Preface

Software testing is required to assess the quality of the developed software. However, it consumes a critical amount of time and resources, often delaying the software release date and increasing the overall cost. The answer to this problem is effective test automation, which is expected to meet the need for effective software testing, while reducing amount of required time and resources. There exists a need for an edited publication illuminating the current state of test automation practices.

In this book, several authors produced chapters related to software test automation and its validity and applicability in different domains. Authors showed how test automation could be used in those different domains and in the different tasks and stages of software testing.

In the first chapter, Izzat Alsmadi introduced a general test automation framework that includes the automation of activities in all software testing stages (e.g. test case design, test case generation, execution, verification, prioritization, etc.). He focused on testing applications’ user interfaces. In the chapter, he used model based testing to convert the application user interface into an XML file or model where test automation activities can be easily implemented. Test cases were then generated automatically from the XML file. They were then executed on the actual application. Results were verified based on defining original accepted results with expected ones. While 100% test automation may not be always feasible, nonetheless, the more tasks that can be automated, the more saving is achieved on software project time and resources.

In Chapter 2, Daniel Bolanos focused on software test automation techniques in the field of speech recognition. The chapter demonstrated the process followed during the development of a new generation speech recognition system that is expected to be released as an open source project. The author discussed testing approaches and methodologies followed to test the program through all its components and stages. The aim of the chapter is to provide practitioners in the field with a set of guidelines to help them through the process of elaborating an adequate automated testing framework to competently test Automatic Speech Recognition (ASR) systems. The chapter first described using the unit testing library CxxTest in speech recognition
testing. Mock objects are used to simulate or substitute using components that are not ready yet. One of the major units to test in speech recognition system is the decoding network which is a graph that contains all the lexical units that are in the recognition vocabulary. Most of the decoding time is spent traversing the decoding network, so testing it thoroughly is crucial. The chapter then described testing the token expansion process. On the system testing level, the author described black box testing approach for a speech recognition system. The system is tested first through its ability to accept valid inputs and reject invalid inputs. Black box testing also includes testing the configuration file setting and command line parameters related to initialization, environment, et cetera. System testing also includes testing the correctness and validity of outputs or results. Application Programming Interfaces (APIs) are also important to test in systems or programs that have relations with hardware, operation system, database, or external systems such as the speech recognition system.

In Chapter 3, Eslam Al Maghayreh described automatic runtime testing or verification techniques on distributed systems. Runtime verification is a technique that combines formal verification and traditional testing. Testing and verification of a program statically does not guarantee that no new errors will appear at run time. The static time is the time before the compilation of the program, and the runtime is the time where the program is dynamically used after it has been successfully compiled. Runtime verification is more practical than other verification methods such as model checking and testing. Runtime verification combines the advantages of formal methods and traditional testing as it verifies the implementation of the system directly rather than verifying a model of the system as done in model checking. After the description of run time verification and its characteristics in comparison with static checking or testing, the author presents a model for a distributed program. Formal methods approaches and techniques relevant to the chapter are described. The author explores some of the logics used to formally specify the properties that a programmer may need to check and presents the main approaches used to verify distributed programs. Later on, author briefly describes the difficulties of checking global properties of distributed programs. Possible solutions for the state explosion problem are then described in later sections. Examples of strategies described to solve such problem include: exploiting the structure of the property, atomicity and program slicing. A brief description for a tool that can be used for runtime verification of distributed programs concludes the chapter.

In Chapter 4, Seifedine Kadry focused his chapter on regression testing and maintenance aspects of software testing. Conventionally, the maintenance of software is concerned with modifications related to software system. These modifications come from the user needs, error correction, improvement of performance, adapt to a changed environment, and optimization. In the chapter, author discussed maintenance aspects,
needs, and costs. In the scope of software maintenance, author described levels of testing unit and system, along with integration testing approaches. As an important testing to maintenance activity, author described debugging which is an activity that comes after detecting errors in order to find their causes and fix them. The author then talks about test automation and started by discussing some of the important questions to answer regarding to test automation. For example, test cases number of repetition, reuse, relevancy, and effort are important questions to answer before deciding whether a test activity should be automated or not. Using a case study, he accordingly proposed drawing a decision tree for making decision on automation based on the answers of those questions. All the proposed questions have a discrete number of answers: ‘High’, ‘Medium’ or ‘Low’, which are represented in the tree by the letters “H,” “M,” and “L.” The author then discussed regression testing as a major activity in maintenance testing. He discussed several possible regression testing strategies: retest-all, regression test selection, prioritization, hybrid, and risk analysis. He then proposed a technique to combine regression testing based on risk analysis and automation techniques to obtain more improvement on the test cost-effectiveness by optimizing the number of cases that should be automated and the number of cases that should we calculate their risk exposures.

In Chapter 5, Natarajan Meghanathan and Alexander Geoghegan focused their chapter on testing software security with a case study on a file reader program written in Java. They followed the static code analysis in order to test possible vulnerabilities associated with the program. They used a source code analyzer and audit workbench, developed by Fortify, Inc. They first introduced static code analysis and its usefulness in testing security aspects of the Software. They then introduced the tool used in the study and how it can be used for source code analysis. Later on in the chapter, they presented a detailed case study of static code analysis conducted on a file reader program (developed in Java) using the described automated tools. They focused on certain security aspects to evaluate. Those include: Denial of Service, System Information Leak, Unreleased Resource, and Path Manipulation. They discussed the potential risk in having each one of these vulnerabilities in a software program and tried to provide the solutions (and the Java code) to mitigate these vulnerabilities. The proposed solutions for each of these four vulnerabilities are more generic and could be used to correct such vulnerabilities in software developed in any other programming language.

In Chapter 6, Praveen Srivastava, D.V. Reddy, Srikanth Reddy, Ch. Ramaraju, and I. Nath discussed test case prioritization using Cuckoo search algorithm. Test case prioritization includes using techniques to select a subset of the test cases (that are usually large) wherein this subset can be an effective representative of the overall test cases in terms of coverage. This is done due to the scarcity of resources available for testing and the small amount of time usually available for such activity. In
This chapter introduces a new test case prioritization technique for version-specific regression testing using Cuckoo search algorithm. This technique prioritizes the test cases based on lines of code where the code is modified. The authors introduced first test case prioritization and related work in the area. They then introduced Cuckoo search algorithm for automating the selection and prioritization of test cases. To determine the effectiveness of this approach, the implementation of the algorithm is done on the sample program in Java.

The proposed algorithm was tested on real-time software (some in-house development and few of open sources). They showed effective test case selection using the proposed algorithm. They compared (on the basis of redundant test case, complexity of a program, dependent test cases, etc.) the proposed algorithm with traditional approaches on test case prioritization. Since Cuckoo search is an optimized algorithm, test case prioritization algorithm based on cuckoo search have better results over procedural methods.

In Chapter 7, Saqib Saeed, Farrukh Khawaja, and Zaigham Mahmoud wrote a chapter on software quality methodologies. They present and review different methodologies employed to improve the software quality during the software development lifecycle. In the start, they discussed with a background the meaning and attributes of software quality. In the next section, they discussed software quality elements through the development lifecycle and all stages: project initiation, project planning, requirement, design, coding, testing, configuration management, evolution, or maintenance. They discussed those different stages with focus on quality elements based on the activities that occur in each stage along with goals and objectives. In a separate section, they then discussed several quality methodologies. In the first part on quality, they discussed software quality standards. Those are important to set goals and objectives in which progress and evaluation can be measured based on international standards such as ISO9000, a set of standards for both software products and processes. Examples of those standards mentioned besides ISO include: IEEE software engineering standards, CMM, TL 9000, etc. On a related subject, authors discussed software metrics as tool to measure and quantize software attributes. Without software metrics, it is not possible to measure the level of quality of software. Review and inspection activities are also described as human methods to evaluate the progress and the observation and following of standards. In the next section different methods of software testing are described. Software audits are also described in this chapter as an important tool to identify the discrepancy much earlier in a software development lifecycle approach.

In the eighth chapter, Thomas Bauer and Robert Eschbach talk about model-based testing of distributed systems. They present a novel automated model-based testing approach for distributed functions that uses informal system requirements and component behavior models. The test modeling notation makes it possible to
model component interactions and composite functions with defined pre- and post-
conditions. Test cases are automatically generated as scenarios of distributed func-
tions represented by sequences of component interactions. A generic approach for
efficient quality assurance is model-based testing, which deals with the automated
generation or selection of test cases from models. A large variety of models have
been used and adapted for automated test case generation. Integration testing is fre-
quently supported by different kinds of models. Distributed systems are systems that
consist of multiple autonomous computers or processes that communicate through
a computer network or a communication middleware. Components are system parts
with defined interfaces and functionalities. Authors discussed Component Based
Testing (CBT) as one software development methodology used to build a software
from ready components. In their case study, authors discussed an automotive example
and other CBT systems related to automotive control systems. They showed the
different distributed functions of the case study and their relationships in an acyclic
hierarchical function graph. In later steps, the graph is the basis for the derivation of
the functional model. The function graph is retrieved from the system requirements
specification by mapping chapters, sections, paragraphs, and cross-references to graph
nodes and transitions between them. The graph is acyclic to avoid design flaws.
Functions may be referenced by multiple functions. Authors also have an extensive
related work for model based testing specifically in the automotive systems. They
described a model based on pre and post conditions. The second modeling notation
used for describing the composition of functions is an operational notation. In the
approach, the modeling notation CSP is used. This is a formal notation for describing,
analyzing, and verifying systems consisting of concurrent, communicating processes.
The combination of CSP and B uses the B machines to specify abstract system states
and operations and CSP models to coordinate the execution of operations. The third
modeling notation used for describing the composition of functions is a transition-
based notation. Transition-based notations describe the control logic of components
with their major states and actions which lead to state transitions. Functional and
component models are analyzed statically to retrieve model faults and inconsisten-
cies. The model shall not contain any deadlocks and all relevant system states shall
be reachable. Furthermore, it has to be assured that each function referred to in the
high-level functional models is represented in the low-level component models,
i.e., with start states, end states, and defined paths. Based on the functional model,
the integration strategy is defined. The integration strategy determines the system
assembly order. Test cases are automatically generated from the reduced models for
each integration step (4). Different coverage criteria are proposed for the test case
generation. The generation starts with the reduced functional models. Integration
approaches are also discussed as important elements of model based testing. The
integration order determines the order of system assembly for integration testing.
For the testing approach, a function-driven bottom-up integration strategy is used. The objective is to integrate and test the functions according to their order in the function graph. For the test case generation, an approach based on stepwise coverage and considering all three model types is proposed. Test cases are generated for each configuration step from its reduced test models. Test cases are automatically generated from the test models. A set of coverage criteria has been proposed for the different modeling notations. Future work will comprise the improvement of the tool chain and further application of the method in industrial case studies.

In the ninth chapter, Kamaljeet Sandhu discussed testing e-learning websites. The chapter aims to explore the effectiveness of e-learning websites in relation to user’s perceptions that are developed when interacting with e-learning system and the features that they have to use. The chapter first described a literature review and popular tools in e-learning. The aim of the study is to investigate the adoption of Web-based learning on websites amongst users. This led to the development of converging lines of inquiry, a process of triangulation. The case study examines the testing of Web-based framework of the University of Australia (not the real name). International students have the option to lodge an admission application through either of: Web-based e-service on the Internet, phone, fax, or in person. The main aim of this approach is to test users’ information experience on websites. The user perception of Web electronic service is a burden and acted as a barrier to their work. The users learnt that it increased the workload, slowed the work process, and brought in complexity to the task. The department did not implement electronic services or introduce technology into jobs at the same time. The chapter includes also details on users’ in the case study and their perception and evaluation of the e-learning tool WebCT.

In Chapter 10, Kamaljeet Sandhu talked about testing the effectiveness of Web based e-services. There are technical issues that impact negatively on users of the e-services. Case analysis reveals that there are wider issues stemming from the interaction with the system which has a low impact on e-services acceptance. E-Services drivers such as user training and experience, motivation, perceived usefulness and ease of use, acceptance, and usage were not clearly understood by the technical development team, hence not integrated when implementing the e-services system. The chapter focused on evaluating an educational website from the perspective on providing and evaluating services. The case study examines the Web Electronic Service framework of the University of Australia (not the real name). The department is in the process of developing and implementing Web-based e-service system. The focus was on evaluating the website user interface and users’ perception and acceptance of the user interface and their ability to understand and use those services. The study and the chapter include results of a survey or question-
naire of users and staff of the evaluated website. User’s acceptance to use e-services relates to the initial use when users trial the use of e-Services system and evaluate its effectiveness in meeting their objectives for visiting the e-Services system. For example, a student may use e-services system on a university website for emailing but its effective use can only be determined if such activities meet with the student’s objectives. If the objectives are not met, the student is unlikely to accept e-services. Another area that the author focused on is to evaluate the gap between paper-based and web-based e-service system. Those are found due to information gap, design gap, communication, and fulfillment gap. The chapter also discussed the risk of asking the same people to compare and evaluate between the earlier and the new systems and what evaluators are expecting.

Izzat Alsmadi
Yarmouk University, Jordan