The Concepts Behind Software Maintenance and Regression Testing

Deborah J. Barndt

Illinois Institute of Technology

[dbarndt@hawk.iit.edu](mailto:dbarndt@hawk.iit.edu)

10-20-18

Abstract

This paper reviews the concepts behind software maintenance and regression testing, how it is important to an organization, and why it is necessary. Software maintenance and regression testing is an important part of the software development life cycle but is often misunderstood what it entails. Many organizations tend to overlook software maintenance due the lack of time they have in a project and the total costs. However, neglecting maintenance testing can have consequences that can be devastating to a company and could result in rival companies getting the upper hand. This paper presents the purpose of software maintenance, the key issues and challenges that are involved, the types of changes that are made in software maintenance and regression testing, current software maintenance standards formulated by the ISO and IEEE, the different software maintenance and regression testing techniques that are used, and the costs of software maintenance and risks involved with neglecting maintenance during the software development process.

Keywords: Software maintenance, Regression testing

The Concepts Behind Software Maintenance and Regression Testing

Software changes continuously. Even after it is released, updates must be made to the system for it to adapt to the changing environment. Patches are then made available online for users to download. These patches could be because of new requirements that were needed, technology changes, new bugs were discovered, or new features being added after the software’s release. For example, after each mainline Linux kernel is released, the bug fixes are backported from a newer version and applied to the older version. Stable kernel updates are released on a required basis, which is usually once a week and new mainline kernels are released every 2-3 months (Linux Kernel Organization, Inc., 2018)! Thus, making software maintenance and regression testing a crucial part of software testing and the software development life cycle.

# Basic Concepts

## The Purpose of Software Maintenance and Regression Testing

The main purpose of software maintenance is to modify and update software applications after it has been delivered to correct any faults and improve the performance (Bennett, Coleman & Co. Ltd., 2018). A common misconception about software maintenance is that it only involves fixing defects, which is not true. “Maintenance testing includes any type of testing of changes to an existing, operational system, whether the changes result from modifications, migration, or retirement of the software or system” (Black, Veenendaal, & Graham, 2017). Software maintenance is needed to repair defects that were missed in the released software, because defect removal and quality control processes are not perfect. It also must be carefully planned from the beginning of the life cycle phase and planning out your maintenance testing will help with creating a schedule for maintenance releases that may be needed in the future.

While software maintenance can account for about two-thirds of the overall cost of production, one necessary maintenance activity that is needed is regression testing (Rothermel & Harrold, 1997). This type of testing is completed on the modified software in order to make sure that the software behaves correctly and that the modifications haven’t impacted the quality of the software. The objective is to execute each new or changed input at least once, so they can identify the input domains that are not modified and to aid in test generation (Leung & White, 1991). This ensures that the changes have not affected the features of the software that should not change. Even though regression testing is expensive, it accounts for about one-half of the cost of software maintenance (Rothermel & Harrold, 1997). The difference between software maintenance testing and regression testing is that there is an accepted suite of tests that are available for reuse during regression testing. Testers can then rerun all the test cases at earlier stages to make sure the program is working correctly.

There are many factors that determine the need for regression testing and maintenance such as correct faults, improving the design, implementing enhancements, interfacing with other systems, making the system adaptable, migrating legacy software, retiring software, or being able to provide continuity of service. In the end, software systems need to keep running, and much of our daily life is run by a computer. There could be catastrophic consequences if there is a system failure such as financial issues or serious disruptions. “Maintenance activities aimed at keeping a system operational include bug-fixing, recovering from failure, and accommodating changes in the operating system and hardware” (Grubb & Takang, 2007). A company also wants to be able to keep a competitive edge against your competitor’s products would involve mandatory upgrades. Furthermore, users will use the system more if it is a better product, which will get the users to request enhancements in the systems functionality or requirements for better performance. The last factor that determines maintenance is being able to facilitate future maintenance work. Taking shortcuts during software production can be very costly and could hurt the company financially in the long run. It could work out better financially and commercially to begin changes geared towards future maintenance, which would involve things such as database restructuring or updating the documentation (Grubb & Takang, 2007).

## The Key Issues and Challenges with Software Maintenance

Many of the problems that are associated with software maintenance usually involve the inadequacy of the development process. In order to effectively maintain those systems, it is essential to have a good base in the relevant theory. “Software maintenance is a key discipline, because it is the means by which systems remain operational and cope efficiently in a world ever more reliant on software systems” (Grubb & Takang, 2007). As a software maintenance engineer, you need to know how long you are maintaining the system whether it be for five minutes or five years. Software maintenance provides distinct technical and management challenges for software engineers (OpenStax, 2018). For example, introducing a new system and having to postpone the date of completion of the new system. This means that the older system must be supported and maintained past its life expectancy. “The motivation to get it right comes from an understanding the wider implications, which in turn comes from an understanding of the theories and complexities of the discipline of software maintenance” (Grubb & Takang, 2007).

The software maintenance engineer needs to have a wide range of skills besides just computer programming, but they also need comprehension skills and analytical powers. Some of the range of skills that a software engineer needs are a limited understanding of where to make a change or correction in the software that the engineer didn’t develop. “Research indicates that some 40% to 60% of the maintenance effort is devoted to understanding the software to be modified” (OpenStax, 2018). Testing is another significant skill of a software engineer, and the cost of repeating full testing can be significant in terms of time and money to a company. Being able to coordinate testing when different members of the maintenance team are working on different problems can be a challenge when it comes to maintenance. The last skill that a software engineer will need is being able to possess close knowledge of the software’s structure and content, then using that to perform the impact analysis by identifying all the systems and products that will be affected by the change request.

# Types of Software Maintenance Changes

# For the maintenance team or engineer to achieve the goals intended for the system, a broad-spectrum of changes may be necessary to the software. These changes can be classified into four types: corrective, adaptive, perfective, and preventive. Each of these changes can be classified individually, they are also intertwined with one another which is shown in Figure 1. Here we will go into detail about each of the changes and why they are important.

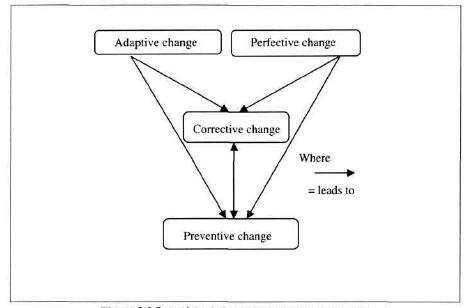


Figure 1. Potential Relationships of Software Changes (Grubb & Takang, 2007).

## Corrective Change

A corrective change makes a reference to the modification that is initiated by defects in the software. A defect could be design errors, logic errors, and coding errors. Design errors occur when changes that are made to the software are incorrect, incomplete, miscommunicated, or misunderstood. Logic errors occur when there are invalid tests and conclusions, the implementation of the design specifications was incorrect, faulty logic flow, or incomplete testing data. Coding errors occur when the implementation of the detailed logic design is incorrect and there is an incorrect use of the source code logic, making defects caused by data processing errors and system performance errors (Grubb & Takang, 2007).

Each one of these errors can prevent the software from adhering to the agreed specification. If there is a system failure due to an error, actions are then taken to try to restore the software system back to its normal operation. “Under pressure from management, maintenance personnel sometimes resort to emergency fixes known as ‘patching’. The ad hoc nature of this approach gives rise to a range of problems that include increased program complexity and unforeseen ripple effects” (Grubb & Takang, 2007). Program complexity usually comes from the degeneration of the program structure and thus makes the program more difficult to understand.

## Adaptive Change

An adaptive change refers to a change that is driven by the need to accommodate modifications in the environment of the software system. The term environment means the completeness of all the conditions and influences that are needed that act from the outside on the system. If there is a change to part or the entire environment it will cause a compatible modification to the software. “Adaptive maintenance includes any work initiated as a consequence of moving the software to a different hardware or software platform – compiler, operating system or new processor” (Grubb & Takang, 2007). A program can be changed because of new compiler that performs additional optimizations in order to create smaller and faster code. A program can also be modified to be able to take full advantage of the additional facilities that is provided by the new version of the system.

## Perfective Change

A perfective change refers to changes that are responsible for expanding the existing requirements of a system. For a software system to be successful, it is subjected to a succession of changes that result in increasing requirements. The increase in requirements is based on whether the software becomes useful, and the users experiment with the software beyond what it was developed for. An expansion of those requirements would then take on the form of enhancements of the existing system functionality. Programs can then also be catered to the users if some of the features are rejected or become redundant by being either removed or modified.

## Preventive Change

A preventive change refers to a change done to a software system to address a problem of a deteriorating structure. This type of change is used to prevent malfunctions or to improve the maintainability of the software itself and is usually initiated with the intention of making the program easier to understand. However, a preventive change does not usually lead to an increase in the original functionality. Also if a thorough impact analysis is not carried out in time before the change is made, unforeseen ripple effects could occur that could affect other sections of the program and be unpredictable as to the results which could lead to a distortion of the system (Grubb & Takang, 2007). “Program slicing techniques and concepts such as modularization and information hiding can be used to address these problems” (Grubb & Takang, 2007).

**Software Maintenance Standards**

If a process is well-defined in software maintenance, it can be observed, measured, and improved upon. It also allows the circulation of effective work practices more quickly. Software maintenance standards have been formulated by ISO and IEEE. “The maintenance standard document from ISO is called ISO/IEC 14764 which is a part of the standard document ISO/IEC 12207 for life cycle processes. The maintenance standard document from IEEE is called IEEE/EIA 1219” (Tripathy & Naik, 2015). In both standard documents they describe the processes for managing and executing the activities for software maintenance. There are seven phases of the maintenance process that are described by the IEEE/EIA 1219 as: problem identification, analysis, design, implementation, system test, acceptance test, and delivery (Tripathy & Naik, 2015). Both standards use the same terminology when describing software maintenance and there’s very little difference in their results. The activities that form the maintenance process are: process implementation, problem and modification analysis, modification implementation, maintenance review/acceptance, migration, and software retirement as shown in Figure 2.

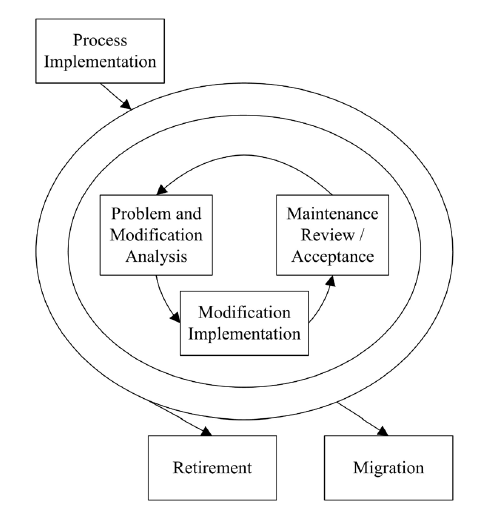


Figure 2. Software Maintenance Process (Bourque & Fairley, 2014).

Process Implementation tasks are involved with establishing procedures for modification requests, developing maintenance plans and procedures, and implementing the CM process. Problem and Modification Analysis tasks are involved with performing initial analysis, verifying the problem, developing options for implementing the modification, documenting the results, and obtaining approval for modification options.Modification Implementation tasks involved with performing detailed analysis and developing, coding, and testing the modification. Maintenance Review/Acceptance tasks involved with conducting reviews of the software and obtaining approval for modifications. Migration tasks involved with ensuring that migration is in accordance with ISO/IEC 12207, developing a migration plan, notifying users of migration plans, conducting parallel operations, notifying users that migration has started, conducting a post-operation review, and ensuring that old data is accessible. Software Retirement tasks involved with developing a retirement plan, notifying users of the retirement plan, conducting parallel operations, notifying users that the retirement phase has started, and ensuring that the old data is accessible.

# Techniques

## Software Maintenance Techniques

There are five best practiced techniques used by software maintainers to ensure effective software maintenance is performed which are: program comprehension, re-engineering, reverse engineering, migration, and retirement.

## Program Comprehension.

Code browsers are a key tool that is used for program comprehension, which programmers spend a considerable amount of time in reading and comprehending their programs in order to implement changes. Clear and concise documentation is also used to help with program comprehension.

## Re-engineering.

“Re-engineering is defined as the examination and alteration of software to reconstitute it in a new form, and includes the subsequent implementation of the new form” (Bourque & Fairley, 2014). It is often used to replace the aging legacy software. One of the re-engineering techniques that is used is called refactoring, which aims to re-organizing a program without changing its behavior and seeks to improve a programs structure and maintainability.

## Reverse Engineering.

“Reverse engineering is the process of analyzing software to identify the software’s components and their inter-relationships and to create representations of the software in another form or at higher levels of abstraction” (Bourque & Fairley, 2014). This is a passive technique that does not change the software or make new software but produces call graphs and control flow graphs from the source code. Tools are the key items that are used for reverse engineering for redocumentation and design recovery.

## Migration.

Software may have to be modified to run in different environments during a software’s life. The maintainer then needs to determine what needs to be done to accomplish the migration, and then develop and document all the steps for it in a migration plan, which covers the migration requirements, migration tools, the conversion of product and data, the execution, verification, and support. Migrating software could also involve additional activities such as: notification of intent, parallel operations, notification of completion, post-operation review, and data archival.

## Retirement.

When a software has reached the end of its life it must be retired, and an analysis must be performed to make that decision. The analysis should be included in the retirement plan, which will cover the requirements, the impact, replacement, the schedule, and effort that is involved with the retirement. The activities involved with retiring software are similar to the activities used for migration.

## Regression Testing Techniques

“Regression testing is defined as ‘the process of retesting the modified parts of the software and ensuring that no new errors have been introduced into previously tested code’” (Kadry, 2011). There are four different regression techniques that are used: retest all, regression test selection, test case prioritization, and hybrid approach as shown in Figure 3.

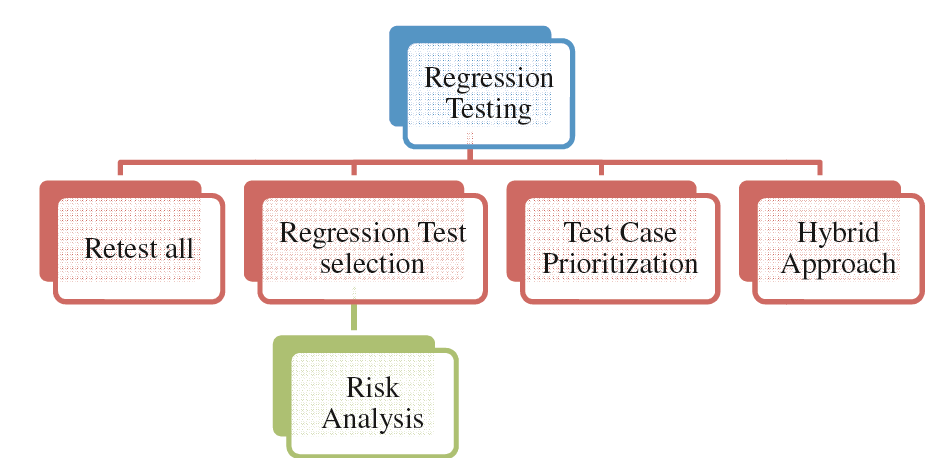


Figure 3. Regression Testing Techniques (Kadry, 2011).

### Retest All.

The retest all method is one of the conventional methods that is used for regression testing in which all the tests are re-run that are existing in the test suite. This technique is very expensive compared to other techniques, because test suites are costly to execute in full due to the time needed and budget.

### Test Case Prioritization.

This type of regression testing technique prioritizes the test cases so as to increase a test suite’s ability to detect faults in the modified program in order to increase reliability. There are two types to this which are the general prioritization and version specific prioritization. General prioritization attempts to select an order of the test case that will be effective to subsequent versions of the software on average. Version specific prioritization is concerned with a specific version of the software.

### Hybrid Approach.

The hybrid approach technique consists of both regression test selection and test case prioritization. This technique has not been finalized, but has a number of researchers working on it that have proposed many algorithms.

### Regression Test Selection.

Regression test selection is performed due to the expensive nature of the retest all technique. This technique selects a part of the test suite to re-run if the cost of selecting a part of the test suite is less than the cost of running all the tests. Regression test selection divides the existing test suite into three different test cases: reusable test cases, re-testable test cases, and obsolete test cases. It may also create new test cases for the program in order to cover areas that were not covered by the existing test cases. The regression test selection can be classified into three categories: coverage techniques which takes the test coverage into account, minimization techniques which is similar to coverage techniques but they select a minimum set of test cases, and safe techniques which do not focus on criteria of coverage but select all the test cases that produce different outputs with a modified program compared the its original version (Kadry, 2011).

# Cost and Risks of Maintenance and Regression Testing

As was mentioned previously, software maintenance is the process of modifying existing software and the importance of software maintenance cannot be overestimated. “It is widely recognized as the highest cost phase of the software life cycle with estimated costs of between 60 and 80% of the total software budget. It has also been noted that over 50% of programmer effort is dedicated to maintenance” (Stark, 1996). Considering this is such a high cost; some organizations and companies are starting to look at their maintenance processes towards their software as a competitive advantage against their rivals. A way to reduce costs of maintenance is by retesting the software with efficient and effective regression testing. Using a selective strategy compared to a retest-all strategy can reduce costs, since a selective strategy typically uses fewer tests than the retest-all strategy.

Some factors that could affect the cost of maintenance can be the organizational processes, which helps with predictability and quality. Another factor is the importance of documentation and programmer experience, which can help determine how complex software maintenance will be. This means that the management and planning of large software maintenance organizations need to be formally standardized and quantified. In order to accomplish this, some organizations use a metrics program to better understand and manage software maintenance that is based on Basili’s Goal/Question/Metric Paradigm and other metrics experiences as shown in the example in Table 1 (Stark, 1996). A metrics program can also help an organization find out if there are any problems in the software currently.

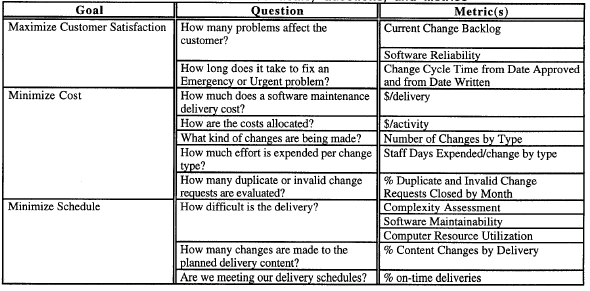


Table 1. SW Maintenance Goals, Questions, and Metrics (Stark, 1996).

If something is deemed a software problem, a software change form is generated which contains 18 data items that include dates for submission and approval, the current method, proposed change, the justification for the change, and the resource estimates. This form is then presented to a user board for validation and a software maintenance engineer for an independent evaluation of the form. This entire process in determining the cost of such changes and what is deemed necessary is all part of the software development process as shown in Figure 4.

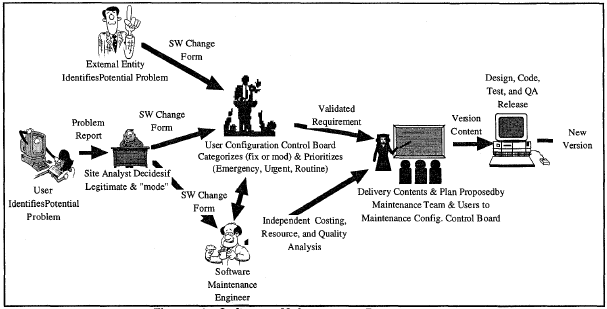


Figure 4. Software Maintenance Process (Stark, 1996).

Once the company decides what is necessary, they must figure out how the costs are allocated. According to Figure 5, the costs associated with the software development activities and hardware system maintenance make up about 88% of the total release cost of the software.

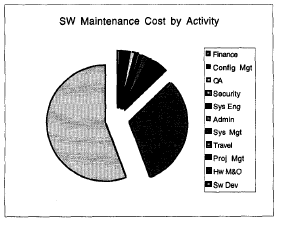


Figure 5. Software Maintenance Cost by Activity (Stark, 1996).

# There are many factors that can affect the cost of testing and maintaining a software system, which can be broken down into two sets of costs: direct and indirect costs. “A direct cost includes the test analyst’s time for performing all activities related to testing and the resources such as the computer and test lab required for executing the tests” (Leung & White, 1991). An indirect cost could include management overhead for running the overall testing process, database costs for storing the test-related data such as test cases, results of the static analysis, and execution histories, and test tool development. Indirect costs are commonly not factored into the total cost though, due to some organizations feel that it will not affect their results. For direct costs, the cost of applying a set of tests to a system has four components: system analysis cost, test selection cost, test execution costs, and result analysis costs. The system analyst must become familiar with the system and time must be taken to study the requirements and documents. Once there is enough knowledge about the system, test cases can then be selected which results in some costs in working out the test input. Test execution includes the cost of setting up the environment for testing the system and the cost of computing resources for the essential execution (Leung & White, 1991). The last step is the result analysis cost, which involves checking the behavior of the system against the specified behavior. These cost components depend on several factors and some are interrelated, and the testing strategy used has a direct effect on the cost.

Without testing software or providing software maintenance, there could be a lose of service, errors or defects missed in the software, and the cost of trying to fix the system could be high. While we cannot test everything or predict every eventuality in the software, it is vital to the safety and correct operation of software systems. Most bugs run an average of more than one per program statement, and programmers can find and fix most bugs. Huge amounts of bugs are fixed in order to release the software at an estimated three or less per 100 program statements but finding the bugs does not fix them only identifies them (Grubb & Takang, 2007). “However, recognizing the best-case scenario is also valuable, because no-one should become so resigned to developing and maintaining software to inadequate standards that they stop trying to improve things” (Grubb & Takang, 2007).

# Conclusion

Software maintenance and regression testing is a crucial part of software testing and the software development life cycle. Some of the factors involved goes beyond fixing defects such as improving the design, implementing enhancements, interfacing with other systems, making the system adaptable, or continuity of service. Many of the key issues and challenges that arise in software maintenance involves the development process being inadequate such as an engineer’s lack of knowledge about the system. This could require many unnecessary changes to the software and become costly to the organization but could be avoided if more attention was paid to maintaining and testing the software. However, having a process that is well-defined will allow the circulation of effective work practices and improve the quality of the software provided. Different techniques can be used to accomplish this such as program comprehension, re-engineering, reverse engineering, and retirement and ensures the effective software maintenance is performed. While software maintenance is the highest cost phase of the software life cycle, it cannot be overlooked or neglected. The consequences for not having effective software maintenance or regression testing could lead to a system failure, financial issues, missed errors or defects, or serious disruptions in service. While we cannot test everything or predict every eventuality, not doing so could be catastrophic.

# References

April, A., & Abran, A. (2008). *Software maintenance management: evaluation and continuous improvement.* New York (NY): IEEE Computer Society Press.

Bennett, Coleman & Co. Ltd. (2018). *Software Maintenance - What is Software Maintenance ? Software Maintenance meaning, Software Maintenance definition*. Retrieved from The Economic Times: Software Maintenance: https://economictimes.indiatimes.com/definition/software-maintenance

Black, R., Veenendaal, E. v., & Graham, D. (2017). *Foundations of software testing: ISTQB certification.* Andover: Cengage Learning EMEA.

Bourque, P., & Fairley, R. E. (2014). *Guide to the software engineering body of knowledge.* Los Alamitos, CA: IEEE Computer Society.

Grubb, P., & Takang, A. A. (2007). *Software maintenance: concepts and practice.* New Jersey: World Scientific.

Hunt, B., Turner, B., & Mcritchie, K. (2008). Software Maintenance Implications on Cost and Schedule. *2008 IEEE Aerospace Conference*, 1-6. doi:10.1109/aero.2008.4526688

Jarzabek, S. (2007). *Effective software maintenance and evolution.* Boca Raton: Auerbach Publications.

Kadry, S. (2011). A New Proposed Technique to Improve Software Regression Testing Cost. *International Journal of Security and Its Applications, 5*(3), 45-58.

Leung, H. K., & White, L. (1991). A cost model to compare regression test strategies. *Proceedings. Conference on Software Maintenance 1991*, 201-208. doi:10.1109/icsm.1991.160330

Linux Kernel Organization, Inc. (2018). *The Linux Kernel Archives*. Retrieved from The Linux Kernel Archives: https://www.kernel.org/category/releases.html

OpenStax. (2018, March 20). *2.6 Software maintenance, Lecture notes, By OpenStax (Page 1/10)*. Retrieved from QuizOver.com: https://www.quizover.com/software/course/2-6-software-maintenance-lecture-notes-by-openstax

Reifer, D. J. (2012). *Software Maintenance Success Recipes.* Boca Raton: CRC Press.

Rothermel, G., & Harrold, M. J. (1997). A safe, efficient algorithm for regression test selection. *1993 Conference on Software Maintenance, 6*(2), 173-210. doi:10.1109/icsm.1993.366926

Stark, G. E. (1996). Measurements for managing software maintenance. *Proceedings of International Conference on Software Maintenance ICSM-96*, 152-161. doi:10.1109/icsm.1996.565000

Tripathy, P., & Naik, K. (2015). *Software evolution and maintenance: a practitioners approach.* Hoboken, NJ: Wiley.

Usaola, M. P., Piattini, M., & Ruiz, F. (2003). *Advances in software maintenance management: technologies and solutions.* Hershey: Idea Group Publishing.

Varga, E. (2017). *Unraveling software maintenance and evolution: thinking outside the box.* Cham: Springer International Publishing.

Wong, W. E., Horgan, J., London, S., & Agrawal, H. (1997). A study of effective regression testing in practice. *Proceedings The Eighth International Symposium on Software Reliability Engineering*, 264-274. doi:10.1109/issre.1997.630875