Sofware Test Automation

Speaker: Jerry Gao Ph.D.

Computer Engineering Department
San Jose State University

email: jerry.gao@sjsu.edu
URL: http://www.engr.sjsu.edu/gaojerry

Presentation Outline

- *What is Software Test Automation?*
- *Why Is Software Test Automation Important?*
- *Software Test Automation Process*
- *Needs, Issues, and Challenges*
- *Classification of Software Test Tools*
- *Test Automation and Tools for Software*

What is Software Test Automation?

What is Software Test Automation?

Software test automation **refers to the activities and efforts that intend to automate engineering tasks and operations in a software test process using well-defined strategies and systematic solutions.**

The major objectives of software test automation:
- To free engineers from tedious and redundant manual testing operations

- To speed up a software testing process, and to reduce software testing cost and time during a software life cycle

- To increase the quality and effectiveness of a software test process by achieving pre-defined adequate test criteria in a limited schedule

The major key to the success of software automation
--> to reduce manual testing activities and redundant test operations using a systematic solution to achieve a better testing coverage.

What is Software Test Automation?

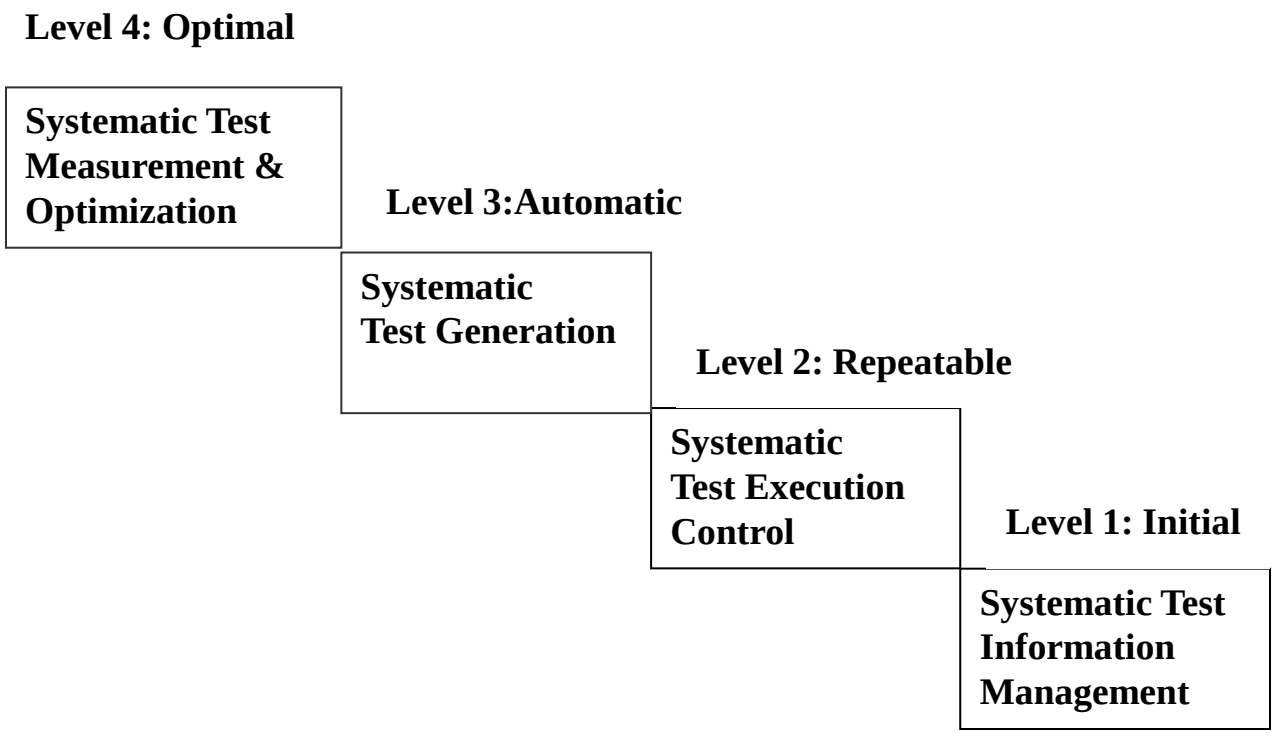
Software test automation activities could be performed in three different scopes:

- Enterprise-oriented test automation, where the major focus of test automation efforts is to automate an enterprise-oriented test process so that it could be used and reused to support different product lines and projects in an organization.

- Product-oriented test automation, where test automation activities are performed to focus on a specific software product line to support its related testing activities.

- Project-oriented test automation, where test automation effort is aimed at a specific project and its test process.

Different Maturity Levels of Software Test Automation



Maturity Level of Software Test Automation

Level 1: Initial
– A software test process at this level provides engineers with systematic solutions and tools to create, update, and manage all types of software test information, including test requirements, test cases, test data, test procedures, test results, test scripts, and problem reports. No systematic solutions and tools are available to support engineers test design, test generation, and test executions.

Level 2: Repeatable
– A software test process at this level not only provides engineers with tools to manage diverse software testing information, but also provides systematic solutions to execute software tests in a systematic manner. These solutions allow engineers to use a systematic approach to run tests and validate test results. However, no systematic solutions and tools are available to assist test engineers in test design, test generation, and test coverage measurement.

Maturity Level of Software Test Automation

Level 3: Automatic

– Besides the test management and test execution tools, a software test process at this level is supported with additional solutions to generate software tests using systematic methods. They could be useful to generate black box or white-box software tests. However, no systematic solutions are available to measure the test coverage of a test process.

Level 4: Optimal

– This is an optimal level of test automation. At this level, systematic solutions are available to manage test information, execute tests, and generate tests, and measure test coverage. The primary benefit of achieving this level is to help engineers understand the current coverage of a test process, and identify the test coverage issues.

Essential Needs of Software Test Automation

- **A dedicated work force for test automation**
- **The commitment from senior managers and engineers**
- **The dedicated budget and project schedule**
- **A well-defined plan and strategy**
- **Talent engineers and cost-effective testing tools**
- **Maintenance of automated software tests and tools**

Basic Issues of Software Test Automation

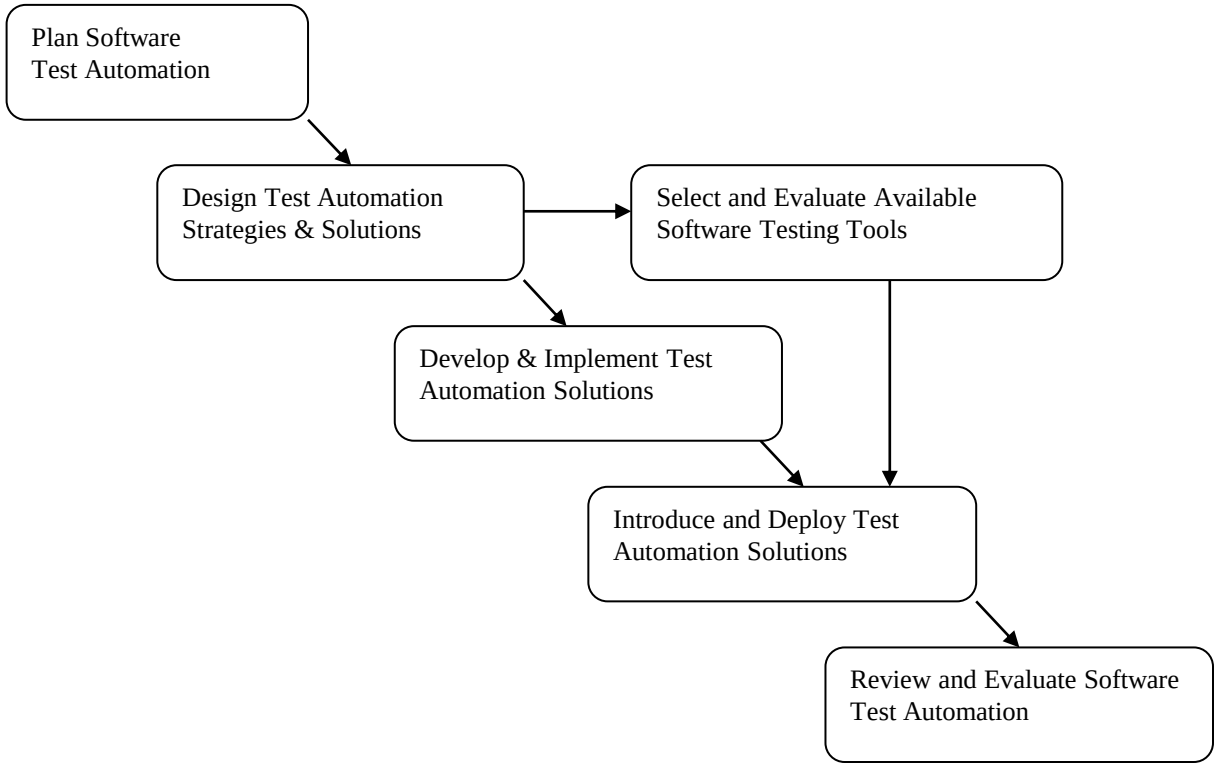
- **Poor manually performed software test process**
- **Late engagement of software test automation in a software product life cycle**
- **Unrealistic goals and unreasonable expectations**
- **Organization issues**
- **Lack of good understanding and experience of software test automation**

Essential Benefits of Software Test Automation

There are a number of essential benefits from test automation. They are listed below.

- ♦Reduce manual software testing operations and eliminate redundant testing efforts.
- ♦Produce more systematic repeatable software tests, and generate more consistent testing results.
- ♦Execute much more software tests and achieve a better testing coverage in a very limited schedule.

A Software Test Automation Process



A Software Test Automation Process

The process consists of the following steps:

Step #1: Test automation planning
– This is the initial step in software test automation. The major task here is to come out a plan that specifies the identified test automation focuses, objectives, strategies, requirements, schedule and budget.

Step #2: Test automation design
– The primary objective of this step is to draw out the detailed test automation solutions to achieve the major objectives and meet the given requirements in a test automation plan.

Step #3: Test tool development
– At this step, the designed test automation solutions are developed and tested as quality tools and facilities. The key in this step is to make sure that the developed tools are reliable and reusable with good documentation.

A Software Test Automation Process

The other steps in a test automation process:

Step #4: Test tool deployment
– Similar to commercial tools, the developed test tools and facilities must be introduced and deployed into a project or onto a product line. At this step, basic user training is essential, and proper user support is necessary.

Step #5: Review and evaluation
– Whenever a new tool is deployed, a review should be conducted to identify its issues and limitations, and evaluate its provided features. The review results will provide valuable feedback to the test automation group for further improvements and enhancements.

Classification of Software Test Tools

Test Tool Types	Basic Descriptions of Different Types of Test Tools
Test Information Management	Systematic solutions and tools support test engineers and quality assurance people to create, update, and maintain diverse test information, including test cases, test scripts, test data, test results, and discovered problems.
Test Execution and Control	Systematic solutions and tools help engineer set up and run tests, and collect and validate test results.
Test Generation	Systematic solutions and tools generate program tests in an automatic way.
Test Coverage Analysis	Systematic solutions and tools analyze the test coverage during a test process based on selected test criteria.
Performance Testing and Measurement	Systematic solutions and tools support program performance testing and performance measurement.
Software Simulators	Programs are developed to simulate the functions and behaviors of external systems, or dependent subsystems/components for a under test program.
Regression Testing	Test tools support the automation performance of regression testing and activities, including test recording and re-playing.

Classification of Software Test Tools

Types of Test Tools	Test Tool Vendors	Test Tools
Problem Management Tools	*Rational Inc.*	ClearQust, ClearDDTS
	Microsoft Corp.	PVCS Tracker
	Imbus AG	Imbus Fehlerdatenbank
Test Information Management Tools	*Rautional Inc.*	TestManager
	Mercury Interactive	TestDirectory
Test Suite Management Tools	*Evalid*	TestSuiter
	Rational Inc.	TestFactory
	SUN	JavaTest, JavaHarness
White-Box Test Tools	*McCabe & Associates*	McCabe IQ2 Junit
	IBM	IBM COBOL Unit Tester IBM ATC - Coverage Assistant - Source Audit Assistant - Distillation Assistant - Unit Test Assistant

Classification of Software Test Tools

Test Execution Tools	***OC Systems***	**Aprob**
	Softbridge	**ATF/TestWright**
	AutoTester	**AutoTester**
	Rational Inc.	**Visual Test**
	SQA	**Robot**
	Mercury Interactive	**WinRunner**
	Sterling Software	**Vision TestPro**
	Compuware	**QARun**
	Seque Software	**SilkTest**
	RSW Software Inc.	**e-Test**
	Cyrano Gmbh	**Cyrano Robot**

Code Coverage Analysis Tools	***Case Consult Corp.***	**Analyzer, Analyzer Java**
	OC Systems	**Aprob**
	IPL Software Product Group	**Cantata/Cantata++**
	ATTOL Testware SA	**Coverage**
	Compuware NuMega	**TruCoverage**
	Software Research	**TestWorks Coverage**
	Rational Inc	**PureCoverage**
	SUN	**JavaScope**
	ParaSoft	**TCA**
	Software Automation Inc	**Panorama**

Classification of Software Test Tools

Load Test and Performance Tools	*Rational Inc.*	Rational Suite PerformanceStudio
	InterNetwork AG	sma@rtTest
	Compuware	QA-Load
	Mercury Interactive	LoadRunner
	RSW Software Inc.	e- Load
	SUN	JavaLoad
	Seque Software	SilkPerformer
	Client/Server Solutions, Inc.	Benchmark Factory
Regression Testing Tools	*IBM*	Regression Testing Tool(ARTT) Distillation Assistant
GUI Record/Replay	*Software Research*	eValid
	Mercury Interactive	Xrunner
	Astra	Astra QuickTest
	AutoTester	AutoTester, AutoTester One

Classification of Software Test Tools

There are three types of test information management systems and tools.

Test information management tool
– It supports engineers to create, update, and manage all types of test information, such as testing requirements, test cases, data, procedures, and results. *Mercury Interactive's TestDirectory* is an example.

Test suite management tool
– It enables engineers to create, update, and manage various software test scripts for test execution. *TestManger* in *eValid* testing tool (www.evalid.com) is a typical example.

Problem management tool
– It helps engineer bookkeeping and manage the discovered problems during a test process.

Classification of Software Test Tools

Test execution tools

– These are programs developed to control and execute program tests and test scripts automatically. They usually consists of the capability to set up the selected test scripts and test data, invoke and execute them, and validate the test results based on the expected testing outputs. *Mercury Interactive's WinRunner* is a typical example.

Test generation tools

– They refer to the programs that generate tests for an under test program using a systematic solution. There are two classes:

- **White-box test generation tools – They generate white-box tests based on program source code and structures using program-based test models and methods, such as the basis path testing technique.**
- **Black box test generation tools – They generate black box tests based on program requirements using black box test methods, such as random testing, boundary value analysis methods.**

Classification of Software Test Tools

Software simulators refer to the programs developed to simulate the functions and behaviors of external software/hardware entities, components, or subsystems. Program simulators are necessary and useful for program integration and system testing.

There are three types of software simulators:

Model-driven program simulators

– They simulate the behaviors of a program process based on a specific model, such as a finite state machine.

Data-driven program simulators

– They simulate the behaviors of a functional program based on the given inputs and return the pre-defined outputs.

Message-driven program simulators

– They simulate the behaviors of a communication process to generate the protocol-oriented outgoing messages, based on the pre-defined incoming messages.

Classification of Software Test Tools

Regression test tools:

Software change analysis – This refers to a systematic facility that identifies various types of software changes and discovers their ripple effects and impacts.

Software test selection – This refers to a systematic facility that assists engineers to select reusable tests for regression testing based on software change information.

Test change analysis – This refers to a systematic solution that identifies the necessary test changes based on the given software changes. The major task of test change analysis is to find reusable tests and identify obsolete tests so that new tests could be added, and existing tests will be updated.

Test recorder and re-player – This refers to a software tool that records the executed tests, and re-plays them for re-testing. *Mercury Interactive's Winrunner* is a typical re-testing tool that is able to record and re-play user-system interactive sequences for window-based programs using pre-defined test scripts.

Test Automation for Software Components

On the road to test automation for software components, we have the following needs:

(A) Systematic Test Information Management Tools for Components

- A Component Test Information Management Tool

- A Component Test Suite Management Tool

- A Component Problem Management Tool

(B) Automatic Test Execution Tool for Components

(C) Automatic Test Generation Tools for Components

(D) Systematic Performance Evaluation Tool for Components

(E) Systematic Regression Test Tool for Components

Challenges in Software Test Automation

Table 12.3. Challenges in Component Test Automation

Areas of Component Test Automation	**Challenge Questions Relating to Component Test Automation**
Component Test Information Management	♦ How to book keeping and manage component-oriented test information in a systematic way to support component evolution?
Component Test Execution and Control	♦ How to construct a plug-in-and-play test bed for software components? ♦ How to generate component test drivers and scripts in a systematic way?
Component Test Generation	♦ How to generate tests systematically for customized software components? ♦ How to generate cost-effective black-box tests for software components?
Component Test Coverage Analysis	♦ How to monitor and analyze component test coverage in a systematic way? ♦ What are the adequate test coverage criteria for components and customized components?
Component Performance Measurement	♦ How to track and measure component performance in a systematic manner? ♦ How to create measurable components? How to create plug-in-and-measure environment for software components?
Building Testable Components	♦ How to increase component testability in a systematic way? ♦ How to create BIT or testable components in a systematic manner?
Component Re-Testing	♦ How to identify component element changes and their impacts on component tests in a systematic manner? ♦ How to select and reuse component tests using a systematic way?

Test Automation for Software Components

Major issues in component test automation:

(A) Ad-hoc test information standards and formats for components.

(B) Most reusable components (such as COTS) are not designed to facilitate software test automation. For example, they do not support component tracking capability for supporting component validation.

(C) Most reusable components (such as COTS) do not provide detailed accessible artifacts about components. It is hard and difficult for test engineers to understand components' behaviors and functions.

(D) It is costly for component users to performance component validation due to the high cost on component test harness.

Test Automation for Software Components

Basic Needs in Automatic Test Execution for Software Components:

(A) Automatic test bed for software components with the following functions:

- Set up component tests (including test suite, test data, and procedures) in a component test bed.
- Execute component tests in a component test bed.
- Collect the test results from test executions and validate them based on the expected results.

(B) A plug-in-and-play unit test environment to support the test execution of different components. This implies that the unit test environment should be able to support different components.

(C) Consistent interaction protocols and interfaces between a component and its test bed, test suites, and a unit test repository.

(D) Systematic solutions to assist engineers to generate component test drivers and stubs based on component API information and available component artifacts, such as a component profile.

(E) New methods to construct testable components that facilitate test executions with standardized component test interfaces and test bed technology.

Test Automation for Software Components

There are three traditional approaches to establishing a component test bed:

•*Script-based approach* – In this approach, a script-based programming language is defined to allow test engineers to configure component test suites by creating, updating, and maintaining test scripts manually. A script compiler and an execution environment are provided to set up component tests using scripting functions, run these test scripts, and validate the test results. ***SUN Microsystem's Java Harness*** is a typical example.

•*Record & replay approach* – The basic idea of this approach is to implement a systematic solution that allows engineers to record the executed tests and results during manual testing, and replay them later for automatic re-testing. This method has been used to develop GUI-based testing tools to perform black-box tests according to system-user interaction scenarios through graphic interfaces. ***Software Research's eValid*** is a typical example.

Test Automation for Software Components

There are three traditional approaches to establishing a component test bed:

•*Component API-based approach* – In this approach, a component test bed is built to interact with components based on component application interfaces. A component test bed usually consists of three parts. The first part plays as a test execution controller that controls test executions of a component. The second part supports component test harness. It assists engineers to create and update component test drivers and stubs manually and/or systematically. The last part works as a test result checker. It collects, monitors, and checks component unit test results. Unlike the first approach, no script languages are used here. Component test drivers and stubs could be generated manually or systematically based on the given component interfaces..

Test Automation for Software Components

Automatic test generation for components refers to generating component tests (such as test cases, test data, and test scripts) using systematic solutions in component testing.

•White-box component test generation

Component tests are created automatically using program-based testing methods to validate component internal structures, logic, data, and functional behaviors. Most existing white box testing methods (such as basis path and state-based testing methods) are still applicable to white box testing of software components.

•Black box component test generation

Component tests are created automatically using specification-based testing methods to validate external visible component functions, behaviors, and interfaces. The essential issue here is how to assist engineers to generate proper test inputs and outputs for each test case. Most existing black box testing methods (such as random test method, equivalence partition, and boundary value analysis) are still applicable to component black box testing.

Test Automation for Software Components

Major difficulties, issues, and challenges in automation test generation:

- Components supporting graphic user interfaces increase the complexity of software test generation. Because each GUI-driven function is performed based on a number of sequential system-user operations involving multiple windows. Each window consists of a number of input and output elements.
- Components involving multimedia inputs/outputs adds an additional complexity factor into the test generation problem.
- Distributed software components and objects bring the distributed communications and interaction issues into component test generation.
- Many existing test tools do not provide test generation features and functions.
- We are lack of systematic solutions and domain specific black box test criteria to support component test generation based on component API specifications.
- Most GUI-based test tools only provide the record-and-replay feature without well-defined test criteria.

Test Automation for Software Components

The primary needs in automatic component test generation.

- **Well-defined black-box test models and adequate testing criteria for software component to enable engineers to develop systematic test generation solutions.**

- **Systematic test generation methods and tools to support test generation of domain-specific application functions.**

- **More white-box test generation methods, criteria, and tools are needed to address the special features of object-oriented software components.**

Test Automation for Software Components

Systematic performance evaluation for software components:

The issues and challenges in automating performance validation and evaluation for components.

- Current components (including COTS components) are developed without consideration how to facilitate component performance validation and measurement. This not only increases the difficulty and complexity in component performance validation and measurement, but also results in a very higher cost.

- We are lack of cost-effective performance testing tools and evaluation solutions that enable component users to validate and measure component performance in a plug-and-measure approach.

- We are lack of well-defined performance measurement frameworks that allow component developers to use them as a simple and seamless connection interface between components and a systematic component performance evaluation solution.

Test Automation for Software Components

To validate and evaluate component performance in systematic way, component test engineers and users have the following needs:

- **Building measurable components that facilitate component performance validation and measurement**

- **Well-defined component performance models and measurement metrics which could be used to implement component performance evaluation solutions.**

- **Systematic component performance validation and evaluation environments and tools that enable engineers and users to measure component performance in a simple plug-in-and-measure approach.**

Test Automation for Software Components

In general, there are four types of engineering activities in component regression testing.

- **Understand and identify various types of software changes for a component, including the changes made in the component specification, interface, design, and program code. It is obvious that component users only need to understand component specification changes and component interface changes.**
- **Understand and analyze component change impacts on component functions, interfaces, structures, and behaviors.**
- **Examine and select the reusable component tests, and form a new component test suite by adding necessary new tests and deleting obsolete tests.**
- **Execute component re-tests and check the test results.**

Test Automation for Software Components

Today, engineers encountered the following issues and challenges on the road to automate component regression testing.

•Although current GUI-based testing tools provide a record-and-replay function to support re-testing of GUI-based software functions, most of them need engineers to manually identify GUI interface changes, manually update and select reusable GUI-based test scripts. Besides, there are no well-defined adequate test criteria for GUI-based software testing and re-testing.

•Existing specification-based test tools do not support component API change analysis and test selection. Hence, engineers must manually identify component API changes and their impacts on component tests, and then, manually update these tests.

Test Automation for Software Components

Existing script-based testing tools do not provide solutions to support engineers to identify and select reusable test scripts in a systematic manner due to the following reasons:

- **There is no direct mapping relationships between component elements and test scripts is tracked.**

- **We are lack of research results on software change identification and impact analysis solutions to address: a) component API changes and their impacts to component tests, and b) internal component changes and their impacts to component APIs .**