International Journal of Engineering Sciences Paradigms and Researches, Vol. 02, Issue 01, December 2012
ISSN (Online): 2319-6564
www.ijesonline.com

A Mathematical Study of Software Testing Regarding Computer Measurements

Suman Lata[1], Navita[2], Dr. Pardeep Goel[3]

[1,2]Research Scholar, Singhania University, Rajasthan (India)

[3]Associate Professor, M.M. College, Fatehabad, Haryana (India)

Abstract

The paper explains the mathematical study of software testing and the measurement regarding software. Here we explain the planning regarding software testing and the metrics regarding software testing. Also the different attributes for the software test planning are discussed in this paper. The benefits of measurement in software testing are explained in detail here. This paper shows that the software measurement is the prime portion in the software testing.

Keywords: *Software Testing, Software Measurement, Software Metrics, Software Testing Attributes.*

Introduction

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs (errors or other defects).

Software testing can be stated as the process of validating and verifying that a computer program/application/product:

- meets the requirements that guided its design and development,
- works as expected,
- can be implemented with the same characteristics,
- And satisfies the needs of stakeholders.

Software testing, depending on the testing method employed, can be implemented at any time in the development process. Traditionally most of the test effort occurs after the requirements have been defined and the coding process has been completed, but in the agile approaches most of the test effort is on-going. As such, the methodology of the test is governed by the chosen software development methodology.

Different software development models will focus the test effort at different points in the development process. Newer development models, such as Agile, often employ test-driven development and place an increased portion of the testing in the hands of the developer, before it reaches a formal team of testers. In a more traditional model, most of the test execution occurs after the requirements have been defined and the coding process has been completed.

History

The separation of debugging from testing was initially introduced by Glenford J. Myers in 1979. Although his attention was on breakage testing ("a successful test is one that finds a bug") it illustrated the desire of the software engineering community to separate fundamental development activities, such as debugging, from that of verification. Dave Gelperin and William C. Hetzelclassified in 1988 the phases and goals in software testing in the following stages:

- Until 1956 - Debugging oriented
- 1957–1978 - Demonstration oriented
- 1979–1982 - Destruction oriented
- 1983–1987 - Evaluation oriented
- 1988–2000 - Prevention oriented

A Sample Testing Cycle

Although variations exist between organizations, there is a typical cycle for testing. The sample below is common among organizations employing the Waterfall development model.

- Requirements analysis: Testing should begin in the requirements phase of the software development life cycle. During the design phase, testers work with developers in determining what aspects of a design are testable and with what parameters those tests work.
- Test planning: Test strategy, test plan, testbed creation. Since many activities will be carried out during testing, a plan is needed.
- Test development: Test procedures, test scenarios, test cases, test datasets, test scripts to use in testing software.
- Test execution: Testers execute the software based on the plans and test documents then report any errors found to the development team.
- Test reporting: Once testing is completed, testers generate metrics and make final reports on their test effort and whether or not the software tested is ready for release.
- Test result analysis: Or Defect Analysis, is done by the development team usually along with the client, in order to decide what defects should be assigned, fixed, rejected (i.e. found software working properly) or deferred to be dealt with later.
- Defect Retesting: Once a defect has been dealt with by the development team, it is retested by the testing team. AKA Resolution testing.
- Regression testing: It is common to have a small test program built of a subset of tests, for each integration of new, modified, or fixed software, in order to ensure that the latest delivery has not ruined anything, and that the software product as a whole is still working correctly.
- Test Closure: Once the test meets the exit criteria, the activities such as capturing the key outputs, lessons learned, results, logs, documents related to the project are archived and used as a reference for future projects.

Benefits of Measurement in Software Testing

1) Identification of testing strengths and weaknesses.
2) Providing insights into the current state of the testing process.
3) Evaluating testing risks.
4) Benchmarking.
5) Improving planning.
6) Improving testing effectiveness.
7) Evaluating and improving product quality.
8) Measuring productivity.
9) Determining level of customer involvement and satisfaction.
10) Supporting controlling and monitoring of the testing process.
11) Comparing processes and products with those both inside and outside the organization.

Methodology

Program testing and fault detection can be aided significantly by testing tools and debuggers. Testing/debug tools include features such as:

Program monitors, permitting full or partial monitoring of program code including:

Instruction set simulator, permitting complete instruction level monitoring and trace facilities

Program animation, permitting step-by-step execution and conditional breakpoint at source level or in machine code

Code coverage reports

Formatted dump or symbolic debugging, tools allowing inspection of program variables on error or at chosen points

Automated functional GUI testing tools are used to repeat system-level tests through the GUI

Benchmarks, allowing run-time performance comparisons to be made

Performance analysis (or profiling tools) that can help to highlight hot spots and resource usage

Some of these features may be incorporated into an Integrated Development Environment (IDE).

A regression testing technique is to have a standard set of tests, which cover existing functionality that result in persistent tabular data, and to compare pre-change data to post-change data, where there should not be differences, using a tool like diffkit. Differences detected indicate unexpected functionality changes or "regression".

Metric Support for Strategy

The attributes falling in the category of strategy included

• Sequence of test cases.

• Identification of areas for further testing.

• Combination of test techniques.

• Adequacy of test data.

Measurement in Different Phases of Software Testing

The software testing cycle starts from project startup and ending to test process improvement. There are ten more steps between these two which are shown in the table 1 with the different measurements Attributes.

Phase	Historical Value	% of Project	Preliminary Estimate	Adjusted Estimate
Project Startup	140	2.6	179	179
Early Project Support (Requirements analysis, etc.)	120	2.2	152	152
Decision to Automate Testing	90	1.7	117	-
Test Tool Selection and Evaluation	160	3	207	-
Test Tool Introduction	260	5	345	345
Test Planning	530	10	690	690
Test Design	540	10	690	690
Test Development	1980	37	2553	2553
Test Execution	870	17	1173	1173
Test Management and Support	470	9	621	621
Test Process Improvement	140	2.5	173	-
Project Total	5300	100%	6900	6403

Table 1: Measurement in Different Testing Phases

Limitations of the approaches used for estimation

1. The projections from the past testing effort cannot be relied upon entirely. The testing professionals are required to use their experience to deal with unusual circumstances.

2. The testing maturity of the organization must be considered while estimating.

3. The scope of the testing requirements must be clear.

4. The introduction of an automated tool introduces complexity into the project and must be considered appropriately.

5. The domain knowledge of tester is an important attribute that affects the time lines.

6. The test team organizational styles must be taken into consideration.

7. The time at which test planning activity begins matters because early start of the test planning activity enables better understanding of requirements and early detection of errors.

8. The presence or absence of documented processes for testing makes up another important factor for scheduling testing time.

9. Highly risky software requires more detailed testing and must be taken into account while estimating testing schedule.

10. There is a likelihood of change in requirements and design during development.

11. The software should be delivered on time for testing.

Conclusion

Measurement is a tool through which the management identifies important events and trends, thus enabling them to make informed decisions. Moreover, measurements help in predicting outcomes and evaluation of risks, which in turn decreases the probability of unanticipated surprises in different processes. In order to face the challenges posed by the rapidly changing and innovative software industry, the organizations that can control their software testing processes are able to predict costs and schedules and increase the effectiveness, efficiency and profitability of their business.

References

[1] National Archives and Records Administration. Testing Management Plan. Integrated Computer Engineering, Inc. a subsidiary of American Systems Corporation (ASC), 2003.
[2] R. Craig. Test Strategies and Plans. Copyright Software Quality Engineering, Inc., 1999
[3] K. Iberle. Divide and Conquer: Making Sense of Test Planning. In the International Conference on Software Testing, Analysis and Review, STARWEST, 1999.
[4] R. Shewale. Unit Testing Presentation. A StickyMinds Article.
[5] http://www.stickyminds.com/getfile.asp?ot=XML&id=6124&fn=XDD6124filelistfilename1%2Eppt, November 2006.
[6] R. Fantina. Practical Software Process Improvement. Artech House Inc., 2005.
[7] R. Black. Managing the Testing Process. Second Edition. Wiley Publishing, Inc., 2002.
[8] J. Bach. Testing Testers — Things to Consider When Measuring Performance. A StickyMinds Article. http://www.stickyminds.com, November 2006.
[9] C. Kaner. Measuring the Effectiveness of Software Testers. Progressive Insurance, July 2006.
[10] The Standish Group. Chaos Report. Twww.projectsmart.co.uk/docs/chaos_report.pdf, 1995T.
[11] K. Molokken and M. Jorgensen. A Review of Surveys on Software Effort Estimation. Simula Research Laboratory, 2003.
[12] N. Bajaj, A. Tyagi, R. Agarwal. Software Estimation – A Fuzzy Approach. ACM SIGSOFT Software Engineering Notes Volume 31 Number 3, May 2006.
[13] J. P. Lewis. Limits to Software Estimation. ACM SIGSOFT Software Engineering Notes Volume 26 Number 4, July 2001.
[14] D. V. Ferens, D. S. Christensen. Does Calibration Improve the Predictive Accuracy of Software Cost Models? CrossTalk, April 2000.
[15] K. Iberle, S. Bartlett. Estimating Tester to Developer Ratios (or Not). Hewlett-Packard and STEP Technology www.kiberle.com/pnsqc1/estimate.doc, November 2006.
[16] L. M. Laird, M. C. Brennan. Practical Software Measurement and Estimation: A Practical Approach. Copyright IEEE Computer Society, 2006
[17] W. E. Lewis. Software Testing and Continuous Quality Improvement. Second Edition. Auerbach Publications, 2005.
[18] R. Patton. Software Testing. Sams Publishing, July 2006.
[19] M. Marre´, A. Bertolino. Using Spanning Sets for Coverage Testing. In IEEE Transactions on Software Engineering, Volume 29, Number 11, November 2003.
[20] S. Cornett. Code Coverage Analysis. Bull Seye Testing Technology. http://www.bullseye.com/coverage.html, Dec 2005.
[21] C. Kaner. Software Negligence and Testing Coverage. http://www.kaner.com/coverage.htm, 1996.
[22] P. Piwowarski, M. Ohba, J. Caruso. Coverage Measurement Experience During Function Test. International Business Machines Corporation. In IEEE Software Engineering Proceedings, 1993.