While challenges faced by small or large organizations are similar in the arena of software engineering in general, some challenges of SOs are unique to SM. Considering the importance and pervasiveness of SOs, few publications have presented solutions for SOs' SM needs. While it is true that solutions exist for large organizations, given their limited resources how can SOs apply processes, techniques, best practices and tools developed for large organizations without introducing unacceptable overhead is one of the questions this dissertation is addressing.

Critical Aspects of Software Maintenance

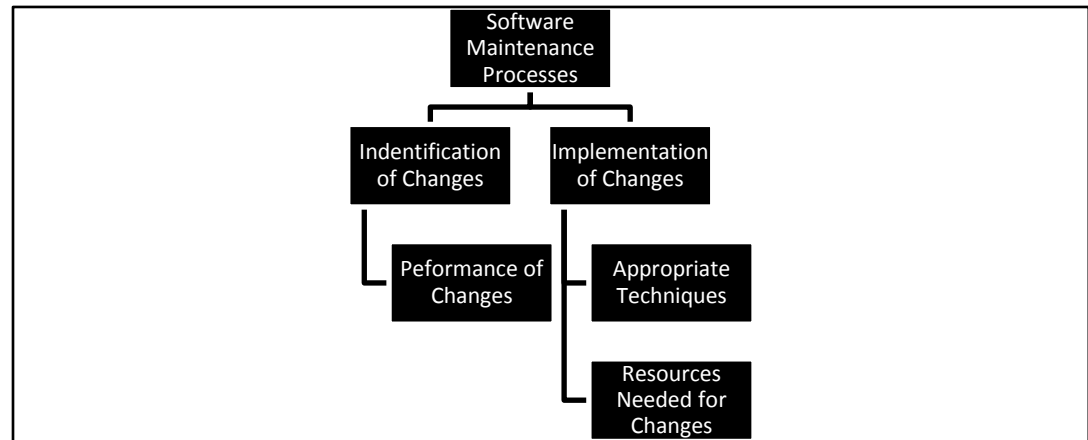
To better understand SM processes and challenges that are faced by small IT organizations, a detailed review of existing literature uncovered three critical aspects.

They are:

- (A) SM Processes
- (B) Identification of Changes
- (C) Implementation of Changes

The first aspect, SM Processes, is the overarching element that encompasses the other elements. Relationships of these elements are presented in Figure 2 which forms a lens through which literature was reviewed for this dissertation.

Figure 2: Critical Aspects of Software Maintenance



The first element, SM Process, is included in this lens because several articles list processes as the key element for improving SM for (any size) organizations (April et al., 2005; Coleman, 2005; Grubb and Takang, 2003). The second element, Identification of Change, is added because it has been cited as the SM activity that occupies maintainers the most of their time (Pigoski, 1997). The third element, Implementation of Change, is inserted because it entails all activities that maintainers perform to implement changes. Included in the third element are techniques and resources needed for change. Staffing and tools are key resources that are needed across all SM activities (Jorgensen and Sjoberg, 2002).

Each critical element of the theoretical lens (Figure 2) is presented in the following paragraphs with further details.

(A) Software Maintenance Processes

The purpose of this section is to present existing software processes and analyze their fitness to SOs. The previous sections indicate that a significant issue related to SM for SOs is the lack of appropriate processes. This premise is explored with a detailed review of existing SM processes that exist for both small and large organizations.

Specifically, in this section, existing SM process methodologies found in literature are evaluated and their appropriateness to SOs is examined. This dissertation's evaluation of these processes is based on three criteria:

- (i) How easy is it to implement the process?
- (ii) Is the process resource intensive?
- (iii) Is the process holistic?

The first criteria relates to the lack of resources (see Figure 1) which was identified as a critical bottleneck for SOs; per this criteria, this dissertation examined the extent to which a methodology is resource intensive. The second criterion pertains to the extent to which the SM solution is easy to implement. The third criterion explores the two types of methods/process in place for SM. One type covers a portion of SM and the other, attempts to, cover all aspects. For the process to be useful, the latter category of processes is more useful to meet the SM needs of SOs. The second category of processes is categorized as "holistic," which shall be used as a criterion to gauge these methodologies. These "holistic" methods are geared toward large organizations. Listed below are various methodologies for SM. They are grouped as either non-holistic approaches or holistic approaches.

Non-Holistic Approaches

Among non-holistic approaches to SM, the following were found in the literature: Quick-Fix Model, Boehm's Model, Iterative Enhancement and Reuse-Oriented Model. They are considered non-holistic for SOs because of their high-resources costs, their lack of implementation or due to their coverage of only a portion of what is needed to be included in a SM effort.

1. **Quick-Fix Model:** 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.
2. **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. Investment stage relates to the release of a new software product when input of resources is low and benefits to the organization are low as well. At the high payoffs stage, the organization's benefits increases quickly. This is where initial problems are ironed out and resources are put into enhancements, thus improving efficiencies and documentation. At the stage of diminishing returns, the product has reached its peak and cumulative benefits have slowed down. This is where management may decide to stop pumping resources into SM. 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.

3. **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-intensive and is relatively easy to implement. It allows for application in adaptive, corrective and perfective maintenance, it cannot be, however, considered holistic as it does not offer prescriptions for preventive maintenance. Since preventive maintenance is not so pervasive (it is 5% of the total mix), this model may be the closest to the SM approach adopted in SOs. This model, however, relies on iterative development that is mainly ad-hoc without many guidelines. While it may be a good approach for very small projects that are undertaken in SOs, it is not suitable for mid-sized to large projects.
4. **Reuse-Oriented Model:** 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 SM, it does not cover all aspects of SM.

Holistic Approaches

Below are 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 Software Maintenance Capability Maturity Model (SM^{cmm}).

1. **Osborne's Model:** This model is based on software life-cycle with provision for maintenance at each stage. Different stages of life cycle include, Identification of Changes, Analysis, Design and Modify Code, Review, Testing, Documentation Update and Standards Audit. 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). He recommends inclusion of maintenance requirements in change specifications and a quality assurance process that verifies that maintenance goals have been met and that provides meaningful feedback to management. While this is perhaps one of the more comprehensive SM approaches, it is inherently resource-intensive and difficult to implement for SOs.

2. **Staged Model:** This model suggests five stages in the life cycle of SM (Bennett and Rajlick, 2000). They are Initial Development, Evolution, Servicing, Phase-out, and Close-down. In the first stage, Initial Development, some features may be lacking that will require development. In Evolution stage, software is adapted to user and environment's needs and errors are

corrected. In the Servicing stage, software is no longer evolvable; changes are limited to patches and wrappers. In this stage, experienced staff is not needed and processes are stable, mature and well-understood. In Phase-out stage, servicing is no longer done, but the system is still in production. The users work around known deficiencies. In Close-down stage, software is no longer used. A replacement is made with another product. This is an excellent model that can be adopted by SOs to better understand maintenance stages and to ascertain specific stages of SM. It is easy to implement, but suffers from being resource-intensive.

3. **ISO/IEC 12207 Model:** 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.
4. **SM^{CMM}:** The final model that was evaluated is the SM Capability Maturity Model (SM^{CMM}) from Software Engineering Institute. The purpose is to

recommend a model that improves upon the existing standards such as ISO/IEC 12207 and that incorporates daily SM activities (Guerrero and Eterovic, 2004). Based on the process domains of Capability Maturity Model (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 on Software Maintenance Processes

Table 1: Evaluation of Software Maintenance Processes for Small Organizations

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
(SM ^{cmm}).	No	Yes	Yes

Table 1 provides a snapshot summary of the evaluation of the SM process models as related to SOs. The 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). These processes 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 finding reemphasizes the need for understanding better the challenges faced by SOs and identifying a process approach that fits optimally to their needs.

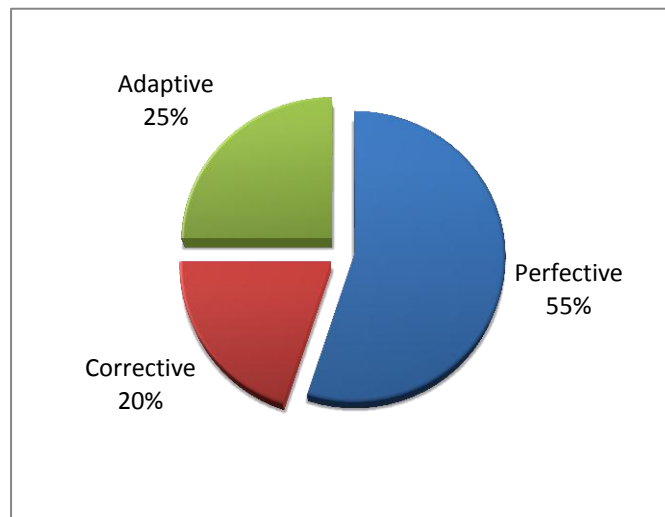
(B) Identification of Changes

According to Lientz and Swanson (1980), SM is all about “change.” Based on their landmark study of 487 organizations, they suggested the following types of SM:

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.

Contrary to popular presumption, SM is not all about fixing bugs (defects). Rather most of SM is about “enhancements.” As depicted in Figure 3 (Pigoski, 1997), Corrective Changes (fixing defects) take only 20% of SM of activities, whereas almost 80% of maintenance efforts are allocated to Non-Corrective maintenance (enhancements). Whereas preventive maintenance (not shown in the Figure 3) typically constitutes less than 5% of the total mix.

Figure 3: Distribution of Software Maintenance Efforts (Pigoski, 1997)



Identification of changes in areas where SM should take place is extremely important for SOs. Due to their limited resources, SOs cannot afford to waste resources on wrong projects. While the previous section dealt with overall SM methodologies, an important beginning phase in SM is Identification of changes. What enhancements are needed and why? What defects need to be corrected? What should be the priority to what project under limited resources? These are crucial questions that must be addressed at the beginning. A feasibility study and detailed analysis can inform decision makers of the modifications needed, alternative solutions, and costs (Pigoski, 1997). Often the analysis stage is overlooked and SM tasks are undertaken, only later to realize that the project is beyond budget and time constraints. Realizing the importance of this phase, as presented below, several publications have addressed identification of changes through Program Comprehension, Reverse Engineering, Defect Analysis and Fault Prediction.

Program Comprehension

More than half of maintainers' time is allocated to this activity. Software Maintainers spend 40% to 60% of their time reading the code and trying to comprehend its logic. (Pigoski, 1997). One specific method of understanding code is Detecting-with-Clones (Basit and Jarzabek, 2009). About 20 to 50% of software code contains clones – code that is used again and again. Identifying and comprehending these code chunks and their repetition is critical for software maintainers. Detection of clones has been done in two ways. One way has been to look at individual instances and the other, as suggested by Basit and Jarzabek, has been to look beyond at the whole structure. Basit and Jarzabek suggested a tool 'Clone Miner' for this purpose. In other words, their emphasis has been to look at the whole forest instead of the trees. This provides a good understanding of simple clones that coexist and relate to each other in certain ways. While such an understanding (overall understanding) is apparent at the time of creation, it gets very complicated during maintenance.

Reverse Engineering

Another proven method of understanding an application (that has no documentation or is getting out of date) is to use Reverse Engineering. Reverse Engineering is the process of analyzing a system to identify its components and their interrelationships and to create representations of the system in another form or at a higher level of abstraction (Grubb and Takang, 2003). The comprehension attained through reverse engineering can be used to implement change (maintenance). Table 2, below provides a summary of objectives and benefits of reverse engineering.

Table 2: Reverse Engineering in Software Maintenance

Objectives of Reverse Engineering	Benefits for Software Maintenance
<ol style="list-style-type: none"> 1. To recover lost information 2. To facilitate migration between platforms 3. To improve and/or provide documentation 4. To provide alternative views 5. To extract reusable components 6. To cope with complexity 7. To detect side effects 8. To reduce maintenance efforts 	<ol style="list-style-type: none"> 1. Maintenance <ol style="list-style-type: none"> a. Enhances understanding, which assists in identification of errors b. Facilitates identification and extraction of components affected by adaptive and perfective changes c. Provides documentation or alternative views of the system 2. Reuse: Supports identification and extraction of reusable components 3. Improved quality of system

Quality and Performance of Software Changes

Quality and performance of software maintenance is extremely important that should be considered at the beginning of any software maintenance project. “Performance refers to the system responsiveness with respect to the time required to respond to specific events, or with the number of events processed in a given time interval” (P201, Devaraj et al. 2010). Furthermore a feasibility study of every change needs to be undertaken early on. As software systems become more complex, performance should be evaluated at the beginning phases of the software lifecycle (Connie, 1990; Du, L., and Hu, Q. 2006). A few approaches are available that can assist in conducting feasibility and in predicting performance against specifications. One of them is Software Performance Engineering (SPE) which is used in the design, coding and testing stages (Connie and Lloyd 2002). Another approach is PRIMA-UML, which uses Unified Modeling Language (UML)

diagrams during analysis phases to produce a performance model. The performance model assigns probability to every node in the proposed system to link a type of user to a use case (Cortellessa and Mirandola 2002). Additionally, to ensure good performance and contain negative ripple effects, proper defect analysis and fault prediction need to be undertaken.

Defect Analysis

While program comprehension is a key step in SM, several articles have been written under the auspices of Defect Analysis (DA). Defect Analysis can basically assist in avoiding the future bugs and predict future enhancements. Defect analysis lead to defects prevention which lead to improvement in software's quality which in turn improves SM (Jalote and Agrawal, 2005). Identifying defects early on not only addresses root causes of problems but greatly helps in the later stages. DA needs Defect logging, categorization similar to the one suggested by IEEE (1993). Jalote further states that efforts must be logged to distinguish activities from review to rework. Jalote, Munshi and Probsting (2006) suggest using a When-Who-How approach for DA process improvement. This approach identifies dependencies between components and signifies co-relations between early and late defects. An example of usage of this approach can be found in earlier version of Windows. The basic idea is that after implementation of large applications, defects are recorded; data is very useful to facilitate tracking, resolution and management. Due to limited resources and pending priorities, SOs tend to pay little attention to DA. However, these (limited resources and pending priorities) are exactly the reasons that they should do more analysis. Being better prepared can greatly assist them in avoiding big pitfalls and in making the best usage of their limited resources.

Fault Prediction

This is another popular topic among researchers and very much relevant to SOs. The idea is to predict and avoid faults. It is not an easy job. Maintenance efforts can be predicted if default data exists. If data does not exist, then an indirect approach could be used (Yu, 2006). Linear regression is an example of an indirect maintenance model. Similar approaches can be used to indirectly predict faults and thus improve SM process. Additionally, future faults in SM can be predicted with Rough Set Theory --- which is utilized when representing incomplete information. Approximation and Logistic Regression is used for prediction of the probability of occurrence of a fault (Morasca and Ruhe, 2000).

Conclusion on Identification of Changes

As SOs have limited resources, it is very important for management of SOs to identify and allocate their resources optimally. While looking at enhancements, management wants to know why modifications are needed and want to spend time in the most productive way. Current literature review shows that several techniques including Program Comprehension and Reverse Engineering are utilized to identify changes. Equally important is considering the performance, feasibility and quality of software changes. Methods such as Software Performance Engineering (SPE), PRIMA-UML, Defect Analysis and Fault Prediction can assist in that. Program Comprehension is an extremely time consuming activity for software maintainers. Sometimes they find themselves spending almost half of their time on comprehending the existing code. It is important to understand the overall code design as well as in-depth details of existing programs – that takes time and in most of the times is more difficult than starting from

scratch. It is a huge challenge for SOs as most of the time documentation is non-existent and software maintainers have to reverse engineer the code to understand. Some methods of identification of changes are defect analysis and prediction. While software maintainers comprehend existing code, to alleviate future headaches, it is a good opportunity to conduct defect analysis and fault prediction. Since this is the time when minds are already engaged in understanding the code, they can analyze and think ahead about strategies of avoiding future problems. Identifying defects early on not only addresses root causes of problems but greatly helps in the later stages. Fault prediction is very important for SOs as they cannot afford to make faults and waste valuable resources

(C) Implementation of Changes

So far, SM methodologies and identification of changes have been discussed. The next logical phase is Implementation of Changes. This is where actual SM takes place. Literature review reveals the following techniques for SM implementation: restructuring, reengineering and reusability (Grubb and Takang, 2003). Most of these are popular implementation methods in SOs and among programmers, as they build upon previous knowledge and attempt to conserve resources. They, however, depend on having experience maintainers and good understanding of existing applications (the previous phase of identification of changes).

Restructuring:

This involves transformation of an application from one form to another without changing its functionality (Grub and Takang, 2003). This activity is based on the premise that after a string of modifications, the structure of programs tend to degrade and become

complex. To control this increase in complexity, the source code needs to be restructured. The purpose of restructuring code is to improve one of these qualities: Maintainability, Flexibility, Reliability, Reusability, Usability, Efficiency, Testability, Integrity, Portability, Interoperability, or Correctness (Arthur, 1988). An example of restructuring code is provided in Table 3.

Table 3: An Example of Restructuring of Software Code

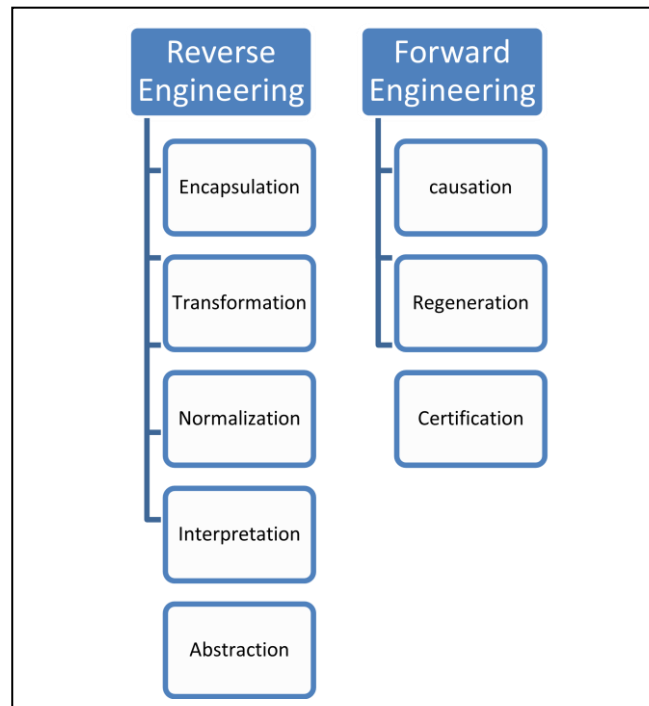
Source Code	Restructured Code
<pre> IF ExamScore >= 90 THEN Grade := 'A' ELSEIF ExamScore >= 80 THEN Grade := 'B' ELSEIF ExamScore >= 70 THEN Grade := 'C' ELSEIF ExamScore >= 60 THEN Grade := 'D' ELSE Grade := 'F' ENDIF </pre>	<pre> CASE ExamScore OF 90..100: Grade:= 'A' 80..89: Grade:= 'B' 70..79: Grade:= 'C' 60..69: Grade:= 'D' ELSE Grade:= 'F' ENDCASE </pre>

Reengineering

This involves examination and alteration of target system to fulfill desired requirements. It consists of two steps: Reverse Engineering to understand the system and Forward Engineering to apply the new modifications. These two steps can be further broken down into an 8-layer Source Code Reengineering Model (SCORE/RM) as suggested by Colbrook, Smythe and Darlison (1990) and provided in Figure 4 below. The first five layers that fall under Reverse Engineering are Encapsulation, Transformation, Normalization, Interpretation and Abstraction. The next three phases that fall under Forward Engineering are Causation, Regeneration and Certification. This model enables

maintainers to work through code, understand it, generate documentation and furthermore modify code in a way that enhances its maintainability.

Figure 4: Reengineering Model (Colbrook et al, 1990)



Reusability

Reusability is the reapplication of knowledge of one system to another similar system in order to reduce the effort of development or maintenance of that other system (Grubb and Takang, 2003). Targets of reuse can be process, personnel's knowledge, product, programs or design. Newer technology calls for new approaches such as *Component- Based* approach recommended for continuous service software such as web-based applications (Wang, Shen, Wang and Mei, 2006). Again this implementation method is very useful for small institutions as they like to overcome their limitations by reusing existing resources.

Resources Needed for Changes

Resources such as staff and tools are critical for all aspects of SM including those of processes, identification of changes, and implementation of changes. Below, staffing and tools are further explained as they relate to SM projects in SOs.

Staffing

Most software development projects get carried by human resources with varying degrees of skills available at a given time. SOs typically have small sized staff with a few people wearing many hats. While some issues in SM are related to the processes and systems in place, appropriate staffing and staff's motivations and interest can greatly influence SM efforts in SOs. Staffing for SM is an important subject matter under which researchers have looked at topics such as required *skills, effort estimation and staffing teams*.

Skills: Wang et al. (2001) investigated the dependencies of code in different versions and skills of programmers who wrote them. They found that that no differences exist between code of original and maintenance programmers. They, however, concluded that skills of programmers make a big difference. The skills and education affected the time it took to write code. Jorgensen and Sjoberg (2002) noted that application and domain experience reduced the frequency of unexpected problems.

Effort Estimation: The question of how to estimate efforts required for corrective maintenance is important for the managers of SOs. Multiple linear regression analysis can be used to construct effort estimation (De Lucia and Stefanucci, 2005). This type of estimation was found to offer better results than those offered through other methods. Determining complexity of maintenance efforts is another important

question that managers and researchers both are interested in (Bocco, Moody and Piattini, 2005).

Staffing Teams: What is the right size for SM and how should staffing be done are important questions. Investigation of size versus cost for staffing reveals that large teams tend to have good distribution of skills but has higher communication, coordination and salary costs. While smaller teams have less salary costs, they have programming biases that lead to SM issues (Pendharkar and Rodger, 2009). For staffing SM projects, a five-step method is proposed (Ramaswamy, 2000). Pair designing is another approach suggested to achieve optimized staffing on SM (Bellini et al., 2005). Pair design encompasses knowledge from domain and architectural components. Lastly, communication and knowledge sharing needs of smaller groups are investigated and social networks that encompass informal methods of communication are suggested (Nielsen and Tjørnehøj, 2010).

Tools

In addition to relying for the most part on human resources, SM can also utilize automation and tools where possible. Tools can be of great help to software maintainers. They can assist them in creating, debugging, maintaining and supporting programs and applications (Kernighan and Plauger, 1976). One of the key tools used by software maintainers is integrated development environment (IDE). IDEs combine many tools into one package and make it easy to do tasks such as searching for content in certain projects. Other tools that can be used are known as software development kit (SDKs). SDKs assist in creating and maintaining of software in particular platform, hardware or software, operating system or computer system. Tools can be sometimes as

simple as an application programming interface (API) in the form of some files to interface to a particular programming language or include sophisticated hardware to communicate with a certain embedded system. Common tools include debugging aids and other utilities often presented in an integrated development environment (IDE).

Literature review shows various tools used in SM. Below, in Table 4, is a listing of available categories and specific tools (Grubb and Takang, 2003). Most of these tools are relevant to SOs. As human resources are typically more expensive, SOs can use tools in lieu of them.

Table 4: Listing of Software Maintenance Tools

Category	Tools	Description
Program Understanding	Program Slicer	Allows to see only those portions that are affected by the change
	Static Analyzer Or Browser	Gives a quick overview of the program. Informs of modules, procedures, variables, data elements, objects and classes
	Dynamic Analyzer	Allows viewing the program when executed. Gives a trace of the execution path
	Data Flow Analyzer	Allows seeing the data flow and controlling flow paths. Provides the underlying logic of the program
	Cross Referencer	Generates an index of such elements as variables and sections
	Dependency Analyzer	Provides interrelationships between entities in a program
	Transformation Tools	Converts between text and graphics. Ex., Generates an ERD for database components
Testing	Simulator	Provides a controlled environment in which real life is mimicked
	Test Case Generator	Produces data sets to test the functionalities
	Test Path Generator	Prior to Integration and Unit testing provides all possible data flow and control flow paths
Configuration Management	Source Code Control System and Revision Control System	Keeps track of objects produced during modification
	Open Development Environment (ODE) built on RCS	Manages parallel development
Documentation and Measurement	Hypertext Tools, data flow and Control Chart generators, Requirement Tracers and CASE Tools	Various tools are available in this category

In addition to these, other tools are mentioned in published work. A fuzzy-based multi-modal tool is utilized to extract knowledge and provide quantitative inputs so future defects in corrective maintenance can be predicted is one such tool (Reformat, 2005).

MANTOOL is an automated tool that is used for managing SM process according to

MANTEMA, a rigorous methodology for maintenance (Usaola Velthuis and Gonzalez, 2001).

Related to automation and tools, four additional areas are found to be of particular interest to SM in SOs.

- (i) Aversano, Canfora, De Lucia, and Stefanucci (2002) mentioned automation of management of maintenance work flow. Some specific examples and workflow management technologies for maintenance are explored.
- (ii) Bachara, Blachnicki and Zielinski (2010) focused on reducing cost with automation. It is noted that software costs increase with size and complexity. If some forms of maintenance are delegated to automatic systems then some costs can be reduced. A tool is presented that introduces elements of adaptability to Java applications using dynamic aspects.
- (iii) Barry, Kemerer and Slaughter (2007) explored the long-term effects of using automation. It included findings from longitudinal empirical data stating that automation has enabled organization to accomplish more work activities with greater productivity and reduce errors over time. Such a study is especially helpful to a manager who has to make informed decisions about resource allocations.
- (iv) Lastly, Silva, Alonso and Torres (2009) recommended specific self-healing techniques, especially for off-the-shelf applications servers. Self-healing is achieved by continuously monitoring of system data

and performance metrics of the application server. If some anomalous behavior is identified, the system triggers automatic rejuvenation action.

Conclusion on Implementation of Changes

After identification of changes and understanding the current systems, the next important step in SM is carrying out the changes. While a lot of time gets spent in identifying changes a good amount of planning and care should be dedicated to this phase. SOs cannot afford to implement a change wrongly; they do not have resources for re-correcting their mistakes or repeating projects. In this section, four dominant techniques that are popular among SOs for implementing SM implementation were covered: restructuring, forward engineering, reengineering, and reusability. One reason for their popularity is they build upon previous knowledge of the domains and applications. If the previous stage of identification is carried out correctly, this phase becomes easy as only with a good understanding of current systems, it can be restructured, re-engineered or re-used. Individuals become key factors in success of this phase for SOs as they are the ones with institutional knowledge and understanding of current systems. This is truer even more due to lack of documentation that is found to be a common issue for SOs.

As noted above, carrying out the SM changes in SOs need two key resources. One is people and the other tools. People, as learned from literature review, are found to be the key resource used in SOs. Tools usage is per the discretion of staff especially power users and experts in SOs. It should be noted that staff in SOs have varying skills; those that are experts have to wear many hats and have to deal with many pressures.

Therefore keeping such a resource optimally, estimating the right staffing and formatting efficient teams are all challenges that SOs have to overcome. Due to the expensive nature of human resource, one would think that automation and tools where possible could reduce some of this burden. However, currently there are not many tools available that can be good substitutes for the tasks undertaken by human resources.

Summary of Literature Review

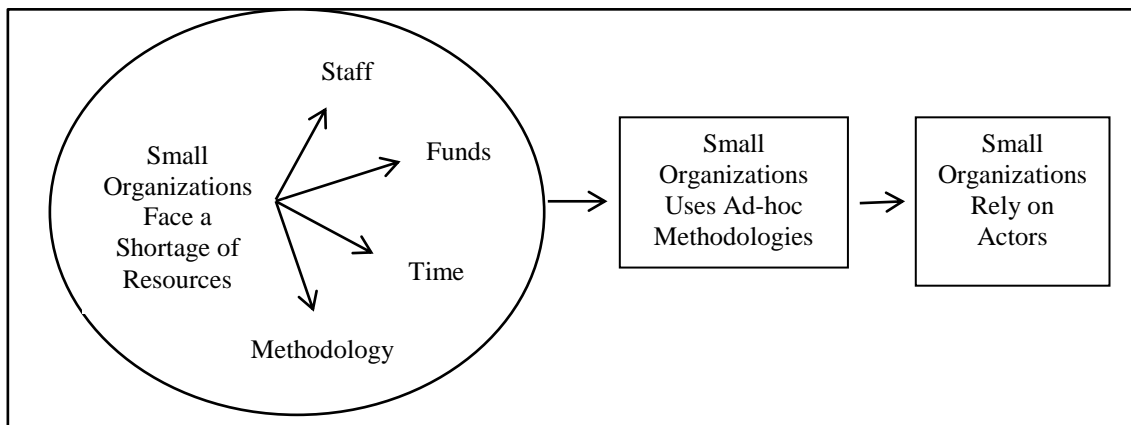
A review of literature regarding SM projects in SOs shows a paucity of attention given to this particular area. While the importance of SOs is found to be gaining ground in the process improvement related publications, work on SM in SOs was not as prevalent. One reason for this appears to be because SM is not regarded as “sexy” as software development is by software professionals.

An investigation into the salient factors of SM showed that certain elements are regarded as highly critical for SM. They are methodologies, being able to identify and implement changes, and resources at the disposal of SOs such as individuals, funds and tools. Existing SM methodologies show that all of them fail to provide a comprehensive solution to the needs of SOs. It is because they fall out of the reach of SOs, due to their high costs, lack of ease in implementation or their inability to provide a holistic solution. Hence no specific methodology was found to be helpful for SOs; rather ad-hoc methods were found to be the norm among small shops. Furthermore, it was learned that identification of changes occupies a huge amount of software maintainers’ time. Identification of changes is achieved through techniques such as program comprehension, reverse engineering, defect analysis and defect prevention. Implementation of changes

requires much planning and care. Given the limitations SOs face, resources especially individuals were found to play a vital role in the success of software.

Hence, as depicted in Figure 5, this literature review enabled me to conclude that small organization face a shortage of resources that include staff, funds, time and even methodologies, which lead to reliance on ad-hoc methods undertaken by the individuals. Individuals come out as the main drivers of SOs' SM efforts. They are the ones that weave together various threads of processes, identification and implementation of changes and utilization of appropriate resources that lead a given SM project's success.

Figure 5: Summary of Literature Review



CHAPTER III

METHODOLOGY

Main Points

- Qualitative Research Approach was used. Specifically, Case Study Method was used for Data Collection and Grounded Theory Method (GTM) was used for Data Analysis.
- Data was collected from two small IT organizations and five SM projects.
- Data was converted into labels and concepts to generate categories.
- Resultant categories show that for SM projects, SOs face shortage of resources and rely on individual heuristics.

To undertake real-life SM activities in small organization, a qualitative research approach with two specific methods was used. An Interpretive case study method was undertaken for data collection (Walsham, 2006) and Grounded Theory Method (GTM) was utilized for data analysis (Glaser and Strauss, 1967). The following sections describe the data collection and data analysis. These sections allowed the development of a theoretical framework that depicts critical elements of SM projects in SOs.

Data Collection

The section on data collection includes a discussion on the adopted case study approach, the instruments and sources of data. Case method enabled a comprehensive study of organizations under study and allowed seeing individuals and institutions in real context as they undertake actions, interactions and decisions related to software maintenance in small organizations. The sources of data, presented below, provide information about the sites and projects that were used in this endeavor.

Approach for Data Collection

A case study methodology was utilized to collect data from information systems that are considered SOs and from projects that were undertaken by these organizations. Case study has been defined by Merriam-Webster's dictionary (2009) as an intensive analysis of an individual or a community stressing developmental factors in relation to a specific context or environment. Developmental factors imply that in order to get a whole picture a case study evolves over time (Flyvbjerg, 2011). In case studies, "data are collected from a small number of organizations through methods such as participant-observation, in-depth interviews, and longitudinal studies. The case study approach seeks to understand the problem being investigated. It provides the opportunity to ask penetrating questions and to capture the richness of organizational behavior, but the conclusions drawn may be specific to the particular organizations studied and may not be generalizable" (Gable, 1994 p.113).

The case study method is a commonly used technique for carrying out qualitative research in information systems (Orlikowski and Baroudi, 1991). It is deemed as an appropriate method for a situation where "how or why questions are asked about a

contemporary set of events over which an investigator has little or no control” (Yin, 1994, p.9). Furthermore, case study research method with an interpretivist approach is used in this dissertation’s study. The interpretivist approach is suited for studying situations where reality is subjective and interpreted by individuals according to their values, context and beliefs (Eisenhardt, 1989; Drake, Shanks and Broadbent, 1998). The interpretive approach requires that "the social scientist must collect facts and data describing not only the purely objective, publicly observable aspects of human behavior, but also the subjective meaning this behavior has for the human subjects themselves" [Lee, 1991:347].

Such an approach, an interpretive case study, is recommended by Yin (1994), especially when the investigator goes into a real life context and explores the phenomenon (Yin, 1994). This matches with research objectives and context of this dissertation’s goals, whereby little work has been done in the understanding of individual processes (for instance, related to political and heuristics) in SM projects within SOs. Support for using case study can also be found in the work of Benbasat, Goldstein and Mead (1987). They encourage using interpretive case study when the research and theoretical development understanding of the particular phenomenon are at a formative stage (Benbasat et al. 1987).

Instruments for Data Collection

Data was collected from interviews and existing documents. Interviews constituted semi-structured, open-ended, questions and ranged from 40 – 60 minute time periods. All interviews were tape recorded and were fully transcribed. Reviewed documents included project plans, customer service requests, software application user

guides and software specification documents. This collection provided good data about the involved actors, processes taking place in SOs and details related to the dimension of size and success/failure of projects, SM resources and tools, and enablers and inhibitors of SM.

Sites for Data Collection

For data collection purposes, two sites were selected. Both are information systems departments (ISD) of two U.S. Mid-Atlantic public universities. The first university has a user base of 20,000 students and 3,000 employees. The ISD under study has a total of sixteen staff employees including seven developers, five analysts, three managers and one director. The second university has a user base of 3,800 students and 826 employees. The ISD of the second university has a total of eight staff employee including four developers, two analysts, one managers and one director. A vast majority of work carried out in both information system departments is in SM. Site data and organizational information is provided in Table 5.

Table 5: Data Collection Sites

	Organization One	Organization Two
Director	1	1
Managers	3 (1)*	1 (1)*
Analysts	5 (2)	2
Software Developers	7 (2)	4 (2)
Department Staff Size	16	8
Majority of work in Software Development (SD) or Software Maintenance (SM)?	SM	SM
Department Type	I.S. Department (Administrative)	I.S. Department (Administrative)
Type of Business	University	University
User Base – Students	20,000	3,800
User Base – Staff	3,000	826
*Actual Interviews		

Projects as Data Sources

Data was collected about SM practices and, in particular, about a few SM projects that were carried out in these organizations. There were two sets of projects that were looked at. The first set included three projects with different sizes (large, medium and small sizes). The second set included two projects with different outcomes (a successful and a failed project)

Data Collected from Projects with Difference Sizes

Data for this set was undertaken in the beginning of the data collection phase. This data provided a good foundation to build the findings upon. Table 6 shows data collected from three projects that represented large, medium and small sized projects.

Table 6: Projects Data

Project	Large	Medium	Small
Name	Portal	Financial Aid	Faculty Roles
Time Span	14 Weeks	11 Weeks	1 Week
SM Type	Adaptive and Perfective	Adaptive and Perfective	Perfective

All of these different sized projects are related to SM and provide a good mix of data that paints a picture that mimics reality. Based on organizational information, large projects represent projects that are carried out over a time period of more than twelve weeks, medium ones are those that are two to eleven weeks in duration and small projects require less than two weeks. A brief description of each of the projects is provided below.

The Portal Project

The Portal project was implemented over a span of fourteen 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. This was a major project that involved adapting to several requirements prescribed by users and new technologies.

The Financial Aid project

The Financial Aid project was initiated to incorporate a process to adjust a student's financial aid information in the student information system. This required updating certain existing programs. Also required were additional components prescribed by legislation that were to be added to a Financial Aid module. This project was undertaken to fulfill legal requirements and provide new functionalities needed by users.

The Faculty Roles Update project

The Faculty Roles Update project entailed making changes to existing program as per new requirements of Registrar's and Human Resource Offices. The objective of this project was to further refine faculty's security roles to make faculty access to student information system more convenient.

Data Collected from Projects with Different Outcomes

Data for this set was collected in the latter part of this dissertation work. This data provided a good way of refining the initial findings. Specifically, this data assisted in the Selective Coding phase of Grounded Theory Methodology. By selective coding phase, a tentative core category is found. It was: individuals utilize heuristics to get goals achieved in SM projects in SOs. At such a point in Grounded Theory development, Glaser (1978) recommends theoretical sampling. Theoretical sampling is the deductive phase of Grounded Theory emerging that involves selectively sampling new data with the core category in mind.

In this phase, data was collected about two projects, one was a successful project and the other was a failed one. Motivation for collecting this data was to further understand core category and to learn more about other categories such as political processes, organizational heuristics and biases. Table 7 shows this data with actors involved in the successful and failed projects under study.

Table 7: Actors in Successful and Failed Projects

Actors Involved	Successful Project	Failed Project
Developers	1	2
Analysts	2	1
I.S. Managers	1	0
I.S. Directors	1	0
I.S. Executives	1	1
Functional Users	30	5
Functional Managers	2	0
Functional Directors	0	2
Functional Executives	2	2

A brief description of the successful and failed project is provided next.

Successful Project

The purpose of the project was to automate the form submission process for students. Previously students would fill these forms online but a particular office would print them for storage, processing and retrieval. This office would print them and stack them into folders, which would be then passed around to multiple people and multiple offices located in different areas. Supplemental documents to each file would be received and sent to the appropriate folder wherever the folder may be. This created a lot of confusion, duplication of efforts, delayed processes and increased errors. A recommended solution was to overcome these problems by using document imaging and workflow technologies. The goals of the project were to enable streamline storage, quick retrieval and faster processing of these forms. All of this was achieved and the project is

considered a successful project because it met the specifications for its initiation, within time and budget.

Failed Project

This failed project was concerned with making it easy for students to review and an order item for sale at a university's vending location. If implemented this would have allowed students to see and order certain items at a web page where they often visit. This project was initiated, but got stalled in the design phase due to political hurdles put up by various actors involved in the project. This is considered a failed project as it did not meet functional specifications for which it was initiated.

Since the enhancements in both of these projects were undertaken per user requirements both successful and failed projects are considered of perfective SM type.

Data Analysis

For data analysis, GTM was chosen due to its strengths in analyzing and interpreting data that is originated from real life phenomenon on one hand and is directly related with current theories on the subject matter on the other hand. GTM is a widely used research method that is based on rigorous fieldwork. This dissertation relied on the inductive approach of GTM in which researchers explores data, allowing them to suggest meanings and explanation that may cumulate into a theoretical model. This approach requires few preconceptions and a close relation with actual data (Payne & Payne, 2004). Strauss and Corbin (1998, p.12) explains GTM in the following way: "The researcher begins with an area of study and allows the theory to emerge from the data. Theory derived from data is more likely to resemble the reality than is theory derived from

putting together a series of concepts based on experience or solely through speculation. Grounded theories, because they are drawn from data, are likely to offer insight, enhance understanding, and provide a meaningful guide to action.”

Specifically, this work used GTM to identify SM processes and generated findings in the abovementioned two projects. These findings were then sensitized with existing theories on critical elements of SM to present key factors that influence organizational adoption of individual heuristics (core category). GTM enabled data analysis was conducted over three phases: (1) Open Coding; (2) Axial Coding; and, (3) Selective Coding. Below the following sections briefly describe each phase and show how the GTM process toward generation of a theoretical framework was used.

Open Coding

In open coding, data is initially broken apart and examined line-by-line. Collected data (interview transcripts) is pondered upon, memos are written and labels and categories are identified. “A concept is a labeled phenomenon. It is an abstraction of an event, object, or action/interaction that a researcher identifies as being significant in the data (Strauss and Corbin 1998, p.103). Tables 8 and 9 are excerpts from the interview transcripts. It shows snapshots related to interviewing questions, memos conducting line-by-line analysis, and conceptual labels.

Table 8: Open Coding (Labels related to Training)

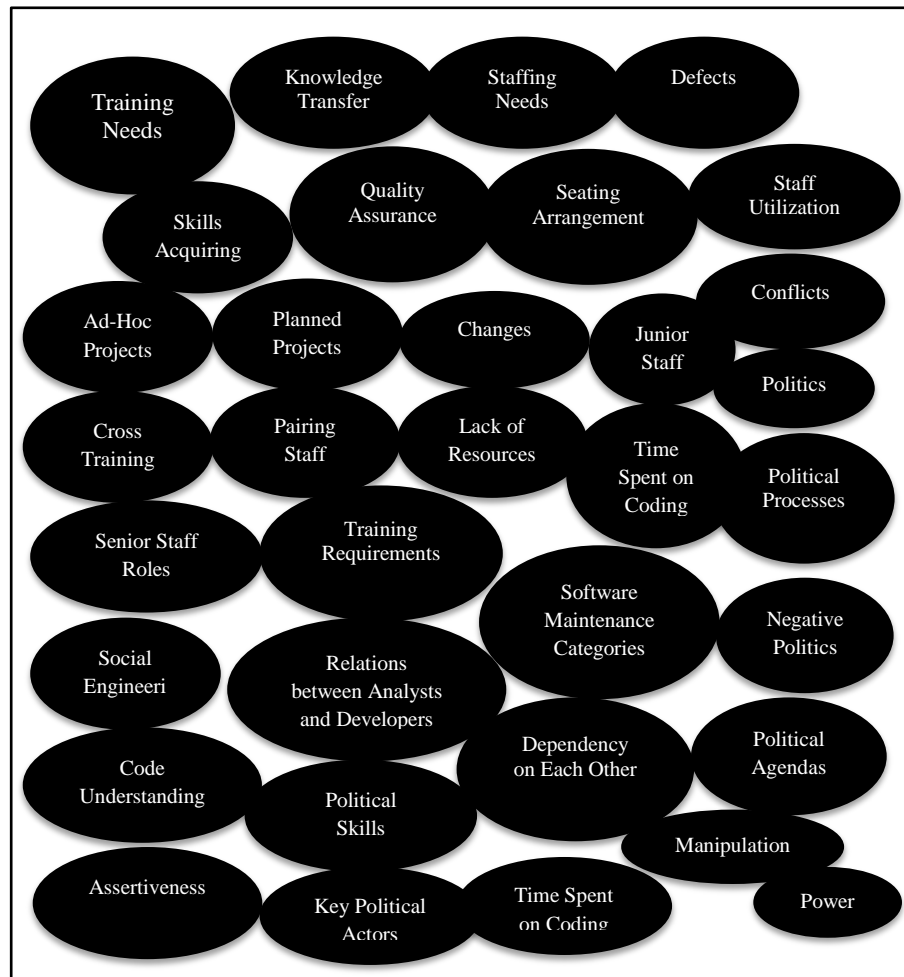
Transcript	Memo and Labels
<p>Below is transcript data from an interview with a senior developer.</p> <p><i>“Our software maintenance process involves cross training which is a good way to train and bring up-to-speed new employees. Cross training needs to be managed though. I have a lot on my plate; it’s not easy for me to get my work done. If I am expected to train juniors, that eats into my time. It gets frustrating, especially if the person on the receiving end is not very motivated”</i></p>	<p>Training is very important for a small shop such as this one. Staff cannot know everything or be trained in everything. Pairing juniors with experienced and senior staff (Cross Training) can help the organization big time in meeting skill shortage factors. The problem of efficiently using cross training is important though it has the potential to waste time of senior staff (Efficient Scheduling of Senior Staff’s Time)</p>

Table 9: Open Coding (Labels related to Politics)

Transcript	Memo and Labels
<p>Below is answer to a follow-up question, after a developer mentioned that sometimes political instincts causes people to mislead him:</p> <p><i>“They don’t want you to succeed. They want credit. They want to show that they know more. Also they don’t want others to know that you are better than them. It’s all about power and position. They want to be praised.”</i></p>	<p>This response shows a classic political maneuvering in action at workplace. There is a race among peers to get ahead of each other. It shows that people have hidden agendas. They are vying for power and positions. They feel this way they can get promotion or have more power. Sometimes others see this behavior and become involved in a political game. The climate becomes very political. People play politics or political games due to many reasons. To get control of resources, enlarge their domains or to overcome problems. People also act certain way due to emotional insecurity, peer pressure, or the politics of grouping (race, age, and seniority). This is politics at workplace. Other reasons for politics: getting ahead, getting praise (self-fulfillment), securing position, etc.</p>

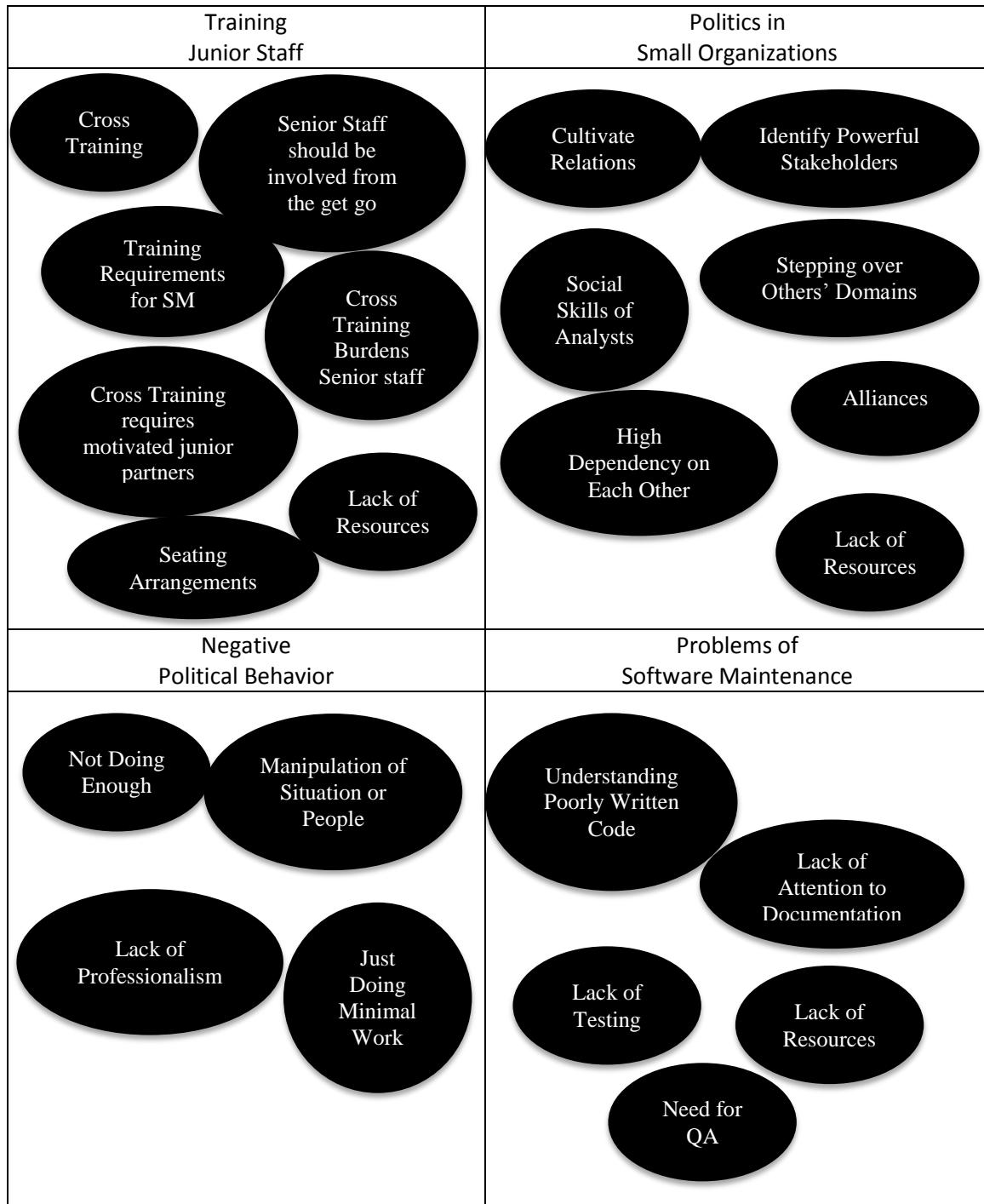
Through this process of writing memos, putting names on events, objects and happenings, a list of labels was generated, such as the one presented in Figure 6. Some of the names of the concepts are derived from the meaning or imagery they evoke when examined in context or in comparison with other related ideas, or they could be exact words of the respondents known as “in vivo codes” (Strauss & Corbin, 1998, p.145).

Figure 6: Labels



The next step in Open Coding is to take similar concepts and labels and formulate categories from them. Categories are abstract representation of labels and concepts that are classified and grouped together (Strauss and Corbin, 1998). Categories provide rich descriptions because of the definition of properties. This is an iterative process of identifying concepts, categories and properties that continues till one is fully satisfied with the results. Samples of similar concepts and labels grouped together from data are presented in Figure 7.

Figure 7: Categorization of Concepts



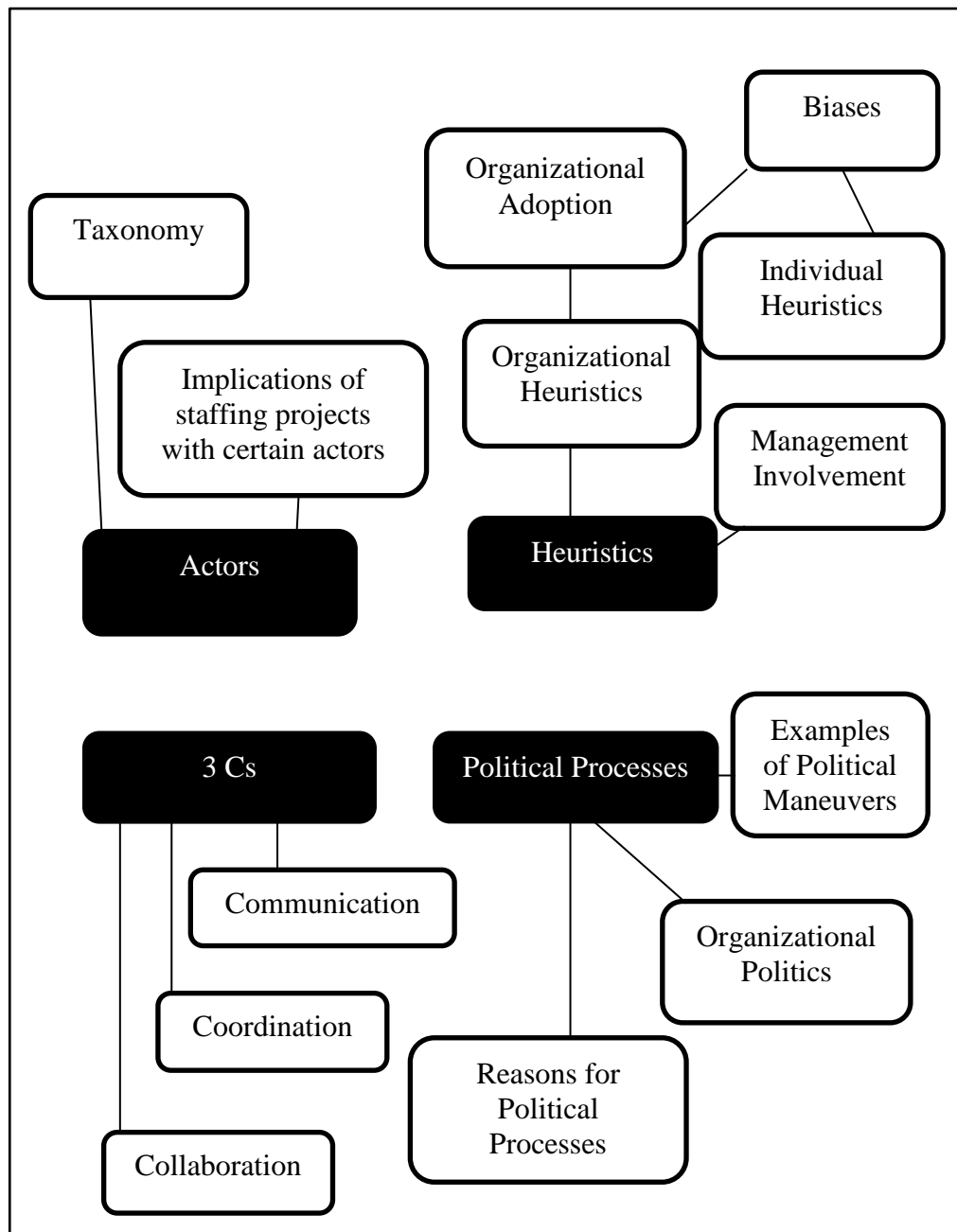
Axial Coding

“The purpose of axial coding is to begin the process of reassembling data that were fractured during open coding. In axial coding, categories are related to their subcategories to form more precise and complete explanations about phenomena” (Strauss and Corbin, 1998, p.124). Figure 8 depicts the axial coding for this dissertation. Although axial coding differs in purpose from open coding, the process of working with them is not sequential. Axial coding requires linking of categories but often time a sense of how categories relate emerges in open coding phase (Strauss and Corbin, 1998). Identification of these connections may require multiple reading of the original transcripts and documents. The Paradigm approach was then utilized, suggested by Glaser and Strauss (1967), to identify conditions, actions / interactions and consequences in the data. This approach enabled to better understand the data and generate meaningful categorization. Table 10 includes the paradigm approach toward understanding of political processes in SM projects in SOs. In this approach, questions are answered that help in producing axial coding diagram (Figure 8).

Table 10: Paradigm Approach

Conditions	<ul style="list-style-type: none"> • How do individual work differently under small organization environment as opposed to under large organizations environment? • How individual behave with set methodologies and in ah-hoc situations? • How individuals develop heuristics? • When is SM effort different than software development efforts
Actions / Interactions	<ul style="list-style-type: none"> • How do individual overcome problems with heuristics bias and implementations? • Who play the critical actions in SM projects in SOs?
With What Consequences	<ul style="list-style-type: none"> • What are results of individuals’ heuristics on project successes? • What are the consequences of individual’s actions such as political games?

Figure 8: Axial Coding - Category Linkages



Selective Coding

“It is the process of integrating and refining categories” (Strauss and Corbin 1998, p.143).

It involves selecting core categories and relating them, in a systematic way, to other categories. Continuing the GTM process, Table 11 shows the core and related categories.

Table 11: Selective Coding Categories

Type	Title
Core Category	Individual Heuristics
Category	Actors
Category	Political Processes
Category	Organizational Heuristics enable 3 Cs

These categories are augmented and facilitated by “Theoretical sensitivity” which is explained by Urquhart (2001) as when authors review the literature in the substantive field and relate that literature to their own work. Individual’s involvement in getting SM moving in small organization has been established (Hasan and Chakraborty, 2011). In the absence of well-defined guidelines, SM gets done through the efforts of individual actors who employ certain heuristics that they have acquired through experience. In implementation of these heuristics, these actors in turn employ some political processes. Sabherwal and Grover (2010) have identified the nature of these political processes to be consisting of three general categories: Empire Building, Tug of War and Obstacle Race. Additionally, work by Kipnis and Schmidt (1988) has been used to understand the actors’ political styles. The purpose of this approach is to generate a theoretical framework (Chapter IV, Results) that explores and explains the phenomenon of individuals’ involvement in the political processes of SOs’ SM efforts, in addition to other components.

Theoretical Sensitivity

This dissertation heavily depended on using theoretical sensitivity to facilitate the emerging theoretical narrative. In developing empirically grounded theoretical models, GTM scholars stress the significance of being sensitive to existing research. They emphasize this as a further source of inspiration. As such, this objective was achieved by drawing on broader theoretical perspectives from related IS Psychological and Organizational research (e.g., Urquhart, Lehman and Myers, 2010; Glaser 1978).

In particular, this work was interested in developing an insight into the nature of the individual heuristics utilized in SOs' SM projects. The core finding of this dissertation is in SOs in order to perform various critical functions of SM, individual actors act as the drivers of organizations. They do this with the help of certain heuristics that they have acquired over a long time period. The term heuristics was coined by Simon (1957) who described heuristics as adaptive strategies undertaken by humans when their capacity is limited for information processing. Existing research shows that people use heuristics all the time. They use different heuristics to make sense of the decisions they make. An example of this can be found in Political Science where citizens employ different heuristics to make sense of politics (Bray and Sniderman, 1985). If a politician is Republican in America, then voters readily infer that s/he is for low taxes, strong defense, against government intervention in the economy, etc. Heuristics are used often and are associated with higher quality decisions. If heuristics did not work at least some of the time, they would not be developed and utilized (Lau and Redlawsk, 2001). Furthermore, in organizational settings, heuristics often represent accumulated wisdom of experts or experienced resources. While explaining experts, Sniderman, Brody and

Tetolck, (1991) mentioned that the comparative advantage of experts is not that they possess a great amount of knowledge, but that they know how to utilize knowledge in the best manner. In other words, not only do experts or experienced resources employ certain cognitive heuristics, but they are very much likely to employ them appropriately (Lau and Redlawsk, 2001).

To better understand the unique value each actor added, politics in the context of SOs was investigated. Politics is an elephant whose existence is often recognized in the room but is discussed only in hushed tones and in dark corners. Political activity is a ubiquitous part of individuals and organizations life. Senior executives, managers, developers, analysts, and employees from all rank and file use political means (Fairholm, 1993). What is politics? Politics in literature refers to having power. Power enables one to achieve certain outcomes or overcome some resistance. Dahl (1957) refers to power as the ability of a person to get others to do something even against their wills. Robey and Sales (1994) explains it as the ability to get things done according to one's wishes. The importance of politics and power in organizations is well established. A significant amount of research has been geared towards study of politics and power in managing information systems (Kling R., 1980; Markus, 1983; Pfeffer, 1992; Jaseperson et al., 2002; Sabherwal and Grover, 2010).

In SOs, politics play an even more significant role. Hasan and Chakraborty (2011) have established that SOs do not have well-defined methodologies to guide its SM efforts. In such environments, as stated by Fairholm (1993), political behavior increases when formalized rules and systems are not in place. Madison (1980) has also noted that politics increases in situations of uncertainty. Furthermore, regarding SOs, Pfeffer

reported that an understanding and channeling of power processes in SOs is very important due to the reasons of heavy interdependence of individuals and increased conflicts (Pfeffer, 1992). While there has been plenty of research work carried out to explore politics in organizations in general and in I.S. in particular, there is a paucity of research on political processes' impact on SOs as it relates to efforts undertaken in SM. As noted in the above sections, individuals in SOs employ heuristics to get things done. However, while utilizing these heuristics, they face resistance and obstacles which only through political means they are able to overcome.

Overcoming resistance through politics in software projects has been discussed in detail by Markus (1983). In her influential paper, she presented a theory of resistance that underscored resistance by people as the most important factor in implementation of management information systems. She listed three basic explanations for resistance: people resist because of their own internal factors, because of poor design of the system or because of interaction of specific system with the organization's components. An example of resistance due to interaction of characteristics of system and people is an Enterprise Resources Planning (ERP) implementation that causes certain staff to feel insecure and that prompts them to find ways (political) to create hurdles in the implementation of ERP in the organization. This dissertation hopes to offer some insights into the mechanisms of such resistances and related political implications.

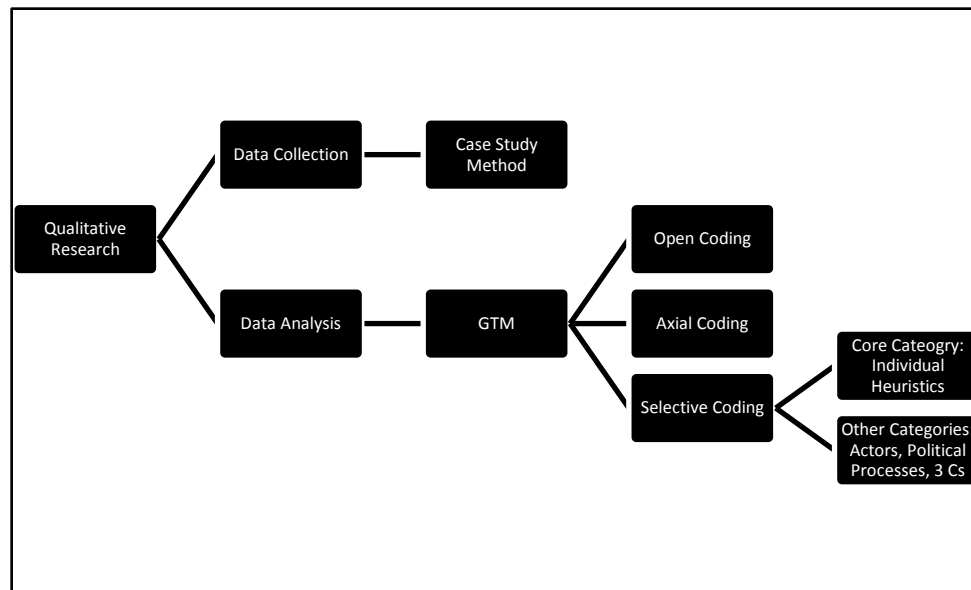
Withholding support, delaying, providing token contributions, acting confused are a few types of tactics employed by individuals to put resistance (Doolin, 2004). Bardach (1997) mentioned that individuals play these political games because they want to achieve certain outcome, deflect goals, dissipate energies or divert resources. Other

political means that impact organizations are coalition building, deceit control and gaining support from a higher authority (Myers and Young, 1997). Jaspersen et al. (2002) conducted a thorough review of existing articles and studies on information systems politics and presented theoretical conceptualizations included workings of authority, participation in decision making, influence, power and politics. Kling (1980) has also provided various theoretical perspectives on people's resistance. He mentioned organizational politics and class politics as the main political categories to study along with areas such human relations, rational reasons, structural issues and human relations. A recent article by Sabherwal and Grover (2010) has further delineated the political maneuvers by presenting a model of political strategies that is employed by individuals in organizations. These political strategies are Empire Building, Tug of War and Obstacle Race. To gain an understanding of political processes and their implications in SOs, this work applied Sabherwal and Grover's (2010) and Kipnis and Schmidt's (1988) models to collected qualitative data. Further explanation and application of these models are presented in Results, Chapter IV.

Summary of Methodology

As shown in Figure 9, the research method adopted for this dissertation was Qualitative Research Method. In particular, for data collection, case study method was used and for analysis Grounded Theory Method (GTM) was adopted. Major findings and a theoretical framework (presented in the next chapter) emerged from GTM's open, axial and selective coding phases. The core category that emerged from GTM analysis was individual heuristics. In other words it was learned from the empirical data that individual heuristics were found to play the most important role in getting SM achieved in SOs. In addition, actors, political processes and 3 Cs (communication, coordination and collaboration) were other important categories that were found to be of critical importance.

Figure 9: Summary of Methodology



CHAPTER IV

RESULTS

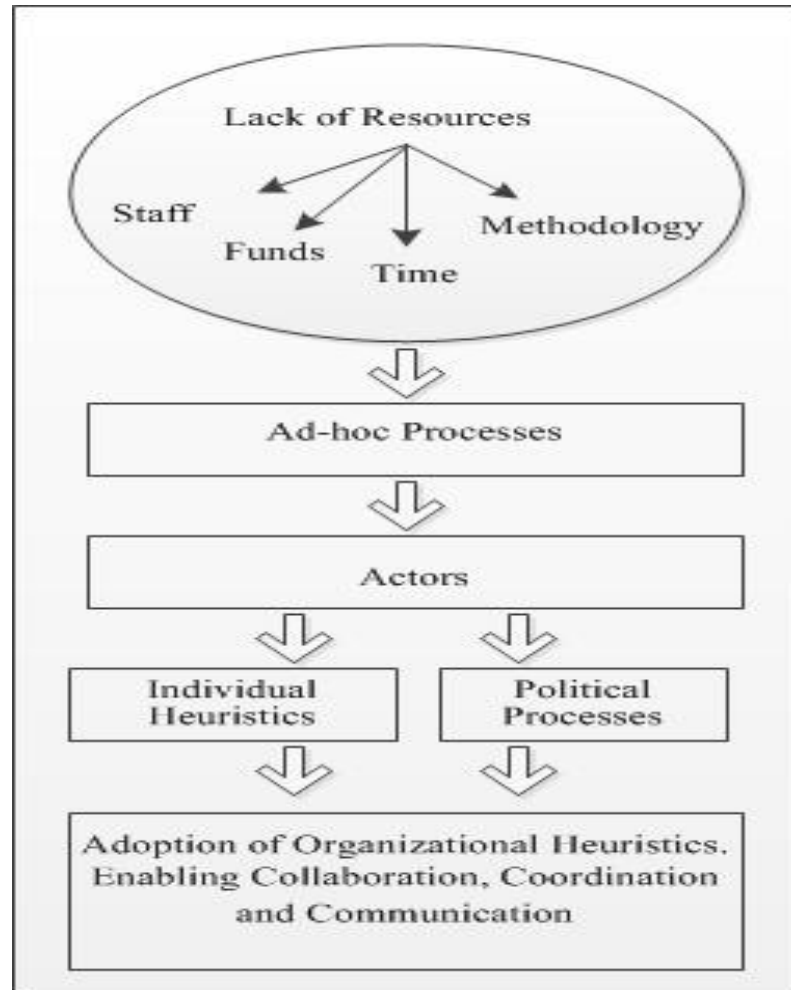
Main Points

Findings from empirical studies into the inner workings of SOs show that certain elements are critical for the success of SM:

- **Actors:** In the absence of systems, methodologies and resources, individuals (actors) play a vital role in SM.
- **Individual Heuristics:** These individuals rely on certain heuristics to get things done. These heuristics are derived from years of experience of working in the field. While using these heuristics, certain biases must be watched for as they could impact the outcomes.
- **Political Processes:** In order to implement their heuristics, individuals rely on political processes.
- **Organizational Heuristics and their Adoption:** Due to repeated usage, individuals' heuristics eventually get adopted by organizations and become organizational heuristics. In SOs' SM projects, these organizational heuristics enable collaboration, communication and coordination (3 Cs) which are salient for the success of SM projects.

As a result of analysis with the Grounded Theory Methodology, a theoretical framework (Figure 10) emerged that explains how SM gets done in SOs.

Figure 10: Theoretical Framework



As noted in the literature review (Chapter II), all SOs struggle with shortage of resources. SOs do not have enough staff, funds or time to allocate to their SM projects. They also do not have a standard methodology that can be readily allocated to software projects. As a result, for the most part projects are undertaken with an ad-hoc approach. As the data

was analyzed, it was clear that despite lack of resources and prevalence of ad-processes, there are some elements in play that help SOs achieve their objectives. The elements depicted in the theoretical framework (Figure 10) are:

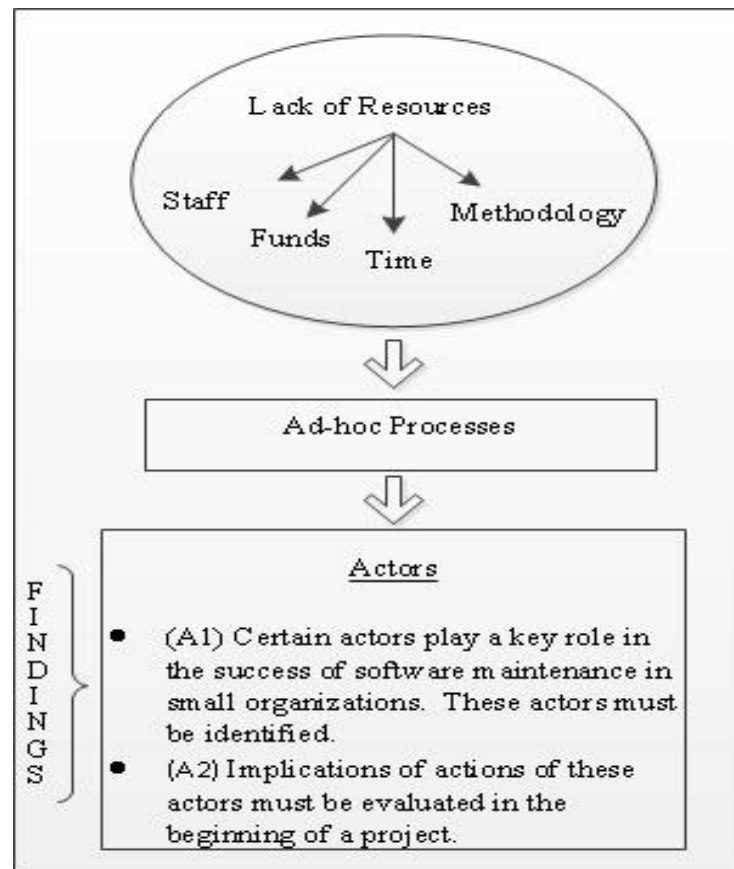
- Actors who play a vital role in SM
- Individual heuristics which are utilized by individuals to get things done in SM
- Political processes that are implemented in conjunction with heuristics to achieve objectives
- Organizational heuristics which originate from repeated usage of individuals' heuristics

This framework emerged as a result of a thorough analysis with Grounded Theory Methodology. Next, each of the components of the framework is presented in full detail along with its associated findings from empirical data analysis and theoretical sensitivity constructs.

Actors

Empirical data shows that actors played the most important part in the success of SM projects for SOs. The important role of individuals has been established in literature also. Pfeffer (1992) mentioned the heavy interdependence of individuals in SOs. Individuals' involvement in specifically getting SM moving in SOs has been reported in published work by the author before (Hasan and Chakraborty, 2011). There are two reasons given for over-reliance on individuals: shortage of resources at the discretion of SOs and paucity of methodologies/guidelines available to SOs.

Figure 11: Actors



This reliance is depicted in Figure 9 with two specific findings:

1. Certain types of actors play a key role in the success of SM in SOs. These actors must be identified.
2. Implications of actors' actions must be evaluated. In order to attain maximum benefit, this evaluation must be done at the very beginning of a project.

Next, details about each of these findings are presented in the following paragraphs.

(A1) Identification of Actors

Given the importance of individuals to SOs, analysis of data regarding existing individuals in SOs resulted in taxonomy of certain actors that are associated with SM projects. The taxonomy is presented in Table 12 and is followed by a brief discussion of different categories of rules found in this table.

Table 12: Taxonomy of Actors in Software Maintenance

Actors Categorization	Actor Type	Description
Users	Power-Active Users	Uses a software application for at least an hour daily. This user is highly motivated, possesses advanced knowledge of the application, and is highly interested in enhancements.
	Power-Passive Users	Uses a software application for least one hour daily and possess application knowledge similar to Power-Active Users. However, their motivation and interest in the application is low. They are apathetic to the related software project and are not willing to invest time and efforts in making it a success. Nonetheless, they demand to be kept well informed about the project. If ignored in communication, they make a lot of fuss.
	Occasional-Active Users	Uses a software application occasionally. This user is interested in giving time to the project; however, its members may not possess enough knowledge of the application.
	Occasional-Passive Users	Uses a software application occasionally and is apathetic to the enhancements project. Not interested in giving time. Vast majority of users is found in this group.
Developers	Active Developers	Developers of this type are involved in multiple projects, possess vast experience and expertise. 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's role.
	Passive Developers	Will start coding from the specs given to them. They are not interested in the reasons behind problems and specifications.
Analysts	Technical Analysts	Want to be more technical. Want to jump in coding. They can cause strife by interfering with developers work.
	Non-technical Analysts	Not interested in coding. Will focus on providing specifications only.
Project Managers	Process-oriented Project Mangers	Want to follow a Waterfall type phased approach for managing projects.
	Process-averse Project Mangers	Do not follow a process. Manage projects on an ad-hoc basis.

This taxonomy of actors lists four broad categories of actors that play a significant role in SM projects. These categories consist of “users” of a given software product, “analysts” who are involved in designing solutions in a particular application per the requirements of users, “developers” who develop a solution and “project managers” who coordinate all aspects of a given software project.

The taxonomy further shows particular types of actors in each of these categories. Users are considered “active” if they have a high interest and motivation in using a particular software application; users are considered “passive” if their interests and motivations are low. Furthermore, if users’ utilization of a particular software application is high, they are considered “power users”; if their usage is low, they are considered “occasional users.”

In the taxonomy, developers are categorized as “active” if they want to work beyond their prescribed duties. These are the types of developers who want to understand the whole business process and justification behind coding. Other types of developers are considered “passive” as they are interested in coding only. They do not want to be involved with other aspects of SM. Analysts can also be technical or non-technical. Non-technical analysts want to be involved in writing functional specifications only; whereas technical analysts are interested in coding also. Lastly the taxonomy lists process-oriented and process-averse project managers. Process-oriented project managers follow a set project management methodology whereas process-averse project managers rely on their soft and political skills to get things done.

(A2) Implications of Actors’ Actions

The taxonomy of actors presented in Table 12 offers two salient implications for SM efforts in SOs. Firstly, it reiterates the importance of individuals in achieving project success. Secondly, it offers unique implications into actors’ types, which can be utilized

to optimally staff a SM project. Some specific uses of the taxonomy (Table 12) are in the following paragraphs.

1. Those developers who are considered active developers would not be a good match to a SM project because the specifications would already have been developed for such a project. As noted in Table 12, Active Developers possess vast experience and expertise and want to be fully engaged in any project they are involved in. However, instead of working to the point, they want to fully understand the business reasons behind the project enhancements. They always have several ideas on how to solve given problems. Given these tendencies, they cause strife in projects by interfering with others roles (such as analyst's role); wasting time by repeating efforts (such as specifications). Thus a key implication from this section is to identify and understand various types of actors and utilize that knowledge to steer certain actors towards greater involvement in a SM project's success.
2. The knowledge gained from actors' taxonomy can be very helpful in addressing appropriate types of SM. As noted in the literature review chapters SM consists of four types: Adaptive, Perfective, Corrective, and Preventive. The knowledge of taxonomy of actors would enable management to undertake appropriate resource allocation strategies as needed for the appropriate SM type. For instance, a combination involving power-active users, passive developers, non-technical analyst

and either type of PMs could prove to be very effective in a project that is of “Perfective SM” type.

An explanation of this is as follows. A perfective SM project involves changes made to an application per users’ requirements. Since the application under consideration is already in use, Power-Active users would provide the best input as they would have the most functional and domain knowledge. Non-technical analysts and passive developers would be useful because skills from both are needed to achieve the job. On the other hand, if an active developer were to get involved, that would be detrimental to the project, as he would involve himself in both coding and the business requirements. That would make other team members feel that all specifications are covered; however, active developers tend to understand business requirements but do not document well. As such, a non-technical analyst would be best as she would capture the required specifications and document them for the developers to develop.

3. Another implication of using the taxonomy is that while the multi-dimensional skills of a technical analyst may be a useful asset in SOs, they do not automatically lead to project success. From our data, it was found that technical analysts have a tendency to shy away from documenting requirements and jumping into 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.

4. Hence keeping a passive developer and a non-technical analyst in a team would be helpful for a particular project which would not only clarify what needs to be done but would also reduce role interference and friction. It would increase accountability as both of these actors prefer to operate within the boundary of their roles. One analyst mentioned the following 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.”

In Table 13, some additional specific implications for using certain actor types are provided. Again, this information can be very useful to management while staffing SM projects.

Table 13: Implications of using Actors Taxonomy

Actor Type	Enablers/Inhibitors of Software Maintenance Projects	Implications for Small Organization
Power-Active Users	Enablers. Their willingness, knowledge and availability greatly assist the project.	This group should be identified early on in the project and brought on-board as soon as possible.
Power-Passive Users	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 Users	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 Users	They provide no 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.
Active Developers	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 Developers	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.
Technical Developers	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 Developers	Enablers. They are helpful to projects as they tend to be good at developing good specifications.	Encourage to keep abreast of technological changes as their technical knowledge may get rusty. Could be paired with active developers to enhance their technological knowledge.
Process-oriented Project Mangers	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 SOs do not have enough resources.
Process-averse Project Mangers	Enablers. They rely on experience and intuitions.	Encourage to learn project management methodologies.

Implications for small organizations

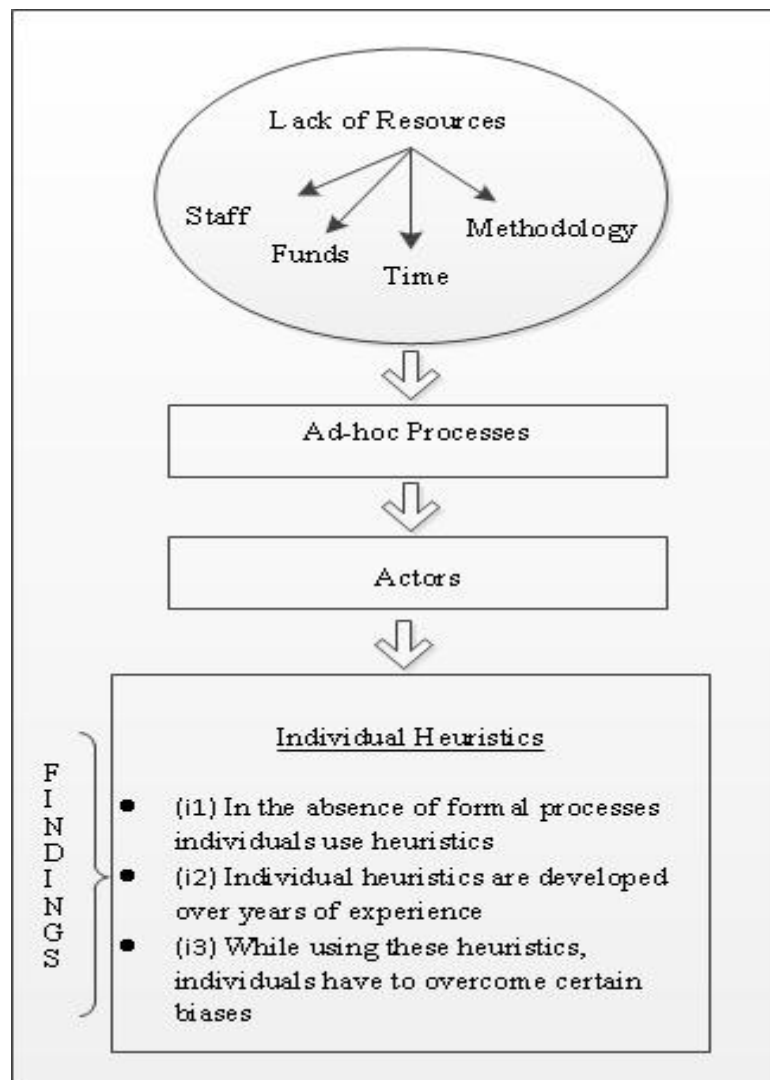
From this study, the following findings were found that can assist in understanding of other SM projects:

1. There are certain actors that played an important role in the success of SM projects. Taxonomy of such actors was presented in Table 12.
2. These actors were identified in the beginning of the project.
3. Knowledge of the taxonomy can be very helpful in staffing of the project and in assigning roles and responsibilities.

Individual Heuristics

The core finding from this study is to get jobs done in SM projects in SOs, individuals utilize certain heuristics. As shown in Figure 12, in SM projects, SOs face challenges of lack of resources - personnel, time, and funds (Pigoski, 1997). Lack of resources in-turn results in ad-hoc processes which results in extreme reliance on key individuals for the success of the project and the heuristics they have developed. Evidence was found that implied that in the absence of formal SM guidelines, the

Figure 12: Individual Heuristics



individuals in SOs have developed a pattern where they rely on their own rules gleaned from the experiences over the years (Hasan, Chakraborty, Dehlinger, 2011). These rules are developed from ad-hoc processes are unwritten guidelines which are used in the maintenance projects and represent heuristics that are followed in any new projects that are initiated. In other words, the vacuum created by formal guidelines is filled by individual heuristics. In this section, three findings are presented from the investigation of individuals' work in SM projects. Firstly, samples of individual heuristics are provided; secondly, means of developing individual heuristics are shown; thirdly, biases that could hinder these heuristics are presented.

(i1) Samples of Individual Heuristics

In the three SM projects (Portal Project, Financial Aid Project and Faculty Roles Project) that were investigated, this work found several samples of individual heuristics. These were rules utilized by staff to overcome shortages of resources or achieve success in SM. A listing of these heuristics is presented in Table 14. Following that is a discussion of the four categories of individual heuristics and after that a detailed explanation related to each rule from the empirical data is provided.

Table 14: Individual Heuristics' application to Projects

	Individual Heuristics	Portal	Financial Aid	Faculty Roles
1	Adopt non-formal methods to stay abreast of changes in the field.	✓	✓	✓
2	Be involved in all phases of a project. This ensures continuity between phases and ensures information is not getting lost in translation.	✓	✓	✓
3	Be prepared for future changes and projects.	✓	✓	✓
4	Identify the problem well.	✓	✓	✓
5	Consider ripple effects.	✓	✓	✓
6	In the absence of documentation, talk to senior staff to initiate projects on the right track.	✓	✓	
7	Make test plans with users prior to undertaking a SM project.	✓	✓	
8	Identify and involve power users.	✓	✓	✓
9	Maintain a personal documentation system.	✓	✓	
10	Prepare for meetings prior to attending.	✓	✓	
11	Use the tool "Find" in an editor for SM coding.		✓	✓

The heuristics presented in Table 14 can be classified into the following four categories:

Maintaining Skills

The first three rules deal with individuals' efforts in maintaining their knowledge base and skills. These are individual efforts to keep up-to-speed with projects that are beyond their skill sets.

Problem Understanding

Rules 4, 5 and 6 of Table 14 emphasize problem identification and ripple effects of fixes that are put in SM. This group emphasizes the need to look at the repercussions of actions.

User Involvement

Rules 7 and 8 of Table 14 pertain to the benefits of involving users in SM.

Tools

Rules 9, 10 and 11 of Table 14 suggest maintaining documentation, preparing for meetings and utilizing the “Find” tool in SM. These are tools used by software maintainers’ to achieve efficiency.

A brief explanation of each of the rules listed in Table 14 is presented in the following paragraphs. Such detail offer helpful implications for software maintainers in SOs.

1. **Maintain skills through self-reliance:** Rely less on formal training and more on yourself to stay abreast of the new technologies: In SOs, it is expected of staff to deliver solutions and perform maintenance. If they do not perform, they are replaced. Such reliance places extreme pressures on individuals to be abreast of relevant technologies and skills. For individuals working in SOs formal training are often thought to be not specific enough for the needs of their environment. In addition such trainings are frequently too costly to attend. Therefore it falls on the staff to do the learning themselves. A good way to learn in such circumstances is either on the job or by creating specifics exercises and projects to practice with and learn from. That requires extra effort but is well worth it. One analyst mentioned:

“Most formalized training is a waste of time for me. They cover only surface area and do not go deep into where we need to go. Thus it becomes a good refresher but not a good source of knowledge. The best way for me to learn is on the job and on the project. Nothing can compensate that.”

Thus the rule is adopting non-formal methods to stay abreast of changes in the field.

2. **Maintain skills by getting involved in all phases of a project:** In various interviews of the current case study, it was learned that staff members felt the need to be more involved in all phases of the project. This was felt to increase awareness as well as visibility of issues across the phases. This however is a tall order and may not always be a realistic possibility in SOs. However, some efforts should be made to place the persons (whether a developer or an analyst) in the same project from start to end in all phases. This ensures continuity and things do not get lost in translation. Again, this is important in organizations where documentation is lacking or methodologies do not exist. While this rule can be categorized as organizational, management finds it impossible to assign a person to all phases. On the other hand, it can be initiated by individuals themselves. Thus the rule of thumb is to use one's own initiative to be involved in all aspects (planning, analysis, and design) of a project. This way one can learn more and be self-sufficient.
3. **Maintain skills for upcoming project:** Be prepared for future projects. A common rule of thumb that emerged seemed to be the need to be inquisitive and informed of the new projects on the horizon. This was deemed to be the best way to get prepared for projects that one may end up working on. One way to be

prepared is to get as much information on one's own as possible. One developer stated:

“I often learn about new projects over the grapevine. We don't have formal methods of knowing about upcoming projects unless of course our manager inform us – which they do when they are sure it is a go.”

Thus, the rule is to keep eyes and ears open for new projects. Once knowledge is gained of an upcoming project, it seems advisable to start acquiring appropriate knowledge, background and training that are available.

4. **Understand problems by spending ample time on them:** When time allows one should spend more time on identification of the problem otherwise one is susceptible to wasting time. One developer stated:

“Sometimes I have spent days trying to figure out a problem and then I make one line of code change and that takes care of the takes care of it.”

Thus the rule is to give good amount of time to identification of the problem.

5. **Understand problems by knowing its ripple effects:** SM change should be considered very carefully and all its effects should be fully identified prior to implementing it. As one developer said:

“Researching a problem is very important and also I need to identify if it's going to fix one problem but break 10 others.”

Thus, the rule is prior to implementation of any change, consider its ripple effects.

6. **Understand problems by talking to experience staff:** While confronting a SM project/task, it is best to find out if the organization has worked on it before.

Unfortunately, SOs do not have good documentation to refer to. So the next best reservoir for institutional memory is senior staff. Therefore every project should

be vetted through senior staff who can lend their experience and quick tips. One of the developers stated:

“Often I would turn my wheels around until I find a solution. It’s best that I get some direction from a senior, experienced, staff member who can put me on the right track. Since we are short on time, such direction is invaluable to me and my organization.”

Thus the rule is for every project, get feedback from experienced staff.

7. **Involve users in testing:** In SOs testers should be involved from the beginning.

When possible test cases should be created prior to changing the code. That way SM will be done per users’ requirements and will have fewer bugs. Thus, the rule is making test plans with users prior to undertaking a SM project.

8. **Involve power users:** Power users are highly motivated individuals and have the highest interest in the success of their specific application. It is best to work with them. They typically do not waste the time of IT resource and often go out of the way to help the project. Thus the rule is to identify and involve power users.

9. **Maintain a personal system of documentation:** In the absence of a central documentation system, one needs to rely on one’s own system. This is especially needed in SOs where maintaining documentation is always a challenge and often non-existent. Therefore one has to have some sort of personal system to keep track of things. It is best to give it thought and consideration so that project information is readily available when needed. One analyst mentioned:

“I have to maintain my own information to be able to maintain my job.”

Thus, the rule is to maintain personal documentation system.

10. **Be prepared for meetings:** Before attending a meeting, doing homework is highly recommended. That allows one to gain the most from meetings, contribute to them and also not be bored during the meetings. One developer said:

“Before going to a meeting, I spend at least fifteen minutes preparing for it, that keeps me in tuned during the meeting; otherwise, it is hard to keep up with stuff”

Thus, the rule is to be prepared for meeting prior to attending.

11. **Use Find:** It was learned in this case study that the most commonly used tool in maintaining systems is the “Find” tool in code editors. It allows one to find relevant matter quickly. One developer stated:

“While I am asked to work on a software maintenance project, the first thing I do is identify the relevant material by the Find feature of a code editor.”

Instead of going through the whole program and understanding it, “Find” is used to get to the relevant information. In most cases documentation is non-existent in SOs therefore developers have to reverse engineer to get to the right location.

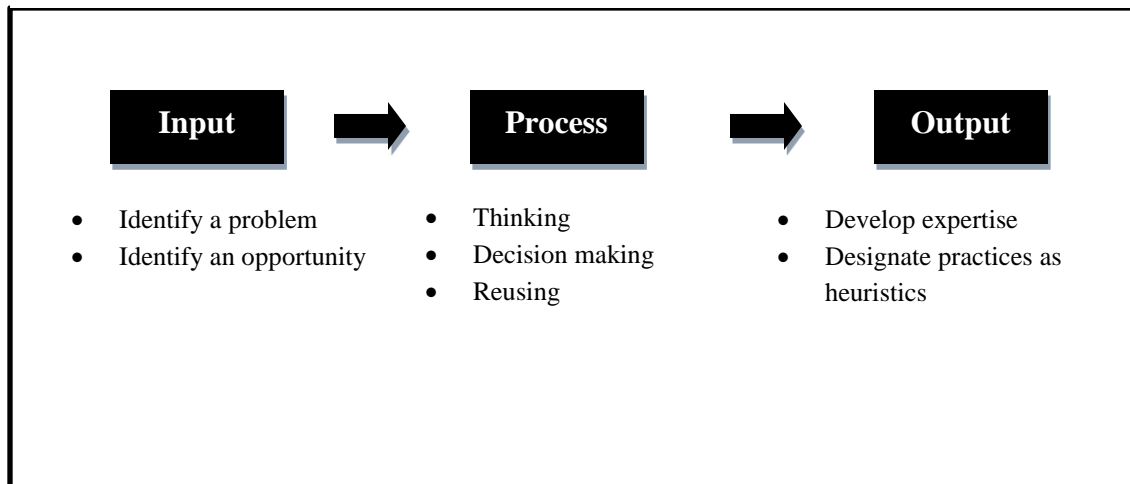
Thus the rule in SM coding is to use a “Find” tool.

(i2) Methods of Developing Individuals Heuristics

The previous section provided ample details on individual heuristics that were found to be in practice in SOs. How these heuristics are formed is explained in this section. From data review and analysis, this work found that these heuristics are formed over a process of formation that begins with an individual looking at a problem or an opportunity, applying certain processing methods, and generating results. If the results are good then

the same rules/heuristics are repeated. With repetition, they become individual heuristics. This process is called the Individual Heuristic Formation Model and is presented in Figure 13:

Figure 13: Individual Heuristics Formation Model



The three-phased (Input, Process and Output) approach to individual's Heuristic Formation Model is strikingly similar to how computers operate. Similarities of computers to humans in processing information has been underscored in the cognitive revolution of the 1960s when computer programs were created to mimic human actions such as arithmetic and chess playing. Newell and Simon (1972) categorized humans and computers as both "information processing systems." Hastie and Dawes (2001) stated that many aspects of human thinking including judgment and decision making can be produced in computational models. As noted in Table 15, both humans and computers deal with the three phases in the same way. Inputs are communicated through external stimuli and are sent in for processing via sensory media such as sight, sound, smell, taste and touch. Received data is then processed in temporary memory (similar to computer's

Random Access Memory - RAM) and then is sent to the brain of a human or a Central Processing Unit of a computer. Brain / CPU is relied upon to glean established rules and syntax from information stored in long term memory (hard drive and ROM). Ultimately both entities make a decision and communicate it through an output medium. Such information processing is depicted in Table 15:

Table 15: Humans and Computers Processing Information

Information Processes	Humans	Computers
Input	Use external stimuli and five senses (Smell, Sound, See, Taste, Touch)	<ul style="list-style-type: none"> • Keyboard • Mouse
Process	Use Temporary Memory	RAM
	Use Permanent Memory	Hard Drive ROM
	Brain makes decision	CPU
Output	Communicate Decisions	<ul style="list-style-type: none"> • Printout • Monitor

From this study, heuristic formation occurs over three phases of input, process and output. Some of the details related to these phases are presented as bullets in Figure 10. Whereas descriptive details related to these three phases are provided.

1. Input

This is the first phase of Individual Heuristic Formation Model. It begins when an individual working for a small organization identifies a problem or recognizes and opportunity for an enhancement. In one of the interviews, when asked how he begins solving a problem, an analyst responded:

“the trigger is always either a problem that needs to be solved or an enhancement or an opportunity that others are availing that we are not. That gets us thinking about a solution”

Thus, a problem or an opportunity starts the ball rolling.

2. Process

This is the second phase of Individual Heuristic Formation Model. This phase occurs when the problem or the opportunity is contemplated upon. This phase includes three components: Thinking, decision making and reusing information. Each of these components is explained in the following paragraphs.

Thinking: The first component of the “Process” phase of the Individual Heuristic Formation Model is “thinking.” Thinking is a vast field by itself and has spawned vast amount of research in Psychology and Cognitive Psychology. Several theoretical reasons have been given regarding what takes place in thinking. Provided here is a list of three explanations of thinking from theoretical models. These are selected based on what was found in case data. One of the managers explained how he thinks about a given SM project:

“I do my thinking by relating new problems to old problems, filling needs of customers and never forgetting the big picture”

From the quote above, three concepts are deduced that are connected with the three theoretical constructs to explain how individuals process the input. The three concepts are:

- Relating ideas
- Filling needs
- Looking at the big picture

This analysis enables us to understand how individuals think – the process of which eventually leads to heuristic formation. The three concepts can be related to

- The view of thinking by Associationists
- The view opposing to Associationists
- The view by Gestalt theorists (Mayer, 1983)

According to Associationists, which goes back to Aristotle, thinking takes place in associations. Thought process follows chains of events and objects that are similar, fall in the same time, or are found in the same space. Hobbes and Locke, who are considered Associationists, explained unit of thinking as an association between two ideas (Humphrey, 1963). They stated that all human knowledge consists of either ideas or associations between ideas.

There are some scientists who oppose this point-of-view. For example, Otto Selz stated that thinking does not represent linkages; rather they exhibit a process of filling gaps in a structural complex knowledge (Hastie and Dawes, 2001).

Yet a third school of thought, represented by Gestalt psychologists, explained thinking as a search process that relates one aspect of a problem situation to another, which results in a structural understanding. Thus “structural understanding,” “overall insight,” or “high-level understanding” of looking at all parts together to satisfy a goal (solve a problem) is what thinking is all about, according to (Mayer, 1983). This study found all of these three views of thinking in use by individuals while SM efforts.

Decision making: The second component of the “Process” phase of the ‘Individual Heuristic Formation Model’ is ‘Decision Making.’ When asked an analyst, how he goes about solving problems, here is what he said:

“ to make a decision, we rely on previous experience of something that has worked before, or we go to someone else who has worked on a similar problem to pick their brains and then we explore doing what they did and weighing their consequences.”

This implies that heuristic formation involves decision making. Hastie and Dawes explained how decision making takes place. Decisions are needed if there is more than one choice available; each choice gets projected to take to certain outcome; and each outcome has consequences that can be assessed based on personal values and goals. Also, people make decisions based on habits, conformity to what others are doing, or based on religion/cultural/authority mandate. The process of decision making can be either rational or due to unconscious dominance or bad behavior due to a childhood trauma as explained in the Psychoanalytic Theory by Freud(Hastie and Dawes, 2001).

Similar motivations for decision making were found in this study. Mostly, individuals began solving a problem or working on an opportunity and then decided among various choices of actions. In most instances, those choices were selected that had worked before in generating certain desired outcomes. So experience of positive results played a big role in choice selection. Intelligent problem solving depends on selective rather than rapid behavior. How experts make selections or make decisions is an area explored by two authors. For example, Gary Klein who has done extensive research on expert intuition in decision making and studied an area known as Naturalistic Decision Making. Another is Daniel Kahneman who has spent much of his career running experiments in which intuitive judgment was commonly found to be flawed; he is identified with the “heuristics and biases” approach to the field of decisions made by experts (Kahneman and Klein, 2009).

Reusing: The third component of the “Process” phase of the Individual Heuristic Formation Model is “Reuse of information.” While there is extensive literature available on reuse of information in software engineering, their focus has been on technical and

organizational factors, largely ignoring cognitive characteristics of individuals. Despite anecdotal evidence that cognitive heuristics play a role in successful artifact reuse, few empirical studies have explored this relationship (Parsons, 2004). Webb and MacMillian (1995) point to the potential impact of certain cognitive biases in software engineering and note a lack of empirical research in this area. Nonetheless, reuse is in full action in SOs, as found in this study. This study found people frequently reused information. When people observe the actions of others and then make the same choice, regardless of their own information, information cascade occurs (Bikhchandani, Hirshleifer & Welch, 1992).

3. Output

This is the third phase of Individual Heuristic Formation Model. This phase relates to how expertise gets developed and how repeated decisions eventually become individual heuristics and get utilized. As noted in the Individual Heuristics Formation Model (Figure 11), the output of heuristics gets developed by initially going through two phases of Input and Process. In other words, the process begins by an individual looking at a problem or an opportunity, applying certain processing methods (thinking, decision making and reusing, and finally generating outputs (heuristics). If the results from applying heuristics are good then they are repeated.

A key finding from data analysis is heuristics are closely related to expertise. It was found in the study sites, the main sources of individual heuristics are individuals who have been working with an organization for a long time and have developed certain expertise that has worked for them repeatedly. On developer stated:

“I know what to do in a given situation because I have certain levels of expertise that I have developed over years of experience, training and trial and errors.”

This expertise gives them a kind of a professional intuition. As noted before, however, expertise is more than intuitions though; it reflects acquired knowledge, experience and skills (Ericsson and Smith, 1991; Hunt, 2006; Yates and Tschirhart, 2006). Furthermore, heuristics acquisition or development depends on the approximate number of years in deliberate practice (experience) to attain a high level of expertise (Chase & Simon, 1973). In their research on expert decision making by chess masters, it was noted that chess masters were able to quickly identify best moves, whereas mediocre players did not even consider best moves. Chess masters do this due to their repertoire of 50k to 100k patterns that they recognize immediately. Strong players need a decade to collect such repertoire (DeGrott, 1978; Chase & Simon, 1973).

Yet, another element that distinguishes experts and lead to heuristics is the speed of decision making. Experts are able to rapidly make good decisions due to heuristics. Heuristics are principles or devices that contribute to reduction of search in problem-solving activities (Newell, Shaw and Simon, 1958). Those practices and rules employed by individuals that succeed get reused by individuals and become individual heuristics.

(i3) Biases

One of the findings from the empirical study is the ways in which experts make decisions is not straight forward. Experts make decisions for problems some of which are structured and some are not. Furthermore, outcomes of the decisions, even though for the most part are positive, are not guaranteed to generate success. One reason for this is in a problem situation we are most likely dealing with partial information. This has been established by Fred Tonge (1961) in his initial research on artificial intelligence. He stated while making decisions, “we are not sure if that information would be enough to generate a good solution or any even any solution.”

Kahneman and Klien (2009) concluded that professional intuition is sometimes marvelous and sometimes flawed. The evaluation of the quality of intuitive judgment depends on predictability of the environment and individual's ability to learn the regularities of the environment. In other words, expert's intuition depends on prediction and learning abilities. These are the characteristics found in experienced staff who utilized heuristics in this work. The main implication that should be noted in this section and that is presented as a precautionary note to the adoption of the overall theoretical model is the following: While heuristics seem to be a useful mechanism for small organizations, there are certain caveats to their use that need to be noted.

Heuristics typically represent wisdom accumulated from experience. Though the experience is real and the lessons captured are often valuable, different experiences easily can lead to contradictory heuristics and guidelines. For instance, there are software engineers who firmly believe that C++ and Ada (statically typed languages) are too inflexible for serious system development. On the other hand, there are those who believe

that the error-prevention properties of such statically typed languages are required for serious system development (Webb and Macmillian, 1995). These types of perceptions and biases in formulation of heuristics are needed to be investigated in the context of SM. For this to be done it is perhaps important to understand how heuristics are formulated within individuals. Kahneman and Tversky (1973, p.273) noted that "In making predictions and judgments under uncertainty, people do not appear to follow the calculus of chance or the statistical theory of prediction. Instead, they rely on a limited number of heuristics which sometimes yield reasonable judgments and sometimes lead to severe and systematic errors." They then defined three cognitive heuristics and biases that can emerge from them: representativeness, availability, and anchoring-and-adjustment.

Representativeness refers to making an uncertainty judgment on the basis of "the degree to which it is: (i) similar in essential properties to its parent population; and, (ii) reflects the salient features of the process by which it is generated". Representativeness skew judgments based on engineers' experience towards the salient characteristics of their own views (Webb and Macmillian, 1995). Availability is used to estimate "frequency or probability by the ease with which instances or associations come to mind".

Availability has been reported to be influenced by imaginability, familiarity, and vividness, and has been supported by evidence of stereotypic and scenario thinking (Kahneman and Tversky, 1979).

Anchoring-and-adjustment involves "starting from an initial value that is adjusted to yield the final answer. The initial value, or starting point, may be suggested by the

formulation of the problem, or it may be the result of a partial computation. In either case, adjustments are typically insufficient (Kahneman & Tversky, 1979). In other words, when people are given a problem for which there exists a correct solution and an initial estimate of its solution, they tend to provide final estimates close to the initial estimate when asked to solve the problem (Parsons and Saunders, 2004). This is termed anchoring. The anchoring heuristic helps humans simplify problem solving in a complex situation without conscious effort. However, since they are not aware they are using the heuristic, people often fail to make adequate modifications to an initial solution and produce estimates that are severely flawed with respect to the correct solution. This is termed an adjustment bias (Shanteau, 1989).

This discussion shows that while heuristics could be potentially useful tools for software maintainers within SOs, they are susceptible to cognitive biases in their formulation and use. These biases can affect the way in which SM artifacts are reused, potentially impeding or enhancing the successful reuse of code and design. It is important therefore to understand how heuristics are formulated in the SM context and categorize them to facilitate further in depth study. Kahneman and Tversky's (1973) taxonomy of cognitive heuristics and biases perhaps form a good foundation for such investigation.

Strategies against Biases

It is also perhaps critical to try and understand how the biases in heuristic formulation can be compensated. Specifically there is a need to investigate what can be done to reverse biases such as representation, availability and a preference for

confirmatory evidence? Webb and Macmillian (1995) provide four recommendations that can alleviate these biases.

1. There is a need to frame problems to signify relevant information that might be otherwise left out.
2. Use specific education/training to raise awareness of common biases. In general, cognitive biases do not disappear just because people know about them. It is probably not sufficient simply to make software engineers aware of the possibility of bias.
3. Biases may be eliminated if heuristics were formulated as a result of empirical investigation as opposed to intuition. It is constructive simply to know that intuition is likely to be flawed. Software engineers should understand that their impressions of an application may be biased, and should undertake empirical investigation whenever possible. If they need to know how often a certain idiom occurs in code, it will often be wise to use an automated tool to find out. If they need to know where an application is spending most of its time, profiling almost always pays off. Though it may not be possible to compensate for cognitive biases, it is possible to be aware of their existence.
4. The final recommendation is to seek non-confirmatory information when applying heuristics. For example when testing and debugging the application, engineers should actively seek non-confirmatory information. This would enable individuals to continuously evaluate the applicability of the adopted heuristics to the context of use.

In conclusion, an important finding of our research is the identification of the fact that SOs often resort to heuristics to compensate for the lack of formal processes. However, while heuristics represent a viable mean to structure SM operations within an organization, the discussion above indicates that there are certain problems associated with their use and formulation and they should therefore be used with caution.

Implications for small organizations

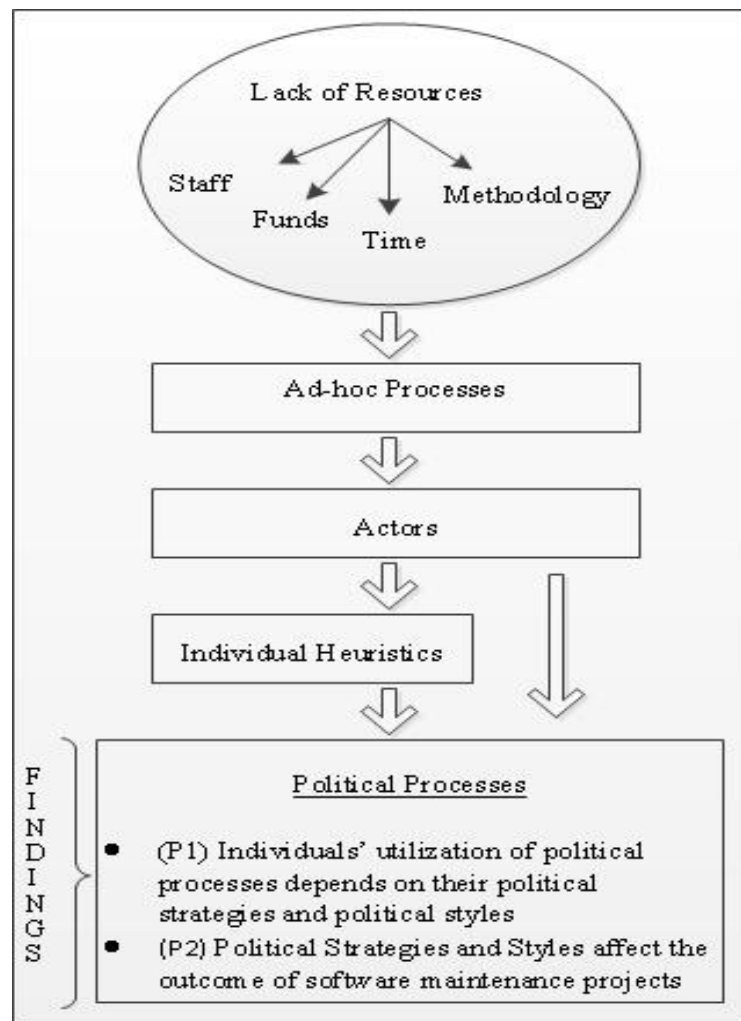
From this study, the following findings were found that can assist in understanding of other SM projects:

1. There was a huge reliance on individuals to lead SM projects to success.
2. These individuals exhibited certain expertise and applied certain rules (heuristics).
3. These experts possessed certain experience, skills, acquired knowledge and intuition.
4. With repeated use, the rules employed by experts became heuristics.
5. Certain biases must be watched for while these heuristics are implemented.

Political Processes

As presented above, in SOs, SM processes are guided by actors and their individual heuristics. Analysis of data exhibits that in carrying out their responsibilities, actors utilize certain political processes. As shown in Figure 14, our analyses of collected data exhibit a heavy presence of political maneuvers and their constant impact on outcomes. Consequences of actors' political behavior as both positive and negative impacts have been noted by researchers (Madison, 1980). On the positive side, it can be

Figure 14: Political Processes



useful to organizations in reaching its goals, coordinating staff, developing esprit de corps, and making decisions. In our case study, political processes clearly enabled the successful project. One analyst noted the following positive impact.

“Politics can be very useful to a small group such as ours. As we are always strapped for resources and depend on each other, conflicts arise over mundane (political) issues. In this project, we faced many hurdles, but senior management got involved in the right time to fix things. Cooler heads prevailed.”

He further added an example of a situation where political strife was found and project had to be put on a positive track:

“At one point, in this project, one of our main technical resources was not working properly. He was using stalling tactics, delaying work, not sharing knowledge and postponing tasks. Don’t know what was going on with him. He was just not performing. Thankfully the project manager involved senior management in the right time; who took corrective action and put the project back on track.”

Political behavior can have negative influences as well. Madison (1980) noted that it can result in inappropriate use of scarce resources; can produce divisiveness, tension and stressful environment; can allow less fully qualified persons to advance in ranks; can reduce communication flow; and can eventually damage the image and reputation of organizations. One of the reasons for the failed project in our case study was politicking. It did not go beyond the design phase due to the political disagreements between the key actors. One manager noted:

“The main reason for the failure of this project is because the functional directors did not agree on the direction of the project. One director wanted the solution to be designed her way. However, the other director, who was more involved with the desired functionality of the project, saw an infringement upon her authority and blocked that initiative. That took it to a stand-still situation.”

These findings show the impact of political behavior in general. In addition, specific instances of political strategies and styles were found in our case study. Below details regarding findings associated with utilizing political strategies (Empire Building, Tug of War and Obstacle Race) and political styles (Shotgun, Ingratiator, Tactician and Bystander) is provided.

(P1) Actors' Political Strategies

An actor's political strategies play a significant role in influencing SM projects within a small organization. This work found that application of Empire Building was quite high in the failed project. The project was considered a failure because it did not get completed. One of the analysts highlighted the empire building strategy in the following way (as a tussle between two main user groups):

“Okay so the second group is not going to budge. So why didn't the first group retreat and let the project succeed. The first group could have agreed to demands of the second group and that would have brought some level of success. But, the first group did not leave its ground even in one inch. Both parties were like at war would each other would not let go of even one inch.”

While Empire Building can be a cause of failure, it can also be a source of inspiration and motivation to dive into new and uncharted territories. This phenomenon was noted in the behavior of a new functional executive responsible for the successful project's department. A manager noted:

“The new executive saw the automation project as a way to solidify and extend his domain”

Application of the second political strategy, Tug of War, was high in the failed project. One analyst mentioned the following about two functional department heads that were maneuvering politically to get more control over the project:

“This project suffered because both department heads wanted to have more control and say in the way the project proceeds. Both heads wanted to adopt a solution that would involve minimal work for their departments.”

Tug of War tendencies were found in the successful project also, but it did not have much impact as the inclinations were moderate. Its tendencies were found among the technical team. One of the developers mentioned the following about another developer:

“The other developer wanted to not get anyone to learn web services technology usage, this way his control will be intact.”

Such inclinations in technical teams could have negative and/or positive implications. On a positive side, it shows competition among developers to learn new technologies. On the negative side, it entails territorialism and extending control which could keep others from gaining technical know-how. This can be a risk for the team and should be mitigated by management through cross-training and rolling responsibilities.

Application of the third political strategy, obstacle race, was found to be quite low in the failing project. This implies fewer efforts were undertaken to overcome the resistances and political obstacles. An analyst noted:

“No efforts were made to overcome the main obstacle of resolving the differences between the two departments’ heads. Each department’s head thought that a concession on technology selection would mean a defeat. Someone should have resolved these differences.”

In the successful project, obstacle race was found as well but it would get immediate treatment whenever found. One of the technical managers mentioned the following remedy for overcoming criticisms:

“We faced users’ ridicule from the functional team when my team did not correct errors in time. Overcoming their mockery and negative propaganda was a challenge for me but I totally ignored them and doggedly kept at fixing errors until they were all resolved. ”

Table 16 provides a summary of findings regarding usage of political strategies in both projects. Depending on involvement, each actor is assigned a grade of low, medium or high is toward Sabherwal and Grover’s political strategies. This work found that individual’s political strategies are directly proportional to the success of the project.

Table 16: Political Strategies in Projects

Type of Political Process	Successful Project	Failed Project
Empire Building	Medium	High
Tug of War	Medium	High
Obstacle Race	High	Low

(P2) Actors’ Political Styles

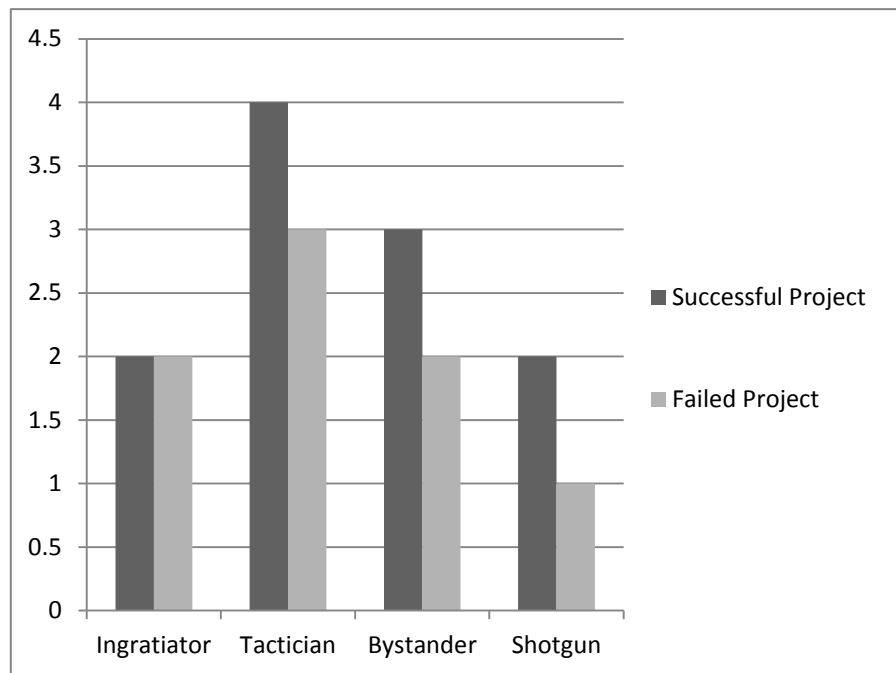
When the specific political styles (Kipnis and Schmidt, 1988) of individuals were applied to project data, it was clear that the actors’ political styles played a vital role in the success or failure outcomes of projects. In both projects, this work found that actors had different political styles. The most common political style type in each project was that of a “tactician.” Projects, however, differed on the second style.

In the successful project, the adoption of the Bystander political style, by some key actors in a senior role made a big difference. This style implies usage of minimal interference and political tactics. In other words, management was not getting in the way. One manager described it as the following:

“We were given a lot of autonomy in this project. This was a complex project that had impact on many offices and departments. We (active actors in the project) had to make many spur of the moment decisions based on our gut intuitions. Many a times the tempers went through the roof in the war room (big conference room where everyone sat together to work on the project). Executives were aware of these tensions and skirmishes, but they did not jump in or show off their authorities.”

In the failed project, there were more “ingratiators” amongst actors (Figure15) playing key and influential role (a functional director and a key developer). These actors were more concerned with their impression management (self-image) which could have contributed to the failure. As noted in the quote from a participant in this project, certain individuals (from management) did not take a stand at the right moment:

“If that particular executive had taken a stronger position, this project wouldn’t have failed. His approach was – don’t rock the boat.”

Figure 15: Political Styles in Both Projects

In comparison while there were instances of actors adopting an ingratiatory style in the successful project, these individuals were not as influential within the project. This seems to indicate that this style can have a detrimental effect on the project if adopted by key and critical resources with decision making capabilities.

Table 17: Successful Project

Actors	Political Style
Developer	Ingratiator
Analyst #1	Tactician
Analyst #2	Bystander
I.S. Manager	Shotgun
I.S. Director	Tactician
I.S. Executive	Tactician
Functional Manager#1	Shotgun
Functional Managers#2	Ingratiator
Functional Executive#1	Tactician
Functional Executive#2	Bystander
Functional Users (30)	Bystander

Table 18: Failed Project

Actors	Political Style
Developer	Ingratiator
Analyst	Tactician
I.S. Executive	Tactician
Functional Director#1	Shotgun
Functional Director#2	Ingratiator
Functional Executive#1	Tactician
Functional Executive#2	Bystander
Functional Users (5)	Bystander

Tables 17 and 18 show a comparative analysis of political styles adopted by actors for the two projects. From the data analyses in Tables 17 and 18, this work found political strategies playing out in the following ways.

Implications for small organizations

From this study, the following findings were found that can assist in understanding of other SM projects:

1. Political processes were found to enable or inhibit project's success.
2. Individuals employed political certain specific political strategies and styles.
3. An increased utilization of political strategy of Empire Building was found to have contributed to the failure of a SM project.
4. An increased utilization of the political strategy of Tug of War led to the failure of a SM project.
5. An increased utilization of the political strategy obstacle race led to the success of a SM project.

6. Adoption of the political style of an ingratiation by certain actors led to the failure of a SM project.

Organizational Heuristics and Their Adoption

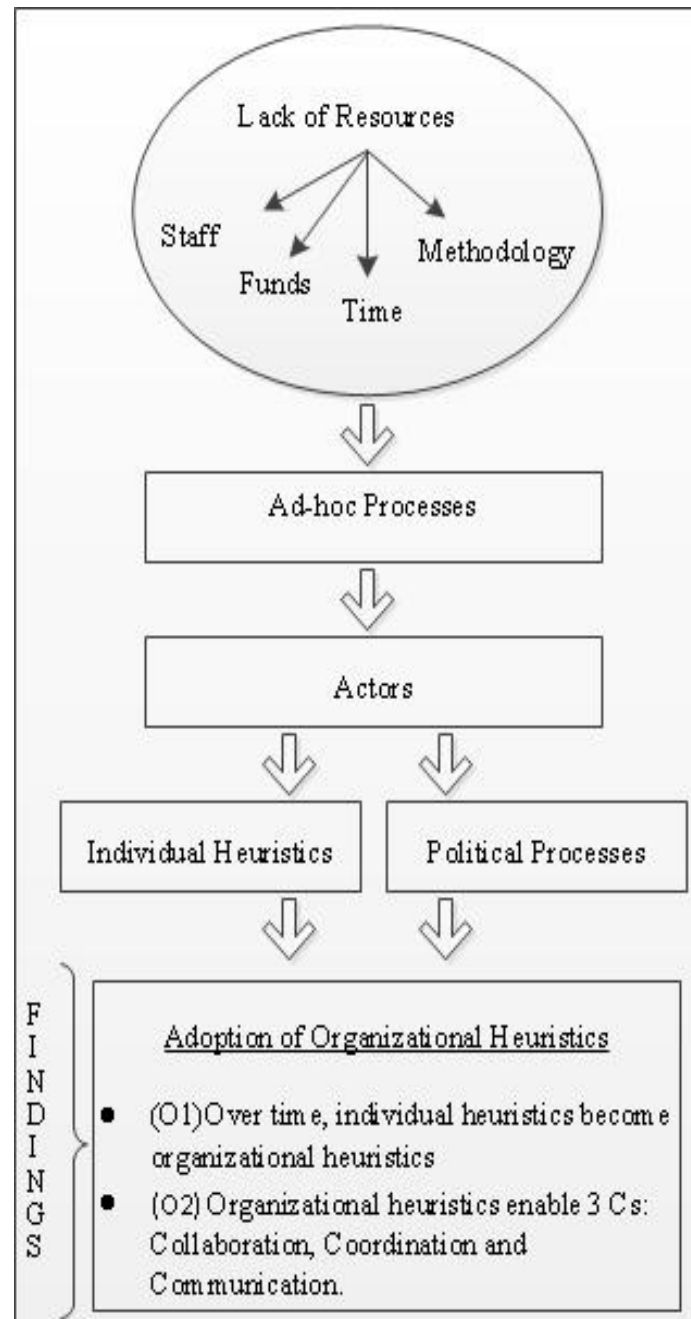
After repeated and successful usage of individuals' heuristics and political processes, they get adopted by organizations and become organizational heuristics. Organizational heuristics are similar to individual heuristics. However, the difference is organizational heuristics have a broader span and are followed across the organization. While they are not propagated officially and cannot be found in policy or procedure manuals, they get themselves adopted through word-of-mouth. In this study, it was found that such organizational heuristics play a vital role in getting work completed. They enable communication, coordination and collaboration which are found to be very important for SOs SM efforts. As depicted in Figure 16, key findings in this area are recognition of organizational heuristics and important role of 3Cs. Details related to these findings are presented in the following paragraphs.

(O1) Samples of Organizational Heuristics

From this study, various organizational heuristics were found that can be utilized to achieve success in SM efforts in SOs. A summary of these are presented in Table 19. There were three broad categories of organizational heuristics found:

Staff development: The first five rules listed in Table 19 deal with overcoming challenges of maintaining skills and staff development. These are important as SOs, despite being short on resources, have to be prepared for looming projects and must deliver solutions

Figure 16: Organizational Heuristics



1. **Resource augmentation:** Rules 6, 7, and 8 in Table 19 deal with strategies of harnessing additional resources when needed.

2. **Coordination strategies:** Rules 9, 10 and 11 in Table 19 suggest ways in which office arrangements, hiring and meetings can be utilized to better assist SOs in SM.

Table 19: Organizational Heuristics

	Organizational Heuristics	Portal	Financial Aid	Faculty Roles
1	If staff is missing key skills, provide formalized training	✓		
2	If a project needs a backup person and junior staff is available, pair with senior member for cross training	✓	✓	
3	If time allows and topics are related to project's subject matter, send your staff to local conferences	✓	✓	✓
4	Send relevant information to staff and let them decide if webinars are worth their time	✓	✓	✓
5	Involve senior staff in all projects – especially in the design phase	✓	✓	✓
6	Use outside consultants for projects that are high profile and are needed in a specified time period	✓	✓	
7	For mission critical needs, acquire formal product support	✓		
8	When confronting a technical problem search first for a solution on Google first	✓	✓	✓
9	In hiring, look for individuals with entrepreneurial background and skills	✓	✓	✓
10	Enable office arrangement that allows ease in communication and quick mutual support. Cubicles are better than walled offices	✓	✓	✓
11	Utilize meetings to manage individuals and get tasks done	✓	✓	

A brief explanation of each of these rules is presented in the following paragraphs.

1. **Staff development through formal training:** Due to financial constraints, SOs do not have a formal training plan and budget for its staff. They still, however, have to stay abreast of cutting edge technologies and for that purpose sometimes send staff for formal training. The cost for sending one person to such training runs into thousands of dollars. The rule for selecting the type of training and the staff member to send it to is a management prerogative. Often, the trigger for such training is an impending large project that requires certain expertise that is lacking in-house. Thus, the rule is if a high profile project is looming and certain skills are missing, adopt formalized training.
2. **Staff development through cross training:** SOs often have varying skilled individuals and one method of bringing low skilled persons up to par is by pairing them with experienced persons. One problem with this approach is it takes away time from experienced staff. One senior developer stated:

“I have a lot on my plate; it’s not easy for me to get my work done. If I am expected to train juniors, that eats into my time. It gets frustrating, especially if the person on the receiving end is not very motivated.”

Nonetheless, this is still an excellent method of bringing up-to-speed those with less experience. Thus, the rule is if a project needs a backup person and junior staff is available, pair with senior members for cross training.

3. **Staff development by attending conferences:** Due to exorbitant costs, frequent formal training is often out of the reach of SOs. Therefore, conferences especially local ones provide an excellent mean of staying current. Also, it allows

networking with peers in the field. It is expected and hoped that participants would share ideas, codes and designs after the conferences. Thus, the rule is if time allows send your staff to local conferences

4. **Staff development through webinars:** Webinars have become a major resource for technical knowledge. Management of SOs actively promotes these to their staff. However, finding and recommending the right webinar is time consuming which is a bottleneck for SOs. Thus, the rule is to send webinar information to staff and let them decide if it is worth their time attending these webinars.
5. **Staff development through senior staff:** This allows the organizations to pick their brains, and make use of the heuristics developed by such individuals who have accumulated wisdom through years of experience. It prevents “re-inventing of the wheel.” One developer mentioned:

“Must involve senior and experienced staff in the beginning of the project. They are like the brain of the system that tells the other parts what to do. They tell you where to start. Without them, you will be starting from scratch and may be are on a wrong path.”

Unfortunately, SOs have a handful of such senior staff members. Organizational heuristics is therefore to involve such people in more design work and less low-level work. Low-level coding is typically assigned to junior staff. One caveat with such approach however is to keep senior staff involved in coding also so they do not lose their skills and can continue to provide good guidance. One senior developer stated:

“I want to be involved in coding otherwise this stuff disappears from brain.”

Involving senior staff is important for yet another reason: SOs often do not have good documentation systems in place; as such senior staff is the only ones who have institutional memory. Thus, the rule is to involve senior staff in all projects, especially in the design phase.

6. **Resource augmentation by involving consultants:** It is best to involve consultants when the project has a very high profile and needs to be completed in a specified time period. In the current site, it was found that when consultants are involved in a project, a sense of urgency comes into play. All organizational resources move at a more rapid pace to get such projects completed. One reason for this is consultants are expensive and for a small organization money is limited and important. One developer stated:

“Everyone knows that consultants charge by the hour and we better provide them what they need to get the job done. They bring focus.”

Thus, the rule is to bring consultants in for projects that are high profile and are needed in a specified time period.

7. **Resource augmentation through formal support:** Oftentimes solutions cannot be easily found through other means and one has to rely on vendor support. While SOs do not have financial resources to acquire such support for all applications, it should consider purchasing it for mission critical applications. Thus, the rule is for mission critical needs, acquire formal product support.
8. **Resource augmentation through Google search:** Current search engines and Internet provide vast and free resources to organizations. Organizations encourage staff to look up solutions to problems and ideas on the web. One developer stated:

“When confronting a problem my manager often asks me: have you googled it?”

Thus, the rule is when investigating a problem, search first for a solution on Google.

9. **Improve coordination by improving hiring practices:** SOs try to get the best resources as they know that the staff members must be multi-skilled. A person working for a small organization must be willing to learn new technologies, work on multiple projects, be able to handle stress and must be willing to progress on projects with or without formal guidelines. These are characteristics of a person who is running his own business. One analyst mentioned:

“Those people who have been entrepreneurs work best in our shop.”

Thus, the rule is when hiring look for individuals with entrepreneurial background and skills.

10. **Improve coordination through meetings:** Most SOs, as also evidenced in our case operate under functional management style. Therefore the position of Project Manager (PM) is created for each project. Typically, analysts play the role of PMs. Without proper title and authority, conflicts and confusion arise. Due to lack of authority, PMs find it very difficult to get team members to commit to deadlines and finish tasks. The strategy employed by PMs to overcome this hurdle is to 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, they are motivated to show results by the meeting time.”

Thus, the rule is to utilize meetings to get tasks done.

- 11. Improve coordination through office arrangement:** As resources are short and projects' list is vast, it is understood that not everyone will know all and quick communication is essential. Quick and earshot communication is enabled through closed proximity of staff offices and utilization of cubicles. For this purpose, cubicle offices or open office configurations are more suited than walled offices. One analyst mentioned:

“Because we sit close to each other, we can quickly exchange information. I don’t want to wait till meetings, back and forth emails or phone tags. Sometime I need a quick yes or no answer. Nothing beats first hand communication. Also, we are involved in creative work, if communication is not instant; sometimes it’s hard to refocus. Unless of course, we have written it down. But who has the time to write down everything?”

Thus the rule is to seat staff close to each other for ease in communication and for quick mutual support.

(O2) Small Organizations Need 3 Cs

In addition, from the interviews data, this work found 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 and their means (heuristics) of getting things done that eventually become organizational.

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, include a developer who is not distracted by other projects and an analyst (non-technical) who would be focused on analysis rather than coding programs. 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).”

Different medium of communication can be used. The first three types of users from Table 12 (actors’ taxonomy) should be communicated to via meetings and emails whereas the last one should be informed via web-bulletins. Both PM and non-technical analysts can split responsibilities regarding communication as it can become a huge job for a single person. Also, as noted in Table 12, 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. Due to lack of authority, PM finds 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, or 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 is best to give them advanced notice and send them agenda for what is needed.

Implications for Small Organizations

From this study, the following findings were found that can assist in understanding of other SM projects:

1. Individual heuristics eventually get adopted at organizational level. The method of propagation is informal though. These rules do not get passed on in

organizational procedures. The prevalent method of propagation is when junior staff learns it from senior staff.

2. One benefit of using these heuristics is they enable better communication, collaboration and coordination among project teams which are essential for the success of projects.

Summary of Results

Overall results of investigations into small organizations' software maintenance exhibit that there exists a paucity of resources (staff, funds, time, skills and methodologies) in SOs. This leads to ad-hoc processes and over reliance on individuals to get job done. Individual actors, it is learned from empirical studies, rely on certain heuristics that they have acquired over years of experience through repeated and successful usage. In addition to using such heuristics, these actors utilize political means to assist organizations in achieving goals and objectives. Eventually, individual heuristics get adopted by SOs and become organizational heuristics. It was also noted that three elements play vital roles in achieving success in SOs. They are communication, collaboration and coordination among individual working in SOs. In the next chapter overall contributions, limitations and conclusion are presented.

CHAPTER V

Contributions

A general contribution of this dissertation is to provide unique insights into the workings of SM projects in small organizations. There are some specific contributions which are provided in this section.

Firstly, a thorough review of existing SM process methodologies is conducted and the fitness of those methodologies to SOs is evaluated. This evaluation reveals that while some of existing processes are compatible to SOs due to their ease of implementation and relatively low need for resources, they do not meet the needs of SOs. This is because they are not complete enough to meet all requirements of SOs. Therefore there seems to be an imperative need for developing SM process methodologies explicitly tailored to suit the needs of SOs. The contribution here is establishing the need for generating a complete methodology for SOs' SM projects.

Secondly, an empirical investigation into the SM processes of SOs indicates that the processes in use are ad hoc in nature. The contribution here is establishing the need of generating an organized methodology for SOs' SM projects.

Thirdly, due to the absences of a formal methodology for SM in SOs, it is learned that individuals play a vital role in getting things done. They are the drivers and fuels of SOs. They help SOs achieve their organizational objectives. How individuals assist in achieving SM objectives is captured in and presented in a theoretical framework that emerges at the end of this investigation into the workings of SM activities in SOs. The contribution here is underlining the critical role of individuals in SOs' SM projects.

Fourthly, a theoretical framework is provided that explains how SM gets done in SOs. This framework is generated based on study of existing IT SOs. The contribution here is providing an understanding of important elements that are in play in SOs SM projects. Elements of this framework can be utilized in study of other organizations. These elements are provided in the following paragraphs:

1. The first element of the framework is the component of actors. A detailed review of existing actors involved in the SM project led me to present a taxonomy of major actors. This taxonomy lists different types of actors that are found to play most important 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.
2. The second element of the framework is the component of individual heuristics. Through empirical studies of two organizations, it is learned that in the absence of systems, methodologies and resources, individuals (actors) play a vital role in SM. These individuals rely on certain heuristics to get things done. The heuristics are derived from years of experience of working in the field. While heuristics seem to be a useful mechanism for SOs, there are certain caveats to their use that need to be

noted. Heuristics typically represent wisdom accumulated from experience.

Though the experience is real and the lessons captured are often valuable, different experiences easily can lead to contradictory heuristics and guidelines. In other words there are some biases (representativeness, availability or anchoring-and-adjustment) that must be watched for as they could impact the outcomes:

3. The third element of the framework is the component of political processes.

Political processes play a vital role in implementation and adoption of individual heuristics by organizations. This element underscores the significant role of individuals' political processes in enabling and impeding organizational objectives. Specifically, individual's political tactics are explored and the strategies and styles' impact on the outcome of SM projects are presented.

4. The fourth element of the framework is the component of organizational heuristics.

Due to repeated usage, individuals' heuristics get adopted by organizations and become organizational heuristics. In SOs' SM projects, these organizational heuristics enable collaboration, communication and coordination (3 Cs) which play critical roles in the success of SM projects.

These are the findings from what was seen in existing SM projects and in project components in small IT organizations. These findings offer an understanding of the inner working of SOs which can be used as a guideline to software maintainer in SOs.

Limitations

While the findings from this empirical study and its implications provided a rich insight into the inner workings of SOs, there were some limitations related to this study that were found and are needed to be mentioned. It should be noted, however, that these limitations did not impede the work; they rather defined the scope and boundaries of the study.

One limitation that was noted was both case studies that generated data for this dissertation were conducted in academic settings (in two universities). As such, a question could arise; do the findings from this work apply equally well to industry or commercial environments? An answer to this question requires further empirical studies in commercial entities and comparing the results. Nonetheless, as the author of the study has worked both in the industry and in academic environments, based on his judgment alone, no differences appeared glaringly. The shortage of resources and methodologies, reliance on actors, and reliance on heuristics appears to be characteristics that both sides share.

A second limitation of the study was data was collected from two organizations only. While it is a limitation in a sense that a generalization cannot be made from such a small-sized sample, the findings from this study are found to be extremely rich. As true with any interpretive case study, the data collected provided a vast amount of theoretical concepts, properties and dimensions (presented in the Methodology, Chapter III) that

were utilized in producing an enriched set of findings with implications and lessons for other SOs.

A third limitation of the study was the variance in size of parent organizations of the study sites. The parent organization for one data site was four times the size of students of the other organization. Nonetheless, it should be noted that both IT organizations had the same number of SM projects and software applications. It appeared that for comparison purposes, the size variant did not make a difference and the quality of data collected was the same from both institutions.

A fourth limitation was the study was conducted on SM projects using in-house resources only. I did not have the opportunity to look at the factors that could have impacted outcomes if SM were carried out by a third party, a vendor, and a consultant or outsourced to another country. Hence the findings are purely related to organizations that carry SM in-house.

A fifth limiting factor that should be noted is that in generation of taxonomy of actors, which is one of the major components of theoretical framework, a full scale stakeholder's analysis was not conducted. The listed actors are only those who were found in the responses of interviewees. These actors were regarded as playing the most important and direct role into the success of the SM project.

A sixth limitation is findings from this dissertation do not claim to be generalizations that can be applied to any other situation. Rather these are findings from looking at two specific organizations with their unique contexts.

Lastly, two limitations were found in the interviewees. One was those interviewees who had high communication skills tended to talk longer and provided more information which was useful in getting a rich set of data. Second, some interviews were conducted in cubicles and others were in closed rooms. Those interviews that were conducted in closed environments, such as a conference room or in an office with a door shut, tended to speak openly and provided more information related to the negative aspects of the projects. Both of these factors, however, did not impact negatively on the data collection, as half of the interviews were conducted in open spaces (cubicles) and the other half in closed rooms (conference rooms). Similarly, half of the individuals tended to be talkers and the other half closed-lipped persons.

Conclusion and Future Research

The purpose of this dissertation was to investigate challenges that SOs face in their SM efforts. Identifying challenges took me through a three-stage journey.

In the first stage, after conducting a thorough literature review, this work found that the main challenges SOs face in SM projects are shortage of resources that include lack of funds, staff, skills and tools. It was learned that there is not an appropriate SM methodology available that caters to the needs of SOs. While there are some methodologies such as CMMI for SOs, SOs shy away from them due to their prohibitive requirements and costs. An appropriate SM methodology was not found for SOs. Rather it was learned that the approach in use is ad-hoc.

In the second stage, empirical data was collected from existing SOs and findings from stage one were evaluated in a real-life context. It was noted that SOs are doing some great things. They are the engines behind word economics. How do they achieve such results was a question that was explored by undertaking an empirical investigation into the inner working of two small IT organizations. The investigation was conducted using qualitative research approach with Case Study and Grounded Theory methods.

In stage three, results of the empirical investigations are shown. A theoretical framework is presented that lists critical elements found to be affecting SM projects in the SOs under study. The framework shows that for SM, SOs depend on key individuals in getting jobs done. These individuals rely on certain heuristics and political processes to achieve organizational objective. With repeated usage these individual heuristics become

organizational heuristics that enable collaboration, coordination and communication which are found to be the main enablers of project's success in SOs.

From this work several good prospects emerge for future research. As SM involves 60-80 percent of software and information technology related tasks, looking at ways of achieving efficiencies in SM with available resources is an untapped area that has a lot of room for explorations. Some of the ideas that can be investigated in the future are.

- Small organization's resource base especially 'individuals' is already highly taxed due to many demands on their time and involvement in multiple projects. How to achieve optimal efficiencies from these individuals is an objective that I tried to explore in this dissertation. In future, this work hopes to expand upon these findings and continue research in the field by adding other related dimensions and properties that can suggest ways of achieving optimal utilization of resources. For instance, additional studies could be undertaken that explore the elements of time and funds utilization in SM projects.
- In this study, presence of political maneuvers was noted. It would be fascinating to find out how much time and money gets utilized or wasted with such political maneuvering.
- Another area that could be investigated is to look into the factors of pressure and stress on individuals involved in SM projects in commercial and academic environments.

- A study, similar to the one in this dissertation, could be conducted in a non-academic, commercial, organization. That would provide good data for comparisons between academic and commercial environments.
- This dissertation focused on in-house SM. Another avenue of exploration would be to look at SM projects that are outsourced, or are conducted by onsite consultants or outsourced consultants.
- Patch management is a big portion of work that constitutes SM. More studies are needed that focus on patch management, regular updates to software and processes and issues associated with them.
- Another area for future investigation is to explore linkages between individuals in the taxonomy of actors to the organizational heuristics and or political processes that were discussed in the Results, Chapter IV.
- Lastly, it would be a good idea to conduct additional studies to have enough data that can lead to formation of a generalized methodology for the SM needs of SOs.

APPENDICES

Appendix A: Interview Questions

1. What is your understanding of the term software maintenance (SM)?
2. How important is this activity for the proper running of the software systems that your department is responsible for? How much of the work is SM?
3. What initiates software maintenance activities?
4. How is software maintenance carried out? (What is the Process?)
5. What are the most critical issues (challenges) with regards to SM?
6. How are the resources and tools allocated for SM?

Appendix B: Institutional Review Board Exemption Letter

EXEMPTION NUMBER: 11-1X49

To: Raza Hasan
From: Institutional Review Board for the Protection of Human
Subjects, Justin Buckingham, Member
Date: Tuesday, April 26, 2011
RE: Application for Approval of Research Involving the Use of
Human Participants

Thank you for submitting an application for approval of the research titled,
Investigating Software Maintenance Challenges in Small Organizations

to the Institutional Review Board for the Protection of Human Participants
(IRB) at Towson University.

Your research is exempt from general Human Participants requirements
according to 45 CFR 46.101(b)(2). No further review of this project is
required from year to year provided it does not deviate from the submitted
research design.

If you substantially change your research project or your survey
instrument, please notify the Board immediately.

We wish you every success in your research project.

CC: Suranjan Chakraborty
File

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. Arthur, L.J. (1988). *Software Evolution*. John Wiley & Sons, Inc., New York, New York. 186.
4. Aversano, L., Canfora, G., De Lucia, A., and Stefanucci, S. (2002). Automating the management of software maintenance workflows in a large software enterprise: A case study. *Journal of Software Maintenance & Evolution: Research & Practice*, 14(4), 229-255.
5. Aysolmaz, B and Demirörs, O. (2011). A detailed software process improvement methodology: BG-SPI. Vol. 172, Part 3, 97-108, DOI: 10.1007/978-3-642-22206-1_9.
6. Bachara, P., Blachnicki, K., and Zielinski, K. (2010). Framework for application management with dynamic aspects J-EARS case study. *Information & Software Technology*, 52(1), 67-78.
7. Bardach, E. (1977). *The implementation game: What happens after a bill becomes a law*. MIT Press, Cambridge, MA, USA.
8. Barry, E. J., Kemerer, C. F., and Slaughter, S. A. (2007). How software process automation affects software evolution: A longitudinal empirical analysis. *Journal of Software Maintenance & Evolution: Research & Practice*, 19(1), 1-31.
9. Basili, V. R. (1990). Viewing software maintenance as reuse-oriented software development. *IEEE Software*, 7: 19-25.
10. Basit, H. A., and Jarzabek, S. (2009). A data mining approach for detecting higher-level clones in software. *IEEE Transactions on Software Engineering*, 35(4), 497-514.
11. Bellini, E., Canfora, G., Garcí'a, F., Piattini, M., Visaggio, C.A. (2005). Pair designing as a practice for enforcing and diffusing design knowledge. *Journal of Software Maintenance and Evolution: Research and Practice* 17 (6), 401 – 423.

12. Benbasat, I., Goldstein, D.K., Mead, M. (1987). The case research strategy in studies of information systems, *MIS Quarterly*, Vol. 11, No. 3 (Sept), pp. 369-386.
13. Benestad, H. C., Anda, B., and Arisholm, E. (2009). Understanding software maintenance and evolution by analyzing individual changes: A literature review. *Journal of Software Maintenance & Evolution: Research & Practice*, 21(6), 349-378.
14. Bennett, K.H, Rajlich, V.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.
15. Bikhchandani, S., Hirshleifer and Welch, I (1992). A theory of fads, fashion, custom, and cultural change in informational cascades. *Journal of Political Economy*, 100(5), 992-1026.
16. Bocco, M. G., Moody, D. L., and Piattini, M. (2005). Assessing the capability of internal metrics as early indicators of maintenance effort through experimentation. *Journal of Software Maintenance & Evolution: Research & Practice*, 17(3), 225-246.
17. Boehm, B. W. (1983). The Economics of Software Maintenance. *Proceedings, Workshop on Software Maintenance*, pages 9 – 37, Silver Spring, MD., IEEE Computer Society Press.
18. Bray, H.E., and Sniderman, P.M.(1985). Attitude attribution: A group basis for political reasoning, *American Political Science Review* 79:1061-1078.
19. Chase, W.G. and Simon, H.A.(1973). The mind's eye in chess. In W.G. Chase (Ed.) *Visual information Processing* , New York: Academic Press.
20. Colbrook, A., Smythe, C., and Darlison, A. (1990). Data Abstraction in a Software Re-engineering Reference Model. *Proceedings, IEEE Conference on Software Maintenance*, Los Alamitos, CA, pages 2-11.
21. Coleman, G. (2005) An empirical study of software process in practice, proc. 38th Ann. Hawaiian Int'l Conf. System Sciences (HICSS 38), Track 9, IEEE CS Press, p.315c.
22. Connie, U.S.(1990) *Performance engineering of software systems*. Addison-Wesley
23. Connie, U.S.and Lloyd, G.W. (2002). *Performance solutions: a practical guide to creating responsive, scalable software'* Addison-Wesley)

24. Cortellessa, V. & Mirandola, R. (2002), PRIMA-UML: a performance validation incremental methodology on early UML diagrams', *Sci. Comput. Program.*, 44, (1), pp. 101–129
25. Dahl, R. (1957). The concept of power. *Behavioral Science*, 2, 201-205
26. DeGroot, A.D. (1978). *Thought and Choice in Chess*. The Hague: Mouton.
27. De Lucia, A., Pompella, E., and Stefanucci, S. (2005). Assessing effort estimation models for corrective maintenance through empirical studies. *Information & Software Technology*, 47(1), 3-15.
28. Doolin, B. (2004). Power and resistance in the implementation of a medical management information system. *Information Systems Journal*, 14, 343–362.
29. Drake, P., Shanks, G., and Broadbent, M. (1998). Successfully completing case study research: combining rigor, relevance and pragmatism, *Information Systems Journal* 8.4, 273-289.
30. Du, L. & Hu, Q. (2006) Analysis of bidding behavior on Ebay auctions'. *Proc. IEEE Int. Conf. on e-Business Engineering, ICEBE'06*, October 2006, pp. 33–38, doi: 10.1109/ICEBE.26
31. Eisenhardt, K. M. (1989). Building theories from case study research. *Academy of Management Review*, 14: 532–550.
32. Ericsson, K. A. and J. E. Smith (1991). *Toward a general theory of expertise: Prospects and limits*. Cambridge, England, Cambridge University Press.
33. Fairholm (1993). *Organizational power politics: Tactics in organizational leadership*. Praeger, Westport, Conn.
34. Flyvbjerg, Bent. (2011) "Case Study", in Norman K. Denzin and Yvonna S. Lincoln, eds., *the Sage Handbook of Qualitative Research*, 4th Edition. Thousand Oaks, CA: Sage, pp. 301–316
35. Gable, Guy G (1994) Integrating case study and survey research methods: an example in information systems. *European Journal of Information Systems* 3(2):pp. 112-126.
36. Garcia, S. (2005). Thoughts on applying CMMI in small settings, *Presented in Carnegie Mellon Software Engineering Institute*, <http://www.sei.cmu.edu/cmmi/adoption/pdf/garciathoughts.pdf>.
37. Glaser B.G. (1978). *Theoretical Sensitivity*, Sociology Press, Mills Valley, CA.
38. Glaser, B.G., and Strauss, A.L. (1967). *The discovery of grounded theory*. Chicago, IL: Aldine.

39. Goldsmith, D., and Siegel, M. (2010). Managing and Valuing a Corporate IT Portfolio Using Dynamic Modeling of Software Development and Maintenance Processes. The MIT Center for Digital Business.
40. Grubb P., and Takang, A.A. (2003). Software Maintenance Concepts and Practice, 2nd Ed. World Scientific Publishing, Singapore.
41. Guerrero, F., and Eterovic, Y. (2004). Adopting the SW-CMM in a Small IT Organization, *IEEE Software*, 29, 30.
42. Harris, M., Aebischer, K., and Claus, T (2007). The whitewater process: Software product development in small businesses,” *ACM Communications Magazine*, vol. 20, no. 5, pp 89-93.
43. Hasan, R. and Chakraborty, S. (2011). Investigating Software Maintenance Challenges in Small Organizations. *AMCIS 2011 Proceedings - All Submissions*. Paper 381.
44. Hasan, R., Chakraborty, S. and Dehlinger, J. (2012). Examining Software Maintenance Processes in Small Organizations: Findings from a Case Study. *Studies in Computational Intelligence*, 2012, Volume 377, Software Engineering Research, Management and Applications 2011, Pages 129-143.
45. Hastie, R. and Dawes, R. (2001). Rational choice in an uncertain world: the psychology of judgment and decision making, Sage Publications, Thousand Oaks, CA.
46. Hofer, C. (2002). Software development in Austria: Results of an empirical study among small and very small enterprises, *Proceedings Euromicro Conference, 2002*
47. Humphrey, G (1963). Thinking: An introduction to its experimental psychology. New York: Wiley
48. Hunt, E. (2006). Expertise, Talent and Social Encouragement. The Cambridge Handbook of Expertise and Expert Performance. K. A. Ericsson, N. Charness, P. J. Feltovich and R. R. Hoffman. Cambridge, Cambridge University Press.
49. IEEE, Std. 1044, 1993. IEEE standard classification for software anomalies, IEEE.
50. Jalote, P., and Agrawal, N. 2005. Using defect analysis feedback for improving quality and productivity in iterative software development, *Proceedings of the Information Science and Communications Technology (ICICT 2005)*, pp. 701–713.

51. Jalote, P., Munshi, R., and Probsting, T., (2006). The when-who-how analysis of defects for improving the quality control process. *The Journal of Systems and Software*, 584-587.
52. Jasperson, J., Butler, B.S., Carte, T.A., Cross, H.J.P., Saunders, C.S., Zheng, W. (2002). Review: Power and information technology research: A metatriangulation review, *MIS Quarterly* December, Vol.26, No.4, pp.397-459.
53. Jørgensen, M., and Sjøberg, D. I. K. (2002). Impact of experience on maintenance skills. *Journal of Software Maintenance & Evolution: Research & Practice*, 14(2), 123-146.
54. Kahneman, D., and Klein, G. (2009). Conditions for intuitive expertise. *American Psychologist*, 64, 515–526.
55. Kahneman, D., and Tversky, A. (1973). On the psychology of prediction, *Psychological Review*, pp.237-251.
56. Kahneman, D., and Tversky, A. (1979). Intuitive prediction: Biases and corrective procedures, *TIMS Studies in Management Sciences* pp. 313-327.
57. Kernighan, Brian W.; Plauger, P. J. (1976). *Software Tools*, Addison-Wesley, pp. 352, ISBN 020103669X.
58. Kipnis, D. and Schmidt, S. (1988). Upward influence styles: Relationship with performance evaluation, salary, and stress, *Administrative Science Quarterly*, 33(4), 528-542.
59. Kling R. (1980). Social analyses of computing: Theoretical perspectives in recent empirical research. *Comput Surv*, 12, 1, 61-110.
60. Ko, A. J., Myers, B. A., Coblenz, M. J., and Htet, H. A. (2006). An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *IEEE Transactions on Software Engineering*, 32(12), 971-987.
61. Lee, A. S. (1991) Integrating Positivist and Interpretive Approaches to Organizational Research. *Organization Science* (2:4), November, pp.342-365.
62. Lau, R. R., and Redlawsk, D. P. (2001). Advantages and disadvantages of cognitive heuristics in political decision making. *American Journal of Political* (add detail) Vol. 45, No. 4, October.
63. Lehman, M.M. (1966). The Programming Process. IBM Res. Rep. RC 2722, IBM Research Center, Yorktown Heights, NY 10594.
64. Lientz B. P., Swanson E. B. (1980). *Software Maintenance Management*. Addison Wesley, Reading, MA.

65. Madison, D.L., Allen, R.W., Porter, L.W., Renwick, P.A., and Mayes, B.T. (1980). Organizational politics: An exploration of managers' perceptions. *Human relations*, 33, No. 2: 79-100.
66. Markus, L.(1983). Power, Politics and MIS Implementation, *Communications of the ACM*, Vol.26., No.6, June.
67. Mayer, R (1983). *Thinking, Problem Solving, Cognition*, W.H. Freeman and Company, New York.
68. Merriam-Webster Online Dictionary. (2009). Case study. Available at <http://www.merriam-webster.com/dictionary/case%20study>
69. Morasca, S., and Ruhe, G. (2000). A hybrid approach to analyze empirical software engineering data and its application to predict module fault-proneness in maintenance. *Journal of Systems & Software*, 53(3), 225.
70. Myers, M.D. and Young, L.W. (1997). Hidden agendas, power and managerial assumptions in information systems development: an ethnographic study. *Information Technology and People*, 10, 224–240.
71. Newell, A., and Simon, H.A. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentice Hall.
72. Newell, A., Shaw, J.C. and Simon H.A.(1958). The processes of creative thinking, The RAND Corporation Paper, P-1320.
73. Nielsen, P. A., and Tjørnehøj, G. (2010). Social networks in software process improvement. *Journal of Software Maintenance & Evolution: Research & Practice*, 22(1), 33-51.
74. Orlikowski, W. and Baroudi, J.(1991). Studying information technology in organizations: Research approaches and assumptions, *Information Systems Research* 2.1, 1-28.
75. Osborne, W.M. (1987). Building and sustaining software maintainability. *Proceedings of Conference on Software Maintenance*, pages 13-23.
76. Parsons, J., and Saunders, C. (2004). Cognitive heuristics in software engineering: Applying and extending anchoring and adjustment to artifact reuse. *IEEE Transactions on Software Engineering*, 30(12), 873-888.
77. 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.
78. Paulk M. et al., (1993). Capability maturity model for software, Version 1.1, tech. report CMU/SEI-93, TR-24, Software Eng. Inst.

79. Payne, G & Payne, J. (2004) Key Concepts in Social Research, London: Sage
80. Pendharkar, P. C., and Rodger, J.A. (2009). The relationship between software development team size and software development cost. *Communications of the ACM*, 52(1), 141-144.
81. Pfeffer, J. (1992). Understanding Power in Organizations, *California Management Review* Winter; 34, 2; pg. 29.
82. Pigoski TM. (1997). Practical software maintenance: Best practice for managing your software investment. Wiley, New York NY, 29-36, 117-138.
83. 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.
84. Ramaswamy, R. (2000). How to staff business-critical maintenance projects. *IEEE Software*, 17(3), 90.
85. Rashid, A., Wang, W.Y.C., & Dorner, D. (2009). “Gauging the Differences between Expectation and Systems Support: the Managerial Approach of Adaptive and Perfective Software Maintenance”, 4th International Conference on Cooperation and Promotion of Information Resources in Science and Technology.
86. Robey, D. and Sales, C.A. (1994). Designing organizations, 4th Edn. Irwin, Burr Ridge, IL, USA.
87. 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*.
88. Reformat, M. (2005). A fuzzy-based multimodel system for reasoning about the number of software defects. *International Journal of Intelligent Systems*, 20(11), 1093-1115.
89. Sabherwal, R. and Grover, V. (2010). A Taxonomy of political processes in systems development, *Info. Systems Journal*, 20, 419-447.
90. Shanteau, J. (1989). cognitive heuristics and biases in behavioral auditing: Review, comments and observations. *Accounting, Organizations & Society*, 14(1/2), 165-177.
91. Silva, L. M., Alonso, J., and Torres, J. (2009). Using virtualization to improve software rejuvenation. *IEEE Transactions on Computers*, 58(11), 1525-1538.

92. Simon, H. A. (1957). Models of man, social and rational: Mathematical essays on rational human behavior. New York: Wiley.
93. Sniderman, P.M., Brody, R.A, and Tetlock, P.E. (1991). Reasoning and Choice: Explorations in Political Psychology. New York: Cambridge University Press.
94. Software CMMI (2005) Mid-year update by Software Engineering Institute, www.sei.cmu.edu/cmmi/casestudies/profiles/pdfs/upload/2005sepSwCMM.pdf.
95. Software Industry Statistics for 1991-2005, Enterprise Ireland, 2006; www.nsd.ie/htm/ssii/stat.htm.
96. 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*, June, Vol. 80, Issue 6, p883-895, 13p.
97. Strauss, A and Corbin, J.(1998). Basics of qualitative research: Techniques and procedures for developing grounded theory, 2nd Ed., Sage Publications, Thousand Oaks, CA.
98. Swanson, E. B., and Dans, E. (2000). System life expectancy and the maintenance effort: Exploring their equilibration. *MIS Quarterly*, 24(2), 277-297.
99. Takang, A.A., and Grubb, P.A. (1996). Software Maintenance Concepts and Practice. Thompson Computer Press London, UK.
100. Tonge, F.M.(1961). The use of heuristic programming in management science: A survey of the literature. *Management Science* 7/3: 231-237.
101. Tosun, A., Bener, A., Turhan, B., and Burak T. (2009). Implementation of a Software Quality Improvement Project in an SME: A Before and After Comparison, *35th Euromicro Conference on Software Engineering and Advanced Applications*.
102. Urquhart, C. (2001) An Encounter with Grounded Theory in E. M. Trauth (Ed.) *Qualitative Research in IS: Issues and Trends*, Hershey, PA: Idea Group Publishing, pp. 104 -140.
103. Usaola, M. P., Velthuis, M. P., and González, F. R. (2001). MANTOOL: A tool for supporting the software maintenance process. *Journal of Software Maintenance & Evolution: Research & Practice*, 13(2), 77-95.
104. Midha, V & Bhattacharjee,A.(2012). Governance Practices and Software Maintenance: A Study of Open Source Projects, *Decision Support Systems*, doi: 10.1016/j.dss.2012.03.002

105. Walsham, G. (2006). Doing Interpretive Research. *European Journal of Information Systems* 15, 320 -330.
106. Wang, Q., Shen, J., Wang, X., and Mei, H. (2006). A component-based approach to online software evolution. *Journal of Software Maintenance & Evolution: Research & Practice*, 18(3), 181-205.
107. Webb, S and MacMillian, J. (1995). Cognitive Bias in Software Engineering, *Comm. ACM*, vol. 38, no. 6, pp. 57-69.
108. Yates, F.J. and Tschirhart (2006). Decision-Making Expertise. The Cambridge Handbook of Expertise and Expert Performance. K. A. Ericsson, N. Charness, P. J. Feltovich and R. Hoffman. Cambridge, Cambridge University Press.
109. Yin, R.(1994). Case study research design and methods, 2nd Ed., Sage Publications, 9.
110. Yu, L.(2006). Indirectly predicting the maintenance effort of open-source software. *Journal of Software Maintenance & Evolution: Research & Practice*, 18(5), 311-332.

CURRICULUM VITAE

Raza Hasan

PROFILE

- Eighteen years of work experience related to Information Systems in Academia and industry
- In Academia, as an Adjunct Professor & Assistant Professor, taught graduate & undergraduate courses
- In Industry, held job: Business Analyst, Project Manager, Software Engineer & Database Administrator
- Strong Research Aptitude. My paper was selected as one of the top papers at SERA 2011
- Excellent communications skills allow me to establish rapport with students and be an ideal teacher

EDUCATION & TRAINING

- Doctor of Science in Information Technology, Towson University. May 2012
- Masters of Science, Management Information Systems, University of Baltimore. May 1992
- Bachelor of Business, Finance, Western Illinois University, Illinois. December 1988
- Project Management Professional (PMP). 2007 - Present
- Microsoft Certified Systems Engineer (MCSE). 1999

TEACHING EXPERIENCE AS AN ADJUNCT PROFESSOR

Towson University, Maryland

August 2008 – Present

Course taught: Computers and Creativity

University of Baltimore, Maryland

May 2003 – May 2009

Courses taught: Visual Basic, Java, Management Information Systems, Information Resource Management, Data Communications, Info. Systems Planning, Wireless Communications & Object Oriented Programming

Johns Hopkins University, Maryland

July 2005 – Aug 2005

Course taught: Wireless Technologies

Baltimore City Community College, Maryland **January 1999 – December 2004**
 Courses taught: Introduction to Computers, Visual Basic, MS-ACCESS and Microcomputer Applications

George Mason University, Virginia **January 2003 – December 2003**
 Courses Taught: Managing Information and Introduction to Business Information Systems

Morgan State University, Maryland **January 2003 – May 2003**
 Course Taught: E-Business

Coppin State College, Maryland **January 1993 – December 1994**
 Courses taught: COBOL, MIS and Systems Analysis and Design

TEACHING EXPERIENCE AS A VISITING ASSISTANT PROFESSOR

University of Baltimore, Maryland **January 2001 to January 2003**
 Courses taught: Information Systems Planning, End-User Computing, Information Resource Management and JAVA. Assisted in establishing Group Decision Support Center and Oracle Lab. Received high marks (top 10% of instructors) in student evaluations

RESEARCH INTERESTS

Software Maintenance, Information Systems in Small Organizations, Collaboration Engineering, Databases, Requirements Elicitation, Social Networking and Enterprise Resource Planning

RESEARCH EXPERIENCE

- Publication: Hasan, Raza; Chakraborty, Suranjan and Chatterjee, Sutirtha, "The Critical Role of Political Processes in Small Organizations' Software Maintenance Efforts." AMCIS 2012 Proceedings (accepted)
- Publication: Hasan, Raza; Chakraborty, Suranjan; Dehlinger, Josh, "Examining Software Maintenance Processes in Small Organizations: Findings from a Case Study" (2011). Software Engineering Research, Management and Applications 2011 Studies in Computational Intelligence. Volume 377, 2012, pp 129-143
- Publication: Hasan, Raza and Chakraborty, Suranjan, "Investigating Software Maintenance Challenges in Small Organizations" (2011). AMCIS 2011 Proceedings - All Submissions. Paper 381. http://aisel.aisnet.org/amcis2011_submissions/381
- Conference Presentation: Research findings from "Investigating Software Maintenance Challenges in Small Organizations" at AMCIS 2011
- Conference Presentation: Research findings from "Examining Software Maintenance Processes in Small Organizations: Findings from a Case Study" at Software Engineering Research, Management and Applications Conference 2011

- Conference Presentation: Project Lessons Learned: “Using Web Services eforms for Early Faculty Provisioning” at Higher Education (HEUG) Mid-Atlantic Regional Conference 2011
- Conference Presentation: Research findings from "Quality Control in a Public Health Colorectal Cancer Screening Program" was presented at the American Public Health Association 135th Annual Meeting, November 05, 2007

INDUSTRY EXPERIENCE

Sr. Business Analyst, Towson University, Baltimore **2008 onwards**

Towson University is the second largest academic institution in Maryland

- Identify user requirements and design / develop solutions with developers
- Document functional requirements with use cases; conduct unit, system and user-acceptance testing; receive users' sign-off

Database Administrator / Analyst, University of Maryland, Baltimore **2004 to 2008**

University of Maryland, Baltimore is a major healthcare provider and education institution in Maryland

- Provided support to full life-cycle implementation of an Enterprise Application that tracks cancer patients. It includes enterprise portal, databases, and reporting capability (Crystal Reports 10). Wrote SQL queries and stored-procedures. Provided training to users throughout the state of Maryland
- Designed and developed database applications that track cancer patients in Maryland

Senior Software Engineer, RWD Technologies, Baltimore **2006**

Worked on two projects with RWD - a professional consulting services provider

- **First Project:** Participated in all phases of project management of developing a data warehouse for an oil hauling company. Business and data modeling was conducted with Rational Rose
- Used SQL Server 2005 Analysis (SSAS). Conducted all phases of Extraction Transformation and loading (ETL). Utilized Integration Services (SSIS) to load data warehouse. Defined and implemented cubes, Key Performance Indicators (KPIs) and reporting with Reporting Services (SSRS)
- **Second Project:** RWD provides eLearning solutions to clients who have implemented ERP systems such as SAP, PeopleSoft and JD Edwards. For this purpose, I participated in customizing User Productivity Kit (UPK) for a client that was implementing PeopleSoft. Some of the functionalities provided were end user specific training, transaction documentation, instructional design tools and test scripts.
- Documentum was used to implement workflow and document imaging

Consultant Analyst, Impact Innovations Group, Maryland **2000 to 2001**

Impact Innovations Group offer I.T. consulting services to newly established dot comes

- Assisted in writing project plans for creating enterprise java based portal application
- Using JAD sessions, defined business requirements. Participated in a project where mortgage applications were automated with Documentum and made available over the web

Systems Manager, Office of Employment Development, Baltimore City **1999 to 2000**

Office of Employment Development is a government agency that offers employment and training services

- Managed staff consisting of PC technicians, network support and help-desk support staff
- Planned and integrated a wide area network located in 6 branches, 300 clients, and several file servers
- Participated as a bridge between different stakeholders in implementation of citywide SAP Financials.

Consultant, TekSystems, Maryland

1998 to 1999

On behalf of TekSystems, an I.S. Service Provider, provided consulting services to two clients:

- For Norwest Financial Services, managed a national rollout of computer applications. Designed and implemented network user setup, end-user training materials, and testing procedures
- For Greater Baltimore Medical Center, administered LAN and WAN analysis to identify bottlenecks. Provided Systems evaluation, and design & implementation for Financial Management Software

Network Administrator, Doctors Health Inc, Maryland

1998

Doctors Health was a major Medical Service Provider in Maryland

- Managed heterogeneous Networks: Windows NT 4.0, and Novell 4.11. Installation included DHCP, DNS, SMS, SQL Server 6.5 & Exchange Server, RAS and Reachout on NetWare
- Created a WEB BASED Solution (Patients Accounting Information) that received scanned-survey-forms input, and generated dynamic reports from that data using SQL Server 6.5, IIS and ASP

Additional experience include

- Entrepreneurial background from establishing two businesses 1996 and 2003
- Programming, help desk support, research analysis 1990 to 1996

SOFTWARE

- ERP: PeopleSoft(HR, Student Records), UPK
- WEB Dreamweaver, ASP, JavaScript, Cold Fusion, XML, HTML
- PROGRAMMING: .Net Visual Basic, JAVA, COBOL Basic, ASP, JavaScript
- DATABASES: SQL Server 2005, Oracle 10/11, R:BASE 5.0, Data Flex, FoxPro
- GROUP SOFTWARE: MS Exchange Server, Lotus Notes, GroupWise, MS Project
- OPERATING SYSTEMS: Windows 7, XP, NetWare 5, Linux, VMS, TCP/IP, IPX

PROFESSIONAL ASSOCIATIONS

- Member, Association of Information Systems, 2011
- Member, International Association for Computer and Information Science, 2011