PREMIER REFERENCE SOURCE

# Advanced Automated Software Testing

## Frameworks for Refined Practice

Izzat Alsmadi

# Advanced Automated Software Testing:

## Frameworks for Refined Practice

Izzat Alsmadi
*Yarmouk University, Jordan*

Information Science
REFERENCE

| | |
|---|---|
| Managing Director: | Lindsay Johnston |
| Senior Editorial Director: | Heather Probst |
| Book Production Manager: | Sean Woznicki |
| Development Manager: | Joel Gamon |
| Acquisitions Editor: | Erika Gallagher |
| Typesetters: | Adrienne Freeland |
| Print Coordinator: | Jamie Snavely |
| Cover Design: | Nick Newcomer, Greg Snader |

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

Library of Congress Cataloging-in-Publication Data

Advanced automated software testing: frameworks for refined practice / Izzat Alsmadi, editor.
p. cm.
Includes bibliographical references and index.
Summary: "This book discusses the current state of test automation practices, as it includes chapters related to software test automation and its validity and applicability in different domains"--Provided by publisher.
ISBN 978-1-4666-0089-8 (hardcover) -- ISBN 978-1-4666-0090-4 (ebook) -- ISBN 978-1-4666-0091-1 (print & perpetual access) 1. Computer software--Testing--Automation. I. Alsmadi, Izzat, 1972-
QA76.76.T48A38 2012
005.3028'7--dc23

2011042036

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

All work contributed to this book is new, previously-unpublished material. The views expressed in this book are those of the authors, but not necessarily of the publisher.

# List of Reviewers

Eslam Maghayreh, *Yarmouk University, Jordan*
Daniel Bolanos, *University of Colorado, USA*
Seifedine Kadry, *American University of the Middle East, Kuwait*
Natarajan Meghanathan, *Jackson State University, USA*
Saqib Saeed, *University of Siegen, Germany*
Kamaljeet Sandhu, *University of New England, Australia*
Thomas Bauer, *Fraunhofer IESE, Germany*
Robert Eschbach, *Fraunhofer IESE, Germany*
Praveen Ranjan Srivastava, *Birla Institute of Technology and Science Pilani, India*

# Table of Contents

# Detailed Table of Contents

It is widely acknowledged that software testing stage is a stage in the software project that is time and resources' consuming. In addition, this stage comes late in the project, usually at the time where pressure of delivery is high. Those are some reasons why major research projects in testing focus on methods to automate one or more of activities in this stage. In this chapter, description of all sub stages in software testing is explained along with possible methods to automate activities in this sub stage. The focus in this chapter is on the user interface of the software as it is one of the major components that receives a large percentage of testing. A question always raised in testing is whether full test automation is possible and if that can be feasible, possible and applicable. While 100% test automation is theoretic and impractical, given all types of activities that may occur in testing, it is hoped that a maximum coverage in test automation will be visible soon.

This chapter provides 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 systems. Thorough this chapter the testing process of such a system is analyzed from different angles, and different methods and techniques are proposed that are well suited for this task.

Ensuring the correctness of a distributed program is not an easy task. Testing and formal methods can play a significant role in this regard. However, testing does not allow for formal specification of the properties to be checked. On the other hand, formal methods verify that a model of a distributed system satisfies certain properties that are formally specified. However, formal methods do not guarantee that a particular implementation of the system under consideration will also satisfy all of the necessary properties. Runtime verification tries to combine some of the advantages of testing and some of the advantages of formal methods. Runtime verification works on a particular implementation of the system and at the same time it allows us to specify the properties of interest formally. In this chapter we have talked about runtime verification of distributed programs. The main components of a runtime verification framework have been presented and discussed in some details. The reasons that make runtime verification of distributed programs a difficult task have been discussed. A summarization of some of the techniques presented in the literature to simplify runtime verification of distributed programs has also been presented.

System maintenance is a general term required to keep a system running properly. The system could be a computer system, mechanical system, or other system. The maintenance in this sense is related to the deterioration of the system due to its usage and age. This context of maintenance does not apply to software, where the deterioration due to the usage and age don't make sense. 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.

The high-level contribution of this book chapter is to illustrate how to conduct static code analysis of a software program and mitigate the vulnerabilities associated with the program. The automated tools used to test for software security are the Source Code Analyzer and Audit Workbench, developed by Fortify, Inc. The first two sections of the chapter are comprised of (i) An introduction to Static Code Analysis and its usefulness in testing for Software Security and (ii) An introduction to the Source Code Analyzer and the Audit Workbench tools and how to use them to conduct static code analysis. The authors then present a detailed case study of static code analysis conducted on a File Reader program (developed in Java) using these automated tools. The specific software vulnerabilities that are discovered, analyzed, and mitigated include: (i) Denial of Service, (ii) System Information Leak, (iii) Unreleased Resource (in the context of Streams), and (iv) Path Manipulation. The authors discuss the potential risk in having each of these vulnerabilities in a software program and 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.

Test Case prioritization consists of proper organization and scheduling of the test cases in a specific sequence. Regression testing is an important issue and concept during software maintenance process, but due to scarcity of resources re-execution of all test cases, is not possible during regression testing. Hence in version or revision specific regression testing, it is more important to execute those test cases that are beneficial. In this chapter, a new prioritization technique is proposed 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.

Pervasive systems and increased reliance on embedded systems requires that the underlying software is properly tested and has in-built high quality. The approaches often adopted to realize software systems have inherent weaknesses that have re-