AUTOMATED SOLUTIONS: IMPROVING THE EFFICIENCY OF SOFTWARE TESTING

Zhenyu Huang, Central Michigan University, huang1z@cmich.edu
Lisa Carter, FedEx Co., lisa.carter@fedex.com

ABSTRACT

The rapid innovation and fierce competition that rule the IT world today force companies to develop the highest quality software using minimal cost at a timely base. This demand has placed a major responsibility on the area of software testing and validation. Testing the software for accuracy and functionality is usually the final frontier in the SDLC process before releasing new or modified software to the end user. Validating that the software performs according to design specifications is time consuming, labor intensive and expensive, with costs that can equal 50%-75% of the total development expense. When a project is behind schedule, often testing is the area to be blamed. There is an obvious need for testing solutions that will reduce the necessary time required to ensure the quality and accuracy of software. This paper examines several methods available for increasing the efficiency of software testing and validation via automation.

Keywords: software, testing, automation, solution, integration

INTRODUCTION

Information technology plays a critical role in all aspects of the operational functions of a company. Rapidly evolving innovations in technology has created a demand on companies to produce and maintain state-of-the-art software in order to stay competitive. Customers, both internal and external, have come to rely on the functionality and accuracy of software. Being the first to market with new technology along with the effective deployment of software has emerged as one of the top determinants of success for a company in the business world today [12].

As features are added and the functionality is expanded, software applications increase in size and in complexity. This increase in complexity coupled with shortened development cycles and higher customer expectations of software quality has placed a major responsibility on the IT departments in a company. In the IT industry, quality of software includes reliability, on time delivery, within budget, and the assurance that the program performs according to the specifications prior to being delivered to the end user. Releasing poor quality software has negative long-term effects where recovery can be very difficult.

Software testing and validation reduces the level of uncertainty about the quality of newly developed or modified software. Research has shown that professional programmers average six defects per 1,000 lines of code [17]. Software tests are executed to affirm the quality of the software by finding and eliminating defects in the program's behavior, to demonstrate the presence of all functionality defined in the specifications, and to estimate the operational reliability of the software prior to release to the end user. Testing does not provide quality but rather helping discern quality holes.

To reduce the amount of time and effort it takes to complete the testing stage, companies are increasing the use of automated testing methods. A recent study conducted by Microsoft indicated that it takes about 12 programming hours to manually locate and correct software defects [17]. At this rate it could take more that 24,000 hours or 11.4 years to completely debug a program of 350,000 lines of code with costs in millions [17]. The results of this study illustrate the necessity of methods to effectively and efficiently test software. Not only does automation speed up the testing process, it also increases the amount of test coverage that can be done in limited time resulting in higher quality software.

This paper is organized as follows. Section 2 describes the software testing process. Section 3 discusses automating software testing and the use of automation in test generation, unit testing, execution and the use of an integrated testing environment. Section 4 concludes this paper.

TESTING

Testing is the standard process used to validate that software conforms to the formal requirements. The main goals of testing include affirming the quality of the product by finding and eliminating faults in the program, demonstrating the presence of all specified functionality in the product and estimating the operational reliability of the software. Software testing involves all the activities aimed at identifying the differences between specifications of the software and the actual behavior. The testing process includes planning, test case creation, execution of the test, validation of the results, and debugging the software after defects are identified.

A successful test requires a good test plan that focuses on testing new functionality in a dedicated testing environment. Regression testing is also necessary to ensure the new features have not adversely affected the previously existing code [19]. The test plan should include both positive and negative scenarios to demonstrate that the software reacts acceptably with good or bad data. A well-planned test attempts to prove that software performs as the specifications state while ensuring that the software will not cause problems when it is formally loaded for use. The responsibility of a test team reaches beyond trying to “break” software. The test team actually acts as an advocate looking out for the well being of both the end users of the software and the software development company itself [7].

Although testing is a very important aspect in the system development life cycle (SDLC), it is also a very expensive component with costs that can amount to 50%-75% of the total expense of software development [7]. Preparing to thoroughly test the requirements of software is very labor intensive and time consuming. Test cases and expected results are prepared to test the performance, functionality and reliability of the software. Executing the test in a controlled test environment provides data on the actual performance of the software. This data is compared to the predetermined expected results for the validation of the software. Necessary code changes are made by the developers to correct the defects that are identified during the testing process, and the software is tested again before the final product is released. Ideally, testing should continue for as long as the costs of finding and correcting defects are lower than the potential cost of software failure after release to the end user [17].

Testing is often considered as an activity that can not be performed until the software is fundamentally complete. In actuality, testing activities should occur throughout the development process. Costs to correct defects increase rapidly throughout the development cycle. There are four levels of development when testing activities should occur. These activities include unit, system, integration and acceptance tests [15]. The unit test isolates individually completed program modules from the rest of the software and uses prepared inputs to generate outcomes that can be compared with results predicted by the specifications. The programmer usually performs the unit test. System testing includes checking the interfaces and interdependencies of the various subsystems individually, then combining them to test the system as a whole to determine if the specified functionality is performed correctly [17]. Integration testing combines all system components, hardware, software and human interaction to ensure that system requirements in real or simulated environments are satisfied. This is the stage where the majority of software testing for validation occurs. Software validation is defined by the IEEE Standard Glossary of Software Engineering as the process of evaluating a system or component at the end of the development process to determine if it satisfies specified requirements [2]. After the software has passed validation, it moves into acceptance testing done by end users. Acceptance testing is the final stage of testing, and is usually performed using a BETA release of the software.

Consumer pressure in the highly competitive IT industry causes companies to accelerate the speed to market software products. Schedules are tightly restricted and developers are forced to weigh the importance of quality against possibility of missing deadlines. When a project is in danger of not being completed on schedule, testing time is often reduced so that the product can be released as planned. This behavior increases the potential for defects, leading to problems with the software that include incomplete design, poor quality, high maintenance costs, and a possible loss in customer satisfaction.

Releasing poor quality software has long-term effects that are difficult and costly to reverse. The costs involved in correcting software defects after release can be 100 times as much as preventing the same defect from occurring [17]. Software maintenance, which includes modifying software after delivery, has been claiming a large proportion of organizational resources with expenditures as high as 50%-80% of the entire IT budget for a corporation [11]. A loss in customer satisfaction is also at risk, which is difficult to recover once it is lost. In some industries, defective software can cause irreversible problems. Within the last 15 years, software defects have wrecked a European satellite launch, delayed the opening of a hugely expensive Denver airport for a year, destroyed a NASA Mars mission, killed four Marines in a helicopter crash, and shut down ambulance systems in London leading to as many as thirty deaths [21].

AUTOMATION

Automated testing uses pre-developed tools that have the capacity to increase the amount of test coverage while simultaneously reducing the time and effort to complete the test. Automation is not a quick solution for projects that are running behind schedule or over budget. Automating a test requires a full development effort involving strategy and goal planning, defining requirements, analyzing alternatives, implementation and evaluation. There is a significant investment required to implement automated testing, but it is cost effective when testing is

performed on a regular basis. Software automation testing tools provide greater efficiency and consistency to the testing process, and have proven beneficial in terms of increasing quality while minimizing project schedules and effort. This provides the company with the advantage of an earlier time to market and increased confidence in the quality of the software.

Automated tools are software products that have the ability to execute other software through the use of test scripts. These tools are able to perform tasks much faster than humans without the probability of error that is present with manual testing. A larger number of test cases can be executed in less time increasing amount of testing and code coverage. When a potential defect is found, a test script can be repeated with exactly the same inputs and sequence, which cannot be guaranteed with manual testing [15]. Reuse of test scripts saves time and helps ensure the stability of the software. Time is not a factor with automation and it usually does not require much human intervention. Using automation, testing could be performed 24 hours a day, seven days a week. If this was fully utilized, the typical test cycle could be reduced by over 75% [15].

For an automation effort to be successfully implemented, it must be treated in the same manner as other software development projects. Automation tools are only as good as the process used to implement them. An initial investment in planning, training, and development is required before any benefits can be attained [18]. Testing goals, objectives and strategies should be defined prior to choosing an automation tool. Once the testing methodology is agreed upon, tools should be evaluated to determine the compatibility with the type of testing planned, computing platform, testing environment and product features. The test system should also be evaluated for the ability of running both the software and the automation tool simultaneously. The implementation procedures should be documented and the users need to be involved throughout the process. Roles and responsibilities need to be defined and realistic expectations should be set. Automated testing rarely works perfectly the first time it's used. It is a good idea to have a contingency plan in place in case the automation fails to produce the needed results.

When implementing automation, it is better to limit the scope and begin with the most repetitive and non-time dependant tasks involved in the test cycle. Regression tests are the best targets for initial automation as regression testing is very repetitious and takes up a large portion of time spent on testing. Automation reduces the time associated with regression testing and allows the test analyst to focus on the new functionality of the software. The return on investment is also very high with regression testing as the same scripts can be used many times with little maintenance.

Specification Based Automated Test Generation

Specification based automated test generation is a relatively new process that is being used to automate the creation of tests. This method generates test cases and expected results using the formal specifications of the software. Tests are generated as multi-part artifacts in a multi-step, multi-level process [1]. The multi-part aspect means that a test case is actually composed of several components: initial input values, inputs necessary to observe the effect of the test case, and expected outputs. The multi-step aspect means that tests are generated from the functional specifications by a stepwise refinement process. This is done by refining the specifications into a graph, which is then refined into test specifications and finally into ready-to-run test scripts. The multi-level aspect means the tests are generated for use in testing the software at several levels.

There are many benefits in using specification-based automation to generate tests. Using this method to develop a test is very efficient and it removes the human element, freeing up these resources to concentrate on planning and other activities. Since the final specifications are available prior to coding the software, tests can be generated before the first line of code is written. Often when tests are generated, the test engineer will find inconsistencies and ambiguities within the specifications [1]. Identifying problems in the specifications before the code is written reduces the amount of code changes needed later in the development process. Using the formal specifications to develop tests guarantees that the software is being tested based on exactly how it was intended to perform. Results from executing these tests precisely identify what percentage of the requirements the software fulfills.

Automation during Unit Testing

Using automation during unit testing identifies defects early in the development process, which can prevent small errors made by programmers from becoming potential defects that could affect the quality of the software. Unit testing is done on individual modules of code prior to integration. Identifying and fixing errors at this stage of development is much cheaper than finding the errors after integration. Up to 80% of code errors can be found in a single pass of unit testing. Over 95% can be identified if multiple passes are used [8]. Static analysis tools are used during unit testing to analyze code without executing it. These tools look for a class of problems and flag them for investigation and fixes. They are great for locating dead code, infinite loops, data items that are stored but never used, and data items that are used prior to initializing. Static analyzers also identify which modules contain increased complexity within the code and require additional testing. Code coverage tools can also be used during unit testing to evaluate how much of the code has been exercised by a set of tests.

Automating Execution

Execution automation tools provide centralized test execution and control thus reducing errors that are inherent with humans. Execution is a complex and time consuming operation because it involves setting up the test environment, executing the test, recording the outcome, reporting the failure information, and cleaning up the environment after the test is complete [5]. Many execution automation tools are available that can be customized to existing test processes. Using these tools to manage the testing process ensures the test conforms to industry standards such as ISO or ATLM [20]. Execution tools support planning, managing and analyzing testing efforts and have the ability to reference test plans and product specifications to create traceability. The use of execution automation greatly enhances testing by providing the ability to schedule and run tests unattended, execute tests across LANs and WANs, support multi-platform testing, support synchronization for multi-test threads, as well as integrate with unit testing tools. Additional benefits include the ability to track both automated and manual tests, the ability to recover from test errors and continue execution, record test results and perform comparisons against expected results, and measure overall test success [20].

Integrated Testing Environment

The future of automated software testing within larger companies lies in the use of an Integrated Testing Environment (ITE). ITE is a single interface that presents a coherent, unified view of test information by using data and control integration with the existing tools used within an

organization [5]. This environment provides testing groups across a company access to all test data and tools as if it was stored in one single location. Testers have access to the unified test data and testing tools via a graphic user interface (GUI) interface. Existing test data is available for multiple teams to utilize. ITE significantly boosts productivity with the benefits of data and control integration and the elimination of redundant work done across testing groups.

The data integration component manages data content and associations. This component provides a coherent way to access and manage all of the test related data within the organization. Testers are given a unified view of the test data, regardless of where the data is stored. This data includes test cases, test suites, defect records, test history information and test configuration management [5]. Testers have the ability to create, read and update the test data. Traceability is provided by automatically logging any changes to existing test data into a history file for future reference. Data entered into the ITE is automatically converted into the standard format of the test architecture used and links are created to the necessary tools for executing the data [5].

There are three tiers to the data integration component: data repositories, server access to the repositories and data abstraction [5]. Existing test data is stored in various repositories that are used by the different testing groups. The repositories are linked together and accessed using a server. Individual test data items have a uniform resource locator (URI) assigned that maps the actual location that the data is stored [5]. Data abstraction abilities are used to create and manage the test data as well as maintain associations among the data.

Control integration offers services for execution and automation, providing the ability to distribute and customize the flow of control among the various tools that are integrated in the testing environment [5]. This is accomplished by using event management and workflow process control. Event management alerts resource providers within a testing environment of asynchronous events when they occur. Workflow process control involves coordinating and enforcing control of information among several distributed applications. [5]

Investing in creating an Integrated Test Environment benefits testers and the company as a whole as an ITE eliminates the fragmented testing caused by the use of various tools for different testing tasks across testing groups within an organization. A tester working in an ITE has a single interface to perform all testing duties including test case creation and reuse, execution, results reporting and analysis, tracking problems, and communicating results with other members involved in the testing process. Data and control integration promote increased testing efficiency, which leads to reduced testing cycle times and increases the speed to market.

CONCLUSION

Testing is a very important step in SDLC to ensure the quality of software. Using automation during testing reduces the time it takes to complete software testing and allows for increased test coverage. The automation methods discussed in this paper provide benefits at different stages of the testing process. Utilizing a combination of these methods would provide great benefits to a testing effort. Proper implementation of an integrated test environment that utilizes specification based test generation, tools for developing and performing unit testing, execution tools linked directly to test data and conducts analyzes the results from testing has the potential to

revolutionize the way testing is performed by an organization as well as raise software quality to a whole new level.

REFERENCES

1. Offutt, J. and S. Liu (1999). Generating Test Data from SOFL Specifications. The Journal of Systems and Software 49:1, 49-62.
2. Institute of Electrical and Electronics Engineers. (1990). IEEE Standard computer dictionary: A compilation of IEEE standard computer glossaries. IEEE, New York.
3. Loveland, S., G. Miller, R. Prewitt, M. Shannon (2002). Testing z/OS: The premier operating system for IBM's zSeries server. IBM Systems Journal 41:1, 55-.
4. Farchi, E., A. Hartman, S.S. Pinter (2002). Using a model- based test generator to test for standard conformance. IBM Systems Journal 41:1, pp. 89-.
5. Williams, C., H. Sluiman, D Pincher, M Slavercu (2002). The STCL test tools architecture. IBM Systems Journal 41:1, pp. 74-.
6. Dustin, E., J. Rashka, J. Paul (1999). Introduction of an automated test to a project, Methods & Tools, pp. 11-15.
7. B. Hailpern, P. Santhanam (2002). Software debugging, testing and verification, IBM Systems Journal 41:1, pp. 4-12.
8. Rankin, C. (2002). The software testing automation framework, IBM Systems Journal 41:1, pp. 126
9. Binder, R.V., (1994). Design for testability in object oriented systems, Communications of the ACM 37:9, pp 87-102.
10. Amman, P., P. Black (1999). Abstracting formal specifications to generate software tests via model checking. National Institute of Scientific Standards.
11. Banker, R.D., G. B. Davis, S. A. Slaughter, Software Development practices, software complexity and software maintenance performance: a field study, Management Science 44 (4) (1998) pp. 433-450.
12. Harter, D. E., M. S. Krishnan, S. A. Slaughter (2000). Effects of Process Maturity on quality, cycle time and effort in software product development, Management Science 46:4, pp 451-466.
13. Austin, R. D. (2001). The effects of time pressure on quality in software development: an agency model. Information Systems Research 12:2, pp 195-207.
14. Buthcer, M., H. Munro, T. Kratchmer (2002). Improving Software via ODC: Three Case Studies. IBM Systems Journal 41:1, pp. 31-.
15. Fewster, M., D. Graham (1999). Software Test Automation: Effective Use of Test Execution Tools, ACM Press, New York, NY.
16. Perry, W. (1986). How to Test Software Practices: A Step- By- Step Guide to Assuring They Do What You Want, John Wiley & Sons, New York, NY.
17. Pham, H. (2000), Software Reliability, Springer, Singapore.
18. Hayes, L. (1995). The Automated Testing Handbook, The Software Testing Institute, Dallas, TX.
19. Kaner, C., J. Falk, H.Q. Nguyen (1999). Testing Computer Software, Wiley Computer Publishing, New York, NY.
20. Dunstin, E. (2001). Automated Test Tool Evaluation Matrix, Quality Web Systems, Addison Wesley.
21. Mann, C.C. (2002). Why is software so bad?, Technology Review, July/August.