Hewlett Packard Enterprise

HPE Digital Learner Software Testing Content Pack

HPE Content Pack number	CP025
Content Pack length	28 Hours
Content Pack category	Category 2
Learn more	**View now**

In this course, you will learn concepts of software testing, its objectives, principles and processes. You will learn about software testing methodologies and the steps involved in creating software tests. You will also gain experience with Cucumber, JUnit, JMeter, and TestNG.

Why HPE Education Services?

- IDC MarketScape leader 5 years running for IT education and training*
- Recognized by IDC for leading with global coverage, unmatched technical expertise, and targeted education consulting services*
- Key partnerships with industry leaders OpenStack®, VMware®, Linux®, Microsoft®, ITIL, PMI, CSA, and SUSE
- Complete continuum of training delivery options—self-paced eLearning, custom education consulting, traditional classroom, video on-demand instruction, live virtual instructor-led with hands-on lab, dedicated onsite training
- Simplified purchase option with HPE Training Credits

Audience

Software developers, technical and user acceptance testers, test analysts, test engineers, test consultants, test managers, project managers, and quality managers; anyone with an interest in testing

Content Pack objectives

This Content Pack provides the information necessary to:

- Identify software testing objectives, methodologies and processes
- Describe the steps involved in creating software tests
- Describe the steps involved in managing, troubleshooting and automating software tests
- Use JUnit to create a test case and use fixtures
- Use Cucumber to create text-based feature requirements and run automated acceptance tests
- Describe static and dynamic testing techniques
- Identify test conditions and document test cases and procedures
- Define automated software testing and common software testing frameworks

*Realize Technology Value with Training, IDC Infographic 2037, Sponsored by HPE, October 2017

Detailed Content Pack outline

Software Testing Fundamentals

Creating Software Tests

Software testing is critical to the development of quality software applications. In this course, you will learn about software testing methodologies and the steps involved in creating software tests. This course is one of a series that covers the objectives for Microsoft Technology Associate (MTA) exam 98-379, Software Testing Fundamentals.

Outline

- Describe software testing and its benefits, and outline metrics for software quality
- Describe the principal components of computer systems
- Distinguish between compiled and interpreted programming languages, describe data types and analyze simple algorithms
- Analyze simple algorithms and step through corresponding flowcharts and code
- Describe sequential and iterative software development models such as the waterfall, agile and spiral models
- Describe and contrast the product and project life cycles
- Describe and contrast manual and automated testing
- Describe black box testing
- Describe white box testing and contrast it with black box testing
- Use Microsoft Test Manager
- Describe unit tests and the goal of unit testing
- Describe integration testing
- Describe regression testing
- Use web performance and load tests in Visual Studio
- Describe usability, accessibility and localization tests
- Describe performance and stress tests
- Describe user requirements and their importance in relation to testing; describe user requirement documents, including use case diagrams and user stories
- Describe test-driven development
- Describe testing hooks and provide examples
- Describe application lifecycle management in the context of an agile project
- Describe briefly ALM tools in Microsoft Visual Studio Team Foundation Server including PowerPoint Storyboarding, Product Backlog, Sprint Backlog/team capacity, Task Board, Request Feedback, and Microsoft Feedback client tools
- Describe how testing strategy is determined and describe the different scopes that test plans may have
- Describe code coverage and demonstrate how it can be analyzed in Visual Studio
- Describe software features and feature testing; describe exploratory testing, usability testing, and UI testing in the context of feature testing
- Describe test case design and considerations including scope and detail, and the use of boundary conditions and validity tests
- Design software tests

Managing, Troubleshooting, and Automating Software Tests

Software testing is critical to the development of quality software applications. In this course, you will learn about the steps involved in managing, troubleshooting and automating software tests. This course is one of a series that covers the objectives for the Microsoft Technology Associate (MTA) exam 98-379, Software Testing Fundamentals.

Outline

- Define process guidance and describe the process template in Microsoft Team Foundation Server
- Describe phases and internal and external milestones
- Describe entry and exit criteria and sign-offs for software development phases
- Describe the agile development process and the role of documentation and feedback in the process
- Describe the Scrum agile implementation
- Describe the Kanban management process
- Describe the XP agile implementation
- Describe the importance of communication in agile projects, how it occurs for co-located teams; describe the challenges and advantages of working with distributed teams
- Describe the scrum of scrums feature
- Describe reporting in agile projects
- Describe burndown charts and show how to access burndown reports in Microsoft Team Foundation Server
- Describe the overview manual, automated and distributed tests, and how to run them
- Describe the ways in which you can run tests in Microsoft Visual Studio and use one or more of these methods
- Describe how to use a test plan to run automated tests in Microsoft Test Manager
- Describe the settings used when logging bugs
- Describe the benefits of IntelliTrace and demonstrate how to enable and use IntelliTrace in Visual Studio to log test data
- Describe how triage can be used to prioritize a list of bugs
- Describe the steps to verify a bug fix
- Describe the different bug states and how bugs are resolved, closed or re-activated
- Describe how bug status and summary reports are used to provide information on bugs
- Describe software test automation
- Describe and demonstrate how and why you would want to associate an automated test with a test case in Visual Studio
- Describe and demonstrate how to use assertions in code using Visual Studio
- Demonstrate how to add comments to test steps in Microsoft Test Runner
- Describe the types of errors that can occur in tests and how to deal with them
- Define virtual machines and describe how they can be used for efficient testing using different operating systems
- Describe the function of a smoke test
- Describe and implement the steps to generate a BVT in Visual Studio
- Describe how lab environments are managed for testing
- Perform software test management tasks

Software Testing Foundations

Testing Throughout the Software Life Cycle

Software testing ensures that software is reliable and does the job it was designed to do. It is a crucial part of software development. In this course, you will learn about testing and its objectives, and software testing principles and processes. The course also covers the psychology and ethics of testing and testing throughout the software life cycle, including test types for different software models, test levels and test types. This course is one of a series intended to align with the Certified Tester Foundations Level Syllabus. The course is intended to help learners prepare for the Foundation Certificate in Software Testing exam (BH0-010) which is provided by the Information Systems Examination Board (ISEB), a globally recognized testing body providing software testing certification.

- Describe why software testing is necessary
- Describe what software testing involves
- Describe how to meet a test objective and use defects to plan tests effectively
- Describe the first three general principles of software testing relating to the presence of defects, the impossibility of exhaustive testing, and the error of confusing absence of errors with product fit
- Describe the last four applied software testing principles relating to early testing, defect clustering, pesticide paradox and context dependency
- Describe test planning and control activities
- Describe test analysis and design activities
- Describe test implementation and execution activities
- Describe evaluation of exit criteria and test closure activities
- Describe the psychological considerations and levels of independence related to software testing
- Describe the importance of good communication for software testing tasks
- Describe the code of ethics related to software testing
- Describe the types of tests associated with the V-model
- Describe the types of tests associated with iterative-incremental models, including rapid application development (RAD)
- Describe component testing
- Describe integration testing
- Describe system testing
- Describe acceptance testing
- Describe the functional software testing type
- Describe the nonfunctional software testing type
- Describe the structural testing type
- Describe change-based testing, including regression testing
- Describe maintenance testing
- Identify an appropriate software testing strategy

Static, Dynamic, Black-box and White-box Testing

There are many different software testing techniques and it is important to choose the best approach for your project. In this course, you will learn about static techniques, including reviews and static analysis by tools. You will also learn about implementing dynamic testing techniques – identifying test conditions and designing and documenting test cases and procedures. This course also covers the various types of black-box and white-box software testing techniques. Finally, it covers experience-based techniques and the process of choosing a testing technique. This course is one of a series intended to align with the Certified Tester Foundations Level Syllabus. The course is intended to help learners prepare for the Foundation Certificate in Software Testing exam (BH0-010) which is provided by the Information Systems Examination Board (ISEB), a globally-recognized testing body providing software testing certification.

- Distinguish between static and dynamic testing and outline the importance of static techniques for assessing software products
- Describe the activities in a formal review
- Describe the roles and responsibilities associated with formal reviews
- Describe different types of review – walkthroughs, technical reviews and inspections
- Describe the success factors for reviews
- Describe the objective of static analysis in assessing software products
- Outline the steps in the test development process and how these are documented
- Describe the different types of test design techniques for dynamic testing
- Describe the black-box technique equivalence partitioning (EP)
- Describe the black-box technique boundary analysis
- Describe how to use decision tables for black-box test design
- Describe the black-box technique state transition testing
- Describe the black-box technique use case testing
- Describe how white-box techniques can be used to measure test coverage and design tests
- Describe how statement coverage is calculated and test design is based on the results
- Describe how decision coverage is calculated and test design is based on the results
- Describe other structure-based techniques, including various forms of condition coverage
- Describe the experience-based techniques, error-guessing and exploratory testing, and how they work with specification-based techniques
- List the factors involved in choosing a test technique
- Practice identifying an appropriate software testing technique and the considerations for its implementation

Test Planning, Management and Tool Support

Software testing can become very complex, with many layers of testing and testers involved. In this course, you will learn about test organization, planning and management, including risk management and risk-based testing. You will also learn how testing activities can be supported by tools and the considerations for using tools in your organization. This course is one of a series intended to align with the Certified Tester Foundations Level Syllabus. The course is intended to help learners prepare for the Foundation Certificate in Software Testing exam (BH0-010) which is provided by the Information Systems Examination Board (ISEB), a globally-recognized testing body providing software testing certification.

- Describe the benefits of independent testers and the way in which complex tests can be organized to use them
- Describe the activities associated with the test leader and tester
- Describe the activities associated with planning a test for a system and typical entry and exit criteria
- Describe two approaches to estimation of test effort – expert-based and metrics-based
- Describe how the test strategy is implemented with a test approach and describe typical approaches
- Describe how test progress can be monitored and define common metrics
- Describe the activities involved in test reporting, including metrics and documentation used
- Define what test control is and describe examples of test control actions
- Define configuration management and its importance in the context of software testing
- Define project risks in relation to testing as a project activity, and describe types of risk, including organizational factors and technical and supplier issues
- Describe how product risks can be identified to develop risk-based testing
- Define incidents in relation to software testing and describe how they should be managed
- Describe the testing activities that tools can be used to support and their aim
- Describe how tools can be classified according to the activities they support and define intrusive tools
- Describe the uses of test management tools
- Describe how tools are used to support static testing
- Describe how tools are used to support test specification
- Describe how tools are used to support test execution and logging
- Describe how tools are used to support test performance, monitoring and specific testing needs
- Describe the potential benefits and risks associated with testing tools
- Describe the special considerations required for some tool types, including test execution tools, static analysis tools and test management tools
- Describe the considerations related to introduction of testing tools into an organization
- Practice identification of planning and management considerations, and identify considerations for using tools for testing

Exploring Automation

Software Quality Assurance

In this course, you will learn about automated software testing as well as the common software testing frameworks such as Agile and Six Sigma.

- Specify the importance of automated software testing
- Describe the connection between continuous efforts to ensure software quality assurance
- Map DevOps to automated software testing principles
- List the benefits of using application containers for application isolation testing
- Use the Docker command to start an application container
- Describe how some, none, or all of a software solution can exist in a cloud environment
- Create a cloud-based web app
- Apply common frameworks such as Six Sigma and Agile to automated software testing
- Provide examples of common software testing solutions

Automated Software Testing

Ensuring software is as bug-free and secure as possible requires knowledge of detailed testing techniques. In this course, various types of software testing techniques are covered.

- Specify the importance of constant software improvements and testing
- Test software changes to ensure proper functionality
- Unit testing
- Describe how fuzzing tests an application for weaknesses
- Describe how WAFs can increase the security of web applications
- Determine how GUI testing improves software
- Determine how API testing improves software
- Describe how headless browsers are used for web application testing
- Navigate the web page where the headless browser results are sent
- Identify how other services and components relate to software testing
- Describe the meaning of testing what the software does
- Differentiate between load and stress tests
- Use GUI testing software to perform a functional test
- Identify when certain testing techniques should be used

Software Testing with Cucumber

Beginning Cucumber and Behavior-Driven Development

Cucumber and BDD provide several analysis and collaboration techniques to complement Test Driven Development. In this course, you will explore the concepts of Cucumber and BDD, including rules, example mapping and Cucumber installation.

- Define the concepts behind ATDD, automated acceptance tests, and how Behavior-driven Development stems from this concept
- Recognize the fundamental principles of BDD, processes which Cucumber supports and the benefits of using it
- Describe the process of example mapping and understand the distinction between rules and examples
- Demonstrate use cases using the application of rules, examples and example mapping
- Identify how Cucumber works and how it can be used with Ruby code
- List the available versions and the process of installing Cucumber with Ruby on Linux, Mac OSX, and Windows development machines, including HTTP proxy settings and additional gem packages
- Install Cucumber with Ruby on a Mac OS X development machine
- Identify Cucumber features and scenarios by writing feature files
- Use regular expressions to create step definitions in Cucumber scenarios
- Run Cucumber scenarios and show how to use the different formatters
- Recognize how to add and manage assertions in Cucumber scenarios
- Create a scenario that includes both features and steps while exploring concepts in BDD design

Cucumber Steps and Scenario Development

Cucumber has various strategies and implementations that provide users the ability to create expressive scenarios. In this course, you will learn Gherkin and the ability to create expressive scenarios which optimize test outputs.

- Describe the purpose and benefits of Gherkin in test-driven development along with the syntax used
- Recognize the purpose of using features in Gherkin files and how to implement it in a project
- Define steps and step definitions and how it is implemented in an example
- Use features such as capture groups, wildcards and multiple arguments in Cucumber steps
- Describe the available result states for a Cucumber scenario and provide examples of scenarios containing steps that lead to these results
- Describe a background section in a Cucumber feature file to outline steps common to all scenarios
- Use doc strings and data tables for data that does not fit on one line in Cucumber scenarios
- Use scenario outlines to define steps for Cucumber scenarios
- Use nested steps in Cucumber scenarios
- Describe the use of transforms to remove duplication in Cucumber step definitions
- Describe the Cucumber World object and refactor steps into Ruby helper methods that are added to the World
- Describe the uses for directories and tag Cucumber scenarios at feature and scenario level
- Recognize the implementations of steps and step definitions and learn how expressive scenarios work

Deep Dive Cucumber

This course focuses on managing scenarios and behaviors. You will work with various scenarios including databases and web services, REST API and troubleshooting test processes.

- Group Cucumber features in subfolders and run a feature from a subfolder
- Demonstrate how to filter Cucumber scenarios to run a subset using tags or lines
- Use hooks in Cucumber scenarios
- Demonstrate how to modify the default output from Cucumber using formatters such as progress, rerun, usage and stepdef, formatting to file, and use backtrace option
- Demonstrate how to store Cucumber command line options in a YAML file and use the profile option to run them
- Recognize handling of asynchronous systems in Cucumber scenarios
- Recognize concepts including ActiveRecord, refactor a Cucumber scenario to use data, and read and write data to a database
- Use transactions and truncation to clean databases for Cucumber scenarios
- Test REST APIs with Cucumber
- Identify issues in Cucumber scenarios such as flickering scenarios, brittle and slow features and unengaged stakeholders; identify underlying causes and provide solutions
- Recognize how Cucumber and Gherkin can be used with IOS, Android, PHP and many more platforms for tests
- Create Cucumber scenarios for an ATM application

Testing with JUnit

JUnit Fundamentals

JUnit is a framework for writing and running unit tests for Java. This course covers how to get started with JUnit, key members of the API, and how to create a test case and use fixtures.

- Set up JUnit in Eclipse
- Understand how to use the @Test annotation
- Run a JUnit test from Java code
- Understand JUnit assertions
- Understand how to use Assert number equality methods in JUnit tests
- Understand how to use Assert object equality methods in JUnit tests
- Understand how to use Assert array equality methods in JUnit tests
- Understand how to use assertThat methods in JUnit tests
- Understand how to use combined assertThat statements in JUnit tests
- Understand how to use assertThat methods with collections in JUnit tests
- Understand how to use custom assertThat matchers in JUnit tests
- Contrast Assume with Assert and use Assume statements in JUnit tests
- Configure JUnit test cases
- Work with JUnit test fixtures
- Practice modifying Java code to ensure proper testing scenarios

Working with JUnit Tests

There are a number of features in JUnit for enhancing and expanding tests. This course covers how to create test suites and categories, optimize JUnit tests using execution procedures, timeouts and rules, manage test data, use theories and mock objects, run tests with Maven, and implement testing in legacy code.

- Create JUnit test suites
- Use categories in JUnit tests
- Manage failures with @Ignore in JUnit tests
- Configure timeouts in JUnit tests
- Customize JUnit test runs
- Use parameters in JUnit tests
- Manage data across multiple JUnit tests
- Configure execution ordering in JUnit tests
- Enhance JUnit tests with rules
- Use rules to manage JUnit tests
- Work with JUnit theories
- Work with mock objects in JUnit tests
- Run JUnit tests with Maven
- Configure JUnit tests for legacy code
- Configure JUnit test suites for legacy code
- Practice modifying existing Java code to accomplish proper testing scenarios

Testing with JMeter

Performance Testing and JMeter

JMeter is a powerful Apache open source tool used in testing and analysis for application services. This course introduces you to Apache JMeter and performance testing, including installing and configuring the tool to create a basic test.

- Define performance testing, conditions, architecture, and how they are used to test a system
- Describe main testing types including load and stress testing, their differences and when to use them
- Describe the steps in the performance testing cycle and the metrics that are measured in testing systems
- Describe best practices associated with performance testing
- Define JMeter concepts and what it is used for in performance testing
- List the requirements for installing JMeter for Linux, Mac OS and Windows, and the process of installing and configuring JAVA JDK
- Perform the procedure of installing JDK8 on Ubuntu Linux in preparation for JMeter
- Perform the procedure of installing JMeter for the Ubuntu Linux operating system
- Perform the procedure of installing JMeter for the Mac OS X operating system
- Perform the procedure of installing JMeter for the Windows 10 operating system
- Demonstrate configuration of required and optional environment variables, and optimize JMeter performance
- Recognize how to run JMeter in non-GUI mode and the various basic commands and scripts available
- Define JMeter test plan and the process to create your first test plan from a template
- Demonstrate the process of creating a simple test plan
- Recognize JMeter concepts including how to install and use JMeter to create a test plan

JMeter Architecture and Operations

JMeter has several different uses, and in this course you will explore these areas. The course includes basic JMeter operations to handle responses, load analysis, using assertions and controllers, exploring sessions and many more.

- Define the different parts of a test plan and error reporting
- Demonstrate the process of performing a basic load test on a web server in non-GUI mode
- Describe the importance of Thread Groups and when they are used for JMeter tests
- Demonstrate the use of listeners with thread groups in JMeter
- Recognize how to use the Test Script Recorder to record test scripts
- Demonstrate the process of record and playback with a proxy using JMeter scripts
- Demonstrate the use of record and playback test scripts
- Demonstrate the use of record and playback test scripts using the BlazeMeter extension for Chrome
- Demonstrate the use of JMeter timers and the various types
- Describe the various assertions and the commonly used assertions when writing tests
- Define the various controllers and practical use of controllers in designing JMeter scripts
- Describe the config elements used for managing sessions in JMeter with cookies and caches
- Recognize the various operations with JMeter including analysis and test designs

Comprehensive JMeter Operations

JMeter can be used with BeanShell and can integrate with automation tools like Selenium and REST API. In this course, you will explore these concepts and recognize BeanShell scripting and dynamic operations to drive powerful tests.

- Define BeanShell, its main features and when it is used for Java, including the scripting basics and use of variables
- List the steps to download and install BeanShell and how to use the BeanShell elements in JMeter
- Demonstrate simple code using BeanShell scripts and interact with the JMeter API
- Describe the various useful JMeter Regular Expressions and how to use the Regular Expression Extractor
- Demonstrate the process of extracting data from various files including using dynamic data
- Define correlation in JMeter and why it is important in load performance testing
- Demonstrate using correlation with dynamic data in a scenario
- Describe the use of JMeter to test REST APIs
- Demonstrate how to test a REST API using JMeter
- Demonstrate how to configure distributed testing in JMeter
- Describe the process of automating JMeter with Selenium
- Define the various important operations performed with JMeter, including examples of scenarios where the operations would come in handy

Software Testing with TestNG

Exploring TestNG

TestNG is a testing framework inspired by JUnit and NUnit. It supports a wide variety of test categories. This course will introduce you to TestNG, including installing TestNG and installing and configuring Eclipse to support it.

- Define TestNG and describe its purpose in unit testing
- Demonstrate specific features and benefits, including test case writing, of TestNG
- List the requirements for installing TestNG for all available platforms
- List the steps for installing Eclipse on all available platforms and mapping the necessary directories
- Describe the steps of installing the TestNG plugin in Eclipse and verifying that it is successful
- Demonstrate the process of creating a project and a basic test using Eclipse and TestNG
- Recognize the concepts behind using the testng.xml configuration file, including when to use it
- Demonstrate the process of creating a test suite and testng.xml file and executing it
- Demonstrate the process of creating a testng.xml file to run multiple tests
- Describe the use of annotations in TestNG, including order of execution and benefits
- Demonstrate the use of Before and After annotations and execute a Java project that includes various types
- Define TestNG, including concepts of installing and configuring the framework, and create a basic test and test suite

Decoding TestNG in Detail

TestNG has various annotations and methods that can be used to create complex tests. In this course, you will explore these areas and more annotations, including groups and dependencies.

- Describe the use of the @Test annotation in TestNG, including the various important attributes
- Identify the purpose of TestNG assertions and create an example using TestNG Asserts
- Define the use of the parameterization feature in TestNG
- Describe the DataProvider feature in TestNG and what it is used for
- Demonstrate the use of DataProvider by creating a test method including DataProvider
- Recall the dependency of tests and the @Factory annotation
- Use the @Factory annotation to implement tests at runtime
- Describe the concepts of grouping test methods, including grouping tests and regular expressions
- Demonstrate the process of running a TestNG group
- Describe the dependency feature in TestNG, including writing a multiple dependency test and regular expressions
- Demonstrate the process of running a test that depends on or inherits from another artifact
- Describe the various TestNG annotations, including dependencies and groups

Complex TestNG Integrations and Implementations

There are several build tools and automations that can be used to enhance and automate unit testing. In this course, you will explore parallelism, logging and reporting, including integrations with Apache Ant, Maven, Subversion and Hudson.

- Describe the multithreading feature provided by TestNG and how to run tests in multithread mode
- Demonstrate the process of writing a test with the multithread feature
- Define build automation and its benefits, including integrating build automation in TestNG
- Describe the various build tools that can be used in TestNG and their benefits
- Describe the process of installing Ant and using it to run TestNG tests
- List the steps in installing Maven and using it to run TestNG tests
- Describe the process of installing and using SVN, including check in and check out
- Specify the process for installing Hudson and Ant
- Configure Hudson to create a job that runs Ant
- Define the use of reporting in TestNG test execution
- Demonstrate the process of using listeners and reporters in tests
- Demonstrate the process of writing custom reporters and loggers
- Describe parallelism and the various automation and build tools for TestNG, including installation, setup and use with TestNG

Learn more at

www.hpe.com/ww/digitallearner
www.hpe.com/ww/digitallearner-contentpack

Follow us:

CP025 A.00, April 2019

Hewlett Packard Enterprise