Abstract
This study uses a grounded theory research method to explore how IT professionals define and select a methodology to maintain existing software. The findings contribute to a better understanding of how standard methodologies are applied in software practice and the critical factors used by professionals when choosing an appropriate methodology for software maintenance activities. This research underscores the need for incorporating the full software life cycle in information systems development research as well as highlighting the need for more comprehensive education in software methodologies.

1. Introduction

Many studies champion using a standard process as a way to create better quality software that meets customer expectations and is delivered on-time and within budget [14,17,20,32]. Prior research has documented the use and relative effectiveness of applying standard processes to initial software development [17,20,28], but studies have shown that software development professionals are reluctant to use standard processes on an ongoing basis to complete their work [3,6,21]. To better understand this reluctance, current research has used qualitative research methods helping researchers study how software development professionals incorporate, change and improve their software processes [10,24,35].

While current research is developing a richer view of how software development professionals use processes, the focus is mainly on the initial creation and implementation of a software product, rather than the entire life cycle of development that is the responsibility of software development professionals. We have very limited understanding of how software development professionals use processes to support the ongoing evolution of a software product.

Software differs from other engineering products because it is comparatively malleable; software can be changed to incorporate new requirements. If the fundamental nature of software is that it can be changed, then the work performed by software development professionals must incorporate and reflect that change. And in “real life,” it does. Several investigations have surveyed the amount of software maintenance taking place in organizations, showing much variation in the amount of effort and costs associated with software maintenance [2]. In the trade press, the rule of thumb is that about 70-80% of the overall systems development budget supports maintenance activities [16]. Thus, the majority of work performed by software development personnel is related to maintaining software, rather than creating it.

Research into development processes, on the other hand, has put most of its resources into exploring initial development rather than ongoing software maintenance [25]. While it could be argued that the processes used to conduct new development are similar to those necessary to support the long term maintenance of software, there are also distinct differences. Research has shown that the characteristics of software maintenance differ from new development based on the scheduling of activities, the types of activities that must be performed, and the scope of knowledge necessary to complete those activities [31].

The purpose of this research is to expand our knowledge of the processes used to maintain existing software products. We want to understand the relative use of a standard process by software development professionals during software maintenance activities, how a software developer views/defines a standard process and the factors considered when making a choice of process. In order to achieve that understanding, we used a grounded theory approach to design our study, guide data collection, and analyze the results.

The next two sections of this paper define software maintenance and processes/methodology from existing literature. Section 4 explains our research approach, while section 5 describes and analyzes our findings. Finally, section 6 presents the implications of our study for research and education of future systems development professionals.

2. Software Maintenance and Evolution

Software maintenance is defined in IEEE Standard 1219 as: “The modification of a software product after delivery to correct faults, to improve performance or
other attributes, or to adapt the product to a modified environment.” Some authors question the use of the term “maintenance” as an effective description of all the activities revolving around software after it has been installed [1,7,14]. These authors suggest that the term “maintenance” carries with it connotations of errors, and the repair of errors, and is thus an inaccurate description of those activities placed under the maintenance umbrella. Research shows that the vast majority of software maintenance is to adapt software to a changing environment by enhancing functionality, applying new technology, and meeting the needs of new users [2]. The term “software evolution” is becoming more frequently used as a way to better describe the ongoing development effort necessary to keep a software application effective for its intended purpose [7,27,28].

Some authors are calling for a more comprehensive inclusion of software evolution and believe that all development and engineering research must incorporate the reality of ongoing software changes as part of the research framework [29]. There is increasing awareness in software engineering research that software evolution and development should be viewed together, rather than attempting to separate the functions [11].

3. Software Processes/Methodologies

Much thought has been put into the terminology surrounding the area of software development [22]. Following that work, we use the term “methodology” to include the guiding principles, values, concepts, phases, and tasks used to guide the participants in the act of developing and maintaining software.

Substantial research effort over decades has gone into defining, examining, and evaluating the processes used to develop software [i.e. 13,20,32]. Much of the initial effort created normative models based on minimal fieldwork and an engineering paradigm. It was assumed that those idealized models would improve the effectiveness of the process.

After much work was done to clarify and delineate methodologies used for software development that showed success in improving the overall software development process [18,20] some researchers found that relatively few software development professionals were willing to incorporate those standard methodologies into their personal daily activities [21]. It was also found that professionals were capable of misrepresenting to themselves and others their actual level of use of methodologies [30]. As a result, additional research was conducted to better understand why some process improvement efforts succeeded and others failed [3]. This additional research demonstrated that software development is a complex social, technical, organizational, and personal task. Current research is now delving into more extensive and detailed fieldwork-based studies of all aspects of the process [10,19,23,28,35].

Research into software maintenance processes is trying to better understand the methodologies that are most effective for the ongoing maintenance of software [1,7,14,26,33]. However, this research into maintenance is adopting the original approach used with research into initial software development of: (1) defining a normative model; (2) testing it in a limited production environment; and (3) extrapolating its potential success. This approach does not take into account fully what has already been learned about the complexity of the development process. This approach does not consider how methodologies are selected and adapted by software development/maintenance professionals.

We believe that another way to study software maintenance methods is to explore as directly as possible those professionals who are performing the work. Our goal is to contribute to the understanding of how professionals maintain existing software.

4. Research Approach

Our study develops theory about whether a software methodology is used by IT professionals to perform software maintenance and also how and why that methodology might be chosen. This study seeks to better understand the relative use of software methodologies during software maintenance. We were especially interested in understanding the factors that affected the use and choice of methodology and what those factors meant to IT professionals.

This study is based on the interpretive research assumption that reality is a social construction by both the human actors and the researchers who are observing those actors [36]. This study uses an adapted version of grounded theory referred to as “constructivist” grounded theory [8]. According to Charmaz [8, p.130], “constructivists study how – and sometimes why – participants construct meanings and actions in specific situations.” But the resulting theory produced “depends on the researcher’s view; it does not and cannot stand outside of it.” Thus, it is important to understand the relevant background of the researchers to evaluate the results of this study. We have performed software maintenance activities within organizations, and as self-employed software consultants. We are familiar with software development methodologies, have taught methodologies, and have used both standard and adapted methodologies for software maintenance.
4.1. Data collection

Data were collected through semi-structured face-to-face and online interviews. The face-to-face interviews occurred in real-time, while the online interviews consisted of text-based answers to predefined questions, with some limited back-and-forth responses between interviewer and interviewee for further clarification of answers. Face-to-face interviews lasted between .5 and 1.5 hours. The face-to-face interviewees were selected to expand/enhance the emerging categories, while the online respondents were self-selected, based upon their interest in the topic under discussion.

Data collection occurred over a two year time period (2007-2009). Data gathering and analysis were performed iteratively, as recommended in the methodology literature [8,12,34] in order to better understand the meaning of the data and to identify what additional data were required to expand understanding of the concepts.

Six face-to-face interviews were performed to start the study and begin to establish categories for further analysis. An initial interview protocol was used for the first interviews. The interview questions were enhanced based on the results of those initial interviews so that more detailed information might be gleaned from the interviewees. After conducting 16 face-to-face interviews, we then posted the most relevant questions online. 32 online responders were included in the analysis. The interview questions were enhanced and refined. An additional 5 face-to-face interviews completed the study. In total, there were 21 face-to-face interviews and 32 online interviews conducted for this study.

4.2. Interviewee characteristics

While the focus of this study was not the interviewees or their organizations, we did gather some data about their characteristics to better understand the context of the answers received. Since the focus of this study is maintaining software after initial implementation, we selected interviewees with experience both doing and managing maintenance activities. Unlike prior studies [10] we interviewed both developers and managers, but focused primarily on developers. Interviewees ranged from 5 to 38 years of experience with software maintenance as IT professionals. Table 1 summarizes some additional characteristics of the interviewees.

Face-to-face interviews were performed at four different organizations. Table 2 provides some information about the organizations.

Two of the organizations produced software for sale (IT is the primary product for those two organizations) while the other two organizations produced products and/or services other than information technology. All four organizations are operating in a volatile external environment. ORG3 and ORG4 were experiencing layoffs, while ORG1 and ORG2 were facing strong competition, a highly dynamic technical environment, and a rapidly changing customer base. While ORG1 remained highly profitable during that time, ORG2 was suffering financial cutbacks and a decrease in overall market share.

4.3. Software characteristics

Past research [33] has hypothesized that different types of software may require different types of software development and maintenance methodologies. While we attempted to find diversity in our interviewees and the organizations for which they worked, we did not wish to expand our results into potentially confounding differences between software types. Our data collection focused on interviewees responsible for maintaining business process-oriented software. For example, our interviewees maintain software that handles such business processes as payroll, inventory control, employee scheduling, business intelligence analysis and reporting, budgeting,
and housing assessment evaluation. Our interviewees do not maintain embedded software, or software with a critical real-time component such as traffic control or software that directly controls the real-time operation of machinery.

4.4. Data analysis

The units of analysis for this research were the core concepts under investigation. These concepts were the relative use of a methodology during software maintenance activities, the choice of software maintenance methodology, and the factors considered when making that choice.

Data were analyzed using a five stage process [8]: (1) Transcription: Interviews that were recorded were transcribed, interviews that were not recorded were reviewed and notes were typed, online interviews were collected; (2) Initial coding: Reviewed and coded all transcripts line-by-line with general codes reflecting interviewees thoughts and actions; (3) Focused coding: Reviewed initial codes and developed more conceptual codes that reflected the intent of the initial codes; (4) Theoretical coding: Identified how the conceptual codes related to each other; (5) Theoretical memoing: Created conceptual memos presenting how the theoretical codes might be portrayed as descriptive, explanatory and potentially predictive theory.

To understand the beginning processes of data analysis, Figure 1 presents a segment of an interviewee transcript with the initial and focused codes.

4.5. Analytical constraints

Our analysis of the data collected, and the resulting interpretation is not presented as an “objective” rendering of reality. We construct an explanation of the concepts that emerged from the interviews from a perspective that includes our own backgrounds and previous knowledge of software maintenance processes. For this study, we followed the guidelines for grounded theory research of keeping an “open mind, not an empty head” [12].

5. Findings – Description and Analysis

Part of the research process was developing a shared vocabulary with interviewees. As discussed earlier, software maintenance is a general term used to cover all changes made to software after it is initially placed into a production environment. This term was refined during the interview process. Interviewees modifying a software product for sale defined maintenance as all activities between major releases of a product. A major release of a product (even if the product was already in production) was considered “new development” by those interviewees. Maintenance for those interviewees included incremental changes required to correct defects, adapt to regulatory environments, or enhance functionality of the software. Interviewees using software to support internal organizational processes viewed maintenance as all work performed after initial implementation. If a

<table>
<thead>
<tr>
<th>Initial Coding</th>
<th>Focused Coding</th>
<th>Online Respondent Excerpt (O12, medium, U.S. only)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using existing knowledge of methodology</td>
<td>Professional knowledge</td>
<td>We have a mix of traditional / waterfall techniques and agile techniques. Waterfall is used largely because that’s what most people learned either in college or on-the-job over many years. Waterfall has clearly been institutionalized - that is, jobs are specifically created to fit into the waterfall cycle. Business Analysts do requirements, Systems Analysts do Design, Developers do coding, Testers do testing, and so on. So this is very well entrenched in our thinking. But the following phrase changed many things within our organization, and led us to agile: “the world is moving faster than we work”. Waterfall generates mediocre results, at best, and we know that. There has to be a better way. Many of us are finding agile is a better way. But the path to agile is extremely difficult.</td>
</tr>
<tr>
<td>Using learned knowledge of methodology</td>
<td>Educational knowledge</td>
<td></td>
</tr>
<tr>
<td>Acquiring knowledge of methodology</td>
<td>Method reinforcement</td>
<td></td>
</tr>
<tr>
<td>Designing an organizational structure to fit the method</td>
<td>Organizational reinforcement of method</td>
<td></td>
</tr>
<tr>
<td>Using a method reinforces the organizational structure</td>
<td>Institutionalization of method</td>
<td></td>
</tr>
<tr>
<td>Entrenching method in thinking</td>
<td>Entrenchment</td>
<td></td>
</tr>
<tr>
<td>Coping with change: world is moving faster than work</td>
<td>Speed of change/speed of development</td>
<td></td>
</tr>
<tr>
<td>Triggering change because of mediocre results</td>
<td>Trigger of change</td>
<td></td>
</tr>
<tr>
<td>Finding a path for changing methods</td>
<td>Path for change</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1. Sample Data Coding – Initial and Focused Codes
software product was purchased, rather than developed in-house, it was considered “maintenance” to install and refine new releases of the product.

Interviewees used the terms “process,” “method,” “methodology,” “framework,” and “maintenance approach” interchangeably throughout our discussions. For the interviewees, these terms encompassed the guiding principles (either implicitly or explicitly defined), steps, rules, guidelines, tools, and documentation used and produced during software maintenance activities.

Interviewees frequently differentiated between methodology and “techniques,” however. A technique was usually defined as a specific way to accomplish a particular goal, such as Fagan Inspections for creating software testing standards or paired programming to facilitate combining the coding and review activities.

The remaining sub-sections discuss our findings, including a definition of the types of methodologies used for maintenance, an examination of experimentation with methodologies, the relative usage of a methodology during maintenance, and the factors used to select a methodology for software maintenance.

5.1. Definition of methodology

Interviewees seldom referred directly to a particular name for a methodology. They often contrasted two general methodologies for software maintenance termed as follows: Traditional/waterfall/heavyweight vs. Iterative/agile/lightweight. They characterized the methodologies they used most often using the distinction of “heavyweight” vs. “lightweight” or “waterfall” vs. “agile”.

Figure 2 depicts the two general types of methodologies as determined from our data. Interviewees most often referred to the two methodologies as “two end points on a continuum or range” rather than a Boolean-type of difference. Some interviewees referenced the Agile Manifesto for the core values of the lightweight methodology, but most provided their own take on those values. The key value for lightweight was to accommodate change. The core values for a heavyweight methodology were more variable, but tended to focus on the desire to pre-define specifications in order to better manage the process.

Many techniques were considered interchangeable among methodologies. For example, while paired programming was associated exclusively with lightweight methodologies, Fagan Inspections were used with lightweight and heavyweight methodologies. While existing literature references prototyping as a methodology [22], respondents tended to view it as a technique to be used within either heavyweight or lightweight methodologies.

After a group of face-to-face interviews, we began using the terms “agile” and “waterfall” in other interviews and with our online respondents for definitional purposes, so we may have contributed to the delineation of those two general methodologies in the minds of some of our interviewees. However, respondents consistently discussed the differences and similarities in heavyweight and lightweight methodologies and referenced the “continuum” rather than distinct points.

The concept of a continuum of methodologies was supported by interviewee comments about “choosing

<table>
<thead>
<tr>
<th>Core Concepts</th>
<th>Heavyweight Methodology (Waterfall)</th>
<th>Lightweight Methodology (Agile)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bring order to chaos</td>
<td>Accommodate and respond to change</td>
</tr>
<tr>
<td></td>
<td>Value repeatability</td>
<td>Value creating working software quickly</td>
</tr>
<tr>
<td></td>
<td>Value project management</td>
<td>Spend time creating rather than planning</td>
</tr>
<tr>
<td></td>
<td>Follow a plan</td>
<td>Not pre-defined</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Iterative</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Concurrent</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Steps</th>
<th>Pre-defined</th>
<th>Avoid excessive planning because the plan will probably change</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Linear with clear feedback loops</td>
<td>Work with customer as available during project</td>
</tr>
<tr>
<td></td>
<td>Perform sequentially</td>
<td>Continue determining requirements during project</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Create minimal documentation - usually at the end of the project</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A single individual may do multiple (or all) tasks in the process</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Guidelines</th>
<th>Work with customer early in project, or possibly not at all if not necessary</th>
<th>Paired programming</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Determine requirements before proceeding to development</td>
<td>Extremely rapid prototyping in sprints</td>
</tr>
<tr>
<td></td>
<td>Gain agreement on requirements</td>
<td>Minimal modeling</td>
</tr>
<tr>
<td></td>
<td>Freeze requirements early in project</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Create comprehensive documentation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Separate tasks by function</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Techniques</th>
<th>Modeling (DFD, ERD, UML)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Project management techniques (Gantt charts, check-lists, milestones)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Project repository</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2. Continuum of Methodologies
the best from any place on the line,” “we have a mix of heavyweight and lightweight techniques,” “there is a spectrum and no team falls entirely on one methodology,” “we don’t use a strict anything because we like to take the best from all”. In contrasting Scrum (a lightweight development methodology [15]) with a waterfall methodology, one interviewee said: “How are you going to write the software infrastructure, get your database right, and then get some infrastructure in place in time for your first iteration? You can’t. So, then you have to structure your development approach differently...Scrum refers to the first release as “iteration zero,” but that’s just language to hide the fact that Scrum needs help from traditional software processes.”

5.2. Experimentation with methodology

As reported in the literature [10,28] the interviewees use “trial and error” to devise an appropriate maintenance process. They combine pair programming without any other agile practices, adopt very short delivery cycles with traditional waterfall phases, and apply project management documentation standards to extremely rapid prototyping.

“WaterScrum” is an example of a process developed through trial and error in one of the organizations in the study (ORG1). This methodology is a hybrid between the waterfall and agile methodologies. In this hybrid, design definition and requirements are established using a waterfall-like approach, but the actual coding is performed using Scrum. Teams in ORG1 mix and match components from waterfall and Scrum to create different combinations depending on the requirements of the project and the expertise of the team. ORG1 uses a hybrid approach because management pushes for documentation and defined milestones associated with a waterfall methodology, while the software professionals want to use the short sprints and extremely rapid prototyping that are part of Scrum.

One of the online interviewees reinforced the experiences of interviewees from ORG1 by stating that “we experiment a lot. We invented our own methodology for software maintenance planning throughout the company, called Big Scrum.” Another online interviewee stated “We looked at our company culture, learned about what was out there, and meshed the two (heavyweight and lightweight methodologies) together.”

Another interviewee pointed out: “I learned how to develop software in college, but we didn’t do any maintenance. The methodology they taught me in school never quite fit what I do in maintenance, so I had to make my own.”

5.3. Relative use of methodology

Defining the relative use of a particular methodology or fixed process was difficult because some interviewees hesitated to discuss how much they used the organization’s pre-defined process. Those interviewees who worked in an organization that left the choice of a methodology to the individual were willing to discuss their relative use, while those who worked in organizations where it was a management choice were more circumspect.

Interviewees from ORG1 used a computer program to help them decide which methodology to use for a particular project. In addition, they have a whole team dedicated to help other teams select a methodology and then facilitate its use. Many members of that organization stated that the methodology selected by the computer program and/or methodology team was helpful, but that all facets of the selected methodology may not be applicable to a particular project, so individuals elected to modify the methodology without informing management.

ORG2 is ISO certified and the processes used are critical to maintaining certification. Interviewees were somewhat unwilling to discuss their actual application of the standard processes. It was implied that it is relatively easy to document the use of a process after work is completed, making it appear that a process was followed fully. It was also implied that the relative quality of the product obtained from using the standard processes was not affected, so the processes appeared to the interviewees as “bureaucratic steps.”

The vast majority of interviewees believed that they are using a process to perform software maintenance activities. According to one interviewee from ORG2: “Management tells us which methodology to use for a particular project, but no one has time to sit over your shoulder and watch how you work...I always use a process, but it changes from project to project depending on what I need...I learned methods in school, I’ve been doing this for 12 years, I read what’s new, and I make processes that actually work...That’s what I use.”

Almost all interviewees felt very strongly that they applied a “process” to their software maintenance work, but many hesitated to brand it an actual “methodology.” It appeared that the term “methodology” required more stringent requirements (i.e. tools, templates, clear steps) than was incorporated in their personally defined processes. But the interviewees believed quite adamantly that they followed a personally defined process to maintain software.
5.4. Factors affecting the selection of a methodology for maintenance

Existing literature has identified a number of factors that might affect the selection of an appropriate methodology [5,9,11]. Our findings are that while the interviewees mentioned a large number of factors, there were just a few very strong and discernible factors that dictated the actual choice and/or refinement of methodology.

For example, interviewees stated that the size of a company, size of a project team, and size of a software product affected methodology choice. However, how size influences choice was unclear. While some interviewees said that large projects had to be structured with more documentation as dictated by heavyweight methodologies, others believed that large projects and groups were best sub-divided into smaller groups and then executed with lightweight methodologies. Some interviewees working for large companies stated that their organization had to use heavyweight methodologies to cope with managerial bureaucracy, while other interviewees stated that their large companies had to use lightweight methodologies to remain nimble and able to compete with other, smaller organizations. While size was often mentioned, it was not clear how size actually affected methodology choice. It appeared that other factors, such as organizational structure or personal expertise were influential, and size was a surrogate that temporarily hid those other factors.

The three primary factors dictating choice of methodology for software maintenance are depicted in Figure 3 and described below.

5.4.1. Change Factors. Existing literature prescribes the use of lightweight methodologies to incorporate flexibility in the maintenance process [5,9,15]. The interviewees frequently mentioned the importance of environment, product and requirement change as a critical factor for choosing a methodology. They emphasized how often the environment (technical, governmental, managerial) changes, as well as how often users/clients change their minds about the functionality of a given product. They stressed the need for a flexible methodology that would help them cope with the immense amount of change brought by every new project.

Many lauded the inherent flexibility of lightweight methodologies and the avoidance of unnecessary planning and documentation. They commented that products were delivered more quickly using lightweight methodologies. They thought that the transparent processes and people centric focus of lightweight methodologies worked best in the highly dynamic environment of software maintenance.

But as absolutely critical as it is for software professionals to cope with change, that factor does not have a clear and consistent affect on the choice of methodology for software maintenance activities. Some interviewees dreaded too much flexibility and spoke about how freezing the requirements at a defined period of time in the heavyweight methodologies helped them cope with change. Others stated that the project management documentation affiliated in their minds with heavyweight methodologies helped customers better understand the importance of not changing requirements at the last minute. Some stated that the lightweight methodologies were a “nightmare” because they provided too much flexibility in the eyes of their customers. They commented that the endless rapid cycle iterations encouraged more change until “we run out of money, patience, and time...and finally get to stop changing everything.”

While change itself does not yield an easily dichotomous evaluation (more change, use lightweight methodologies; less change use heavyweight), it does affect the choice of a methodology. Change factors are mediated by the perception of the developer, as depicted on Figure 3. It is how the developer views the factors of change that yields a more consistent methodological choice. As stated by an interviewee from ORG4: “You wouldn’t think that much changes in county government, but you’d be wrong. We grow, we need new processes, we have new management who want new reports...It all depends on how you think about change. I embrace change. I love it – it makes...
my job new. I love to work with my clients to give them new reports and new information. It makes them happy and it makes me a hero – I get to be a hero all the time by just adding new data or a new screen.”

Contrast that with another interviewee from the same organization: “I have to freeze specifications as quickly as possible or my clients take advantage of me... Each time I have to change something it’s like looking at another wart in my code. Some of my systems are just a big pile of warts.”

Those developers who embrace change prefer the collaboration and communication that accompanies lightweight methodologies, while those who constrain change either choose heavyweight methodologies, or modify the process to obtain better control over “spurious” requests.

Coping with ambiguity is related to change for the people we interviewed. Ambiguous requirements for maintenance requests are the norm, not the exception, for our interviewees. The reason we list ambiguity on the model is that the interviewees felt very strongly that lightweight methodologies are most appropriate in situations with ambiguous specifications. The interviewees felt that it was better to create a working model as quickly as possible when specifications for maintenance are ambiguous. They stated that documentation and written specifications were of little help in nailing down specifications; that users/clients are best able to define what they need/want from an application if they can see a working model of that application. Thus, lightweight methodologies, with their focus on getting something working quickly, were more appropriate when the product specifications are ambiguous.

5.4.2. Developer Factors. Software development and maintenance is considered a knowledge based task frequently performed autonomously by people who are treated as professionals within their organizations [4]. As noted earlier, many software development professionals work without direct supervision; they are trusted to perform their work as they deem appropriate. Past research has shown that developers may modify a methodology without publicizing that information throughout the organization [30].

Our research found that developers ultimately choose how they will maintain an application. Management or process certification may dictate that a certain methodology be used, but the level of adherence to that methodology is at the discretion or manipulation of the developer.

We found that our interviewees had consistent factors they considered when selecting a methodology. Their initial training (either through formal education or job training) was critical. Many interviewees referred to their initial education as “the core of my approach,” “the basis of my processes,” or “my fundamental knowledge.” This initial knowledge and/or training was the lens used to view and modify new methodologies. Those with initial training in a particular type of methodology continued selecting that methodology. Interviewees with a formal education in methodology tended to shy away from declaring their current processes as a true “methodology” and preferred to discuss their general approach to maintenance.

Another key factor related to the developer was the relative enjoyment of collaboration with the users/clients of the system. Many interviewees discussed the need to collaborate more closely with users/clients when using lightweight methodologies. Those developers who found satisfaction working more closely with users/clients embraced lightweight methodologies. They felt that more frequent meetings and shorter cycle times helped them gain greater understanding of the application domain and ultimate use of the application. On the other hand, many interviewees stated that they prefer lightweight methodologies, but then explained how they “avoid” having to actually participate in meetings with those users/clients. A frequent modification to lightweight methodologies was the attempt to freeze requirements at a relatively early point in the process.

Finally, a key developer factor was the need to stay current or appear to be current. Many interviewees discussed the need to “be in fashion,” “to look current,” and not “appear to be dated.”

5.4.3. Organizational Factors. Organizational factors had a two-way relationship with choice of methodology. While these factors do have a strong relationship with the choice of methodology, it also appears that the choice of methodology influences the factors themselves. For example, the waterfall methodology divides tasks by function, such as “analysis,” “coding,” and “testing.” Some organizations follow this structure by creating positions, groups, or departments that reflect those tasks, such as a “business analyst” position or “quality assurance” group. The people who fill those positions or staff those groups then specialize in those tasks as dictated by the methodology. To adopt an agile methodology, the people would need to fill multiple roles concurrently, since agile methodologies frequently guide an individual to perform analytical, design, coding, and testing tasks concurrently during a given iteration. If an individual specializes in completing only one of those tasks, and the organizational design reinforces that specialization, it may be very difficult to adopt any other approach. Thus, while the choice of methodology may have initially influenced the organizational design, the
organizational design institutionalizes the original methodology.

We found that four organization cultural factors also play key roles in the choice of methodology: The level of trust among colleagues and management; the level of empowerment of individual developers; the level of interest in collaboration among colleagues; and the relative communication with users/clients. The level of trust and empowerment are related to each other. According to an interviewee, to use an agile methodology, "management must trust the developer and allow him/her to assume the risk/responsibility that comes with being the "decider" and at the same time the developer must be ready to assume that risk/responsibility."

Another strong cultural factor was the level of interest in collaboration among colleagues. Some agile methodologies and techniques encourage more frequent interactions among development personnel [15]. Organizations structured for more autonomous work (i.e. siloed areas for specific tasks) are better designed for heavyweight methodologies, while those organizations that support greater cross collaboration among groups are more conducive to lightweight methodologies. In addition, relative levels of collaboration may also be a personal preference on the part of IT personnel. Those who enjoy collaborating with their colleagues may prefer lightweight methodologies, while those who like to work alone or avoid the inevitable conflict that arises during collaboration and communication may find the required meetings onerous.

Finally, the relative communication level with users/clients is a critical cultural factor. If the organization does not encourage or force communication between users/clients and developers, then a methodology more skewed towards the heavyweight side of the continuum is chosen. It was remarked that "...a maintenance project may not be considered important enough to allow you frequent access to the client and good communication...is needed for continuous integration, testing, etc." As mentioned previously, the most frequent modification to lightweight methodologies was to reduce the amount of time spent communicating with users/clients by freezing specifications as quickly as possible. While some of the interviewees enjoyed working directly with users/clients, it was definitely a minority viewpoint.

5.5. Summary of findings

This study found that software maintenance professionals believe they are using personally defined and created processes to maintain software. They find it distasteful that someone would consider what they do to be ad-hoc, unplanned or ill-defined; while they might not use an organization-defined process, they are following a process. They place their personal process on a continuum of methodologies that range from heavyweight (waterfall) to lightweight (agile). Software maintenance professionals create their methodology based on their attitude towards change and how best to cope with change, their previous knowledge and training in software development, organizational culture, and the structure/management hierarchy of the organization.

6. Implications for research and education

This study explored how people responsible for maintaining business-oriented software select and use methodologies to guide their work. Applying the lessons learned from current studies in initial software development [9,10,19,24] we used a qualitative research method to better understand the issues confronted by those who are actually performing software maintenance activities, rather than relying exclusively on a management perspective of those activities. We found that initial answers given by interviewees to questions frequently required more explanation and description to better understand the meanings attributed to given words. Creating a shared vocabulary in software development would surely help us avoid confusion in both research and practice.

Our research demonstrated that education and training are critical to those developing and maintaining software. People use what they learn initially, so teaching methodology is as important as teaching technology if standard processes yield better outcomes. In addition, we need to incorporate maintenance tasks more fully into the curriculum in order to reflect the activities that are performed in the "real world." Finally, if organizations change to accommodate lightweight methodologies, people may be performing more diverse tasks requiring both comprehensive technical and domain knowledge to remain competitive in the job market.

7. References


