Toward a Model for Customer-Driven Release Management

Completed Research Paper

Simon Cleveland
Nova Southeastern University
sc1674@nova.edu

Timothy J. Ellis
Nova Southeastern University
ellist@nova.edu

ABSTRACT

Undetected software bugs frequently result in service disruptions, productivity losses, and in some instances significant threat to human life. One way to prevent such bugs is to engage customers in acceptance testing prior to the production software release, yet there is a considerable lack of empirical examination of the release process from the customer's perspective. To address this research-practice gap, this study proposes a model for customer-driven release management that has been shown to minimize the number of software bugs discovered in production systems. The model is evaluated during a 27 month study at a municipality using the action research method. Following the model, 361 software bugs were detected and eliminated prior to final production releases, confirming the value of customer-driven release management for elimination of production software bugs.

Keywords

Software deployment, project management, release management, ITIL, action research

INTRODUCTION

Over 60% of organizations discover major software errors in their production systems (Sahoo, Criswell and Adve, 2010; Schroeder and Gibson, 2010; Tassey, 2002). Undetected bugs can result in billion dollar losses (Anderson, Santos and Haimes, 2007), threat to human life (Geppert, 2004), and considerable loss of customer loyalty (Le Goues, Nguyen, Forrest and Weimer, 2012). Never-the-less, vendors continue to sell buggier software products to attain earlier market entry (Arora, Caulkins and Telang, 2006), omitting unit testing due to lack of time, personnel, automation, and high testing infrastructure costs (Electric Cloud, 2010; Taipale, Smolander and Kalviainen, 2006). Furthermore, production software bugs are up to 1.33 times more likely to require rework (Zimmermann, Nagappan, Guo and Murphy, 2012) since vendors are frequently unable to reproduce the bugs due to differences in the environments, customer privacy concerns regarding the data shared for off-site diagnosis (Tucek, Lu, Huang, Xanthos and Zhou, 2007), and lack of automated feedback regarding the root cause of each failure (Sahoo et al., 2010).

Researchers have proposed a number of methods and frameworks to improve vendor's software quality and release management processes. For example, Rautiainen, Lassenius, Vähäniitty, Pyhäjärvi and Vanhanen (2002) recommended a software product development framework that incorporates agile-based principles to streamline the vendor's release and integrations processes. Greer and Ruhe (2004) proposed a software development method called EVOLVE to address a vendor's allocation of customer requirements and stakeholder priorities to increments. Kajko-Mattsson and Meyer (2005) proposed a blended framework based on Microsoft's and Oracle's industrial models that addressed the acceptor's release processes. Other studies examined challenges associated with vendor release automation (van der Hoek and Wolf, 2003), vendor patch management (Sihvonen and Jantti, 2010), and vendor service provision (Lahtela and Jantti, 2011).

Presently, there is a considerable lack of empirical research recommending a customer-driven approach to detecting software bugs during the release process. To address this gap, the current study seeks to address two research goals: 1) propose a model for a customer-driven release management process, and 2) test that model. The model blends best practices from two widely accepted methodologies: the Information Technology Infrastructure Library (ITIL) release framework and the project management methodology developed by the Project Management Institute (PMI).

The remainder of this article is structured as follows: first, a review of relevant literature in the areas of ITIL release management framework and project management methodology is performed with special focus on gaps associated with customer release management activities. Next, the methodology used to develop the model is presented. The results section presents the customer-driven release management model and outcomes from its evaluation, followed by the study's limitations and recommendations for future research.

LITERATURE REVIEW

Release management is the process associated with the delivery of high quality software to users (Levin and Yadid, 1990). It is temporary in nature (with specific start and finish dates) and consists of steps to gather customer requirements and build, test, package, and deploy the final software effectively into the customer's production environment (OGC, 2007). Release management is a challenging process for software vendors, because a vendor frequently provides release packages to multiple customers with different environments and needs. Some customers require fixes to software bugs, while others require new enhancements. Depending on the complexity and component integration of the software application, a release may involve multiple resources and tasks, increasing the chance for software and process errors (Ballintijn, 2005; Lahtela and Jantti, 2011).

Vendor release managers play a key role in the release processes, are expected to have complete control over releases (Erenkrantz, 2003), and can impact software quality. Danesh, Saybani and Danesh (2011) showed that 97% of Malaysian software companies depend largely on release managers to ensure successful software deployments. Release managers have been shown to spend 60% of their time planning the software releases and the remaining time on re-planning activities due to changes in requirements and customer priorities (Momoh and Ruhe, 2006). Vendor's release managers are also involved in directing their development team, assessing the risks of proposed changes, examining each software line of code, writing the release notes, and interacting with the customers (Michlmayr, Hunt and Probert, 2007). However, the literature doesn't differentiate between a vendor and a customer release manager roles. While vendor release managers successfully coordinate their company's resources, they frequently fail in coordinating customer resources (Momoh and Ruhe, 2006). As a result, software products are not thoroughly tested by end users, leading to acceptance of poor quality software. It is incumbent on the customers to be vigilant during the release process and ensure no corners are cut at the expense of their software quality.

Software delivery methodologies have proven cost effective in reducing software bugs and improving service delivery. One of the most successful such frameworks is the ITIL (Tan, Cater-Steel and Toleman, 2010), which has been shown to increase customer satisfaction and improve operational performance (Potgieter, Botha and Lew, 2005). ITIL was created by the United Kingdom's Central Computer and Telecommunications Agency (CCTA) and represents a set of documented procedures designed to increase the value of IT department operations and service availability (Zeng, 2008). A key component of ITIL is the structured approach to the release of software. Since ITIL addresses IT departments' responsibilities as service providers, it is vendor-centric and lacks specific customer-driven guidelines to detect unexpected production bugs.

ITIL is comprised of nine release management guidelines, the first addressing release policy. Release policy includes an agreement between the vendor and customer on the: a) infrastructure used in the release; b) acceptable schedule; c) definition of major vs. minor releases; d) deliverables for each release; e) roll-out and back-out plans; f) documentation of releases; and g) roles, responsibilities, escalation steps, contacts for vendors, and end-users (Rasa, Kumar and Banu, 2010). However, while customers rely on internal procurement departments to negotiate this policy at the time of the purchase, procurement specialists frequently lack the skills and expertise necessary to make the appropriate decisions (McCue and Gianakis, 2001; Tassabehji and Moorhouse, 2008). As a result, customers often settle for vendor-defined schedules that take into account vendor resource availability. Furthermore, since vendors service a number of customers with different needs, one customer's release deliverables may conflict another's production environment leading to the introduction of new software bugs.

Release planning, the second ITIL release management area, engages the customer in establishing a predefined acceptance criterion specifying whether a release should be accepted in the production environment. Research demonstrates that release planning is considered a 'wicked' problem not guaranteeing stable software for customers regardless of release frequency (Carlshamre, 2002; Michlmayr et al., 2007; Ruhe, 2005). As already noted, vendor release managers coordinate well their internal resources, but fail with customer's resources (Momoh and Ruhe, 2006). This failure creates disagreements of release procedures, unplanned software features and unexpected software bugs.

Design and development of the software, the third ITIL release management area does not directly provide for customer involvement. While customers should be included in the process of writing the acceptance test cases (Connolly, Keenan and Mc Caffery, 2010; Leamsaard and Limpiyakorn, 2011), the majority don't do so (Electric Cloud, 2010).

Build and configure the release is the forth ITIL release area, which also discounts customer involvement, despite the importance of coordinating configuration parameters. The area involves vendor compiling modules stored in the software library to create the derived objects from the source objects. Yin, Ma, Zheng, Zhou, Bairavasundaram and Pasupathy (2011) studied 546 real world misconfigurations and found that over 70% of them are due to improperly set configuration parameters leading to hard-to-diagnose failures, crashes, system hangs, or severe performance degradation.

The fifth release area is fit-for-purpose testing. ITIL does not consider customer engagement in software testing during this stage. Software bugs can be minimized, while vendor confidence in the operations of the system increased, if testing is performed to mimic the actual customer use (Farooq and Quadri, 2011). As a result, lack of customer involvement in this area leads to omission of new software bugs due to software not being tested on typical usage scenarios.

Release acceptance is the sixth release area and involves vendor coordination of the software deployment into the customer's test environment. Typically, testing is performed by end-users, and release acceptance is based on specific conditions the released package must satisfy (Erenkrantz, 2003). However, release package quality is frequently poor due to lack of quality control customer acceptance procedures (Sihvonen and Jantti, 2010).

The seventh release area is the roll-out planning, which involves heavy interaction between the vendor and customer in order to create a time table, responsibilities, and events during the distribution and installation of the software. Proposed processes and tools to streamline this area include: predefined schedule with a cut-off date for inclusion of additional features, release checklists to ensure no steps are missed during the release, timetable and action plan for the release installation (Michlmayr et al., 2007; Lahtela and Jantti, 2011). Once again, the emphasis is placed on the vendor as key coordinator of customer resources in preparation for the release

Communication, preparation and training is the eight release area and includes notifications to the customer of the upcoming release date, hosting of roll-out meetings, and training sessions for the new functionality. Vendor release managers are expected to have deep involvement with the customer's end-users in preparation of the release deployment by providing clearly defined release procedures and comprehensive training manuals (Sihvonen and Jantti, 2010). However, vendors frequently provide standard training manuals and expect customers to customize these based on their own needs (Cochran, 2011). Khoo (2006) found that training is considered one of the biggest issues that impact customer software upgrades. Lack of appropriate training can result in improper use of the system and could lead to reporting of new software bugs.

The final ITIL release management area – distribution and installation – concerns the deployment of the final changes into the live environments. The distribution and installation is usually a manual process typically handled by the vendor's release manager (Ramakrishnan, 2004), however it should be considered the end-users' responsibility since configuration and deployment issues associated with customer infrastructure can result in error-prone software.

From the scant attention paid to the role of the customer release manager in the ITIL framework, it can be gathered that: 1) a designated customer release manager with specified assignments and responsibilities can increase the quality of release and reduce the number of incidents occurring after installation into the customer's production environment (Lahtela and Jantti, 2011) and 2) customer release managers need to be experienced project managers with extensive skill set related to sound judgment, community building, attention to detail, management and communication (Jiang, Sarkar and Jacob, 2011; Sihvonen and Jantti, 2010). A study of over 30,000 projects revealed that nearly half of the successful projects were managed through a formal project methodology (Johnson, 2001). As a result, this paper proposes that releases are projects due to their temporary nature and should be managed by the customers by experienced project managers through the prism of a formal project methodology such as the structured process group approach developed by PMI.

The PMI approach splits a project into five process groups. The first process group is initiating, in which the project manager works to establish the project objectives. This group is followed by the planning process group in which the overall work is defined and comprehensive plans are developed to guide the project to completion. In the execution process group, the work is performed, while the project manager coordinates and communicates with the project team and stakeholders. During the controlling process group, the project manager tracks the performance, uses corrective actions to bring the project back in line with the plan and re-plans the remaining work to be done. Finally, during the closing, the project manager ensures work is completed, documented, and archived for future reference (PMI, 2008).

This study proposes that a customer-driven release management based on a blended model of best practices from both the ITIL and PMI's methodologies can result in the detection of unexpected software bugs.

RESEARCH METHOD

Based on the literature review, a customer-driven release management model was developed that treated each ITIL release management area as unique sub-project. It also incorporated PMI's structured process groups and their respective tasks to each release management area in order to improve the customer's practice of detecting unexpected software bugs. The model evaluation was conducted during the final implementation phase and the subsequent maintenance stage of an enterprise software system at a municipality in the southern quadrant of the United States. The system was procured with the goal of replacing a legacy system, which lacked vendor support and was entirely maintained by the municipality's limited IT staff. The goal of the new system was to improve business operations, streamline data access for over 200 employees, and to provide fully dedicated software vendor technical support. The system, originally designed as an out-of-the box solution, was heavily customized to fit the municipality's business processes and internal work flows. Inter-departmental reorganizations during the integration resulted in frequent system updates to accommodate new work flows. Releases were handled by the vendor and were performed ad hoc with the municipality accepting software into its production environment without rigorous user acceptance testing. Prior to the study, the organization and one of the researchers conducted an analysis of the existing release process. Two main issues were identified: 1) inefficient customer roll-out planning, and 2) inefficient distribution and installation.

In testing the model, participatory action research strategy was adopted. This research method addresses real-life issues by allowing researchers to implement frameworks in empirical settings, and to observe, document, and make changes based on the results that otherwise may not have been observed by external researchers (Grant and Ngwenyama, 2003; Yates and Paquette, 2011). According to Chein, Cook and Harding (1948), participant action research is a method to diagnose a problem, create an action plan, and implement it to solve the problem in collaborative ways between the researcher and the organization's system. Rapoport (1970) expanded on the applicability of action research by designating it as a collaborating method for addressing an organization's practical issues and the researcher's scientific goals via the use of an ethical framework. Action research consists of five phases (Susman, and Evered, 1978): diagnosing, action planning, action taking, evaluating, and specifying learning. During the diagnosing phase of the study, problems of specific release area were identified by the CRM and the project team. During the action planning phase, the CRM and the project team identified activities to address the problems. During the action taking phase, the CRM implemented the planned activities. During the evaluating phase the CRM tracked the status of the release area and reported changes and action results. In the learning phase the CRM documented and archived specific lessons for improvement during the subsequent releases.

The proposed model and action steps taken to address each problem are identified in the results section.

RESULTS

Developing the model

Figure 1 shows the proposed customer-driven release management model. The model consists of five PMI process groups applied to each ITIL release area. In the initiating group, the CRM conducts thorough planning with careful consideration of the enterprise's environmental factors influencing the release area's success. CRM tasks include: determining initial scope, schedule, costs, external stakeholders and obtaining authorization to proceed to the next release area; the planning group requires the identification of specific tasks and end-user responsibilities; the executing group consists of the CRM coordinating, directing, and managing the work done by the end users and communicating on status with the internal and external stakeholders. The CRM enforces any changes discovered

during the monitoring and controlling group; the monitoring and controlling area has the CRM tracking the progress of work, monitoring and using corrective actions to realign the work with the plan, documenting any changes and frequent communication with end-users and stakeholders to inform all parties of pending changes; In the closing group, the CRM ensures all deliverables have been accomplished, documenting the lessons learned and archiving them for future use.

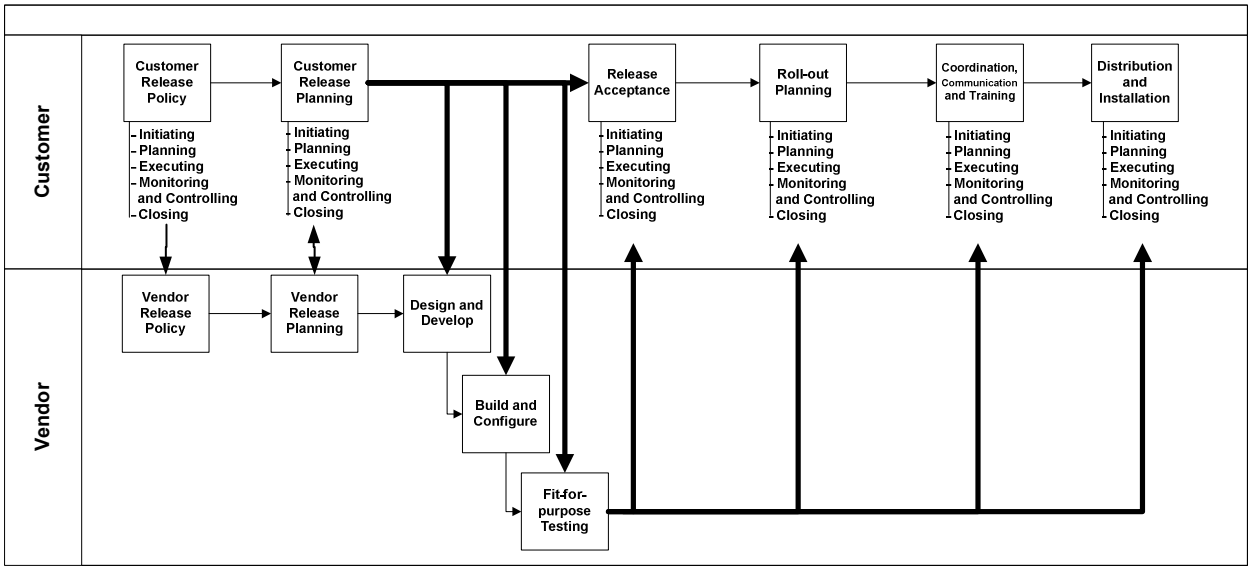

Figure 1: Customer-driven release management model

Testing the model

The model was tested by addressing the two identified problems via two cycles. In the first cycle, the inefficient customer roll-out planning problem was addressed. During the initiating group, the CRM and the project team determined that there was minimal involvement from end-users during the roll-out planning process. A list of tasks for the delivery and installation was prepared by the vendor; however, coordination of customer IT resources for the roll-back procedures was frequently omitted. During the planning group, the CRM developed detailed release time table with release tasks, their start and end dates and times, responsible testers and test cases, and roll-out procedure plan in case the installation fails. During the executing group, the CRM held a pre-roll-out meeting and inform the stakeholders of his role in the release and asked each participant to read out their tasks. The CRM solicited participants' feedback about the tasks and addressed any inconsistencies or misunderstandings. At the end of the pre-roll-out meeting, each stakeholder was made aware of the expectations of his or her role, including start and end times and contact information for all participants and the CRM. The CRM also informed the UAT team of the test cases to be executed in the production environment after the completion of installation. During the monitoring and controlling group, the CRM was able to evaluate the roll-out plan, address any inconsistencies and update the release checklist. In the closing group the pre-roll-out meeting was used as a test run for the real installation. This exercise assisted the CRM to determine tasks missed in the initial release planning.

In the second cycle, the inefficient distribution and installation of software problem was addressed. During the initiating group, the CRM determined that this area frequently experienced a number of issues due to the lack of coordination between the vendor and customer. On one occasion, the vendor account permissions were not enabled due to lack of communication between the teams, which caused a delay of the installation. Furthermore, during the installation, security updates commenced on the servers, resulting in further delays. As a result, the release was delivered over a period of 16 hours during the weekend. Similar issues caused other installations to also last over entire weekends. Frequently on the following business day, end-users discovered configuration issues with the application that could have been avoided by performing production testing after the installation. In the planning group, the CRM ensured the installation process followed the pre-established roll-out checklist. During the executing, the CRM made all release participants aware of their specific tasks and roles associated with the upcoming installation. The CRM also made available a conference bridge for all stakeholders to obtain status on each specific task and direct the execution of the next task on the list during the installation. The CRM downloaded

all installers from the vendor site prior to the release installation and ensured regular backups were disabled from the production server at the start of the installation. The UAT team performed production testing immediately after the installation of the release to ensure no errors with the release existed in the live system. During the monitoring and controlling, the delivery and installation that used a roll-out checklist were completed within six hours of initiation. There were no issues reported during the installation, but three issues were discovered by the UAT team during production testing. The issues were related to server configuration settings and were resolved by the vendor. The CRM certified the release as successful at the end of the sixth hour and informed all stakeholders. On the following business day, the CRM and the UAT team were available to respond to user calls; however no installation issues were reported by the end-users. Finally, during the closing, the use of a time table, checklist with tasks, a conference bridge for communication was determined to eliminate stakeholder confusion and improve the delivery.

Using the proposed customer-driven model the UAT team discovered 361 software bugs that were resolved via seven major and 39 minor releases (major releases are defined as those that included more than one customer resource and took more than three man hours to complete, while minor releases included no more than one customer resource and took no longer than three man hours to complete). Production system uptime data was collected for normal business hours to determine release successes based on the newly delivered builds. Uptime of the production server improved by an average of 0.15% or nearly one hour and five minutes per month compared to the period prior to the changes. System downtime was reduced from 440 minutes prior to the model testing to only ten minutes after the model implementation.

CONCLUSION

While the ITIL framework holds great promise for leveraging a structured method of delivery and acceptance of release the lessons from this study show a number of disadvantages for the customer if the framework is only applied from the vendor's perspective. As a result, the proposed customer-driven release model addresses the research gaps from the perspective of the customer. By viewing the impact of the release management process through the lens of the proposed customer-driven release model, customers can accomplish effective release management while minimizing the number of production software bugs. Furthermore, by establishing a central CRM role, customers can achieve elective communication and coordination between stakeholders to further improve the system quality.

This study appears to be the first to propose a model for customer-drive release management with a predefined CRM role and responsibilities. The paper presents many opportunities for future research on the release management process. For example, in release management it would be valuable to examine the impact on software quality via use cases written by the customer team and used by the vendor software development team during the coding process. This process was not investigated in this case, but can prove valuable in order to not only understand how use cases impact software development, but also how the interaction between the customer UAT team and the vendor's software development team can work together to increase software quality.

Further research, through a multi-case study approach on the use of the proposed customer-driven release model and the results achieved with it, could answer questions on its advantages to other systems across a broad range of industries. Research on understanding the customer's privacy concerns related to data sharing for troubleshooting (Cleveland, 2012a) and lessons learned during release process (Cleveland, 2012b) can further contribute to the elimination of production software bugs. This research will hopefully set the stage for investigating these and other important questions for more effective releases that minimize software production bugs.

REFERENCES

1. Anderson, C., Santos, J. and Haimes, Y. (2007) A risk-based input–output methodology for measuring the effects of the August 2003 northeast blackout, *Economic Systems Research*, 192, 183-204.
2. Arora, A., Caulkins, J. and Telang, R. (2006) Research note-Sell first, fix later: Impact of patching on software quality, *Management Science*, 523, 465-471.
3. Ballintijn, G. (2005) A case study of the release management of a health-care information system, *Proceedings of the IEEE International Conference on Software Maintenance*, 34–43.
4. Carlshamre, P. (2002) Release planning in market-driven software product development: Provoking an understanding, *Requirements Engineering*, 73, 139-151.

5. Chein, I., Cook, S. and Harding, J. (1948) The field of action research, *American Psychologist*, 3, 43-50.
6. Cleveland, S. (2012a) In search of user privacy protection in ubiquitous computing, *Proceedings of the 13th International IEEE Conference on Information Reuse and Integration*, Las Vegas, NV, 694-699.
7. Cleveland, S. (2012b) Using microblogging for lessons learned in information systems projects. *7th Pre-ICIS International Research Workshop on Information Technology Project Management (IRWITPM)*, Orlando, FL, 122-128.
8. Cochran, M. (2011) Evaluation of application embedded knowledge migration Issues, *Proceedings of the 2nd International Conference on Information Management and Evaluation*, Toronto, Canada, 134.
9. Connolly, D., Keenan, F. and Mc Caffery, F. (2010) Acceptance test-driven development by annotation of existing documentation, *Proceedings of the European Systems & Software Process Improvement and Innovation Conference*.
10. Danesh, A., Saybani, M., and Danesh, S. (2011) Software release management challenges in industry: An exploratory study, *African Journal of Business Management*, 520, 8050-8056.
11. Electric Cloud. (2010) Electric cloud survey: 58% of software bugs result from test infrastructure and process, not design defects, Retrieved on May 14, 2012 from http://www.electric-cloud.com/news/2010-0602.php
12. Erenkrantz, J. (2003) Release management within open source projects, *Proceedings of the 3rd Workshop on Open Source Software Engineering*.
13. Farooq, S. and Quadri, S. (2011) Evaluating effectiveness of software testing techniques with emphasis on enhancing software reliability, *Journal of Emerging Trends in Computing and Information Sciences*, 212, 740-745.
14. Geppert, L. (2004) Lost radio contact leaves pilots on their own, *IEEE Spectrum*, 4111, 16-17.
15. Grant, D. and Ngwenyama, O. (2003) A report on the use of action research to evaluate a manufacturing information systems development methodology in a company, *Information Systems Journal*, 13, 21-35.
16. Greer, D. and Ruhe, G. (2004) Software release planning: An evolutionary and iterative approach, *Information and Software Technology*, 46, 243-253.
17. Jiang, Z., Sarkar, S. and Jacob, V. (2011) Postrelease testing and software release policy for enterprise-level systems, *Information Systems Research,* 23, 3-Part-1, 635-657.
18. Johnson, H. (2001) Micro-projects cause constant change: Abstract of research report, Extreme Chaos, The Standish Group International, Inc, West Yarmouth.
19. Kajko-Mattson, M. and Meyer, P. (2005) Evaluating the acceptor side of EM3: Release management at SAS, *Proceedings of the 4th International Symposium on Empirical Software Engineering*, November 17-18, Noosa Heads, Australia, IEEE Computer Society Press, 10.
20. Khoo, H. (2006) Upgrading packaged software: An exploratory study of decisions, impacts, and coping strategies from the perspectives of stakeholders, *Computer Information Systems Dissertations*, Paper 5.
21. Lahtela, A. and Jantti, M. (2011) Challenges and problems in release management process: A case study, *Proceedings of the 2nd International Conference on Software Engineering and Service Science*, 10-12.
22. Le Goues, C., Nguyen, T., Forrest, S. and Weimer, W. (2012) GenProg: A generic method for automatic software repair, *Software Engineering, IEEE Transactions on*, 381, 54-72.
23. Leamsaard, C. and Limpiyakorn, Y. (2011) On integrating user acceptance tests generation to requirements management, *Proceeding of the International Conference on Information Communication and Management*, 248-252.
24. Levin, K. D. and Yadid, O. (1990) Optimal release time of improved versions of software packages, *Information and Software Technology*, 32,1, 65–70.
25. McCue, C., and Gianakis, G. (2001) Public purchasing: Who's minding the store? *Journal of Public Procurement*, 1, 1, 71-95.
26. Michlmayr, M., Hunt, F. and Probert, D. (2007) Release management in free software projects: Practices and problems, *ITP International Federation for Information Processing, 234, Open Source Development, Adoption and Innovation, eds. J. Feller, Fitzgerald, V. Scacchi, W., Sillitti, A.*, Boston: Springer, 295-300.

27. Momoh, J. and Ruhe, G. (2006) Release planning process improvement – An industrial case study, *Software Process Improvement and Practice*, 11, 295-307.
28. Office of Government Commerce (OGC). (2007) ITIL service operation, The Stationary Office, UK.
29. Potgieter, B., Botha, J., and Lew, C. (2005) Evidence that use of the ITIL framework is effective, *Proceedings of the 18th Annual Conference of the National Advisory Committee on Computing Qualifications*.
30. Project Management Institute (PMI). (2008) A guide to the project management body of knowledge (PMBOK guide), PMI, Philadelphia.
31. Ramakrishnan, M. (2004) Software release management, *Bell Labs Technical Journal*, 9, 205-210.
32. Rapoport, R. (1970) Three dilemmas of action research, *Human Relations*, 23, 499-513.
33. Rasa, G., Kumar, J., and Banu, W. (2010) Release and deployment management using ITIL, *Global Journal of Computer Science and Technology*, 1015, 2-8.
34. Rautiainen K., Lassenius, C., Vähäniitty, J., Pyhäjärvi, M., and Vanhanen, J. (2002) A tentative framework for managing software product development in small companies, *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, 3409 –3417.
35. Ruhe, G. (2005) Software release planning, *Chang, S.K., ed.: Handbook of Software Engineering and Knowledge Engineering. Volume 3 Recent advances*, World Scientific, 365-394.
36. Sahoo, S. K., Criswell, J., and Adve, V. (2010) An empirical study of reported bugs in server software with implications for automated bug diagnosis, Paper presented at the *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*,1, Cape Town, South Africa.
37. Schroeder, B. and Gibson, G. (2010) A large-scale study of failures in high-performance computing systems, *IEEE Transactions on Dependable and Secure Computing*, 74, 337-351.
38. Sihvonen, H. and Jantti, M. (2010) Improving release and patch management processes: An empirical case study on process challenges, *Proceedings of the 5th International Conference on Software Engineering Advances*, 232-237.
39. Susman, G. and Evered, R. (1978) An assessment of the scientific merits of action research, *Administrative Science Quarterly*, 234 , 582-603.
40. Taipale, O., Smolander K., and Kalviainen, H. (2006) Cost reduction and quality improvement in software testing, *Proceedings of the Software Quality Management Conference*, Southampton.
41. Tan, W., Cater-Steel, A., and Toleman, M. (2010) Implementing IT service management: A case study focusing on critical success factors, *Journal of Computer Information Systems*, 502, 1-12.
42. Tassabehji, R., and Moorhouse, A. (2008) The changing role of procurement: Developing professional effectiveness, *Journal of Purchasing and Supply Management*, 141, 55-68.
43. Tassey, G. (2002) The economic impacts of inadequate infrastructure for software testing, RTI Project Number 7007.011, National Institute of Standards and Technology.
44. Tucek, J., Lu, S., Huang, C., Xanthos, S., and Zhou, Y. (2007) Triage: Diagnosing production run failures at the user's site, *Proceedings of the 21st symposium on Operating systems principles*.
45. van der Hoek, A. and Wolf, A. (2003) Software release management for component-based software, *Software Practice and Experience*, 331, 77 – 98.
46. Yates, D. and Paquette, S. (2011) Emergency knowledge management and social media technologies: A case study of the 2010 Haitian earthquake, *International Journal of Information Management*, 31, 6-13.
47. Yin, Z., Ma, X., Zheng, J., Zhou, Y., Bairavasundaram, L. and Pasupathy, S. (2011) An empirical study on configuration errors in commercial and open source systems, *Proceedings of the 33rd ACM Symposium on Operating Systems Principles*, ACM, New York, NY, USA, 159-172.
48. Zeng, J. (2008) A case study on applying ITIL availability management best practice, *Contemporary Management Research*, 44, 321-332.
49. Zimmermann, T., Nagappan, N., Guo, P., and Murphy, B. (2012) Characterizing and predicting which bugs get reopened, *Proceedings of the 34th International Conference on Software Engineering*.