Understanding Factors Contributing to the Escalation of Software Maintenance Costs

Completed Research Paper

Michel Benaroch
Whitman School of Management
Syracuse University, Syracuse, NY
mbenaroc@syr.edu

Abstract

We examine the main drivers of software maintenance effort and cost. We use the 'Distributed Cognition' framework to hypothesize about how 'discovery work' in maintenance is effected by two types of cost drivers: system attributes (size, complexity, age, etc.) and personnel attributes (number of maintainers, location dispersion, etc.). We test our hypotheses using archival data about over 5,000 maintenance projects carried out between 2009 and 2011 on 412 different operational systems in a large financial institution. We find that personnel attributes are significantly more influential than system attributes. In particular, a marginal change in personnel factors is associated with effort growing much faster than cost, indicating an escalating marginal cost of spreading maintenance work across more maintainers and site locations. We also find, counter to expectation, that two system attributes are negatively linked to maintenance effort and cost. Implications of these findings for research and practices are discussed.

Keywords: software maintenance, cost drivers, system attributes, personnel attributes, discovery work, distributed cognition

1. Introduction

Between 50 and 80 percent of the lifetime cost of a software system is due to maintenance work (Boehm et al. 2000). Software maintenance is challenging because it requires understanding and documenting changes to the system, retaining the system's functional integrity during and after maintenance work, minimizing structural decay and growth in complexity of the system's code due to the changed or newly added functions, and protecting intellectual control over the system from deteriorating due to staff turnover and offshoring.

This paper studies the main cost drivers of software maintenance work and their relative contribution to costs. Our focus is on two groups of cost drivers. One covers application characteristics, including: system size, system age, software complexity, and so on. Another group covers maintenance personnel attributes, including: number of maintainers, their location dispersion, their skills diversity, and their system familiarity, among others. Unfortunately, little is known about the relative strength of cost drivers in both groups.

The software maintenance literature has not paid equal attention to these two groups of cost drivers. It has been traditionally preoccupied with the effect of system attributes on maintenance cost, on the widely accepted premise that maintenance productivity is directly related to the maintainability of the product (Banker et al. 1998). On the whole, a multitude of empirical studies confirm this association for multiple system attributes (e.g., system size and complexity). By contrast, far fewer studies examine the effect of maintenance personnel factors. What we know about these factors comes mostly from research on software development, not maintenance. However, there are notable differences between development and maintenance. For example, system familiarity plays a much greater role in maintenance than in development (Banker et al. 1987), and the number of IT personnel maintaining a system over its lifetime

could grow significantly larger than the number of original system developers (Thummadi, Lyytinen and Hansen 201). Such differences increase our interest in maintenance personnel factors, especially in light of the growth in using multi-site software maintenance and offshoring strategies.

To address this gap in the literature, this paper studies the effect of software maintenance cost drivers using proprietary archival data from a large financial services institution. The data documents software maintenance work carried out in the US, UK and several offshore locations, using over 7,500 maintenance projects that jointly charged over 15,000,000 person-hours over a three years period (Jan. 2009 - Dec. 2011). These projects support over 400 different software systems. The data we use include: ongoing UNIT (hour) and USD charges made per project per person per week; system attributes such as age, size, complexity, change evolution frequency, and information quality constraints (e.g., meeting stringent government regulations); and maintenance personnel characteristics such as the number of maintainers working on a system, their site locations, and their skills diversity.

The paper has two distinguishing features. First, recognizing the importance of 'discovery' in maintenance (Davison et al. 2000), we use distributed cognition theory to understand how discovery challenges impact maintenance productivity. Discovery is the process of learning what one needs to know in order to complete effectively a maintenance task subject to software design constraints underlying an existing system. Compared to software development, discovery in maintenance presents unique challenges that become more crucial as a system and its environment evolve and when the maintenance organization is subject to personnel migration, turnover, and geographic dispersion. Distributed cognition theory permits suitably reasoning about the effects of such settings on software productivity (Flors and Hutchins 1992; Hansen, Robinson and Lyytinen 2012). Another unique feature of this study is the use of both hour and dollar charges per project per person. We are able to avoid problems related to aggregating the effort of experienced and novice maintainers (Krishnan et al. 2000; Banker et al. 1987).

Our analysis reveals the relative strength of the main cost drivers in software maintenance. We find that maintenance personnel attributes are substantially stronger cost drivers than system characteristics. In particular, while a marginal change in system attributes results in both effort (hour charges) and cost (USD charges) growing slowly and at the same pace, a marginal change in maintenance personnel factors results in effort growing much faster than cost. This indicates the escalating marginal cost of spreading maintenance work across more maintainers and more site locations, even when involving cheaper labor and offshoring.

With regard to system attributes, our results also indicate that information quality constraints and system age are negatively related to maintenance effort and cost. For information quality constraints, the explanation may relate directly to the importance of personnel cost drivers. IT managers at our research site explained that these systems require greater care and, therefore, tend to be maintained by the same smaller and localized teams (i.e., personnel with higher system familiarity). For system age, although the result is contrary to the conventional thinking (Lehman 1996), a few studies do report the same negative link we observed (Jørgensen 1995; Jones 2012; Espinoza et al. 2007). One explanation could be that some maintenance work is done to reduce the effect of age on complexity.

Our study has important implications for software maintenance research and practices. It informs about the relative importance that should be given to the main cost drivers in planning and budgeting maintenance work for existing and new systems, in policies for retiring aging systems, in strategies for sourcing maintenance work, and so on. Of particular importance is the need to balance gains expected from assigning maintenance work to lower wage labor or offshore locations against the escalating marginal cost of dispersing maintenance work across more maintainers with lower system familiarity and greater location dispersion.

The paper proceeds as follows. Section 2 reviews literature on maintenance cost drivers. Section 3 develops the research model and hypotheses. Section 4 presets the data, empirical analysis, and results. Section 5 discusses the main findings, their implications, limitations, and directions for future research.

2. Literature Review

This section reviews what we know about cost drivers, or factors driving the effort and cost, in software maintenance.

2.1. Maintenance Cost Estimation Models

Models for estimating software maintenance effort and cost are one source of information about cost drivers. These models vary on the scope of maintenance work they target: a single maintenance *task*, a system *release*, or routine maintenance over a long *phase* or even the entire maintenance *phase* (Nguyen 2010). Unfortunately, these models insufficiently inform us about the relative strength of cost drivers.

Task- and release-level estimation models have a common trait, namely: they treat 'task size' as the dominant cost driver. Specifically, as seen in (Nguyen 2010), most of these models include task size as the only cost driver, where size may be measured as the sum of added, modified and deleted SLOC (Basili *et al.* 1996), function points (Sneed 1995), or modules or subsystems (Ramil and Lehman 2000, 2003). These models typically focus on maintenance tasks of one specific type, for example, corrective tasks (De Lucia et al. 2005; Basili 1996; Sneed 2004) and adaptive tasks (Fioravanti and Nesi 2001).[1] Focusing on a single type of maintenance task simplifies the estimation of effort and lowers the need to rely on additional cost drivers. In fact, Abran (2000) shows that project effort and functional size are correlated strongly enough to yield good models that use only size as an input. Select models use one or two additional cost drivers, primarily software complexity and software quality (e.g., Sneed 1995, 2004).

Phase-level models, which target routine maintenance work of all kinds performed over a lengthy period, consider more cost drivers. Examples are COCOMO, SEER-SEM, PRICE-S, and SLIM. In these models, maintenance effort is just one of the estimates produced for a new system, based on the same models used to estimate the system's development cost but with some adjustments. COCOMO for Maintenance, for example, employs the reuse model for new software development on the premise that software maintenance involves adaptation of existing code, just like reused software is adapted to a new system context. COCOMO for Maintenance uses 26 cost drivers, of which 22 are used in software development effort estimation (Boehm et al. 2000).[2] Among those cost drivers are numerous maintenance personnel factors, including ones unique to maintenance (e.g., 'software understanding' and 'programmer unfamiliarity'). However, extant phase-level models insufficiently inform us about the relative importance of cost drivers for two reasons. First, they take the amount of maintenance needed for a newly developed system of a known size to be a generic quantity of some sort. Second, they use forward-looking input values for cost drivers based on assumptions made when the new system is developed, not values reflecting conditions when the actual maintenance work is done.

2.2. Factors Effecting Software Maintenance Costs

Beside maintenance cost estimation models, a wealth of empirical studied identify a host of factors that influence maintenance cost but not their relative importance. Chief among these factors are characteristics of the system and of maintenance personnel.

System characteristics generally include the age of the system, its size, the complexity of the software architecture and code, the volatility of the system domain, and the need to meet special information quality constraints, among others. By now there is little debate about the expected impact of these

---

[1] Software maintenance addresses three groups of activities: *correcting* a system from actual errors; *adapting* a system to external changes in business rules, government regulations, and technology platforms; and, *perfecting* a system by improving the quality of software.

[2] The 22 cost drivers include effort multipliers and scale factors. The effort multipliers fall into four groups. *Product characteristics*: (1) Required software reliability, (2) Database size, (3) Documentation match to life-cycle needs, (4) Product complexity, (5) Required Reusability. *Platform characteristics*: (6) Execution time constraint, (7) Main storage constraint, (8) Platform volatility. *Personnel characteristics*: (9) Analyst capabilities, (10) Programmer capabilities, (11) Personnel Continuity, (12) Applications experience, (13) Platform experience, (14) Programming language experience. *Project characteristics*: (15) Use of software tools, (16) Multisite Development as a function of collocation & communication, (17) Schedule compression. The Scale Factors are: (18) Precedentedness, (19) Development Flexibility, (20) Architecture / Risk Resolution, (21) Team Cohesion, and (22) Process Maturity.

characteristics on maintenance effort and cost. Generally, they are all argued to have a positive association with maintenance effort and cost, and there is abundant empirical evidence to support this assertion. One small exception might be system age: a few studies actually report that age is negatively associated with maintenance effort and/or cost (Espinoza et al. 2007; Jørgensen 1995; Jones 2012).

Maintenance personnel characteristics have a natural link with software maintenance effort. They generally include: the number of maintainers working on a system over a given period, maintainers' geographic and/or temporal dispersion, familiarity of maintainers with the system and each other, and the diversity of skills of maintainers. However, research on the effect of these characteristics is spares (Benestad et al. 2009), and some of its findings are conflicting (see Table 1). For example, some find maintenance effort or maintenance productivity to have a weak or no association with maintainers' system experience (Jørgensen 1995), while others find the association to be negative (Banker et al. 1998; Espinoza et al. 2007). Most importantly, this research is silent on the relative strength of maintenance personnel factors as cost drivers.

Table 1. Maintenance personnel factors studied and main findings

Study	Independent Maintenance personnel Variables	findings
Graves and Mockus (1998)	• *Developer id* (who performed the change task?)	No significant relationship with maintenance effort.
Jørgensen (1995)	• *System experience* • Maintenance experience (in general)	Maintainers experienced with the system had lower productivity than inexperienced maintainers.
Banker et al. (1998)	• *Team system experience* (percentage of members working on a maintenance project who had worked on the system for three or more years)	Negatively related to maintenance project effort
Banke et al. (1987)	• *Team capability* (percent of members working on a maintenance project rated 'high' by other members) • *Team system experience* (percent of members working on a maintenance project who are not system novices; had 24+ months system experience prior to the project) • *Staff loading* (number of work-months divided by total project duration; higher loading means higher parallelism and more communication on project)	Software maintenance productivity (Size / Maintenance Cost) positively associated with project team capability and team system experience, and negatively associated with staff loading
Banker et al. (1991)	• *Team capability* (--- " ---) • *Team system experience* (--- " ---)	Software maintenance productivity positively associated with project team capability, but not with system experience
Banker et al. (1993)	• *Maintainer skill* • *System experience*	Maintenance effort has a negative relationship only with maintainer skill, not system experience
Krishnan et al. (2000)	• *Team technical capability*	Significant positive link with maintenance productivity.
Herbsleb and Mockus (2003)	• *Developer span* (how many developers were involved in performing the change task?) • *Location* (where were human resources located physically?)	Larger, multi-site teams take longer to complete same maintenance task.
Schneidewind (2001)	• *Developer span* (how many developers were involved in performing the change task?)	No significant relationship with change caused defect.
Mockus and Weiss (2000)	• *Developer span* (no. developers involved in maintenance task) • *System experience* (no. changes made by developer on the same system)	System experience negatively related to probability of (post maintenance) software failure.

3. Theory and Hypotheses

This section presents our theoretical lens and research hypotheses. Our premise is that much of the effort and cost associated with maintenance is linked to the need to do 'discovery' work. Building on this premise we use the 'distributed cognition' framework to theorize about how different cost drivers intensify the need for discovery work and, in turn, increase maintenance costs.

3.1. 'Discovery' in Software Maintenance and Distributed Cognition

A key source of difficulty in software maintenance is *discovery*, the process of learning what you need to know in order to address a task effectively. *Discovery*, also called "getting started" effort (Fraser et al. 1997) typically consumes between 30% and 70% of the time required to complete a maintenance task (Davison et al. 2000; Pfleeger 2002, p. 475; Pigoski 1996, p. 35). In software development, discovery centers around the capture of system requirements and their specification, based on which a system is then designed (Hansen et al. 2012). Discovery in maintenance in addition entails understanding the existing software design and its underlying design principles, methods, and tools as well as all changes previously made to the system (Davison et al. 2000). Thus, discovery in maintenance requires mastering not only knowledge of the system domain and system requirements, but also knowledge of the system architecture, "library" of system modules and components, design paradigm(s) of the original development, and the software development environment and maintenance tools. Mastery of this knowledge is the product of continuous work on the system, interaction with other developers and maintainers of the system, reading documentation, reading code, experimenting and debugging in a lab setting or with a simulator, participating in reviews/testing, building simplified examples, and so on (Davision et al. 2000).

Distributed Cognition (DC) is a framework that offers a rich theoretical lens for understanding how challenges in discovery impact software maintenance costs. DC is motivated by research on teams engaged in complex tasks (Flors and Hutchins 1992, Hansen et al. 2012). Complex tasks often require collaboration between a number of different individuals and technical artifacts. In these settings, information processing activities are distributed across members, and much of the cognitive workload is "shouldered" by the technical artifacts employed by group members. Together, group members and technical artifacts comprise a complex cognitive system within which effective interactions are necessary to successfully complete a task. These interactions facilitate exchanging task relevant information, in part, through the creation, modification and utilization of artifacts required for the task's completion. Numerous authors have applied the DC framework in the software development context. DC has been used to study software maintenance activities (Flor and Hutchins 1991), collaborative work (Rogers and Ellis 1994), pair programming teams (Sharp and Robinson 2006), information flow within a dispersed agile development teams (Sharp et al. 2012), team strategies in perfective software maintenance (Ramasubbu, Kemerer and Hong 2013), discovery of requirements in open source development (Thummadi et al. 2011), and discovery of requirements distributed across individuals, organizations, and artifacts (Hansen et al. 2012).

Recognizing that software development and maintenance can be highly social activities involving people, software code, and development tools (Flor and Hutchins 1991; Ramasubbu et al. 2013), the DC framework lends itself to three closely linked assertions (Hansen et al. 2012):

1. *Cognitive processes are distributed socially across members of a group*, where each member may play a specific role with respect to the processing of information and initiation of cognitive actions. This idea of social distribution has obvious ramifications: the development and comprehension of software systems is a function of how well the 'distributed cognitive system' performs as a whole. Individuals working together on a collaborative task are likely to possess different kinds of knowledge and skills, and so they will engage in interactions that allow them to pool the various resources necessary to accomplish the tasks. Thus, the ability to bring together individuals from a wide variety of technical and functional domains is essential. Moreover, sharing access and knowledge enables the coordination of expectations and of actions (Yvonne and Ellis 1994).

2. *Cognitive processes intertwine internal and external representational structures*. Individuals and groups integrate their internal representations with external representations of the environment as part of their thought processes. Thus, socially-distributed cognitive processes are likely to employ both internal and external structures during individually-intensive cognitive tasks. External structures in software maintenance include the software code itself, its documentation, and other structures that materialize in CASE tools. Consequently, the existence of such artifacts and their quality could both *enable* and *constrain* software maintenance work.
3. *Cognitive processes may be temporally distributed as well* (i.e., cognition is path dependent). The distribution of cognition over time implies the use of both social transmission (e.g., project team interaction) and material artifacts (i.e., legacy code, software frameworks) to support memory and cognitive structures needed to make use of requisite knowledge for completing a task. Indeed, maintenance efforts draw heavily upon requirements, software designs and other formal artifacts inherited from earlier development and maintenance efforts. Moreover, like in open source development, maintenance efforts may rely increasingly on evolving electronic media artifacts for social transmission and sharing of requisite knowledge, for example, community forums, email threads, chat rooms, and bulletin boards (Thummadi et al. 2011).

3.2. Hypotheses

The distributed cognition lens is next to speculate about how maintenance costs are effected by the two classes of cost drivers of our concern: system attributes and maintenance personnel factors.

3.2.1. System Characteristics

It is widely accepted that maintenance productivity is directly related to the maintainability of the software product – its size, complexity, age, and so on.

System size is a key indicator of maintenance cost (Banker et al. 1998; Boehm et al. 2000). Larger systems simply require more maintenance on an ongoing basis. Moreover, in larger systems, more components need to be updated, and it is more difficult to know all parts of the software that will be affected by changes. Therefore, maintainers need to expend more intellectual effort and time. They also need to understand both functional aspects of the software base and technical details of the associated code (e.g., file names, variable names, and interface requirements). All of this translates into more time and effort spent on discovery work.

System complexity, which generally refers to traits of data structures and procedures within a software product, is another major driver of maintenance cost (Banker et al. 1993, 1998) and error frequency (Ohlsson and Alberg 1996). Put in DC terms, the primary artifacts – the software code itself – becomes more complex and thus more difficult to integrate with maintainers' internal representations. In other words, software complexity increases the amount of information that needs to be processed to carry out maintenance activities. In addition to comprehending each of the individual software modules, maintainers must expend additional cognitive effort to understand how the modules are connected through input/output relationships and envision how the changes made will affect various modules (Banker et al. 1993, 1998). Greater complexity increases the number of those input/output relationships modules and obscures them, also necessitating more testing to ensure that modified modules do not negatively affect related modules (Boehm-Davis et al. 1992, Gill and Kemerer 1991).

System age, too, generally has a positive link with maintenance effort and cost. The number of maintenance activities is reported to increase with age (Barry et al. 2007). As we said, most systems are subject to continuous growth and continuous defect repairs that gradually degrade the software structure and increase its complexity. In DC terms, the quality of artifacts (code, documentation, design charts, etc.) degrades, making it harder for maintainers to integrate internal and external representations. Hence, over time more discovery work is needed since the system becomes less understood by both maintainers and users. This, in turn, also increases the rate of so-called bad fix injection and the amount of maintenance work necessary to cope with this phenomenon.

Information quality constraints refer to special system characteristics that imposes greater demand on maintenance resources, both relative to discovery work and to the actual software change. Examples include satisfying government regulations (e.g., SOX, HIPAA), handling sensitive (customer) data, and meeting unusually high software reliability needs. Some maintenance cost estimation models use special parameters to adjust their cost estimates accordingly, for example, 'required software reliability' in COCOMO for Maintenance (Boehm et al. 2000) and 'required maintenance rigor' in SEER-SEM (Galorath 2002).

Software volatility reflects the degree of change of the system domain and technological environment. It has been used in a variety of contexts as a measure of the frequency and intensity of system changes (Heales 2000; Barry and Slaughter 2000). Naturally, this factor is a fundamental driver of software maintenance costs (Banker and Slaughter 2000; Barry and Slaughter 2000).

In summary, every one of the above system characteristics leads us to postulate that:

> **Hypothesis 1**: *System characteristics – size, complexity, age, information quality constraints, and software volatility – are positively associated with software maintenance effort and cost.*

3.2.2. Maintenance Personnel Characteristics

We focus on three primary personnel characteristics that could be linked to maintenance effort and cost – number of maintainers who worked on a system over a given period, their geographic dispersion, and their skills diversity.

Number of maintainers is a primary cost driver in maintenance (Herbsleb and Mockus 2003). In software development, adding developers contributes more productive resources, but it also makes it more difficult for all developers to interact and to integrate their individual work into a single working product (Herbsleb and Grinter 1999). In software maintenance, an additional and more crucial issue is declining system familiarity and declining familiarity among maintainers. A system in its first year or two of operation is maintained by people from the original development team who are relatively familiar with the system, but as time passes personnel migration and resource constraints necessitate bringing on board maintainers with little or no familiarity with the system. Declining system familiarity makes it harder for maintainers to integrate external representations with their internal representations. This makes it more difficult to pinpoint swiftly and accurately where in the software code changes need to be made (Banker and Slaughter 2000), and complicates the implementation, testing, and integration of changed into the software code base (Curtis et al. 1979, 1988; Walz et al. 1993). Declining familiarity among maintainers has different implications. Maintainers who worked together on the same system start a new task with better expectations about each other, are more effective at locating specialized knowledge and coordinating expertise (Faraj and Sproull 2000), and have a better understanding of how their individual work contributes to each other's tasks. Declining familiarity among maintainers, therefore, means less effective communication (Cramton 2001), greater difficulty pooling the various necessary skills and resources, poorer coordination and division of labor, and greater uncertainty about other maintainers' capabilities. In sum, a larger number of maintainers translates into greater discovery effort and lower maintenance productivity.

Skills diversity, too, increases the need for discovery work. Typical maintenance tasks involve a common set of skills: business analysts, application developers, application support analysts, quality assurance analysts, and the like. Additional skills are necessary as system familiarity degrades, including: IT architecture managers, database management analysts, infrastructure services support analysts, business management analysts, information risk and business resiliency analysts, performance engineers, distributed computing engineers, and even IT program managers. Such additional skills are more costly but necessary to do discovery work on less familiar systems. Naturally, greater skills diversity also means increased coordination complexity and overhead.

Location dispersion of maintainers is another factor complicating discovery work. Maintainers may be drawn from diverse site locations because of resource constraints and/or attempts to reduce cost (e.g., offshoring). As a result, the frequency and timeliness of communications can be adversely affected, often due to the channeling of communications into less interactive media with fewer contextual references. A study of co-located and dispersed agile software development teams found that: (1) dispersed teams rely on more digital mediating artifacts (OneNote, recordings etc.), making information less accessible to

newcomers or outsiders than with co-located teams; and (2) meetings of dispersed teams used no clear equivalent to such artifacts as Story Cards and Wall, but rather screen sharing was used to focus attention and even then members often used just audio contact instead of sharing of screens (Sharp et al. 2012). Other results of location dispersion are less shared context for and understanding of task work (Cramton 2001, Hinds and Mortensen 2005), and fewer opportunities for informal and spontaneous communication (Kiesler and Cummings 2002; Herbsleb and Mockus 2003). Again, all of this translates into more coordination overhead and substantial delays when initiating contacts, locating and pooling expertise, and resolving issues (Carmel 1999). Sharp et al. (2012) specifically report that situational awareness is lower in dispersed teams because information flows were open (i.e., anyone could contact anyone else) but restricted (i.e., contacts had to be targeted and explicit), and individuals were the ones to decide what artifacts to share and with whom (unlike in co-located teams where social context played an important role). More broadly, Herbsleb and Mockus (2003) found that distributed work items take over twice as long to complete compared to when work is co-located. The explanation is twofold. First, co-located members interact with more people working on the same project, get useful work-related information through casual conversations, and receive from coworkers timely information about changes in current plans. Second, distributed members have more difficulties identifying distant colleagues with needed expertise and to communicate effectively with them. Thus, distributed maintenance work is likely to require more people in order to tap all the expertise needed to complete those tasks. The main implication is that distributed maintainers have less system familiarity and familiarity with one another. Moreover, distributed maintainers are less aware of the work going on in other site locations and are not well positioned to ask questions when the need arises, making it harder to avoid software changes that conflict with code written and maintained remotely. Therefore, more work would be necessary to resolve a greater number of conflicts.

In light of the above, we formulate our second hypothesis:

> **Hypothesis 2**: *Maintenance personnel factors – number of maintainers, their skills diversity, and their location dispersion – are positively associated with software maintenance effort and cost.*

As organizations become more concerned with combating the effect of software entropy due to aging and increased complexity, they turn some of their maintenance attention to perfective activities aimed at improving software quality (e.g., system re-engineering). This assertion is supported by studies observing a negative association between system age and maintenance effort and/or cost (Jørgensen 1995; Jones 2012). In any event, such efforts increase the demand for maintenance personnel, and the possibly that they would be more geographically distributed. Thus, while the impact of system attributes on maintenance effort and cost may reach and pick and start diminishing at some point, the impact of maintenance personnel attributes may continue increasing over time. In this light, we postulate that:

> **Hypothesis 3**: *Maintenance personnel factors are more strongly associated with software maintenance effort and cost* than *system characteristics.*

4. Data Analysis and Results

This section presents the data and analyses used to test the hypotheses as well as the results. Examination of the data leads us to log-transform most measurement items in the data, consistent with past research, and to map some of those items to two orthogonal factors. Then, we use ordinary least square regression (OLS) to test the hypotheses and verify the robustness of results using numerous statistical diagnostics. Finally, we report the main results in relation to the research hypotheses.

4.1. Data and Descriptive Statistics

To test the research hypotheses, we use archival maintenance data from a large financial institution. Our data cover maintenance work done on 412 different operational systems over a 3-year period (2009-11). In our research site, a system may be subject to ongoing maintenance activities of all kinds (corrective, adaptive, and perfective), carried out by multiple maintenance projects. Every project is executed by maintainers having different skills (or job titles) and working at different site locations. We compute the cost of a project maintaining a particular system as the sum of daily person-hours (UNITs) and US dollars (USD) charged for every person working on the project.

Data on hour and USD charges come from the organization's project reporting system, offering two benefits. First, these data are used to chargeback labor costs to requesting system owners, so it is accurate enough for the purposes of this study. Second, inclusion of hour and dollar charges per project per person helps to avoid problems related to aggregating the effort of experienced and novice maintainers (Krishnan et al. 2000; Banker et al. 1987). Using data on both UNIT and USD charges per project *per person* allows us to (indirectly) characterize the actual work-hours expended along experience and skill level.

The variables we use and their item measures in the data are listed in Table 2. The maintenance costs of a system over the 3-year period is the dependent variable, measured by the total UNIT and USD charged by all projects servicing that system for years 2009-11.

Table 2. Variables and their definition

	Variable	**Definition & Item Measures** (shown in parentheses)
System Attributes (independent)	Age	Number of years from a system's go-live date (till Dec. 2011)
	Size	System size measured by the: number of source lines of code (SLOC), number of classes (NoClasses), and number of functions (NoFunctions).
	Complexity	Software complexity measured by a complexity 'density' factor (ComplexDensity), a composite factor of class complexity, function complexity, and more granular complexity measures
	Information Quality Constraints	Indicator of whether a system handles sensitive customer data, including financial holdings information, account information, and personal details (PersonInfo).
	Software Volatility	System's change evolution scale and frequency, proxied by the number of maintenance projects (NoProjects) over a given period; past research similarly measured software volatility as the number of enhancements (projects) per unit of application over a specified time frame (Banker and Slaughter 2000; Zhang et al. 2003).
Personnel Factors (independent)	Number of Maintainers	Number of different IT personnel who did maintenance work on a system over a given period (NoMaintainers).
	Location Diversity	Number of site locations where the maintainers are located (NoLocations).
	Skills diversity	Number of different job titles of maintainers (NoJobTitles).
System Maintenance Costs (dependent)	Maintenance Effort	Sum total of daily person-hours charged for every person working on projects maintaining a system during 2009-11 (AppUNIT).
	Maintenance Cost	Sum total of daily US dollars charged for every person working on projects maintaining a system during 2009-11 (AppUSD).

Preliminary analysis of the data shows most item measures to have asymmetric distributions with a long right tail. Moreover, univariate analyses show those items to have power-type relationships with UNIT and USD costs. This pattern is consistent with the economies and diseconomies of scale reported in the literature on software development and maintenance (Banker and Slaughter 1997; Banker et al. 1994; Barry et al. 2007; Boehm et al. 2000; Herbsleb and Mickus 2003). Consequently, we log-transform those items and identify them as such in the remaining analyses.

Table 3 offers descriptive statistics and the pairwise correlations matrix. Most log-transformed item measures are strongly correlated with the dependent variables, but correlations among two subsets of items are also high. We hence run a factor analysis to distil two orthogonal factors for those subsets: (1) SizeComplexity (log SLOC, log NoClasses, log NoFunct, and log ComplexDensity), and (2) MaintainPersonnel (log NoMaintainers, log NoLocations, and log NoJobTitles). The Cronbach's alphas for these orthogonal factors are close to 1.0, indicating their internal consistency reliability and convergent validity (see Table 4). Finally, because these log-transformed items are on relatively comparable scales, values for the orthogonal factors are computed simply as averages of their item members weighted by item loadings.

Table 3. Descriptive statistics and correlation matrix

Variable	N	Avg.	StdDev	Min	Max	1	2	3	4	5	6	7	8	9	10	11	12
1. log AppUNIT)	412	9.68	1.87	2.37	13.70	1.000											
2. log AppUSD ($)	412	12.73	1.93	3.37	16.75	**0.936**	1.000										
3. log NoProjects	412	2.57	1.18	0.00	5.04	**0.822**	**0.816**	1.000									
4. log Age	412	1.76	0.63	0.00	3.74	**0.186**	**0.179**	**0.299**	1.000								
5. log ComplexDensity	412	9.16	1.56	4.43	13.55	**0.399**	**0.370**	**0.388**	**0.183**	1.000							
6. log SLOC	412	11.04	1.55	6.24	15.25	**0.403**	**0.377**	**0.386**	**0.179**	**0.958**	1.000						
7. log NoClasses	412	6.13	1.45	1.79	9.75	**0.433**	**0.417**	**0.404**	**0.129**	**0.946**	**0.921**	1.000					
8. log NoFunctions	412	7.99	1.55	3.22	12.02	**0.396**	**0.370**	**0.375**	**0.152**	**0.972**	**0.927**	**0.966**	1.000				
9. PersonInfo	412	1.46	0.50	1.00	2.00	0.058	0.017	0.096	0.092	0.097	**0.132**	0.044	0.080	1.000			
10. log NoMaintainers	412	4.49	1.23	0.00	7.12	**0.828**	**0.788**	**0.453**	**0.259**	**0.343**	**0.347**	**0.363**	**0.329**	**0.102**	1.000		
11. log NoLocations	412	2.41	0.77	0.00	3.76	**0.746**	**0.691**	**0.391**	**0.261**	**0.338**	**0.344**	**0.343**	**0.320**	**0.111**	**0.915**	1.000	
12. log NoJobTitles	412	2.99	0.78	0.00	4.20	**0.780**	**0.703**	**0.370**	**0.241**	**0.330**	**0.333**	**0.335**	**0.307**	**0.109**	**0.924**	**0.912**	1.000

Correlations in bold face indicate a statistical significance at the level of <0.05

Table 4. Standardized Cronbach's Alphas

Factor (Construct)	Variable (Item)	Correlation with Total	Deleted Item Alpha	Cronbach's Alpha
SizeComplexity	log ComplexDensity	0.8874	0.8893	**0.877**
	log SLOC	0.8686	0.8908	
	log NoClasses	0.8829	0.8897	
	log NoFunctions	0.8737	0.8910	
MaintainPersonnel	log NoMaintainers	0.6506	0.9062	**0.907**
	log NoLocations	0.6397	0.9073	
	log NoJobTitles	0.6326	0.9081	

4.2. Analysis and Results

Consistent with the nature of our data, we adopted a log-linear specification for our statistical models. The specification test for non-nested models (the J-test) supported our log-linear over linear model specifications (Davidson and Mackinnon 1995). Thus, we use the next regressions to test the hypotheses.

$$LN(AppUNIT) = \alpha_0 + \alpha_1 LN(NoProjects) + \alpha_2 LN(SizeComplexity) + \alpha_3 LN(Age) + + \alpha_4 PersonInfo + \alpha_5 LN(MaintainPersonnel) \quad \textbf{(1)}$$

$$LN(AppUSD) = \beta_0 + \beta_1 LN(NoProjects) + \beta_2 LN(SizeComplexity) + \beta_3 LN(Age) + + \beta_4 PersonInfo + \beta_5 LN(MaintainPersonnel) \quad \textbf{(2)}$$

We estimated the two equations using OLS regression and summarized the results in Table 5. For robustness, we tested standard assumptions of the OLS estimators. The assumption of normality is not rejected for any of the models at the 1% significance level using the Shapiro-Wilik test (Shapiro and Wilik 1965). The presence of heteroskedasticity in both models was tested using White's (1980) test, and no evidence of it was found; the significance of coefficients has changed only marginally. The effect of multicollinearity was examined by computing a condition index for the entire model and variance inflation factors (VIF) for each of the independent variables (Belsley, Kuh and Welsch 1980), and no diagnostic problem was revealed; the condition index is 3.79 and all VIF values are below 3.26, well below the recommended thresholds for strong dependencies to start affecting the regression estimates. We did not detect any influential outliers in the equations using Cook's distance (Cook and Weisberg 1982) and the guidelines specified by Belsley et al. (1980). Finally, we also estimated both equations using seemingly unrelated regression (SURE), in case the error terms are correlated as a result of a common effect (Greene 2005), and the regression coefficients were very similar in sign, magnitude, and significance to the OLE estimates, indicating the absence of any correlation across the error terms. In light of these robustness test outcomes, we use the OLS estimates for the interpretation of our results.

Table 5. OLS regression estimation for maintenance effort and cost

Variable	*LN(AppUNIT)* Model			*LN(AppUSD)* Model		
	Est. Param. (α)	**Std Error**	**Pr>|t|**	**Est. Param. (β)**	**Std Error**	**Pr>|t|**
Intercept	4.444***	0.332	<.0001	8.290***	0.372	<.0001
LN(NoProjects)	0.722***	0.073	<.0001	0.980***	0.082	<.0001
LN(Age)	-0.218***	0.077	0.005	-0.225***	0.086	0.010
LN(SizeComplexity)	0.127***	0.034	0.000	0.105***	0.038	0.006
PersonInfo	-0.128	0.093	0.170	-0.266**	0.105	0.011
LN(MaintainPersonnel)	0.869***	0.093	<.0001	0.547***	0.104	<.0001
N	412			412		
Adj. R-Sq.	**0.740**			**0.696**		
F-value	242.96			195.12		
Pr > F	<.0001			<.0001		

***, **, * coefficient significant at the level of <.001, <0.01, <0.05, respectively

The sign and magnitude of coefficients reveal which of our research hypotheses is supported. First, system characteristics are significantly associated with maintenance effort and cost, but the coefficients for log Age and PersonInfo are negative (i.e., opposite direction to the one hypothesized). This means that Hypothesis 1 is only partially supported. Second, maintenance personnel variables have a positive and significant relationship with maintenance effort and cost, consistent with our expectation. Hypothesis 2 is therefore supported. Lastly, the magnitude of coefficients in both models indicates that system attributes (log NoProjects, log Age, log SizeComplexity, and PersonalInfo) are relatively weaker cost drivers than maintenance personnel attributes. For maintenance effort (AppUNIT), personnel factors have the largest coefficient; this is so even for the coefficient of log NoProjects, the variable indicating the change evolution frequency (or "amount" of maintenance work on a system). For maintenance cost (AppUSD), personnel factors have a coefficient larger than those of all system characteristics except for log NoProjects. These results on the most part support of hypothesis 3.

The dominance of personnel factors as drivers of maintenance effort is surprising. It lends credence to the importance of personnel factors in relation to discovery work in maintenance. However, more insightful is the fact that personnel factors influence maintenance effort (AppUNIT) more strongly than maintenance cost (AppUSD). It could be explained quite simply: if one of the reasons for expanding the number of maintainers and their geographical dispersion is to lower costs, then the rate of maintenance cost reduction is outpaced by the rate of maintenance effort increase. To verify that this result is driven by maintenance personnel factors, we run a third OLS regression where the dependent variable is the ratio of cost to effort:

$$\text{LN(AppUSD)/LN(AppUNIT)} = \gamma_0 + \gamma_1\text{LN(NoProjects)} + \gamma_2\text{LN(SizeComplexity)} + \\ + \gamma_3\text{LN(Age)} + \gamma_4\text{PersonInfo} + \gamma_5\text{LN(MaintainPersonnel)} \quad \textbf{(3)}$$

The results are shown in Table 6. Only LN(MaintainPersonnel) has a significant coefficient, and this coefficient is negative. Thus, only LN(MaintainPersonnel) explains the difference in the growth rates of AppUNIT and AppUSD. Moreover, since both maintenance effort and cost increase in maintenance personnel factors, the negative coefficient of LN(MaintainPersonnel) implies that maintenance effort increase faster than maintenance cost.

5. Discussion

This paper investigates software maintenance cost drivers from the perspective of ongoing maintenance work that is carried out over a lengthy period. A primary insight from our analysis is that maintenance personnel factors (number of maintainers, their location dispersion, and their skills diversity) are substantially stronger cost drivers than system characteristics (age, size, complexity, and information quality constraints). In particular, while a marginal increase in software size or complexity results in both

effort (hour charges) and cost (USD charges) growing slowly and at the same pace, a marginal increase in the number of maintainers or location dispersion results in maintenance effort growing much faster than maintenance cost. In fact, the effect of personnel factors on maintenance effort outpaces even the effect of the number of projects indicating the "amount" of maintenance done on a system. One plausible explanation is simple: while use of extra personnel and site locations might be driven by reliance on cheaper (offshore) labor aimed at lowering maintenance cost (USD charges), it also results in more discovery work and coordination inefficiencies that increase the maintenance effort (hour charges) at a pace that exceeds the cost savings.

Table 6. OLS regression estimation for maintenance cost-to-effort ratio

	LN(AppUSD) / LN(AppUNIT) Model		
Variable	**Est. Param. (γ)**	**Std Error**	**Pr>|t|**
Intercept	**1.643*** **	0.041	<.0001
LN(NoProjects)	0.008	0.009	0.3473
LN(Age)	0.008	0.009	0.3622
LN(SizeComplexity)	-0.005	0.004	0.1707
PersonInfo	-0.007	0.011	0.5477
LN(MaintainPersonnel)	**-0.076*** **	0.010	<.0001
N	412		
Adj. R-Sq.	0.323		
F Value	40.25		
Pr > F	<.0001		

***, **, * coefficient significant at the level of <.001, <0.01, <0.05, respectively

Our results also indicate that information quality constraints and system age are negatively related to maintenance effort and cost. For information quality constraints, the explanation may relate directly to the importance of personnel factors. Our expectation was that systems facing information quality constraints (e.g., handling sensitive confidential data) require more maintenance care and, therefore, more effort and cost. However, our opposite result triggered IT managers at our research site to speculate that those systems are more "sensitive" and, hence, tend to be maintained by the same people in the same small number of site locations (i.e., by maintainers with higher system familiarity and low geographic dispersion). Our preliminary examination found some empirical support for this explanation, but a more careful examination is necessary.

Also surprising is the negative relation of system age with maintenance effort and cost. This is contrary to Lehman's law according to which system age, a direct measure of software entropy, drives up costs over time (Lehman 1996). Yet, a few studies report the same negative link we observed (Jørgensen 1995; Jones 2012; Espinoza et al. 2007). One common explanation is that much maintenance work is 'perfective' in nature and aims at reversing the effect of age on software complexity. More research is needed to validate this explanation. We are exploring the possibility of classifying maintenance projects by their type of work (corrective, adaptive, or perfective) and digging deeper into the relationship with system age. In particular, our post-analysis examination of suggests that the negative relationship with age holds only for relatively older systems, not systems in their first five or six years of operation.

Overall, our results have three primary implications for research and practices. First, less importance should be given to system attributes in making maintenance budget allocations, in formulating system retirement decisions and policies, and in choosing the mix of systems in the IT portfolio. Second, work allocation and personnel assignment practices, including the choice to off-shore work, ought to be sensitive to the fact that IT personnel factors have an overall negative marginal effect on maintenance cost. Third, with regard to system age, it may make sense to assume that the amount and types of maintenance work done on a system vary across the system's lifecycle stages. Lastly, in relation to the theoretical lens used in this paper, we and other researchers have found the distributed cognition framework to be helpful in better understanding the effect of various factors in the software development and maintenance context, however, additional research is needed to develop more direct connections

between elements of the framework (team size, degree of distribution, number of artifacts, age and quality of artifacts, etc.) and the degree of efficiency and effectiveness of maintenance work.

We cannot conclude without pointing out three limitations of our work and how they may be overcome with additional work. First, our data comes from a single organization. This could somewhat limit the generalizability of our model, especially in light of the possibility that the model is influenced by organization-specific practices. Second, with regard to the assessment of marginal impact of individual-unit cost drivers, our analysis essentially assumes the "average" unit of every cost driver in the model. For example, when estimating the extra cost associated with increasing the number of maintainers by one additional person to get the same work done, the model does not consider the qualifications or familiarity of that person with the system. In principle, the richness of our data permits analyzing maintenance effort and cost *by person*, based on skills (or job title) and past system familiarity. We plan to do so in follow-up work. Third, our analysis lumps all 412 different systems together, rather than distinguish between classes of presumably similar systems. For example, systems can be classified by line of business or by owner (e.g., business vs. IT organization). Our preliminary analysis indicates no qualitative differences across classes of systems, but a closer examination is warranted.

References

Abran A., Silva I., Primera L., "Field studies using functional size measurement in building estimation models for software maintenance," *Journal of Software Maintenance and Evolution*, 14(1), 2002, 31-64.

Banker RD, Slaughter SA. "The moderating affects of structure on volatility and complexity in software enhancement," *Information Systems Research,* 2000; **11**(3):219–240.

Banker R., Davis A., and Slaughter S., "Software Development Practices, Software Complexity, and Software Maintenance Performance: A Field Study," *Management Science*, 1998.

Banker R., Datar B., Kemerer K., and Zweig G., "Software Complexity and Maintenance Costs," *Communications of the ACM*, 1993.

Banker, R. D., S. Datar, and C. F. Kemerer, "Factors AffectingSoftware Maintenance Productivity: An Explanatory Study," *Proc. Eighth International Conf. on Information Systems*, ACM, New York, 1987, 160-175.

Barry E.J., Kemerer C.F. and Slaughter S.A, "How software process automation affects software evolution: a longitudinal empirical analysis," *Journal of Software Maintenance and Evolution: Research and Practice,* 19(1), 2007, 1-31.

Barry E, Slaughter SA., "Measuring software volatility: A multi-dimensional approach," *Proceedings of the 21st International Conference on Information Systems*. Association for Information Systems: Atlanta GA, 2000: 412–413.

Basili V.R., Briand L., Condon S., Kim Y.M., Melo W.L., and Valett J.D., "Understanding and predicting the process of software maintenance releases," Proceedings of International Conference on Software Engineering, Berlin, Germany, 1996, 464–474.

Belsley D.A., E. Kuh, R. E. Welsch. *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity*. Wiley and Sons, New York, 1980.

Boehm B.W. et al., *Software Cost Estimation with COCOMO II*, Prentice Hall, 2000.

Boehm-Davis D.A., Holt R.W. and Schulz A.C., "The Role of Pro-gram Structure in Software Maintenance," *International J. Man- Machine Studies*, 36, 1 (1992), 21-63.

Carmel, E. *Global Software Teams*. Prentice-Hall, Upper Saddle River, NJ, 1999.

Cook R.D. and Weisberg S., *Residuals and Influence in Regression*, Chapman & Hall, London, England, 1982.

Cramton, C.D. "The mutual knowledge problem and its consequences for dispersed collaboration," *Organization Science*, 2001, 12(3) 346–371.

Curtis, B., H. Krasner, N. Iscoe. "A field study of the software design process for large systems," *Communication of the ACM,* 31(11), 1988, 1268–1286.

Curtis B., "In search of software complexity," Workshop on Quantitative Software Models, 1979, 95-106

De Lucia A., Pompella E., and Stefanucci S., "Assessing effort estimation models for corrective maintenance through empirical studies," *Information and Software Technology*, 47, 2005, 3–15

Davison J.W., Mancl D.M., and Opdyke W.F., "Understanding and Addressing the Essential Costs of

Evolving Systems," *Bell Labs Technical Journal*, April–June 2000, 43-54.
Davidson R. and MacKinnon J., "Several tests for model specifications in the presence of multiple alternatives," *Econometrica*, 49, 1995, 781-793.
Espinosa A.J., Slaughter S.A., Kraut R.E., and Herbsleb J.D., "Familiarity, Complexity, and Team Performance in Geographically Distributed Software Development," *Organization Science*, Vol. 18, No. 4, July–August 2007, pp. 613–630.
Faraj, S., L. Sproull. "Coordinating expertise in software development teams," *Management Science,* 2000, 46(12) 1554–1568.
Fioravanti F. and Nesi P., "Estimation and prediction metrics for adaptive maintenance effort of object-oriented systems," *IEEE Transactions on Software Engineering*, 27(12), 2001, 1062–1084.
Flor N.V. and Edwin H.L., "Analyzing Distributed Cognition in Software Teams: A Case Study of Team Programming During Perfective Software Maintenance," in *Empirical Studies of Programmers: Fourth Workshop*, J. Koenemann-Belliveau, Ed., 1991, pp. 36-64.
Fraser S., Beck K., Booch G., Coplien J., Johnson R., and Opdyke W., "Beyond the Hype: Do Patterns and Frameworks Reduce Discovery Costs?" *Proc. 12th Annual Conf. on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '97)*, Atlanta, Ga., Oct. 5–9, 1997, pp. 342–344.
Galorath 2002, *SEER-SEM Software Estimation, Planning and Project Control - User's Manual*, Galorath Incorporated.
Gill, G. K. and C. F. Kemerer, "Cyclomatic Complexity Density and Software Maintenance Productivity," *IEEE Transactions on Software Engineering*, 17, 12 (1991), 1284-1288.
Graves TL, Mockus A. "Inferring change effort from configuration management databases," *Proceedings of the Fifth International Symposium on Software Metrics*. IEEE Computer Society Press: Los Alamitos CA, 1998; 267–273.
Greene W.H., *Econometric Analysis*, Fifth edition. Pearson Education, 2005.
Hansen, S.W., Robinson, W.N. and Lyytinen, K.J.; "Computing Requirements: Cognitive Approaches to Distributed Requirements Engineering," 45th *Hawaii international conference on system sciences*; HICSS 2012, 7, 5224-5233.
Heales J., "Factors affecting information system volatility," *Proceedings of 20th International Conference on Information Systems*. Association for Information Systems: Atlanta GA, 2000; 70–83.
Herbsleb JD, Mockus A. "An empirical study of speed and communication in globally distributed software development," *IEEE Transactions on Software Engineering* 2003; **29**(6):481–494.
Herbsleb J.D. and Grinter R.E., "Architectures, coordination, and distance: Conway's law and beyond," *IEEE Software*, 16(5), 1999, 63–70.
Hinds P. and Mortensen M., "Understanding conflict in geographically distributed teams: The moderating effects of shared identity, shared context, and spontaneous communication," *Organization Science*, 2005, 16(3) 290–307.
Jones C., "Software Maintenance, Sustaining Engineering, and Operational Support: Estimating Software Maintenance Costs for U.S. Army Weapons Systems," Office of the Deputy Assistant Secretary of the Army for Cost and Economics (ODASA-CE), 1-27 June 2012, PSM User' Group Meetings and Workshops, 29 July 2012
Jones C., "Geriatric Issues of Aging Software," *CROSSTALK: The Journal of Defense Software Engineering*, December 2007, 4-8.
Jørgensen M., "Experience with the accuracy of software maintenance task effort prediction models," *IEEE Transactions on Software Engineering*, 21(8), 1995, 674–681.
Jørgensen, M. and Sjøberg, D., "Impact of experience on maintenance skills," *Journal of Software Maintenance: Research and Practice*, 14, 2002, 123-146.
Lehman M., "Laws of Software Evolution Revisited," in European Workshop on Software Process Technology. Springer, 1996, 108–124.
Mockus A, Weiss D.M., "Predicting risk of software changes," *Bell Labs Technical Journal*, **5**(2), 2000, 169–180.
Nguyen V., Improved Size and Effort Estimation Models for Software Maintenance, Doctoral Dissertation, Computer Science Department, University of Southern California, December 2010.
Ohlsson N. and Alberg H., "Predicting fault-prone software modules in telephone switches," *IEEE Transactions on Software Engineering*, 22(12), 1996, 886–894.
Pfleeger S.L., "What software engineering can learn from soccer," *IEEE Software*, 19 (6), 2002, 64–65.
Pigoski T.M., *Practical Software Maintenance: Best Practices for Software Investment*, John Wiley & Sons, Inc., 1996.

Ramasubbu N., Kemerer C.F., and Hong J., "Structural Complexity and Programmer Team Strategy: An Experimental Test," *IEEE Transactions on Software Engineering*, 38(5), 2012, 1054-1068.

Ramil J.F., Continual Resource Estimation for Evolving Software, PhD Thesis, University of London, Imperial College of Science, Technology and Medicine, 2003.

Ramil J.F., "Algorithmic cost estimation software evolution", *Proc. of International Conference on Software Engineering*, Limerick, Ireland, 2000, 701–703.

Rogers Y. and Ellis J., "Distributed Cognition: An Alternative Framework for Analysing and Explaining Collaborative Working," *Journal of Information Technology*, **9**(2), 1994, 119-128.

Schneidewind N.F., "Investigation of the risk to software reliability and maintainability of requirements changes," *Proceedings of the 2001 International Conference on Software Maintenance*. IEEE Computer Society Press: Los Alamitos CA, 2001; 127–136.

Shapiro S. and Wilk M., "An analysis of variance test for normality," *Biomnetrika*, 52, 1965, 591-612.

Sharp H. and Robinson H., "A Distributed Cognition Account of Mature XP Teams," in *Lecture Notes in Computer Science 4044*, P. Abrahamsson, M. Marchesi, and G. Succi, Eds., 2006, pp. 1-10.

Sharp H., Rosalba G. and Grigori M., "Information flow within a dispersed agile team: a distributed cognition perspective," in *Agile Processes in Software Engineering and Extreme Programming Lecture Notes in Business Information Processing*, Volume 111, 2012, pp 62-76'.

Sneed H., "A cost model for software maintenance & evolution." *IEEE*, 2004, 264–273.

Sneed H.M., "Estimating the Costs of Software Maintenance Tasks," *IEEE International Conference on Software Maintenance*, 1995, 168-181

Thummadi B.V., Lyytinen K., Hansen S., "Quality in Requirements Engineering (RE) Explained Using Distributed Cognition: A Case of Open Source Development," *Proceedings of JAIS Theory Development Workshop*, 2011. (http://aisel.aisnet.org/sprouts_all/465)

Walz, D. B., J. J. Elam, B. Curtis. "Inside a software design team: Knowledge acquisition, sharing, and integration," *Communication of the ACM*. 36(10), 1993, 63–77.

Zhang X., Windsor J. and Pavur R., "Determinants of software volatility: a field study," *Journal of Software Maintenance and Evolution: Research and Practice*, 15, 2003, 191–204

White H., "Heteroskedasticity-consistent covariance matrix estimator and a direct test for heteroskedasticity," *Econometrica*, 48(5), 1980, 817-838.