Testing and Rolling Out Enterprise Applications

5

Learning Goals

After completing the chapter, you will be able to:

- Describe different types and methods of testing enterprise applications.
- Explain testing levels and testing approach.
- Classify enterprise applications environments.
- Explain integration testing.
- Explain different types of system testing.
- Explain user acceptance testing.
- List the tools used for different types of testing.

You have so far learnt that thorough requirements engineering, flexible and robust architecture and design, and good coding practices are essential to raise successful enterprise applications. Along with these crucial elements, comprehensive testing of enterprise applications is equally important. Testing of enterprise applications is performed not only to ensure that the application delivered has the right set of functionalities but also to guarantee the required quality attributes or "ilities". Testing constitutes approximately 30 percent of the overall efforts spent in raising enterprise applications and hence designated as one of the most important life cycle phases of raising enterprise applications.

This chapter will introduce the concept of testing enterprise applications and the testing approach. Different types of testing—integration testing, performance testing, penetration testing, interface testing, user acceptance testing—are discussed. Various enterprise application environments and tools used for testing are discussed. Towards the end of the chapter, you will learn about the procedures for rollout of enterprise applications.

5.1 Testing Enterprise Applications

Companies across industries depend on mission-critical enterprise applications to drive the business and business initiatives. The goal invariably is to strive for high speed of development, yet creating reliable high-quality applications, and most importantly, staying within budgets. Companies would not want to compromise on the quality of these applications, considering the potential fallout of losing customers and business.

The testing of enterprise applications encompasses the answers to the questions listed in Table 5-1, which includes identification, planning and execution of testing.

Table 5-1 Testing enterprise applications

What	What needs to be tested? What kind of testing to be done? On what conditions do we enter into testing? On what conditions do we exit the testing?
How	How does the software need to be tested? How much testing is required before rollout? How long does each type of testing need to be done?
Where	Where does the software need to be installed and tested?
When	When should we plan to start the testing? When should we end the testing?
Who	Who are involved in the testing?

Let us address the above questions one by one to have a complete understanding of testing enterprise applications.

5.1.1 Types and Methods of Testing

Each enterprise application undergoes various types of testing to ensure that it demonstrates the desired functionalities and quality attributes. Different types of testing are performed in different phases of raising enterprise applications. Testing can be categorized using various perspectives. Two of the most important perspectives to categorize the testing are "What" needs to be tested and "How" to test.

Based on "What" needs to be tested, testing can be divided into two categories:

(a) **Functional testing.** Testing the functionalities as per the stated requirements is termed as *functional testing*. This testing typically involves testing the screen flows, data flows, business logic, user and system validations and codified business processes.

(b) **Nonfunctional testing.** *Nonfunctional testing* caters to testing all other things except functionalities to ensure that the quality attributes are as per the requirements. Typically, these include security, performance, maintainability, reliability, availability, scalability, etc.

Based on "How" to test, the method of testing can be divided into two categories:

(a) **White-box testing.** This method of testing tests the internal structure of the application in detail. This method typically checks for the flow of data, control and branching sequences. Knowledge of algorithms and data structures used in the application is required in *white-box testing*.

(b) **Black-box testing.** This method of testing, on the other hand, tests the software without need for understanding the internal structure and focuses on testing the application from an external perspective. This testing method treats the unit/component/application to be tested as black box and assumes no knowledge of algorithms and data structures used in the application.

In *Gray-box* testing is a hybrid form of white-box and black-box testing methods. In this method of testing, the test planning is done keeping the internals in mind, but finally testing happens like in black-box testing.

5.1.2 Testing Levels

Testing has to be done at several levels before rolling out enterprise applications. This answers the questions: "How" much to test and "When" to test enterprise applications. Figure 5-1 depicts the different levels of testing that enterprise applications undergo starting from unit testing, integration testing, system testing, acceptance testing and finally the production testing.

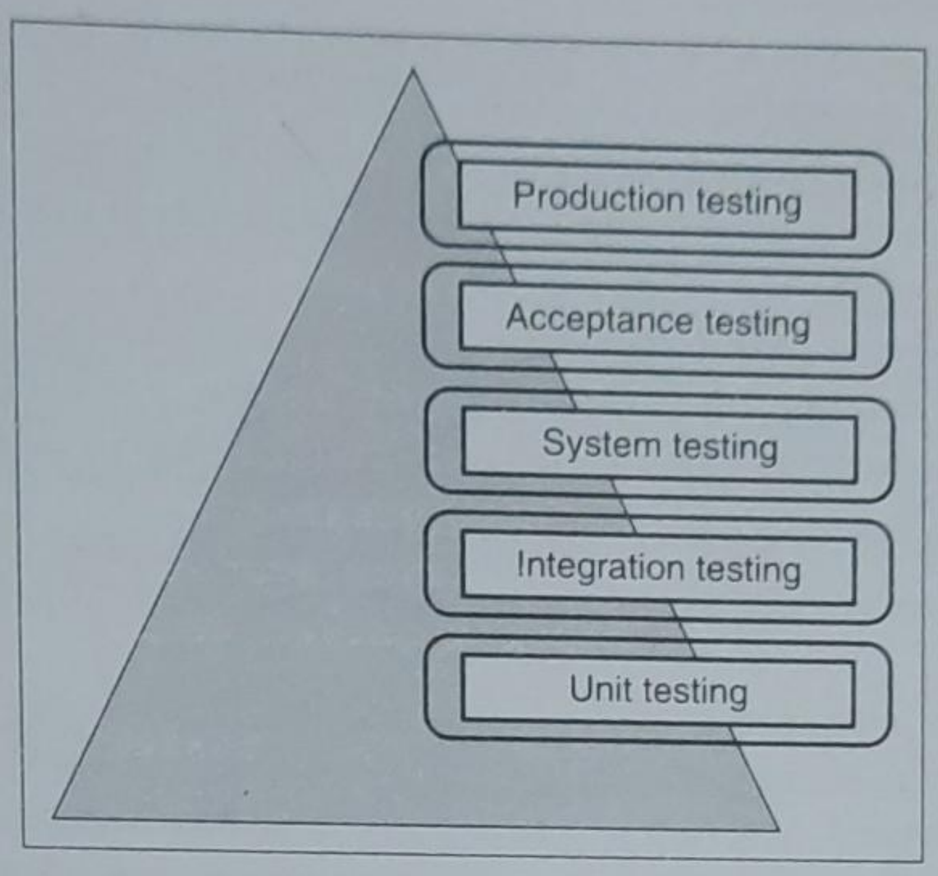


Figure 5-1 Testing levels of enterprise applications.

The following are different types of testing levels:

(a) **Unit testing.** *Unit testing* is performed to ensure that a standalone function/code/module/class works as per the specifications. This testing is the first level of testing that happens in the construction phase of enterprise applications development. You can refer to Chapter 4 on the construction for details on unit testing.

(b) **Integration testing.** *Integration testing* is to ensure that the software code/modules/functions/classes are grouped together consistently. The units of these groups are tested including the interaction among these units to find out any inconsistencies among these units. In essence, integration testing is performed to test that a combination of classes/units correctly works together.

(c) **System testing.** A *system testing* takes place after the integration testing is completed. This type of testing involves testing the entire system as a whole by providing different types of input to the enterprise application and verifying that the different functionalities of the system work end-to-end. Interfaces to and from the system under test are tested in this phase. This is the third level of testing in which the system is also investigated holistically from the perspective of functional as well as nonfunctional requirements.

(d) **Acceptance testing.** *Acceptance testing* is done towards the end of testing phase, just before the rollout of the enterprise application, by the end users and primarily includes the test cases based on a selective subset of end user requirements. This is the fourth level of testing, and only upon successful completion of this testing the software product is deemed accepted by the client.

(e) **Production testing.** The last level of testing in the testing phase is the production testing. It's basically a pilot release of the enterprise application in the production environment with a subset of user base, product base and transaction volumes to have a firsthand feel of the live enterprise application.

5.1.3 Testing Approach

Testing need not have to occur only after the construction of enterprise applications. Testing typically starts as early as inception of enterprise applications. In fact, testing needs to be planned and executed in parallel to the overall life cycle phases of raising enterprise applications. One of the popular models of testing is the *V-Model*, which is essentially a software development model with a focus on testing spread across the entire life cycle.

We covered the various types of testing, testing methods and levels of testing in the preceding sub-section. Let us now explore the approach used to test enterprise applications. Figure 5-2 depicts the generic approach used for enterprise application testing, comprising of four stages—test strategy, test planning, test execution and test analysis:

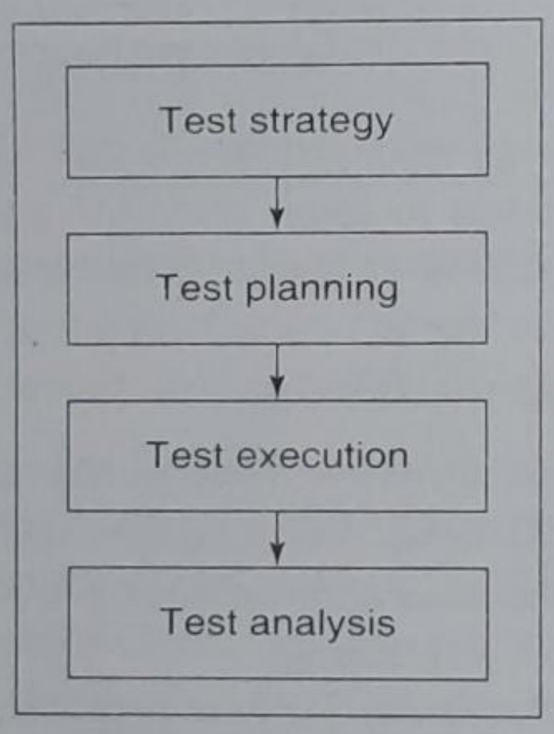


Figure 5-2 Enterprise application testing approach.

(a) **Test strategy.** *Test strategy* refers to the overall approach to test the enterprise application. It takes into account the environment, the types of testing to be done, the testing team, the effort, the cost and the timeline of testing, defect tracking, tools to be used, test planning and when to end the testing. The output of this phase is a test strategy document.

Test strategy also includes the entry and exit criteria for various types of testing. Entry criteria and exit criteria of testing answer the questions related to "when" to start the testing and "when" to end the testing. Before getting into any type of testing, the software should be ready in certain aspects. These aspects are termed as *entry criteria* that specify the prerequisites for any type of testing activity. No amount of testing can guarantee the complete elimination of defects, but it is important to know when to stop. Exit criteria specify when to stop a particular testing phase/cycle and move on to the next one.

(b) **Test planning.** Test planning is used to plan the execution of tests as decided in the test strategy. A good test plan can even find the gaps in requirements or bring clarity to requirements. It contains information about the tests to be performed and provides the overall direction to the testing activity along with test cases and test procedures. Test cases specify the specific features and scenarios for testing, the steps required to carry out the test and the test data along with the expected results. Test procedure explains the steps involved in executing the test for a specified test case.

(c) **Test execution.** It is easy to execute the tests if there is a detailed test plan prepared. Test plans have to be kept up to date and expected results correctly documented. Development team may be responsible for executing the unit testing and integration testing. Typically separate teams are formed including external ones to execute system testing for a large enterprise application. It is important to ensure that all the test preparations are done and that entry criteria for the tests are met along with the test data required for testing. Each of the test cases is then executed and the results are logged. All the integration points also need to be checked. Defects are entered in the defect tracking system and passed on to the application development team. Test execution is done for both functional and nonfunctional testing. Testing is considered complete when the quality goals set in the test strategy are met.

(d) **Test analysis.** The defects captured are analyzed for complexity, type, severity, impact and many other things. Defects found during testing can be a good leading indicator for what can be expected in acceptance testing. There is a saying that the more number of defects you find upfront, there are still some more defects crossing that stage. Corrective actions are taken based on the defect analysis.

Table 5-2 provides a bird's eye view of the enterprise application testing activities spectrum and how they are typically distributed across the development lifecycle.

Table 5-2 Enterprise application testing matrix

	Strategy	Planning	Execution and analysis
Inception	• Acceptance test strategy	• Acceptance test planning	
Architecture and design	• Unit test strategy • Integration test strategy • System test strategy	• Unit test planning • Integration test planning • System test planning	
Construction			• Unit testing
Testing			• Integration testing • System testing • Acceptance testing

Several tools are used in testing: automated regression testing tools, traceability tools, code coverage tools, performance and load testing tools, defect tracking tools, static analysis tools, compliance checking tools, test case generation tools, test data generation tools, etc. Depending upon the need and the complexity of the enterprise application, different tools can be used to increase the effectiveness of testing and the overall productivity of raising enterprise applications.

5.2 Enterprise Application Environments

You are already familiarized with the technical architecture building blocks and the deployment view in Chapter 3. The physical representation of these building blocks is referred to as enterprise application environment. There are several types of application environments that an enterprise application passes through in its lifecycle:

- Development
- Test
- Quality assurance
- Staging
- Production

These application environments are shown in Figure 5-3 and explained in Table 5-3.

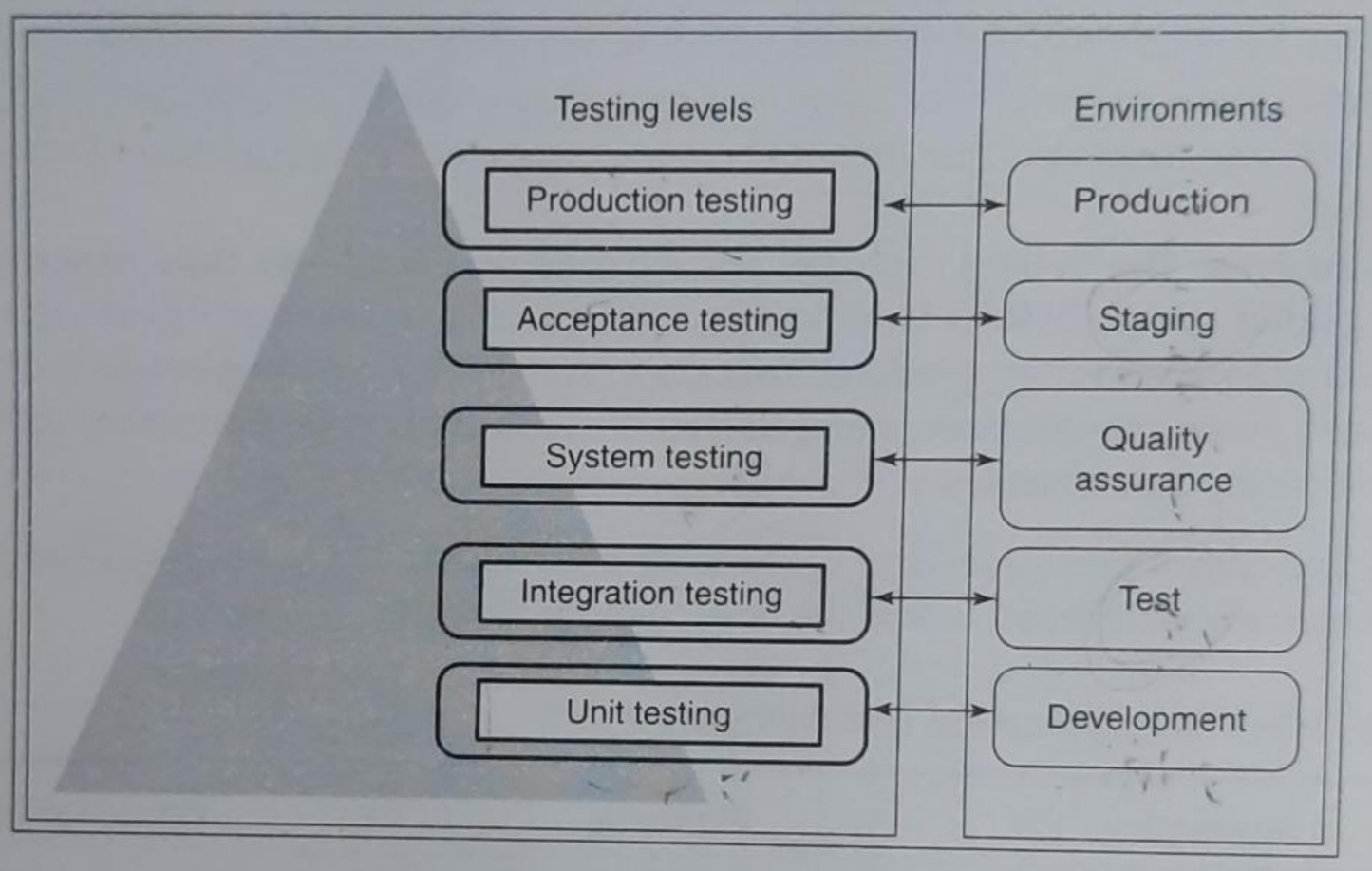


Figure 5-3 Enterprise application environments and testing levels.

Table 5-3 Purpose of application environments

Application environment	Purpose
Development	Development environment is a high-access, low-budget and low-security environment used mainly by the developers for construction activities including unit testing.
Test	This environment is used by the developers to perform integration activities and to do integration testing. Test environment will typically have moderate access control and moderate security.
Quality assurance (QA)	QA environment, having high-access controls, is typically used for QA activities by an external QA team. Most of the nonfunctional requirements are tested in this environment.

(*Continued*)

Table 5-3 (*Continued*)

Staging	Staging environment mimics production environment in all aspects, possibly except for capacity and security. This environment may be used for training end users. This ensures all physical configurations are tested and ready for service.
Production	Production environment is the environment where the final enterprise application eventually resides. This environment has high security and high access control. The capacity of this environment is designed to cater to real-life requirements. This also has to ensure high availability.

You may wonder why so many environments are required to raise enterprise applications. Each environment serves a specific purpose in the overall enterprise application landscape and is normally mapped to one or more of the testing levels.

Apart from the testing levels, other considerations, such as security, availability, access control, ownership of environment, and available budgets and capacity, determine the type of enterprise application environments. By now, you would have noticed that the maturity of the enterprise application increases as it moves from development environment to the production environment.

Let us now discuss how the LoMS application production environment needs to be built. To start with, let us go back to the nonfunctional requirements of LoMS discussed in Chapter 2. A few key nonfunctional requirements are as follows:

- LoMS shall be able to process a peak of 60 and an average of 40 concurrent transactions per second, support 250 concurrent user sessions and process 1000 loan applications in a day.
- LoMS should be able to support an average growth of 20% in volume of loan applications every year.
- LoMS is an Internet facing application with availability requirement of 99.999%.

Based on the above nonfunctional requirements, the server configuration of the production environment has to be worked out. For example, the following need to be considered in the planning of the LoMS application's production environment:

- **Concurrency.** Based on the requirements for concurrency, the production environment should be planned with load balancing capabilities.
- **Capacity.** Based on the average and peak load requirement, the production servers—both database and application—need to be sized.
- **Security.** Since LoMS is an Internet-facing application, the firewalls and DMZ need to be in place.
- **Scalability.** The configurations of the production servers need to be modeled based on the current and future requirements of the user base and number of loan applications expected.
- **Availability.** Based on the availability requirements of LoMS, it should be fault tolerant. For this, LoMS production servers should be clustered.

The deployment architecture of enterprise applications has to be planned based on such factors. Figure 3.33 (in Chapter 3) on architecting and designing depicts the deployment view of LoMS application.

5.3 Integration Testing

The software units/components are integrated into assemblages after unit testing. The interfaces among these underlying units/components are tested to ensure the working of these units, when

combined together, and is referred to as *integration testing*. Integration testing teams can use software construction maps, along with other UML diagrams such as sequence diagram, to understand the big picture of these units/components.

Integration testing is a classical example of gray-box testing. Integration testing team needs to know about the individual units/components to test the assemblages but need not always necessarily delve into the specific details of individual units. Entry criteria for integration testing are completion of integration of components, and addressing the requisite functionality as part of the construction phase.

Let us take the sample scenario of submitting and saving the loan details of the "initiate loan" use case. The sequence diagram is presented in Figure 5-4.

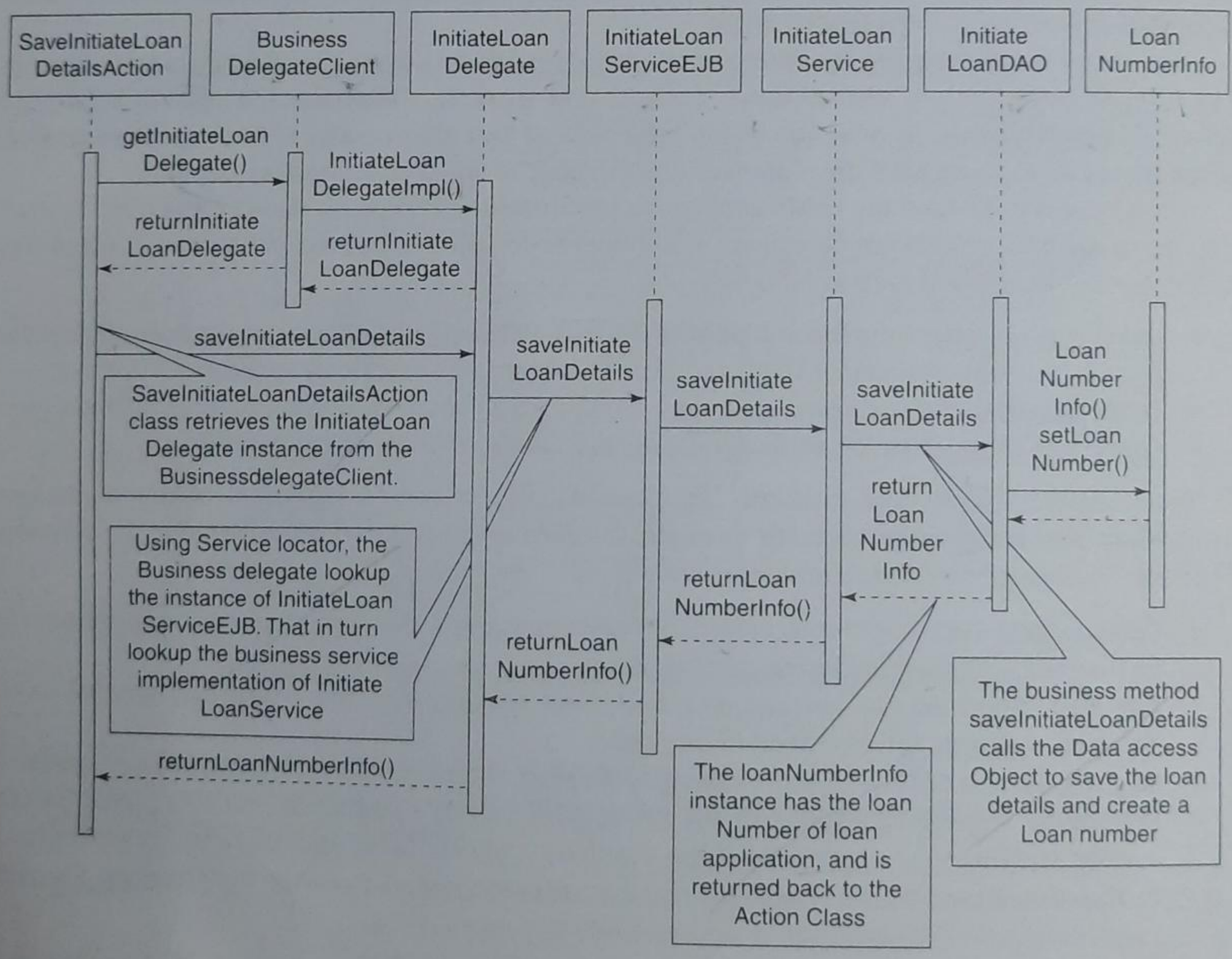


Figure 5-4 Scenario for submit and save loan details.

Entry criteria for the integration testing of this scenario is to have the order form displayed with all the details available to initiate the loan. All the components depicted in the sequence diagram (Figure 5-4) must be completed and integrated to start the integration test.

Table 5-4 Sample integration test cases

Test case	Components under test	Expected result	Result
Enter the details of loan application in the loan initiation form and submit the same.	`InitiationForm.jsp` and `Save InitiateLoan DetailsAction` class	`SaveInitiateLoan DetailsAction` should be invoked	Pass/Fail
Store loan application attributes of InitiateLoanDAO in the underlying loandetails table	`InitiateLoanDAO` and `loandetails` table	All loan application attributes are stored in loandetails table and respective loan number is generated	Pass/Fail

Similarly there are various scenarios, which exist across the entire application that need to be tested as part of the integration testing. Multiple cycles of integration testing are conducted till all the scenarios are successfully verified. Exit criteria for integration testing are when the system provides the expected result for all the test conditions and scenarios under integration testing.

Xp Should integration testing be done after all the components are developed as a big bang or it be incremental in nature? we suggest the incremental approach as it helps validate the application and framework components at the earliest!

Though some tools help in automating integration testing, it is more a manual intensive testing. Once all types of components related to enterprise application are integrated and tested, the application is ready to move into system testing phase.

In iterative methodologies, the same component and assemblages have to go through multiple transformations throughout their life cycle. It is important to ensure that a change in component does not break another part of the application. To ensure this, after each iteration the application needs to be regression-tested. Regression testing includes running the previous test cases and comparing the results with the previous test results. Usage of regression testing tools helps significantly in saving effort for testing frequently changing and complex components and assemblages.

Tt HP Quick Test Professional and IBM Rational Functional Tester are some of the regression testing tools.

5.4 System Testing

System testing is testing the enterprise application and its interfaces as a whole. It is a black-box testing, where a completely integrated enterprise application is tested for the specified functional and nonfunctional requirements.

Though testing functionality, using end-to-end scenarios, is integral to the system-testing phase, this chapter focuses more on the nonfunctional testing aspects. To answer "what" kinds of testing need to be done, let us explore a few of the nonfunctional testing types that are carried out in the system testing phase of enterprise applications in the following sections.

5.4.1 Performance Testing

Performance testing is one of the most important aspects of system testing carried out from the perspective of validating the responsiveness of the system under a given workload profile, scalability to handle growth in number of users and data volume and stability without incurring visible performance degradation. Typically performance testing comprises of activities such as load testing, volume testing, stress testing and endurance testing. All these types of testing enable verifying the application's performance in a holistic manner.

Load testing

Load testing is where the enterprise application is subjected to a variety of loads, mainly in terms of the number of users accessing it, to baseline the performance of the enterprise application. Typically the application is subjected to a model of expected usage to determine the performance capabilities of the system.

Volume testing

Volume testing is somewhat similar to load testing carried out by varying the volume of data processed by the system. This type of testing is usually done on all key, frequently-used functionalities.

Stress testing

Stress testing is done to ensure that the enterprise application can handle the spikes in load due to unusual circumstances like a sudden surge in user hits. It is to test the breaking point when the enterprise application is put to extreme stress in form of a large volume users and data. It provides very useful information to plan the configuration of system live/production environment.

Endurance testing

Endurance testing is performed primarily to identify problems related to memory leaks by putting the enterprise application through sustained load conditions.

Performance testing has the following objectives:

- Determine enterprise application performance at a given user load and data volume—load testing and volume testing
- Determine the breaking point of the system under mounting load—stress testing
- Simulate the conditions to determine the reasons for performance bottlenecks—endurance and other forms of performance testing mentioned above
- Benchmarking the performance

Xp

Performance should not be an afterthought. It should be engineered right from the inception of an enterprise application.

Performance testing is a part of the overall performance engineering life cycle, as shown in Figure 5-5. It starts with the elicitation and validation of performance NFRs in the first phase of raising enterprise applications. Performance test strategy and plans are laid out in the architecture and design

phase. Design considerations specific to performance perspective are also taken care of in this phase. Size of infrastructure to meet the requirements is also determined in this phase. Performance testing happens as part of system testing in the testing phase. The testing results help in determining the performance bottlenecks in the application that are tuned to meet the expected performance requirements. These results also help in benchmarking the performance of the application and capacity planning before the rollout of the application.

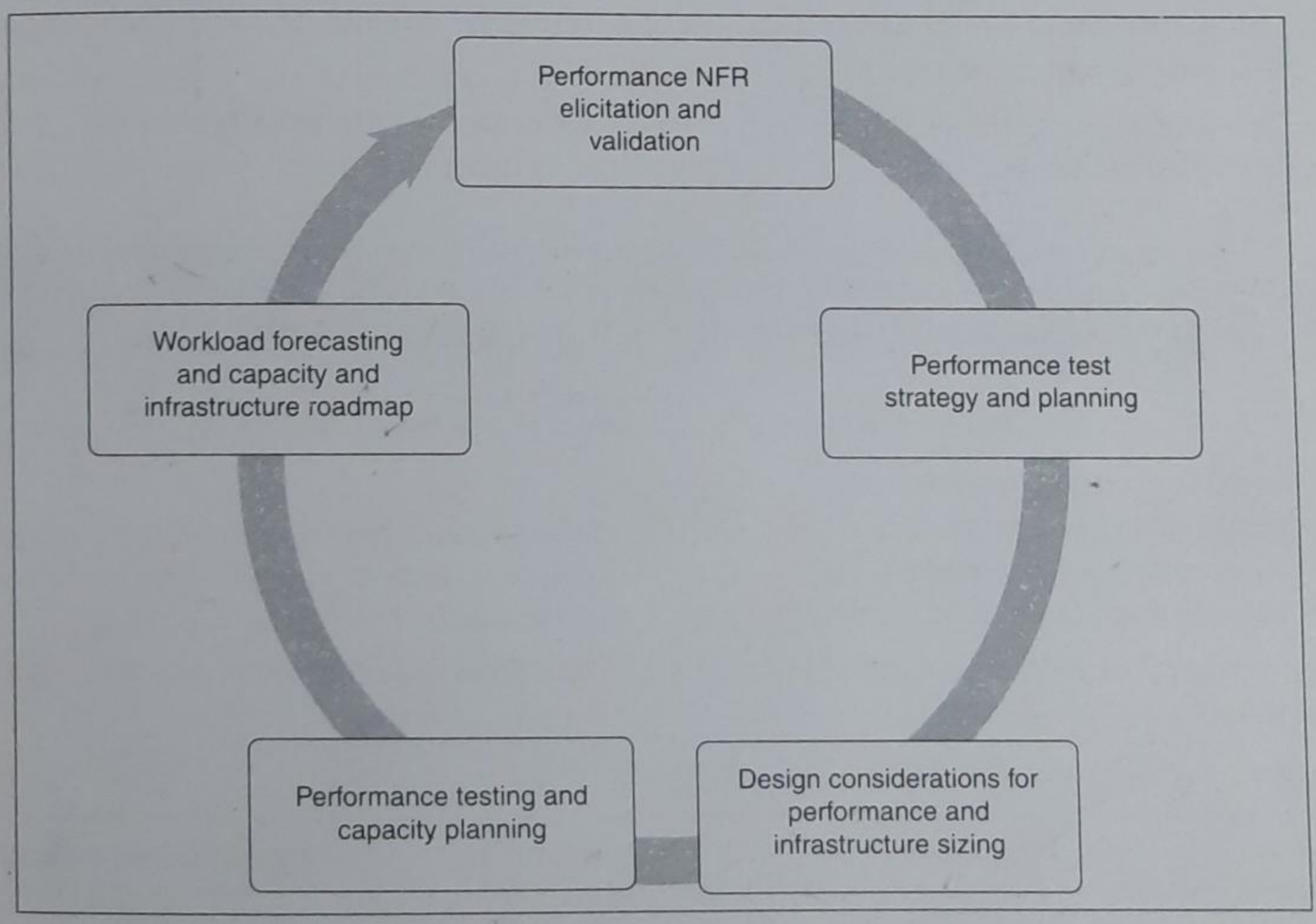


Figure 5-5 Performance engineering life cycle.

Performance engineering is a continuous endeavor to keep enterprise application up and running, and sustaining the required performance levels of the application. Performance testing is usually performed in the quality assurance environment with the use of specifically designed test scripts and tools. The test results are used to gauge and investigate the performance related issues. The output of performance testing is the recommendation for corrective action.

Tt

Apache JMeter and Loadrunner are examples of tools used to test the performance of enterprise applications. They provide support for various kinds of performance testing, including load testing, volume testing, stress testing, etc.

The performance NFRs of LoMS are listed in Table 5-5 for easy reference.

Table 5-5 Performance NFR metrics for LoMS

Maximum number of concurrent users	250
Peak throughput (TPS or hits/sec)	60
Average throughput (TPS or hits/sec)	40
Maximum refresh/response time (in seconds)	5
Number of loan applications per day	1000

Based on the above performance NFRs, the load test is designed to be performed using JMeter with the following load conditions:

- Number of users: 250 (in JMeter, users correspond to threads, i.e. virtual users)
- Ramp up period: 60 seconds

Xp

For a performance test to simulate the real behavior of the system, it is important to identify the appropriate workload mix for the system. Workload mix is the collection of various activities that system users may perform in the system during a given period of time.

The test plan that is used for load testing comprises of the following test cases:

- Login to the LoMS application
- Initiate loan application form
- Loan application submission
- Logout from LoMS application

These test cases are further broken down into the individual user activities/transactions, as shown in Table 5-6.

Table 5-6 Performance (load) test cases

Test case	User transactions	JMeter transaction names
Login to the LoMS application	1. Load the login page of the application in the browser by entering the address of the Web site. 2. Provide user name and password of a valid user and submit.	1. Load login page 2. Login transaction
Initiate loan application form	Clicks on initiate loan link in the menu and waits for the loan application form to be loaded. This test case is executed by a virtual user a predefined number of times in a loop.	3. Load the home page
Loan application submission	Fills the application loan form and submits. The loan application ID gets generated in response. This test case is executed by a virtual user a predefined number of times in a loop.	4. Initiate loan action
Logout from LoMS application	Clicks on logout link.	5. Logout transaction

The above user transactions are recorded in JMeter using an HTTP Proxy server element of JMeter, as shown in Figure 5-6.

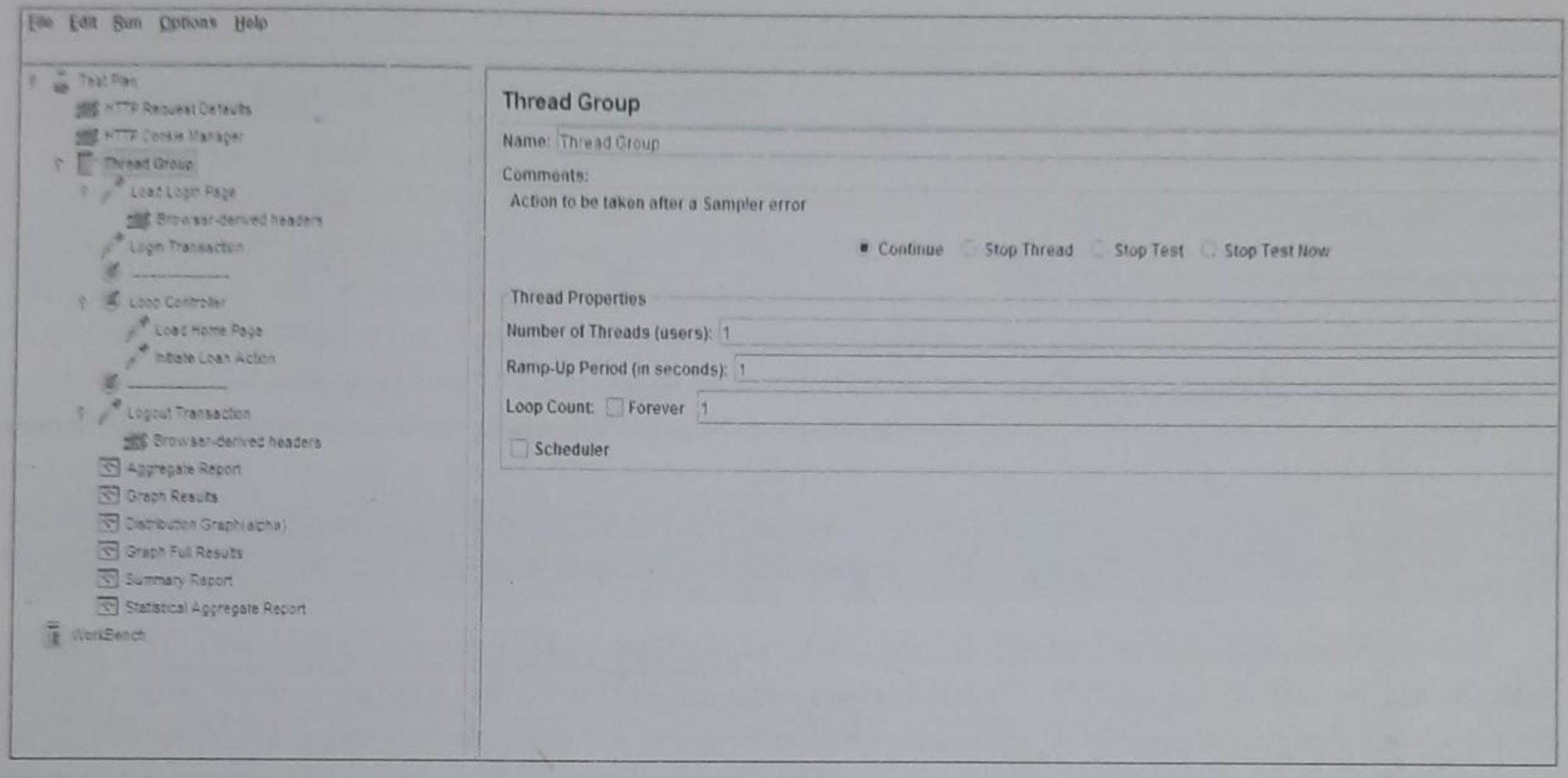
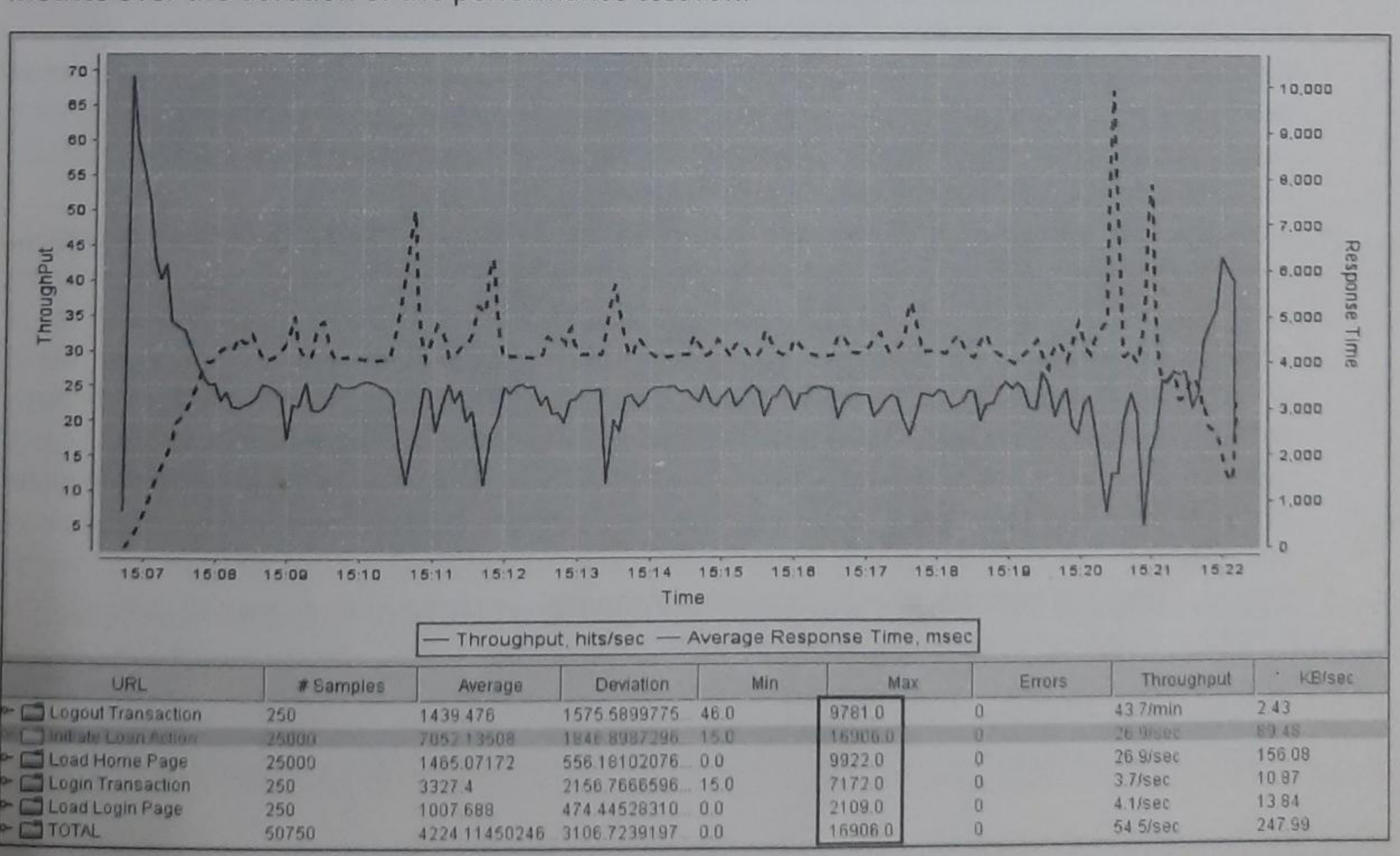


Figure 5-6 JMeter test plan setup.

The user transactions are captured in a single-thread group element and appropriate listeners are added to it to capture the metrics of the performance run. A statistical aggregate graph is generated using JMeter, as shown in Figure 5-7, which shows the variation of the key application performance metrics over the duration of the performance test run.



URL	# Samples	Average	Deviation	Min	Max	Errors	Throughput	KB/sec
Logout Transaction	250	1439.476	1575.6899775	46.0	9781.0	0	43.7/min	2.43
Initiate Loan Action	25000	7057.13508	1846.8987296	15.0	16906.0	0	26.9/sec	89.48
Load Home Page	25000	1465.07172	556.18102076	0.0	9922.0	0	26.9/sec	156.08
Login Transaction	250	3327.4	2156.7666596	15.0	7172.0	0	3.7/sec	10.87
Load Login Page	250	1007.688	474.44528310	0.0	2109.0	0	4.1/sec	13.84
TOTAL	50750	4224.11450246	3106.7239197	0.0	16906.0	0	54.5/sec	247.99

Figure 5-7 Statistical aggregate report.

The results of the performance (load) run are depicted in Table 5-7.

Table 5-7 Results of the performance (load) run

Performance NFR	Expected result	Actual result	Status
Maximum number of concurrent users	250	250	✓
Peak throughput (TPS or hits/sec)	60	69	✓
Average throughput (TPS or hits/sec)	40	54.5	✓
Maximum refresh/response time (seconds)	5	16.91	x -
Number of loan applications per day	1000	25000	✓

Xp In a scalable enterprise application, while the system resources' utilization increases with increase in user load, the response time should ideally remain constant.

As mentioned above, the test results are used to investigate performance bottlenecks. The bottlenecks could be due to any part of the application—the application code, database, Web server, application server, operating system or the network. The reasons for poor performance are several—poor programming practices including database queries or incorrect configurations of servers, operating system and network. Corrective action plans are laid out after performance testing to optimize the performance that typically involves the following activities:

- **Application code analysis.** Run-time behavior of application code is analyzed to investigate the performance bottleneck. It primarily includes profiling the application code base to determine time consuming methods, call sequences, thread deadlock information, list of live threads and thread states, etc., as explained in Chapter 4.
- **Database optimization.** Database schema and stored procedures are also profiled to determine recursive calls, physical reads, number of deadlocks, wait events, etc., to point out the performance bottlenecks as explained in Chapter 4. Database schema objects, like indexes and tables, are also tuned to optimize the database performance.
- **Infrastructure resources and capacity optimization.** Resources primarily include networks, servers—application, Web and database and operating systems. Network latency rate, collision rate, incoming/outgoing packet rate, etc., are the performance parameters to watch for a network. Memory and the processing capacities of various servers also play an important role in determining the performance of enterprise applications. Memory profiling is achieved through dynamic analysis of the application code base, which provides information particularly on object instance counts on the heap, size of the objects, call tree, etc. This helps in determining the memory leaks and setting the right heap size. CPU profiling is performed, which also involves dynamic code analysis, to identify most time-consuming methods, call sequences, etc., which help in identifying the computation related performance bottlenecks in enterprise applications. Some of the concepts were discussed in Chapter 4.

In In case of Java based enterprise applications, JVM profiling is also performed to identify the internal processes of the JVM, most importantly the garbage collection process.

5.4.2 Penetration Testing

Penetration testing is the black-box testing of an enterprise application from the security perspective. This testing not only helps in fortifying the application against the various types of online threats but

also ensures the application's compliance to the industry regulations. This is one of the important types of testing from the perspective of building the enterprise brand and the customer confidence. It is mandatory for B2C applications and community sites involving large number of users.

Penetration testing is a part of the overall application security engineering life cycle, as shown in Figure 5-8. It starts with the elicitation and validation of security NFRs in the inception phase of raising enterprise applications. Security test strategy and planning is laid out in the architecture and design phase. Threat modeling is also performed in this phase. This exercise is performed to determine the security objectives, threats and vulnerabilities of enterprise applications. The output of threat modeling helps in determining the design decisions for application security. The application code base is analyzed in the construction phase for security vulnerabilities. Penetration testing happens as part of system testing in the testing phase. The testing results in unearthing the security vulnerabilities in the application, which need to be fixed before the production rollout of the application. These results also help in ensuring legal compliance of the application from the data security perspective.

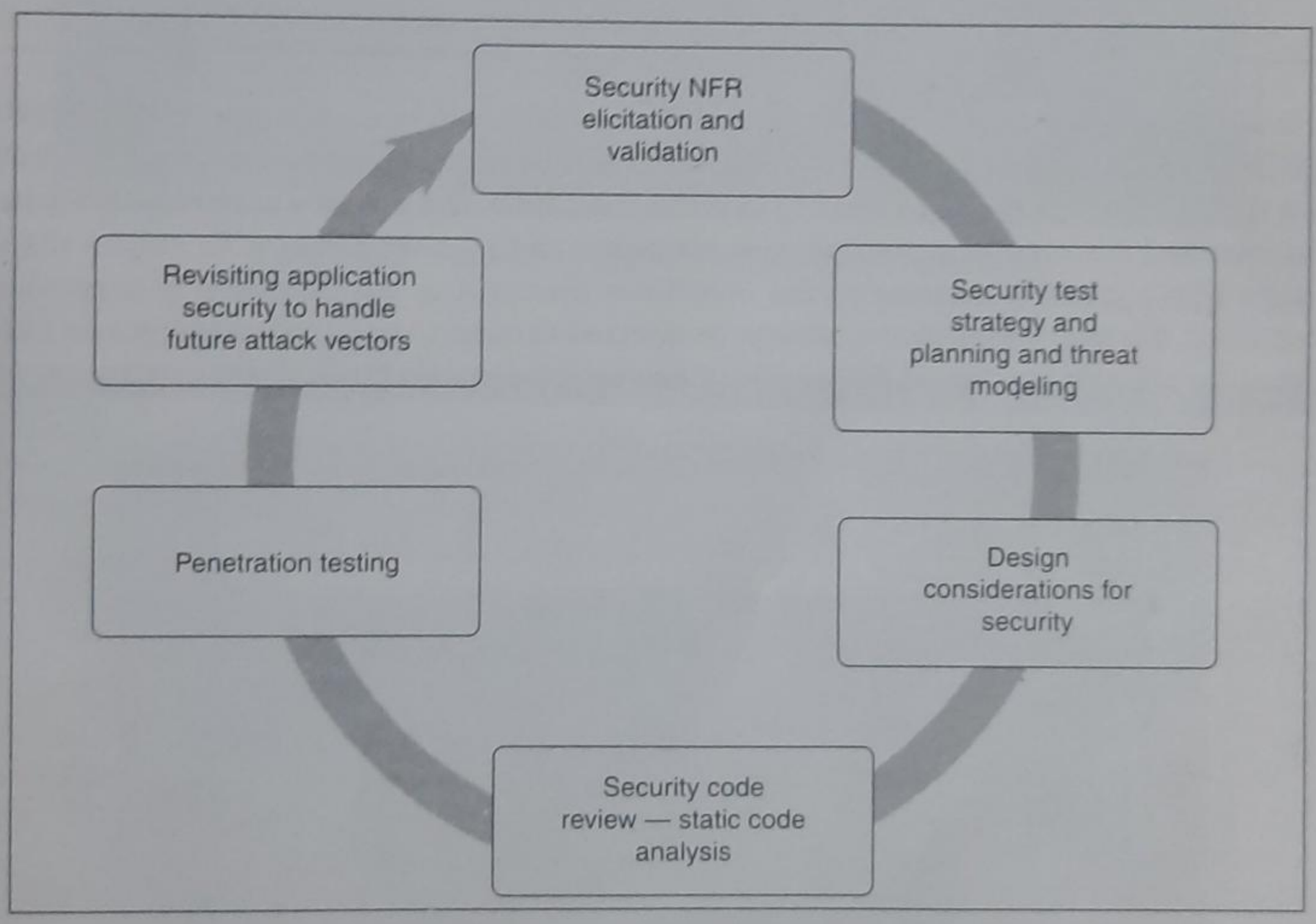


Figure 5-8 Application security engineering life cycle.

Application security engineering is a continuous endeavor to guard enterprise applications from current and future attack vectors and helps in fortifying the application against different forms of attack. Penetration testing is typically a mix of manual and automated testing. Some of the security vulnerabilities can be unearthed by manual testing while others are easier to figure out by automated testing using penetration testing tools.

Tt Paros Proxy and OWASP WebScarab are a few of the freely available and popular tools for Web application security assesment. Fortify 360 is one of the commercial tools available for performing penetration testing.

Security requirements for LoMS dictate taking care of the OWASP top-10 security vulnerabilities. A subset of penetration test cases based on this is shown in Table 5-8.

Table 5-8 Penetration test cases

Test case	Expected result	Result
Input the following text in the username field of the login screen. `Scott' # <iframe src="http://www.google.com" width="500" height="500"></frame></b><b>`	System should gracefully alert the user regarding the bad or malicious input	Pass/Fail
Crawl the Web site using tools like WebScarab.	The tool shouldn't be able to generate the Web resources tree unless the user performing this activity is authenticated to do so.	Pass/Fail

As illustrated in test case 1 (Table 5-8), the field username is populated with the malicious input containing HTML tags. The string "`Scott' #`" is used to inject SQL and the remaining part of the malicious input is a simple JavaScript in order to inject XSS in the application. This scenario is depicted in Figure 5-9.

The intended outcome of providing valid username and password field is to display the user's home page with a greeting message to the user. Since the input to the username field contains the malicious input, it is interpreted by the browser as directed to open an unintended content (in this case, http://www.google.com) within an iframe. This scenario is depicted in Figure 5-10.

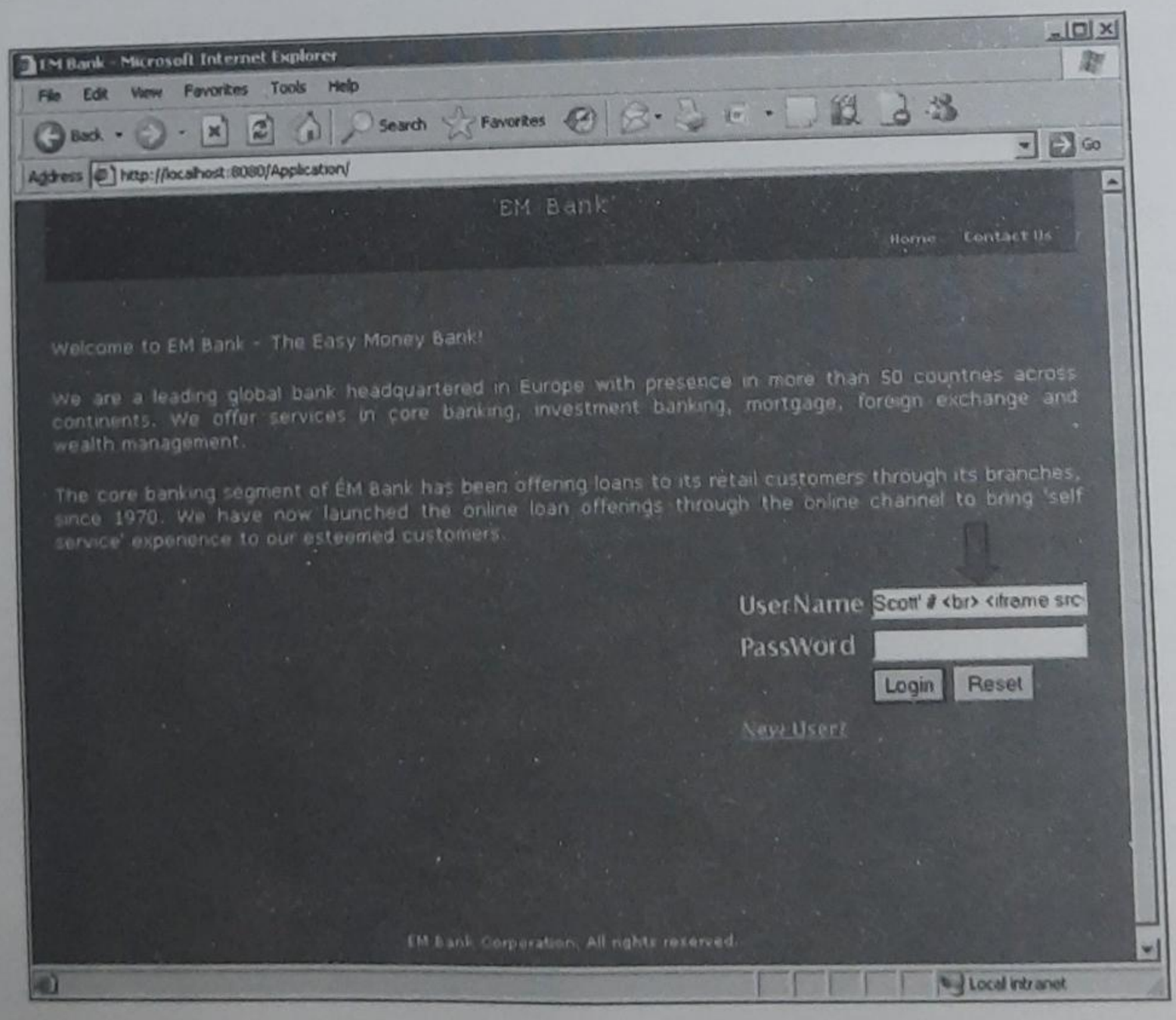


Figure 5-9 SQL and XSS injection demonstration.

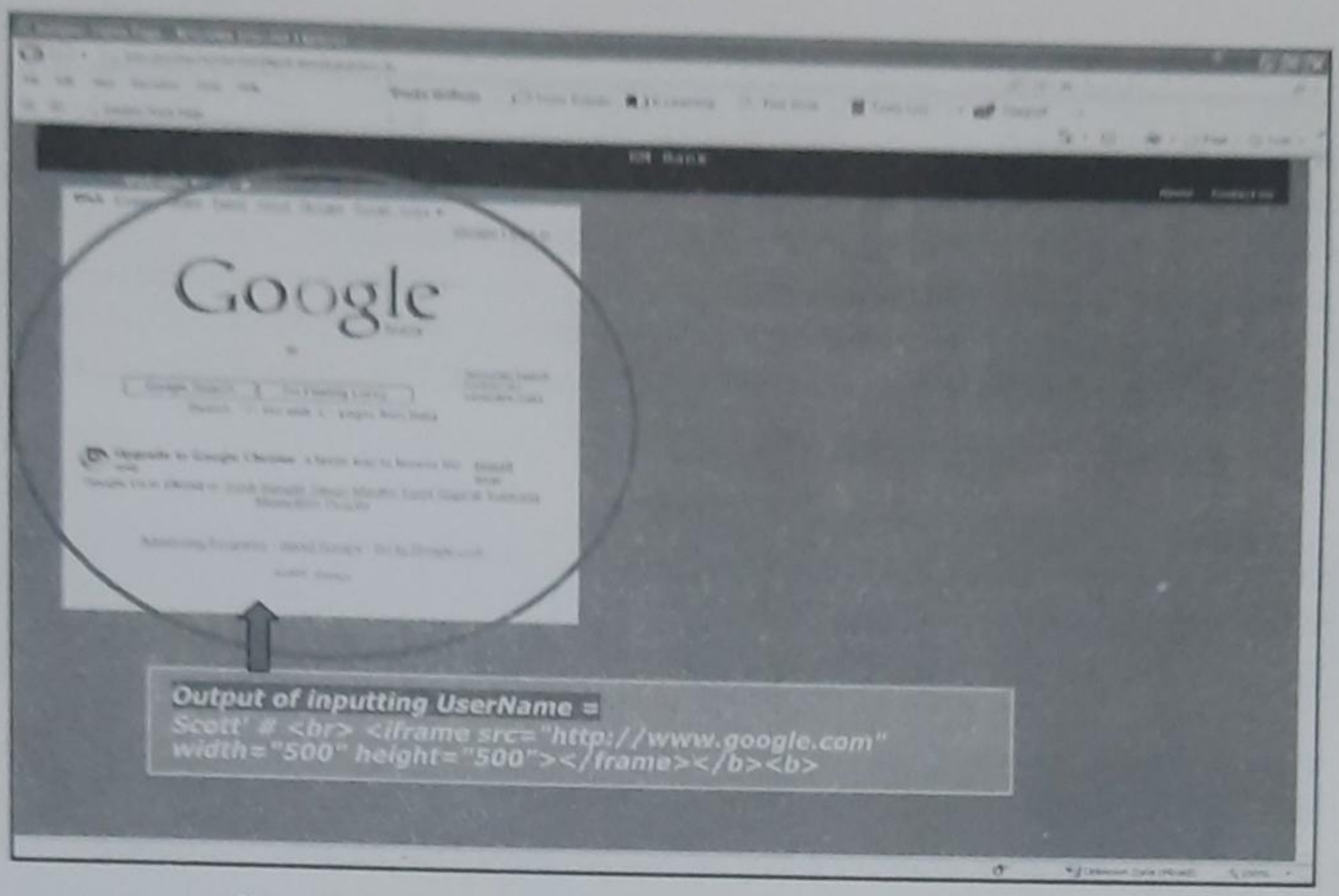


Figure 5-10 Application security vulnerability—unearthed.

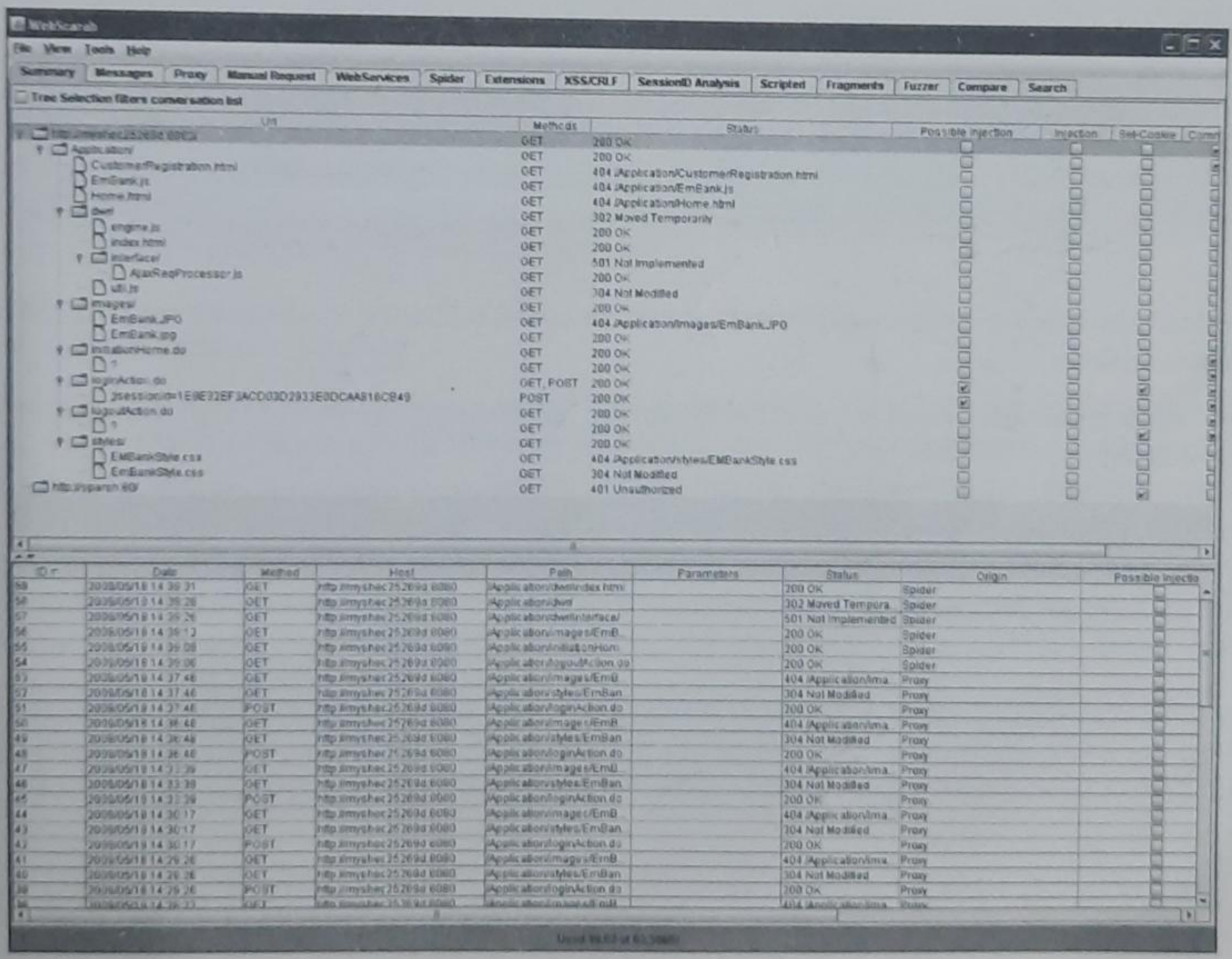


Figure 5-11 Web layer resource tree—outcome of crawling.

As stated in test case 2, the WebScarab tool is used to crawl over the Web layer resources after authentication as a user who is restricted from accessing them. The Web layer resource tree is generated by the tool, as depicted in Figure 5-11, which means that the Web layer resources are not secure. In such a case, any un-authenticated user or a user with inadequate privileges is able to download static resources from the site by typing their address in the browser's address bar.

Various coding best practices are exercised to fix security vulnerabilities. To avoid XSS injection, measures like implementing a security filter for a servlet and escaping HTML characters are used. To avoid SQL injection, usage of prepared statements is advised. The security vulnerabilities and the measures taken to curb them were discussed in Chapter 4.

Xp

Penetration testing tools should be selected based on factors like how accurately they report security vulnerabilities, the extent to which they can be customized and configured, availability of the tool documentation and other necessary support for tool.

5.4.3 Usability Testing

Usability testing is a form of black-box testing of the enterprise application to validate the usability requirements typically mentioned as part of the nonfunctional requirements. This type of testing is primarily useful to ensure that the enterprise application is easy to understand and use. Usability testing is especially important for B2C applications. This is normally performed either by the end users themselves or alternatively by business analysts who represent end users.

Xp

Often usability is misinterpreted as performance of the system. Performance is tested separately as part of performance testing. Together, usability and performance can contribute significantly to end user productivity.

Let us look at the usability requirements stated for LoMS in Chapter 2, which are reproduced below for easy reference:

- LoMS application requires customer and prospects to be self-guided through the loan initiation process.
- For bank representatives, the system should facilitate easy navigation combined with an easy workflow system.
- Since EM Bank is a global bank, the LoMS application needs to be internationalized.
- Context-sensitive help should be provided to a user.

In

Demographics of users needs to be considered in testing the accessibility features of the system. For example, if a significant of user population is above 40 years of age, then it may be useful to design the user interface with larger fonts.

Usability testing has test cases based on the user behavior and nature of use. Considering the requirements stated above, a part of the usability test plan prepared is as shown in Table 5-9.

Table 5-9 Usability test cases

Test case	Expected result	Result
Loan initiation. Test whether the user is self-guided through the whole loan initiation process	Simple and easy-to-use loan initiation process with ease of navigation to complete the initiation process. Already available customer information for loan processing should be pre-populated	Pass/Fail
Loan approval. System should have a simple work flow from initiation to approval and disbursement of loan	Workflow should be completed in a maximum of three steps	Pass/Fail
Globalization. Multi-lingual support	Site to be multi-lingual and all messages and forms are displayed in the correct local language	Pass/Fail
Help. System should provide an adequate help facility for both customers and bank representatives	System should have form level help, tips, process help available based on context.	Pass/Fail
Personalization. System should display the information relevant to the customer	Personalized information displayed to the customer	Pass/Fail

The success of an enterprise application depends on its acceptance by the end users which, in turn, depends primarily on the usability characteristics. Hence, usability testing becomes one of the important steps in system testing. Significance of usability testing increases when the system exposure to the outside world increases. For example, the system would be tested for usability more rigorously in a B2C environment compared to B2B or in-house applications.

5.4.4 Globalization Testing

Enterprises are reaching out to users across the globe, which mandates applications that can be localized per the user needs to ensure customer friendliness. *Globalization testing* is a manual testing to validate whether an enterprise application meets the internationalization requirements. It also validates whether the application can be localized based on a given locale without any architectural or design change. Globalization (G11N) is the term typically used for a combination of internationalization (I18N) and localization (L10N), and is a key aspect of the usability of an application. This testing is typically performed by test engineers and business analysts, who are aware of the language of choice, by running the application in different locales.

Globalization testing involves testing of various things related to data and content of the enterprise application. Language translations, writing direction, currency symbols, units of measures, date and time format, time zones, number formatting, fonts, sizes, colors, images, collation and ordering are the predominant things which are typically validated as part of this testing. Based on the LoMS globalization requirements, a sample of the globalization test cases is depicted in Table 5-10.

Table 5-10 Globalization test cases

Test case	Expected result	Result
Login as a bank representative using locale different from the default one, and sort the customers on customer first name	The sort order should be correct based on the locale selected	Pass/Fail
Login as a customer and select the locale different from the default one, and check the date and currency format	The date and currency format should be correct based on the locale selected	Pass/Fail
Login as a customer and select the locale different from the default one, and check the language translation on customer page	The language of the form fields and other form content should be correctly translated based on the locale selected	Pass/Fail

Xp

In practice, the rollout of a globalized application typically happens in several phases based on the successful completion of the globalization test suites for each locale. A single universal rollout of the application across all the required locales may be difficult to achieve due to considerations such as availability of language experts for all the locales.

5.4.5 Interface Testing

Interface testing validates the incoming and outgoing interfaces of the application under test with respect to all other applications that it interacts with. It is also known as *intersystem testing*, which validates the coexistence of the application under test with other enterprise applications.

Objectives of interface testing are dependent on several factors such as nature of communication of interfaces, request-response handling of the system, data formats exchanged through the interfaces, types of validations performed and nature of errors and exceptions that could arise. In synchronous interfaces, the response time is a major consideration as against in asynchronous interfaces.

Xp

During interface testing, especially synchronous ones, the system interfaces should not be just validated from the exchanged content or data perspective. Optimum performance need to be validated by performing performance test on the participating system interface for the end to end scenarios.

The entry criteria for interface testing are the readiness and correct documentation of all the participating systems in the interface testing. The documentation should properly document the data passed between the systems. Based on the LoMS interface requirements, a sample interface test case is shown in Table 5-11.

Interface testing involves a mix of manual and automated techniques and is typically a collaborative effort by multiple teams responsible for different systems whose interfaces are integrated with the

Table 5-11 Interface test cases

Test case	Expected result	Result
Login as customer and check the pre-approved loan amount at the time of loan initiation	The pre-approved loan amount will be made available to the customer at the time of loan initiation by credit rating system	Pass/Fail

in-bound and/or out-bound interfaces of the system under test. As presented in Chapter 3, there are various integration mechanisms and technologies to integrate enterprise applications. Each one of them has its own testing mechanisms.

Interoperability and platform independence are the need of almost all enterprise applications of modern times. To facilitate these, interfaces based on Web services constitute a considerable share of system interfaces of enterprise applications. Testing system interfaces developed using Web services is different in several ways, as it involves validating security, interoperability, platform independence, etc. Lack of a user interface also poses a few challenges in testing interfaces developed using Web services.

Tt SoapUI is one of the open source Web services testing tool to test SOAP and REST based Web services. It provides features like Web service inspection, Web service invocation, load testing, compliance testing, Web service simulation and mocking among other features.

5.5 User Acceptance Testing

User acceptance testing is a black-box testing of an enterprise application performed by the end users of the system, primarily from the perspective of system's stated functionalities and usability. This is the last stage of testing before the application rollout, and is required to certify the readiness of enterprise application to go live in production. The team to certify the "acceptance" of the enterprise application typically represents a cross-section of the end users of the application. This testing is mostly manual and is carried out in the staging environment.

The entry criteria for user acceptance testing are successful completion of unit, integration and system testing. The bugs identified during these testing should either be fixed or documented as "known issues". The objectives of a typical user acceptance testing are to validate:

- The stated functionalities in the system requirement specification document
- Part of nonfunctional requirements like usability and response time
- The access control of the modules of the application
- The complete business processes
- The business data from the outgoing and incoming interfaces
- Batch processes, if any
- The report format and contents

Xp The acceptance criteria for the enterprise application should be worked out with the end user in the early life cycle phase of the application development.

User acceptance testing plan is prepared by taking inputs from the system requirement specification document. A part of the user acceptance testing plan for LoMS is presented in Table 5-12.

Table 5-12 User acceptance test cases

Test case	Expected result	Result
Login as an existing customer and check the pre-approved loan amount	Pre-approved loan amount should be displayed based on credit rating of the customer	Pass/Fail
Login as a prospect and try to initiate loan	Prospect should not be able to initiate the loan	Pass/Fail
Login as an existing customer and check the fonts, colors, toolbars, tooltips, text wrap, static content on the customer home page	The customer home page should be correct in content and consistent with the overall presentation of the enterprise application	Pass/Fail
Login as a bank representative and upload the signed loan application	The contents and format of signed loan application should be appropriate and as per the EM Bank's guidelines	Pass/Fail

Xp

The members of UAT team should be trained using manual and training aids to get acquainted with the features of enterprise application under test, before the commencement of user acceptance test.

A controlled and organized user acceptance test plan and test cases help in keeping testing focus on the important features of the enterprise application and gain end user confidence. A successful user acceptance test sets the stage for the acceptance of the application by the end user community. By this time, the enterprise application is all set to be rolled out in the production environment.

5.6 Rolling Out Enterprise Applications

Rolling out enterprise applications means making the application live on the production environment for the end user. This is the time when the enterprise applications are handed over to the enterprise application maintenance team. Release engineers facilitate the rolling out procedure of enterprise applications. There are a few high-level strategies to be considered for rolling out enterprise applications:

- **Brand new rollout.** This rollout strategy is for the first roll out of a brand new application. This rollout is possibly the easiest to manage among all other rollout strategies.
- **Parallel switch.** The parallel switch rollout strategy is to bring up a new application environment in parallel. The new environment is verified, and application traffic is switched from the old to the new environment. The old environment is kept on standby for a predetermined period. This rollout strategy has nil to minimal downtime.
- **Rolling rollout.** The rolling rollout strategy is applicable in a clustered and highly managed environment. In this strategy, a node of the cluster is taken offline, upgraded, verified, brought

to online status, and then the next node follows suite. This is not applicable to all types of systems, as it poses challenges in terms of transaction integrity and data continuity of the enterprise applications.

- **Phased rollout.** Phased rollout is a slower variant of the rolling rollout strategy. The rollout is phased out on the basis of particular region or time.
- **Up-down-up rollout.** This rollout strategy starts with bringing everything down. The system is upgraded, verified and brought up again. This strategy is the most time consuming and requires a lot of upfront planning. This is the safest rollout strategy for large-scale deployments.

Whatever be the selected strategy to rollout enterprise applications, the following are the key prerequisites for a successful application rollout:

- The rollout plan that should consist of sequence of rollout related events, rollout schedules, key personnel along with their contact information and the escalation mechanism
- Communication plan for rollout that describes who gets involved at what stage, handshake protocol for handover from one group to another and escalation protocol
- Certification plan to ensure that everything is working which is necessary for rollout activities; release notes, scripts, configuration files, installable packages, etc., should all be in place
- Plan B or the rollback plan which is required if things do not go well as per the rollout plan; the plan should also contain the impact in money terms, brand image and the impact on customer for every hour of delay during the rollout
- Identify the best time and duration for launching the rollout of enterprise applications
- Depending on the criticality of the rollout, a dress rehearsal can be done before roll out, possibly in the test or staging environment. Dress rehearsal is similar to a smoke test. *Smoke testing* is a cursory check of the crucial pieces of the enterprise application without getting into the finer details of them

Once the rollout strategy is finalized and the team is ready to roll out the enterprise application, the actual rollout of application takes place. Although steps to rollout the application may vary, depending on type of deployment, a typical rollout goes through the following steps:

1. Make sure all the communication systems like emails, instant messengers, pagers, cell phones, war room phones are working.
2. Identify and verify the binaries, configuration files, scripts and artifacts to be used in the application deployment. Typically a quality analyst should have all these tagged and labeled.

Xp Do not check out code from version control systems for the production rollout. Everything should be prebuilt, verified and readily available on the target systems prior to starting the rollout activity.

3. Run backups and setup recovery points.
4. Turn OFF monitoring, alerting and alarm notification systems. Increase queue depths and cache ratios, if you expect traffic to pile up for subsequent processing after the system is back online.

Tt BMC Patrol, Tivoli monitors, Queue threshold monitors, pager notification systems, etc., are a few examples of monitoring, alerting and alarm notification systems.