Chapter 1

AuTester a framework for Automated Testing and Test Management

Arkadiusz KUMPIN¹, Marian JURECZKO²

1. INTRODUCTION

The dependability of software systems become recently a crucial feature in many areas. Failures in software systems or software defects removing are enormously costly. Software engineering puts a lot of attention on developing software with high level of quality and dependability. Many researchers and practitioners believe that following a well defined process should assure obtaining of high quality artifacts as the process output. Therefore, we have so many software development processes and organization maturity models, like CMMI. However, the most important activity that leads to increase in the dependability of software based systems is testing. Since manual testing is a dull and time-consuming task, a number of tools exist to automate it. This paper presents an open source framework for automated testing and test management - AuTester. This system is not yet ready for commercial use, but all main features are implemented, so it is possible to check its capabilities. It provides a complex functionality to build test scenarios and to execute them, as well as efficiently manage all the test artifacts according to the IEEE 829 testing standard [28] and Regulation of Ministry of Science and Computerization for acceptance tests and examination of interface software and verification of this examination (Polish regulation) [19]. Support for those two testing guidelines is extremely rare, especially in tools that are free of charge. AuTester design emphasizes extensibility, modularity, usability and portability across common platforms. Therefore, further changes are easy to apply and the AuTester tool may be adapted to new requirements without big effort.

This paper is organized as follows: the next section describes motivation and goal. Section 3 discussed related work. Section 4 briefly introduces to the IEEE 829 and the Polish regulations. The 5th section discusses the functionalities and architecture of the AuTester. Some concluding remarks are presented in the final section.

¹ Institute of Computer Engineering, Control and Robotics, Wrocław University of Technology, Wybrzeże Wyspiańskiego 27, 50-370, Wrocław - Poland, 140066@student.pwr.wroc.pl
² Institute of Computer Engineering, Control and Robotics, Wrocław University of Technology, Wybrzeże Wyspiańskiego 27, 50-370, Wrocław - Poland, marian.jureczko@pwr.wroc.pl
2. MOTIVATION AND GOALS

A modern test process requires test automation [5,7,12,20]. Having great number of automated test cases implicates the need of test management. There is an IEEE standard [26] and some guidelines of application of the standard [3,25] where the test management and documentation are discussed. Unfortunately, there are very few tools that cover both of the areas, the test automation and the test management. The IEEE 829 [26] is worldwide known and therefore, quite well covered by tools, but there are also other, local regulations that may be relevant. In our case that is the Polish regulation [19]. The Polish government is going to launch several big software projects. The goal is to have a complex e-administration system. Each of the applications that will be implemented within the confines of e-administration has to be tested according to the Polish regulation [19]. There are no tools that support those regulations, hence all tests must be performed and documented manually. That indicates enormous costs. The main goal of this work is to fulfil this gap by designing and implementing a test automation and management tool. In the field of test automation it should be easy to use and to integrate with existing software development environments and continuous integration systems and on the field of test management it must support the Polish regulation [19]. In order to allow worldwide usage of our tool, we decided to add support for the IEEE 829 [26] as well. The tool (called AuTester) has been already implemented. The paper briefly presents its design, discusses its functionalities and considers its further usage and development.

3. RELATED WORK

There are many test automation tools, a survey of the most popular open source solutions (Fit/FitNesse³, JfcUnit⁴, Proven! and Selenium⁵) can be found in [8], there are also several commercial tools, like HP Quality Center or Rational quality management solutions. Considerable research has been performed on designing and implementing new test automation tools. Xie et al. [27] suggested a test framework (called Symstra) that is not only suitable for test automation, but test generation as well. Although the framework works only with unit tests and does not consider test management. Okresa and Solcany [22] have recently presented a very interesting tool (called Ascella) that supports test automation through application GUI. Unfortunately, according to the authors’ knowledge the Ascella tool is still in development phase and therefore has not been made publicly available yet. Memon presented a comprehensive approach to GUI test in his PhD thesis [13]. He discusses comprehensively the theoretical basis and a test

---

³ http://www.fitnesse.org  
⁴ http://jfcunit.sourceforge.net  
⁵ http://seleniumhq.org
framework. Some interesting findings about GUI tests are presented in his further works [14,15,16,17,18,26,28] or in the Kasik’s paper [9], where genetic algorithms were used in order to automatically generate user events in GUI-based test cases. Unfortunately none of those works consider test management. A tool (called Dart) where test generation and test execution may be performed completely automatically were suggested by Godefroid et al. [6]. The Dart tool extracts the interface of a program using static source code parsing performs random testing to simulate the environment the program may operate in and analyzes the program behaviour. Random testing was used in Claessen’s and Hughes’ tool called QuicCheck [2] too. It is an interesting tool, but it may be used in very few software projects because it supports only programs written in the Haskell language. Bousquet et al. suggested another test automation tool, called Lutess [1]. The Lutess tool operates in the context of synchronous reactive software. Ostrand et al. implemented an experimental test development environment intended to test through GUI [21]. It uses commercial capture / replay tools as a test generation engine and links it with a test designer and a test design library in order to raise the productivity of the GUI testers. There are some strongly specialized test automation tools too. An example of such a solution may be found in [10]. The V-Sat tool that is described in [10] performs design space exploration of system-on-chip architecture.

Most of the mentioned above tools support neither test management nor test documentation. The several exceptions (the commercial solutions, Fit/FitNesse and Proven!) that consider test management or test documentation do not support the IEEE 829 [26]. In a typical case there is a need to use second test tool that is dedicated to test management. The most popular test management tools are TestLink\(^6\), Testopia\(^7\) and QaTraq\(^8\). Only the QaTraq tool supports the IEEE 829 [26]. There are some approaches that did not become popular but are quite close to our. Sneed [24] performed an experiment where a prototype test automation tool that supports the IEEE 829 [26] was used to test AS/400 COBOL programs. Experience with this tool in ongoing projects showed that it is a promising direction, but the potential usage is limited to AS/400 COBOL programs. Similar findings were presented by Desai [4]. He described a Software Development and Testing Environment with a Test Case Management System (TCMS). TCMS allows managing of test cases and test results and executing manual as well as automated tests and generating customized reports. Test case formats and test results format are based on the IEEE 829 [26], but the standard is not fully supported. The method of automation is not perfect too. Automation of a test case is complicated and requires very much time. Siepmann and Newton suggested a test management system for object-oriented software [23]. It is an interesting approach, but applicable only for Smaltalk systems. Luiz da Cruz et al. implemented a tool to support test

\(^6\) http://blog.testlink.org
\(^7\) http://www.mozilla.org/projects/testopia
\(^8\) http://www.testmanagement.com
documents generation and traceability among test artefacts [11]. It is a prototype tool, but it already (only partly) supports the IEEE 829 [26].

Implementation of test automation tools has recently become so popular that a book about the desired tool architecture and the implementation process has been written [12]. The 5th section briefly presents the architecture of our test tool, but as test tool creation guide we definitely recommend the Li’s and Wu’s book [12].

4. TEST DOCUMENTATION STANDARDS

Two standards for managing test documentation are discussed in this section. Those standards define both the test process and the required documents.

*IEEE 829-1998 standard for Software Test Documentation* is a worldwide known standard defining set of documents that could be used as it is or as base for defining documents that will suite particular organization.

*Regulation of Ministry of Science and Computerization for acceptance tests and examination of interface software and verification of this examination* is a document published by Polish government that describes process and documentation for acceptance tests that needs to be done when any public institution is ordering any software. Those tests are part of whole ordering process and needs to be performed exactly as they are described by this document.

4.1. IEEE 829-1998 STANDARD FOR SOFTWARE TEST DOCUMENTATION

![Fig. 1. Relationship of test documents to testing process after IEEE 829 [26].](image)
The purpose of this standard is to describe a set of basic software test documents. A standardized test document can facilitate communication by providing a common frame of reference (e.g., a customer and a supplier have the same definition for a test plan). The content definition of a standardized test document can serve as a completeness checklist for the associated testing process. A standardized set can also provide a baseline for the evaluation of current test documentation practices. In many organizations, the use of these documents significantly increases the manageability of testing. Increased manageability results from the greatly increased visibility of each phase of the testing process.

This standard specifies the form and content of individual test documents. It does not specify the required set of test documents. It is assumed that the required set of test documents will be specified when the standard is applied.

The documents outlined in this standard cover test planning, test specification, and test reporting.

The test plan prescribes the scope, approach, resources, and schedule of the testing activities. It identifies the items to be tested, the features to be tested, the testing tasks to be performed, the personnel responsible for each task, and the risks associated with the plan.

Test specification is covered by three types of documents:

- A test design specification refines the test approach and identifies the features to be covered by the design and its associated tests. It also identifies the test cases and test procedures, if any, required to accomplish the testing and specifies the feature pass/fail criteria.
- A test case specification documents the actual values used for input along with the anticipated outputs. A test case also identifies constraints on the test procedures resulting from use of that specific test case. Test cases are separated from test designs to allow for use in more than one design and to allow for reuse in other situations.
- A test procedure specification identifies all steps required to operate the system and exercise the specified test cases in order to implement the associated test design. Test procedures are separated from test design specifications as they are intended to be followed step by step and should not have extraneous detail.

Test reporting is covered by four document types:

- A test item transmittal report identifies the test items being transmitted for testing in the event that separate development and test groups are involved or in the event that a formal beginning of test execution is desired.
- A test log is used by the test team to record what occurred during test execution.
- A test incident report describes any event that occurs during the test execution which requires further investigation.
- A test summary report summarizes the testing activities associated with one or more test design specifications.
4.2. REGULATION OF MINISTRY OF SCIENCE AND COMPUTERIZATION FOR ACCEPTANCE TESTS AND EXAMINATION OF INTERFACE SOFTWARE AND VERIFICATION OF THIS EXAMINATION

Polish regulation [19] is a document that describes process and documentation for acceptance tests. All public organizations are obligated to do tests according to this regulation when they are ordering any software.

Process described by this paper is very similar to one described by IEEE 829 standard. The only different is that in this regulation we can find some additional steps. In those steps verification of the tests and the test results is performed.

Documents that must be used are described and they are equivalent to documents described in the IEEE 829 standard. There are also three more documents which are related to additional steps in process and the fact that this regulation defines a highly formalized process and this require official documents to be used.

Unlike the IEEE standard, this regulation needs to be used as it is. There are no changes allowed in process or documents.
4.3. COMMON PARTS AND DIFFERENCES BETWEEN IEEE 829 AND THE REGULATION OF MINISTRY OF SCIENCE AND COMPUTERIZATION

Because those standards were created for different purposes, they are defining different sets of documents, but still we can find some common parts. In both norms we have documents that describe test case, test design, test log and test summary report. Those documents have the same functions, but are defined in different way. In test design and test case we can find some equivalent fields (for test design: Identifier, Test items, Input specification and Output specification; for test case: Identifier, Features to be tested, Tests identifications). There are no similar fields in test log and test summary report.

In IEEE 829 norm process is ended after creating test summary report which contains data from all test logs gather after performing all tests. In Regulation there is one additional step. This is verification of examination. In these step all test documentation is verified by independent organization.

5. AUTESTER FUNCTIONALITIES

AuTester is designed to create and manage test documentation and to manage automated tests created with AutoIt\textsuperscript{9} language with usage of functions from SecureFunctions library (this library is part of AuTester).

User is able to choose which standard he wishes to use for documentation: IEEE 829 or the Polish regulation [19]. Because regulation does not allow making any changes, user is not able to edit any settings in forms.

When user choose IEEE 829 standard, he is allowed to choose which fields to use on forms. Still after making all possible changes, all documentation is covering IEEE 829 norm.

Another site of system is tests automation. As engine for scripts we decided to use AutoIt scripting language. It is language designed to automate tasks in Windows environment. It is developed almost like open-source project, so there is big community behind this project. Main advantage of this situation is easy access to help and support and very quick development of new libraries. We believe that AutoIt is one of the best freeware systems for automation available at this moment. Still it is suitable for use as testing tool. This is why SecureFunctions library was created. This library contains modified versions of commonly used functions. These modifications are responsible for changing simple automation scripts into test tools. Every function is able to first make some simple tests, gather results of this tests, log every important information (create screenshot if needed) and decide whether it is possible to continue test. All this changes

\textsuperscript{9} http://www.autoitscript.com/
are made in a way that makes use of these functions almost identical to use of native AutoIt functions. Also format of logs is compatible with AuTester so our system is able to look inside it and get all important information. These simple functions allow system to monitor results of executed tests and generate test logs.

![Fig. 3. AuTester – main window.](image)

According to Dustin [5] and Myers [20], the best usage for automation is during regression tests. This is where AuTester will give most advantages. Mainly it is possible to cover all functional tests that consider usage of GUI with AuTester: regression tests, functional tests or acceptance tests.

6. AUTESTER SYSTEM ARCHITECTURE

AuTester has modular architecture. We can basically distinguish 4 modules:
- documentation module
- automation engine
- script executor
- test functions library
Documentation module is the main AuTester application. It allows user to manage all tests documentation. As automation engine we decided to use AutoIt scripting language. This requires installing the AutoIt software before using AuTester. Script executor is simple application written in the AutoIt language called TestRunner. It is responsible for executing scripts added to particular test design. Test functions library is an AutoIt library that contains functions used for testing.

Because the modules are independent elements and can operate at different times, communication between them is based on files. The first file needed for communication between AuTester and TestRunner is a file containing details of the test design. This file is created by the AuTester program when user adds test cases to the test design. It contains information about test cases that are part of the design. The program TestRunner is able to locate that file on the basis of data that are transmitted to it, as arguments when running from the command line. Based on this information, program generates a list of scripts to be run.

The program TestRunner has its own file with the logs, which are recording information about every execution of this program. Program is also logging information about every single test to the test log. Test log is a separate log for each run of the script. It contains information about any incidents that happened during test execution. Some of this information came from the TestRunner, and some from the test script. Responsibility for reporting any incidents to the log, rest on special functions from the SecureFunctions library. They generate entries, on the basis which, you can find out what actions were taken, after the unexpected event, and in which line of the script incident appear.
TestRunner can be run from command prompt. Thanks to this it is possible to run it automatically.

Test log file is also serving to communicate within the system. This file is used by the program AuTester to generate a report of the test performed. This document is automatically added to the appropriate project as the test log document.

7. CONCLUSIONS AND FUTURE WORK

A tool called AuTester has been presented. AuTester is a valuable tool that allows easy managing of test cases and test results according to the IEEE 829 standard [26] as well as the Polish regulations [19], executing manual and automated test cases and generating customized test reports. Support for the Polish is especially valuable, because according to authors' knowledge AuTester is the only available tool with such functionality.

There is considerable number of test tools reported in the Related work section. According to the authors' knowledge none of them has become popular or is currently used by industry. Why AuTester may succeed? AuTester is going to be developed as an open source project. Therefore, it will be easy to obtain, free of charge, and eventually an AuTester community may grow. There is also a software testing company Test Solution that is already interested in using AuTester in their software projects.

AuTester is an original product of the authors. At this moment it is not yet ready for commercial use. Current version is good enough for making tests of all concepts that was applied into this system. At this point we can tell that this tool has potential to become a powerful application. The implemented tool constitutes a solid foundation for the future work. A web interface for AuTester is going to be developed in the near future. The web interface will be used only for working with documentation. Improvement in the collaboration model in order to increase the comfort of working in bigger groups will also be considered in the future. This includes modifications in data access and applying user accounts system.

REFERENCES


\[10\] http://sourceforge.net/projects/autester
\[11\] http://www.testsolutions.pl


