



Introduction to the Special Issue on Managing Software Development and Maintenance

At the dawn of the new millennium, software managers face a particularly difficult set of challenges [1]. More than ever before, organizations depend upon computer software for their competitive survival. Large, complex, and inter-networked software systems play a critical role in many aspects of organizations' value chains. Users need software that can meet stringent requirements, can be produced quickly and productively, and can be easily maintained to keep pace with an ever-increasing demand for functionality, quality, and cost-effectiveness. The rise of the World Wide Web and electronic commerce have intensified the challenges of software development by dramatically shortening product development cycles and elevating time to market as a critical dimension of software development performance.

Managers and decision makers envision software systems developed with high quality, within budget, and without delays. However, "software is a place where dreams are planted and nightmares harvested, where terrible demons compete with magical panaceas, a world of werewolves and silver bullets" [2, p. 27]. Despite pressures for increased productivity, quality and timeliness, and the introduction of major software process innovations, many software projects continue to experience significant schedule delays, cost overruns, and quality problems.

In this environment, methodologies, tools, management techniques, and insights that lead to improved software productivity, quality, and timeliness are of primary interest. This special issue seeks to consolidate and disseminate leading research at the intersection of software engineering, management, and economics. The papers in this issue can be divided into several major areas of focus. The first set of papers examines the effectiveness of methodologies and tools used in software design, development and maintenance. The second set of papers evaluates new issues and techniques for software project management. The final paper evaluates the benefits and performance impacts of a large software system.

The papers examine software development and maintenance issues in a variety of organizations (such as Boeing, Hitachi, Fujitsu, and the Wyoming Department of Administration and Employment); in numerous industries (such as manufacturing, software, utilities, transportation, banking, and employment services); across new web-based as well as legacy development platforms; and in different countries (including the US, Japan, Hong Kong, Israel, and the UK). As a whole, the papers provide a number of interesting and important insights into software development and maintenance

from many different perspectives. In the following sections, we briefly review each of these papers in turn.

Methodologies and tools for software design, development, and maintenance

The first two papers in this set focus on evaluating the impact of methodologies and managerially controllable factors on software development and maintenance. Both papers examine actual software performance data collected in a variety of organizations and across a number of development platforms.

Nelson and Ghods evaluate the performance impacts of a vendor-supplied structured software development and maintenance methodology in a large manufacturing organization. Projects using this methodology are compared against projects in the same organization that do not use this methodology as well as projects across eleven other organizations. Findings indicate that the software systems developed and maintained using this methodology realized significant cost and quality performance benefits over those not using the methodology. However, these systems had lower levels of configuration management, a feature lacking in the methodology.

Banker, Datar, Kemerer, and Zweig analyze two years of error log data at a commercial site to identify managerially controllable factors that affect software reliability. They find that application systems that underwent frequent modification, were maintained by programmers with little application experience, had high reliability requirements, and had high levels of software complexity all realized high error rates.

Both studies have implications for the management of software development and maintenance. The work by Nelson and Ghods provides guidelines and measures to organizations seeking to evaluate software development and maintenance methodologies. The paper by Banker et al., implies that managers can implement a number of tools and methods to reduce software error rates, such as implementing release control, assigning more experienced programmers, and establishing and enforcing standards for software complexity metrics.

The second two papers in this set focus on developing and evaluating tools to solve particular design and development issues. One important issue in software development is reuse. The reuse of design specifications for business processes and information can help to alleviate bottlenecks in software development and reduce the cost of design. However, there can be many barriers to reuse. In their paper, Kobayashi, Onoda, and Komoda describe the creation of a library of standard business process and information specifications for reuse as business templates. The templates and tools created by Kobayashi et al., are directed toward workflow systems that automate and manage a total business process. The authors evaluate the effectiveness of their templates and tools using a case study of a workflow system for attendance management and expense reimbursement. They find that by applying the workflow business template and implementation support tools to reuse design patterns, the effort required for workflow system development can be drastically reduced.

Another important issue in software development is defect detection. Finding defects early in the software development process is important because the costs to repair defects increase exponentially the later in the life cycle that the defects are discovered. Software inspection is a valuable technique for early detection of defects in software products. However, tools to facilitate software inspection lack the ability to monitor the inspection process and provide the information needed to determine whether to terminate or extend the inspection process. Miller, MacDonald, and Ferguson describe their research in the area of computer support for software inspection and their development and evaluation of an automated inspection tool called ASSIST. They introduce a capture/re-capture feature into their inspection tool that draws upon insights learned from the capture and release of animals in the zoological domain. They evaluate this feature of their inspection tool using data from software inspection experiments. Although the capture/re-capture techniques show a tendency to underestimate the number of software defects, the authors find that the information from the tool is useful in helping the software inspector decide whether to continue or terminate inspection processes.

Both of these papers demonstrate the value of using carefully designed tools to facilitate the software development process.

New issues and techniques for software project management

The four papers in this set examine different issues and new techniques for software project management. The first paper by Gao, Itaru, and Toyoshima studies global software production. Many large companies have established global software production lines to develop and deliver software products using collaborative development processes involving multiple teams at different sites. In this paper, Gao et al., discuss the real world issues, experiences, and lessons learned in building and deploying a web-based problem information management system to support global software development in Fujitsu. Overall, they find that a web-based system provides a cost-effective infrastructure to support the management of concurrent development processes in global software production. The system facilitates information sharing, problem management, project management, and coordination. However, the system lacks support for other features that were found to be important in global software development, i.e., the synchronization of different problem procedures and active monitoring and tracking of problems.

The second paper by Barry, Mukhopadhyay, and Slaughter studies the common problem of scope creep in software projects. Over time, a dynamic environment contributes to the expansion of user requirements, increasing the scope and effort required to complete a project. In this study, Barry et al., develop and empirically evaluate a two-stage model to relate project duration and effort. They find a positive relationship between project duration and effort, implying that scope creep is more likely to occur the longer the duration of the project and the higher the rate of environmental change. Barry et al., demonstrate the practical implications of their study by showing how their model can be used in conjunction with time boxing techniques to better scope software projects.

The third paper in this set by Fichman and Kemerer proposes an activity based costing (ABC) approach to enable project managers to account for and recognize the gains from component-based software development. ABC creates a project management environment in which there are appropriate incentives for the development and reuse of software components. In this paper, the authors present data from a large software vendor that uses ABC in a traditional software development environment. They also present a chart of accounts for a modern component-based model of software development.

The final paper in this set by Carr and Wagner examines the problem of project prioritization in software maintenance. In contrast to software development project managers, maintenance project managers do not first estimate project costs. Instead, they first prioritize maintenance projects, trying to determine which projects to do first within their fixed budgets and resources. In this study, Carr and Wagner examine the process of project prioritization as an expert problem solving and decision-making task, using verbal "think aloud" protocols obtained from expert software professionals in industry. They find that these experts rarely make use of formal mathematical models to determine project priorities, such as COCOMO or function points. Rather, estimators qualitatively consider cost, urgency, and difficulty of a maintenance task, and then prioritize maintenance projects accordingly. Their decision-making process uses case based reasoning and heuristics. The existence of a common decision making process across experts suggests the potential for their techniques to be encapsulated in tools or methods that can aid novice project managers.

Considered as a group, these four papers provide a number of interesting insights into important project management issues. The papers also outline several promising directions for the improvement of critical problems in software project management, including project coordination, scoping, costing, and prioritization.

The value of software application systems

The final paper in this special issue by Mason and Ragowsky provides an approach for estimating the performance impacts and benefits of a particular software application. Past research has struggled to identify the tangible benefits from information systems. In this paper, Mason and Ragowsky identify the characteristics that have a significant impact on the benefit an organization gains from using a material requirements planning system. Their model predicts that the benefits of this system are positively correlated with the number of potential suppliers, the average monthly order volume, the existence of differences between potential suppliers' offers, the presence of price discounts based on quantity, and the share of raw materials in overall costs. They predict a negative correlation of benefits with organization size. Using a sample of 310 manufacturing firms in Israel, they confirm each of their predictions.

The findings by Mason and Ragowsky imply that to determine the benefits of a particular software application, the specific factors that are likely to be influenced by that application must be identified and considered. That is, a general model of information systems value may not apply across applications. Therefore, to properly evaluate

and select potential software projects on the basis of their expected business value, managers must identify the unique value characteristics that are likely to be impacted by the proposed applications.

References

- [1] P. Brereton, D. Budgen, K. Bennett and M. Munro, The future of software, Communications of the ACM 42(12) (1999) 78–84.
- [2] B. Cox, Planning the software industrial revolution, IEEE Software (November 1990) 27.

Guest Editors:

Rajiv D. Banker

University of Texas at Dallas

Sandra A. Slaughter

Carnegie Mellon University