Journal of Information Technology Management

ISSN #1042-1319

A Publication of the Association of Management

ATCG: AN AUTOMATED TEST CASE GENERATOR

CHARLES P. MORGAN
FEDEX SERVICES CORPORATION
chuck.morgan@fedex.com

MARK L. GILLENSON
UNIVERSITY OF MEMPHIS
mark.gillenson@memphis.edu

XIHUI ZHANG
UNIVERSITY OF NORTH ALABAMA
xzhang6@una.edu

SON N. BUI
TEXAS A&M UNIVERSITY–COMMERCE
son.bui@tamuc.edu

EUNTAE "TED" LEE
UNIVERSITY OF MEMPHIS
elee@memphis.edu

ABSTRACT

Testing software to validate its functionality requires the development of carefully crafted test cases. Test cases can be developed algorithmically, they can be taken from existing data, or they can be developed from requirements. In this paper, we describe an Automated Test Case Generator (ATCG) that takes requirements statements as inputs and creates test cases as outputs. The ATCG represents a better, systematic way to develop test cases automatically from requirements, and provides multiple benefits and in the meantime, alleviates some potential issues with manually developing test cases from requirements.

Keywords: Software testing, test case, test case generator, requirements document.

INTRODUCTION

Every newly developed product of any kind has to be tested to ensure that it correctly performs the functions for which it was designed. This is true of an airplane, an oil refinery, or a washing machine. It is also true of software, whether it is systems software or application software. Software can vary from simple applications to very complex applications and systems, and it all must be tested. The fact that in today's business environment, software applications have to function on or be accessed from a panoply of devices ranging from smart phones to mainframes, each with a variety of operating systems and versions, multiplies the testing problem considerably [6].

Testing software to validate its functionality requires the development of carefully crafted test cases. A test consists of executing a test case and comparing the actual result to the expected result. Broadly speaking, test cases can be developed in one of three major ways. They can be developed algorithmically using techniques such as pairwise analysis [6] [9], they can be taken (possibly with modifications) from data from an existing application that is being replaced or upgraded [6] [15], or they can be developed from requirements [5] [6] [8] [9] [13]. This paper will focus on developing test cases from requirements.

Developing test cases from requirements has several distinct advantages [5] [6] [8] [9] [13]:

- The assurance that the software that supports every requirement is being tested.
- The involvement of the business personnel who requested the application in the testing process.
- Traceability of a defect, based on a test case failure, to the requirement and thus to the portion of the software that was written for the requirement.

However, there are also some potential issues with manually developing test cases from requirements [5] [6] [8] [9] [13]:

- The process is labor-intensive and can be unacceptably time consuming.
- If the person writing the test cases is not one of the business people who developed the requirements (e.g., a tester) then there is a significant time element in reviewing and understanding the requirements.
- The requirements may not have been broken down into fine enough divisions for a comprehensive set of test cases to be developed.
- A test case developed from a requirement may not correctly test it.
- Too much time may be required to review requirements documents and ask for clarifications.
- Additional time may be required to enter test cases in a test case management system.
- Sometimes there are missed test cases requirements due to miscommunication in the requirements or a missed requirement itself.
- Extra time is required to add missed test cases to the test bed and re-test.
- Test cases that affect multiple systems sometimes are either missed or not tested.
- If necessary test cases are not included, it can lead to defects not caught that may show up in production.

This paper describes an Automated Test Case Generator (ATCG) that takes requirements statements as inputs and creates test cases as outputs. We start by presenting a review of the related work with a focus on scripted testing and model-based testing. We then describe the Automated Test Case Generator (ATCG), a software tool that can read requirements documents and automatically detect and generate test cases. And finally, we summarize the paper, discuss the advantages of using the ATCG, and outline future research.

RELATED WORK

Software testing is indispensable in ensuring software quality [2]. It has been estimated that software testing uses up to 50% of the overall development cost [10] and the testing activities consume approximately 40% of the overall development time and effort [14]. To improve the effectiveness and efficiency of testing, testers need to create high-quality test cases. Writing test cases, however, is an arduous task and often times involves human errors. Thus, it is crucial that we can find a way of automatically generating high-quality test cases which can be used in making testing activities more effective and efficient to assure software quality.

Automatically generating test cases for testing purposes is not a new idea. For instance, Weyuker et al. [15] introduced an approach to automatically generate test sets that would be highly effective at detecting faults. In requirement-based testing, the need of automation is even more serious. Requirement-based testing normally requires multiple manual processes where software testers have to design test cases from requirements, run the test cases, and verify whether requirements of the software/system under test are met. If the requirement-based testing processes do not run correctly or consistently, the size of the test cases may be too enormous to complete testing in reasonable time or the test cases cannot provide the expected results [1]. To minimize the effort and cost on testing, many companies have started investing in the automation of testing processes and tools. Automation has proven to be an effective way to reduce cost and shorten the product release time to market, and will be a major factor for the success of software testing [13].

In general, this effort can be divided by two types of automated test case generation techniques: white-box (code based approach) and black-box (specification based approach). While each of the two techniques has its own advantages and disadvantages and both are used commonly in automated test case generation, the emphasis in this paper is on black-box test automated test case generation [13].

There are several black-box test approaches to automatically generating test cases in requirement-based testing. These methods are semi-automated ones that require software/system specifications expressed in precise notations. However, most of the software/system requirements currently used in the software industry are written in natural language and thus normally require software testers or systems analysts to translate them into precise software/system specifications. The two most common requirement-based testing methods that can semi-automatically generate test cases, scripted testing and model-based testing, are reviewed as follows.

Scripted Testing

Scripted testing follows a single path that is written by software testers. The path normally follows multiple steps and phrases that test cases are designed to follow.

Scripted testing has three important characteristics: repeatability, objectivity, and auditability. Repeatability refers to the ability to execute the scripts multiple times in an identical way. If product A passed the first test but failed the second test, the scripted testing has to be able to reproduce the testing process to test product A. Objectivity refers to the ability to follow good test design principles rather than well-skilled testers. For example, scripts are used not only in early testing phases but also in later testing activities to catch defects. Finally, auditability refers to the ability to trace from requirements, to design, to execution, and back again. This ability allows software testers to go back to fix any defects that early testing activities failed to identify [3].

Oftentimes, scripted testing is used in the waterfall method. The "IEEE Standard for Software Test Documentation" [7], defines eight documents that can be used in software testing, and these eight documents correspond to eight steps that scripted testing needs to cover: test plan, test design specification, test case specification, test procedure specification, test item transmittal report, test log, test incident report, and test summary report [3].

In the testing procedure described above, each of the eight steps can be automated except for the first four steps (i.e., test plan, test design specification, test case specification, and test procedure specification) due to the complexity of test design. These four steps are usually based on the system requirements that the software testing team is required to specify and document. Thus, scripted testing still requires a lot of effort to translate system requirements to test case specification and test procedure specification in order to automatically execute the test cases [3].

Model-Based Testing

Another popular black-box automated test case generation method is model-based testing. Model-based testing is the method that automatically generates test cases from models derived from system requirements and system behaviors [4]. Typically in order to generate test cases, software testers need to develop models to represent how the intended systems could work from system requirements documents. The goal of the model is to capture the functional requirements of the system in a clear, concise and executable way that the language cannot express [5]. The requirements model can be used to evaluate the completeness of the system when developing and testing test cases. In order to develop a requirements model, modeling languages such as Unified Modeling Language (UML) and SysML are typically used to create interactions and descriptions for system behaviors [8] [12].

For example, Nebut et al. [11] introduced a semi-automated method of generating test cases by using the UML method. This approach creates simulation models from systems requirements to automatically generate test cases and execute them.

While much of this method is automated, an important phase that converts the requirements language to system models is manually done by the software testing team. This process, however, is very time-consuming and prone to error.

In summary, neither scripted testing nor model-based testing has been effective or efficient in automatically generating test cases for requirement-based testing. Thus, in this paper, we introduce a new method that can automatically translate the system requirements to test cases in a more effective and efficient way. Our method can read requirements documents from business and development teams, and then automatically analyze them and generate test cases accordingly.

AUTOMATED TEST CASE GENERATOR (ATCG)

The Automated Test Case Generator (ATCG) is a software tool that can read requirements documents from the business and development teams, for the purpose of automatically detecting and generating test cases. The ATCG is run on the Windows desktop by software testing personnel. After a requirements document is selected for analysis using the ATCG tool, a test case can be generated as follows:

1. The ATCG electronically “reads” the document as a human would do by parsing lines of text.
2. When a sentence appears to contain a requirement, it is decomposed and analyzed for content, word-by-word:
   - An attempt is made to analyze the sentence by subject-verb-predicate order.
   - Action verbs and key words are looked for in the sentence, using user configurable word lists.
   - If the sentence proves to contain a test case, then the test case is created.

Once the test cases are created, they are stored in a centralized database so that everyone can review them. Test cases can be edited for revisions, printed, and imported to a test case management system and emailed to others. The architectural overview of the ATCG is shown in Figure 1. A search feature allows the search for test cases that are common across multiple systems.

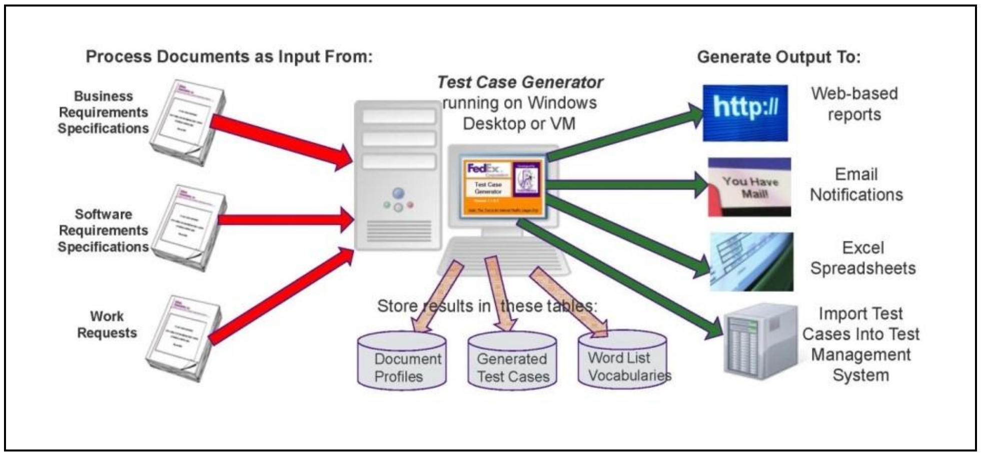
Process Documents as Input From:
Business Requirements Specifications
Software Requirements Specifications
Work Requests
Test Case Generator running on Windows Desktop or VM
Store results in these tables:
Document Profiles
Generated Test Cases
Word List Vocabularies
Generate Output To:
Web-based reports
Email Notifications
Excel Spreadsheets
Import Test Cases Into Test Management System

Figure 1: Architectural Overview of the Automated Test Case Generator (ATCG)

The Detailed Steps to Automatically Create Test Cases

There are 5 steps to automatically generate test cases based on the requirements document. Each step will be discussed with relevant examples.

Step 1

The ATCG loads into memory the lists of key words and phrases that will help it to identify test cases. The lists by the ATCG are shown in Table 1.

Table 1: The ATCG Lists and Purposes

List	Purpose
Action Verbs	Used for determining verbs that indicate some actionable phrase
Technology Terms	Words that are commonly used technology terms in requirements
Nouns	Commonly used nouns that may appear as test inputs
Adjectives	Adjectives used to describe the types of test inputs
Prepositions	Prepositions that may be used to indicate placement of test inputs
Test Data Values	A list of test data values that can be static, random, or in a range

Step 2

The ATCG parses the requirements document to look for phrases that may contain requirements reference numbers. This is accomplished by identifying numerical values formatted in patterns that typically are used for software requirements numbering. If a reference is found, it will remember its requirements reference number and apply it to subsequent test cases generated (this is an optional step and if no reference is found, test cases can still be generated). The examples of requirements reference numbers are SR 5421503, SR 5421522, SR 5421527, and SR 5421539 as shown in Table 2.

Table 2: Examples of Requirements Reference Numbers

Requirement ID	Requirement Name	Requirement Description
6.7.21.1 SR 5421503 *Version: 6.0* *Owner: John Doe* *Project/Load: 2014.TEST.01* *Testable: true* *Work Request* WR5912	Support Variable Length Postal Codes	The system shall support alpha-numeric postal codes of variable length not to exceed the database limit of 5 characters. BR 6.1.1
6.7.21.2 SR 5421522 *Version: 2.0* *Owner: John Doe* *Project/Load: 2014.TEST.01* *Testable: true* *Work Request* WR5912	Support Alpha-Numeric Sector Codes	The system shall support two character alpha-numeric sectors on the user interface. BR 6.1.2
6.7.21.3 SR 5421527 *Version: 1.0* *Owner: John Doe* *Project/Load: 2014.TEST.01* *Testable: true* *Work Request* WR5912	Modify Time Segment	The system shall support minute-resolved time segments. Current default: 15 min resolution Proposed change: 1 minute resolution BR 6.1.3
6.18.1.8 SR 5421539 *Version: 3.0* *Owner: John Doe* *Project/Load: 2014.TEST.01* *Testable: true* *Work Request* WR5912	Support Mass Updates	The system shall provide a "mass update" capability to manage a one route to many 'relationship. BR 6.1.4

Step 3

The ATCG parses the requirements document looking for complete sentences. A complete sentence must start with a capital letter and end with a period. Once a sentence is identified, the ATCG looks for an action verb in the sentence from a list of user-configurable verbs. It then separates the sentence into three parts: subject, verb, and predicate. The following is an example sentence found in a requirements document: "The user interface shall support 5 digit postal codes for US locations."

Table 3: An Example Sentence in a Requirements Document

Subject	Verb	Predicate
The user interface	shall support	5 digit postal codes for US locations.

Step 4

Using the predicate found in the sentence, the ATCG searches through the vocabulary lists to look for phrases that could be expected test inputs. Once a test input is found, it looks in the list of test data values to obtain a value to use as the expected input.

If a sentence, "The user interface shall support 5 digit postal codes for US locations," is found, then "5 digit postal codes for US locations" will be identified as a predicate, and the two test input names will be found within the predicate along with matched test data values in the test data list as shown in Table 4 below:

Table 4: Test Input Names and Test Data Values

Expected Test Input Name	Test Data Value Found in List
5 digit postal codes	38103
US locations	Memphis, TN

Step 5

Finally, a test case is generated, including the following:

- A sequential test case number is generated.
- The requirements reference number identified in Step 2 is included.
- The test case description is derived from the requirements sentence.
- Test case type is specified as positive, negative, boundary analysis, etc.
- The test inputs that were identified in the requirements, along with test data values chosen from the test data lists available.
- The expected results are created from the sentence found in the requirements by making it into a declarative sentence.

Table 5 shows an example of a generated test case.

Table 5: An Example of a Generated Test Case

Test Case Number	03901	
Reference Number Found	SR39029	
Test Case Type	Positive	
Test Case Description	Test for support 5 digit postal codes for US locations	
Expected Test Inputs	Test Input Name	Test Data Value
	5 digit postal code	38103
	US Location	Memphis, TN
Expected Results	The user interface supports 5 digit postal codes for US locations	

Correcting Expected Test Inputs

If a generated test case has expected test inputs, but the ATCG cannot locate any equivalent test data to use in the list of test data input, then the user will be notified that the test case is missing some test data. The user will then be shown a list of expected test inputs that are missing test data, and will be prompted to edit the test data inputs list to provide test data for these expected test inputs. For example, suppose that the ATCG cannot locate test data for an expected test input "US locations" as follows.

Table 6 shows some examples of test input names and test data values to use (partial):

Table 6: Test Input Name and Test Data Value to Use

Test Input Name	Test Data Value to Use
postal codes	38103
City	Chicago
Employee ID	99999

Table 7 shows an example of "No Test Data Found" for an expected test input in a test case:

Table 7: An Example of No Test Data Found for an Expected Test Input

Expected Test Input Name	Test Input Value Found
5 digit postal codes	38103
US locations	No Test Data Found

If needed, the ATCG can generate an email to the requirements author to clarify the expected test input's purpose.

Available Functions in the Automated Test Case Generator (ATCG)

To make the ATCG more use-friendly and easier to use, the following features/functions are available as tabs on the Windows interface: Documents, Requirements, Test Cases, Vocabularies, Tools, Preferences, Test Data, and Help. The full descriptions of these features/functions are provided in Table 8.

Table 8: The ATCG Tabs and Functionalities

Tab	Functionality
Documents	View the list of documents that have been submitted for test case generation, and search all documents for test cases using key word search capability
Requirements	Create a requirements document profile to be analyzed for test cases, and begin the process of test case generation
Test Cases	View the test cases generated for the currently opened requirements document
Vocabularies	View/edit the action verb and terminology vocabularies used for test case analysis
Tools	Access online tools useful in the testing process (requirements management, etc.)
Preferences	Set the test case generation preferences to be applied when generating test cases
Test Data	View/edit the list of test data parameters used for automatic test data generation
Help	Perform a sentence analysis test of the test case generator to generate a test case

SUMMARY AND CONCLUSION

Testing software to validate its functionality requires the development of carefully crafted test cases. Test cases can be developed algorithmically using techniques such as pairwise analysis, they can be taken (possibly with modifications) from data from an existing application that is being replaced or upgraded, or they can be developed from requirements. In this paper, we try to find a better, systematic way to develop test cases automatically from requirements in order to alleviate some potential issues with manually developing test cases from requirements as mentioned earlier in the Introduction section.

To improve the process of creating test cases, the ATCG can read requirements documents from business and development teams, for the purpose of automatically detecting and generating test cases. After a requirements document is selected for analysis using the ATCG, a test case can be generated as follows:

1. The ATCG tool electronically "reads" the document as a human would do by parsing lines of text.
2. When a sentence appears to contain a requirement, it is decomposed and analyzed for content, word-by-word.

Once the test cases are created, they are stored in a centralized database so that everyone can review. Test cases can be edited for revisions, printed, and emailed to others.

The Automated Test Case Generator (ATCG) will be evaluated for use on a regular basis. It should be noted that the accuracy of test case generation is heavily dependent on the quality and standardization of the requirements documents. When used correctly, the ATCG can provide the following benefits:

- To improve the productivity of test case design
- To increase test coverage over manual test case writing methods
- To improve end-to-end testing by identifying test cases that affect multiple systems
- To save time in generating test data by having the ATCG automatically choose test data

In addition, we believe that an agile version of the automated test case generator could also be created that reads user stories, and then generates test cases continuously, as the user stories change. In future research, we plan to conduct an empirical study by collecting experiential data to confirm whether the promised benefits are actually realized and to what extent.

REFERENCES

[1] Bender RBT Inc. "Requirements Based Testing Process Overview," http://benderrbt.com/Bender-Requirements%20Based%20Testing%20Process%20Overview.pdf, 2009, pp. 1-18.

[2] Cohen, C.F., Birkin, S.J., Garfield, M.J., and Webb, H.W. "Management Conflict in Software Testing," *Communications of the ACM*, Volume 47, Number 1, 2004, pp. 76-81.

[3] Copeland, L. *A Practitioner's Guide to Software Test Design*, Artech House Publishers, Norwood, Massachusetts, USA, 2004.

[4] Dias Neto, A.C., Subramanyan, R., Vieira, M., & Travassos, G.H. "A Survey on Model-based Testing Approaches: A Systematic Review," *Proceedings of the 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies*, Atlanta, Georgia, USA, November 5-9, 2007, pp. 31-36.

[5] Escalona, M.J., Gutiérrez, J.J., Mejías, M., Aragón, G., Torres, J., and Domínguez, F.J. "An Overview on Test Generation from Functional Requirements," *Journal of Systems and Software*, Volume 84, Number 8, 2011, pp. 1379-1393.

[6] Hass, A.M. *Guide to Advanced Software Testing* (2nd Edition), Artech House Publishers, Norwood, Massachusetts, USA, 2014.

[7] IEEE Std. 829-1998. "IEEE Standard for Software Test Documentation," http://faculty.ksu.edu.sa/mohamedbatouche/SWE%20434/IEEE%20Std%20829%20-%201998.pdf, sponsored by the Software Engineering Technical Committee of the IEEE Computer Society, September 16, 1998, pp. 1-59.

[8] Lee, C.C. and Friedman, J. "Requirements Modeling and Automated Requirements-based Test Generation," *SAE International Journal of Aerospace*, Volume 6, Number 2, 2013, pp. 607-615.

[9] Mathur, A.P. *Foundations of Software Testing* (2nd Edition), Dorling Kindersley (India) Pvt. Ltd, Noida, India, 2013.

[10] Nagpal, K. and Chawla, R. "Improvement of Software Development Process: A New SDLC Model," *International Journal of Latest Research in Science and Technology*, Volume 1, Number 3, 2012, pp. 217-224.

[11] Nebut, C., Fleurey, F., Le Traon, Y., and Jezequel, J.M. "Automatic Test Generation: A Use Case Driven Approach," *IEEE Transactions on Software Engineering*, Volume 32, Number 3, 2006, pp. 140-155.

[12] Pretschner, A., Prenninger, W., Wagner, S., Kühnel, C., Baumgartner, M., Sostawa, B., Zölch, R., and Stauner, T. "One Evaluation of Model-based Testing and Its Automation," *Proceedings of the 27th International Conference on Software Engineering*, St. Louis, Missouri, USA, May 15-21, 2005, pp. 392-401.

[13] Tahat, L.H., Vaysburg, B., Korel, B., and Bader, A.J. "Requirement-based Automated Black-box Test Generation," *Proceedings of the 25th Computer Software and Applications Conference*, Chicago, Illinois, USA, October 8-12, 2001, pp. 489-495.

[14] Vijay, N. "Little Joe Model of Software Testing," Software Solutions Lab, Honeywell, Bangalore, India, 2001, pp. 1-12.

[15] Weyuker, E., Goradia, T., and Singh, A. "Automatically Generating Test Data from a Boolean Specification," *IEEE Transactions on Software Engineering*, Volume 20, Number 5, 1994, pp. 353-363.

AUTHOR BIOGRAPHIES

Charles P. Morgan is a Technical Advisor for FedEx Services Corporation, located in Memphis, Tennessee, and is a graduate of the University of Memphis, B.S. in Engineering Technology, 1979. Charles has over 38 years of experience in the information technology field in various disciplines, including software development, software testing, and technology research. He is currently active in the areas of software testing tools development, mobile device software, and solutions architecture. Charles developed the ATCG: Automated Test Case Generator software, as well as other applications which utilize innovative techniques to more efficiently automate manual processes.

Mark L. Gillenson is Professor and formerly Department Chair of Business Information and Technology at the University of Memphis. He is also the Director of the Big Data and Analytics Research Cluster in the university's FedEx Institute of Technology. He is the author of several books on database management and numerous journal articles. His current research interests include advanced database systems and software testing.

Xihui Zhang is an Associate Professor of Computer Information Systems in the College of Business of the University of North Alabama. He earned a Ph.D. in Business Administration with a concentration in Management Information Systems from the University of Memphis, 2009. He has published in such leading journals as the *Journal of Strategic Information Systems*, *Information & Management*, and *Journal of Database Management*. He serves as the Managing Editor of *The Data Base for Advances in Information Systems*, and he also serves on the editorial review board for several academic journals, including the *Journal of Computer Information Systems*, *Journal of Information Systems Education*, and *Journal of Information Technology Management*.

Son N. Bui is an Assistant Professor of Marketing & Business Analytics department in the College of Business of the Texas A&M University–Commerce. His work focuses on the application of business analytics for small and medium enterprises. His researches have been published in Journal of Information Technology Management and Information Technology & Management.

Euntae "Ted" Lee received his Ph.D. in Management Information Systems from University of Nebraska and M.S. in Computer Science from The Pennsylvania State University, and B.S. in Atmospheric Science from Seoul National University, Seoul, Korea. His primary research interests are knowledge engineering and management, business rule managements systems, database systems, business intelligence and analytics, software testing, and strategic use of organizational information systems. Dr. Lee's work has been published in many IT related journals and conference proceedings.