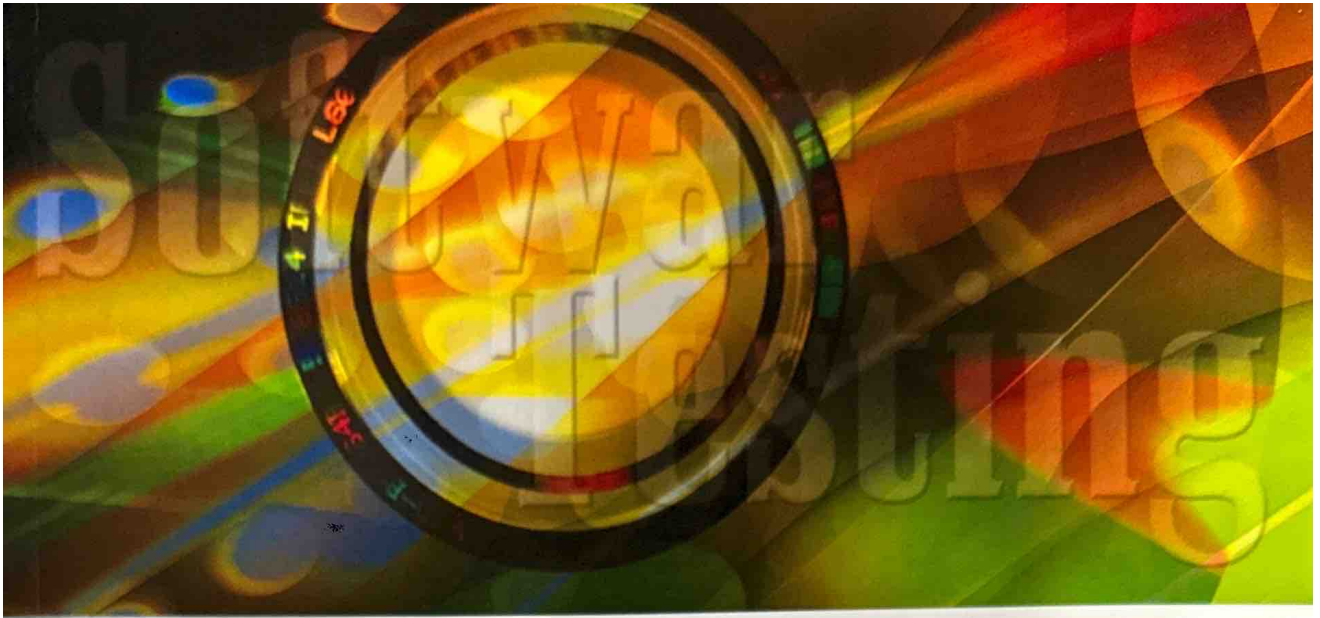




Software Testing

Principles and Practices

Srinivasan Desikan

Gopalaswamy Ramesh



Pearson

Software Testing

Principles and Practices

Srinivasan Desikan
Gopalaswamy Ramesh



Editor in Chief: **Prof H N Mahabala**

Titles under the series include

- Software Maintenance
Gopalaswamy Ramesh and Ramesh Bhattiprolu
- Software Testing – Principles & Practices
Gopalaswamy Ramesh and Srinivasan Desikan
- Managing Global Software Projects
Gopalaswamy Ramesh
- Mobile Computing – Technology Applications & Service Creation
Asoke K Talukder and Roopa Yavagal

Acquisitions Editor: Sojan Jose

Senior Production Editor: M. E. Sethurajan

Composition: Sigma Business Process, Chennai

Printer:

Copyright © 2011 Dorling Kindersley (India) Pvt. Ltd

This book is sold subject to the condition that it shall not, by way of trade or otherwise, be lent, resold, hired out, or otherwise circulated without the publisher's prior written consent in any form of binding or cover other than that in which it is published and without a similar condition including this condition being imposed on the subsequent purchaser and without limiting the rights under copyright reserved above, no part of this publication may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording or otherwise), without the prior written permission of both the copyright owner and the above-mentioned publisher of this book.

Although the author and publisher have made every effort to ensure that the information in this book was correct at the time of editing and printing, the author and publisher do not assume and hereby disclaim any liability to any party for any loss or damage arising out of the use of this book caused by errors or omissions, whether such errors or omissions result from negligence, accident or any other cause. Further, names, pictures, images, characters, businesses, places, events and incidents are either the products of the author's imagination or used in a fictitious manner. Any resemblance to actual persons, living or dead or actual events is purely coincidental and do not intend to hurt sentiments of any individual, community, sect or religion.

In case of binding mistake, misprints or missing pages etc., the publisher's entire liability and your exclusive remedy is replacement of this book within reasonable time of purchase by similar edition/reprint of the book.

ISBN 978-81-775-8121-8

First Impression, 2006

This edition is manufactured in India and is authorized for sale only in India, Bangladesh, Bhutan, Pakistan, Nepal, Sri Lanka and the Maldives. Circulation of this edition outside of these territories is UNAUTHORIZED.

Published by Pearson India Education Services Pvt. Ltd, CIN: U72200TN2005PTC057128.

Head Office: 15th Floor, Tower-B, World Trade Tower, Plot No. 1, Block-C, Sector 16, Noida 201 301, Uttar Pradesh, India.

Registered Office: 7th Floor, SDB2, ODC 7, 8 & 9, Survey No.01 ELCOT IT/ ITES SEZ, Sholinganallur, Chennai – 600119, Tamil Nadu, India.

Website: in.pearson.com, Email: companysecretary.india@pearson.com

Digitally printed in India by Saurabh Printers Pvt. Ltd. in the year of 2022.

One of the methods of achieving interfaces is by providing Application Programming Interfaces (APIs). APIs enable one module to call another module. The calling module can be internal or external. For example, JDBC is an example of an API used by a Java program to make certain SQL calls. Even though both the API and interfaces appear to be similar, it is important to realize that integration is the *purpose* to be achieved while API is a *means* of achieving the purpose. API is just one of the means of providing an interface between two modules. One can think of other means of integration among the various modules: Some of these could be simple function calls, public functions, and some could be facets of programming language constructs like global variables and some could be operating system constructs like semaphores and shared memory. In this chapter, we will not discuss the details of the vehicles used for integration (as it is primarily a development issue), but rather look at how we can test the interfaces (which is the focus for testing).

Not all the interfaces may be available at the same time for testing purposes, as different interfaces are usually developed by different development teams, each having their own schedules. In order to test the interfaces, when the full functionality of the component being introduced is not available, *stubs* are provided. A stub simulates the interface by providing the appropriate values in the appropriate format as would be provided by the actual component being integrated.

Integration testing is done with test cases, which goes through the internal and exported interfaces, and tests the functionality of the software. Internal interfaces are for other developers inside an organization and external interfaces are for third party developers or other users outside the group. Testing for internal interfaces requires a complete understanding of architecture and high-level design (HLD) and how they impact the software functionality. In cases where exported interfaces are provided from the software, one needs to understand the purpose of those interfaces, why they are provided, and how they are actually used by developers and solution integrators. Hence a knowledge of design, architecture, and usage is a must for integration testing.

Initially, the exported (or external) interfaces were provided through APIs and *Software Development Kits* (SDKs). The use of SDKs required an understanding of the programming language on which the API/SDK is provided. Later, the interfaces became available through scripting languages, without the need for SDKs. (Some of the popular scripting languages include Perl, Tcl/Tk). These scripting languages eliminated or minimized the effort in learning the languages in which the API was written. This also made it possible for the interfaces to be called from programming language environments different from the one in which the interface was originally written. This significantly simplified the usage of exported interfaces. For testing interfaces, we now have dynamically created scripts, which can be changed at run time, by a few clicks of the mouse.

All these have made the use of interfaces a lot more widespread. The number of purposes for which the interfaces are provided have been on the increase. These interfaces are becoming increasingly generic in nature and

requirements and the business flow is tested. In replication, customer data and process is obtained and the product is completely customized, tested, and the customized product as it was tested is released to the customer.

As discussed in the chapter on integration testing, business verticals are tested through scenarios. Scenario testing is only a method to evolve scenarios and ideas, and is not meant to be exhaustive. It's done more from the perspective of interfaces and their interaction. Having some business verticals scenarios created by integration testing ensures quick progress in system testing, which is done with a perspective of end-to-end scenarios. In the system testing phase, the business verticals are completely tested in real-life customer environment using the aspects such as customization, terminology, and syndication described in the above paragraphs.

6.4.3 Deployment Testing

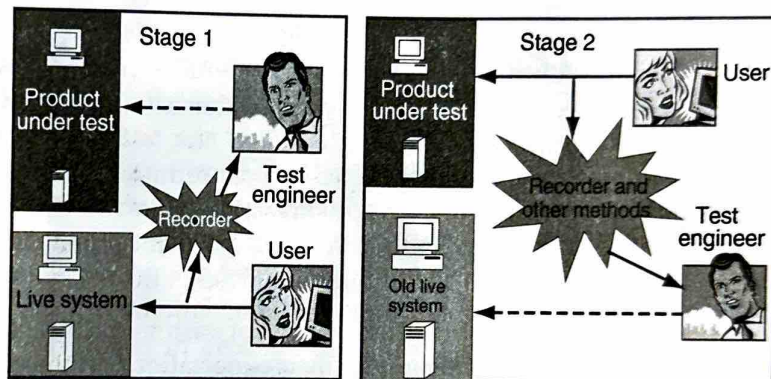
System testing is the final phase before product delivery. By this time the prospective customers and their configuration would be known and in some cases the products would have been committed for sale. Hence, system testing is the right time to test the product for those customers who are waiting for it. The short-term success or failure of a particular product release is mainly assessed on the basis of on how well these customer requirements are met. This type of deployment (simulated) testing that happens in a product development company to ensure that customer deployment requirements are met is called *offsite deployment*.

Deployment testing is also conducted after the release of the product by utilizing the resources and setup available in customers' locations. This is a combined effort by the product development organization and the organization trying to use the product. This is called *onsite deployment*. Even though onsite deployment is not conducted in the system testing phase, it is explained here to set the context. It is normally the system testing team that is involved in completing the onsite deployment test. Onsite deployment testing is considered to be a part of acceptance testing (explained later in this chapter) and is an extension of offsite deployment testing.

Onsite deployment testing is done at two stages. In the first stage (Stage 1), actual data from the live system is taken and similar machines and configurations are mirrored, and the operations from the users are rerun on the mirrored deployment machine. This gives an idea whether the enhanced or similar product can perform the existing functionality without affecting the user. This also reduces the risk of a product not being able to satisfy existing functionality, as deploying the product without adequate testing can cause major business loss to an organization. Some deployments use *intelligent recorders* to record the transactions that happen on a live system and commit these operations on a mirrored system and then compare the results against the live system. The objective of the recorder is to help in keeping the

mirrored and live system identical with respect to business transactions. In the second stage (Stage 2), after a successful first stage, the mirrored system is made a live system that runs the new product. Regular backups are taken and alternative methods are used to record the incremental transactions from the time mirrored system became live. The recorder that was used in the first stage can also be used here. However, a different method to record the incremental transaction is advised, for sometimes failures can happen due to recorder also. This stage helps to avoid any major failures since some of the failures can be noticed only after an extended period of time. In this stage, the live system that was used earlier and the recorded transactions from the time mirrored system became live, are preserved to enable going back to the old system if any major failures are observed at this stage. If no failures are observed in this (second) stage of deployment for an extended period (for example, one month), then the onsite deployment is considered successful and the old live system is replaced by the new system. Stages 1 and 2 of deployment testing are represented in Figure 6.2. The coloured figure is available on page 463.

Figure 6.2
Stages of deployment testing.



In Stage 1 of Figure 6.2, it can be seen that the recorder intercepts the user and live system to record all transactions. All the recorded transactions from the live system are then played back on the product under test under the supervision of the test engineer (shown by dotted lines). In Stage 2, the test engineer records all transactions using a recorder and other methods and plays back on the old live system (shown again by dotted lines).

6.4.4 Beta Testing

Developing a product involves a significant amount of effort and time. Delays in product releases and the product not meeting the customer requirements are common. A product rejected by the customer after delivery means a huge loss to the organization. There are many reasons for a product not meeting the customer requirements. They are as follows.

1. There are implicit and explicit requirements for the product. A product not meeting the implicit requirements (for example, ease of use) may mean rejection by the customer.