Test automation

From Wikipedia, the free encyclopedia

In software testing, **test automation** is the use of special software (separate from the software being tested) to control the execution of tests and the comparison of actual outcomes with predicted outcomes.[1] Test automation can automate some repetitive but necessary tasks in a formalized testing process already in place, or add additional testing that would be difficult to perform manually.

Contents

Overview

Some software testing tasks, such as extensive low-level interface regression testing, can be laborious and time consuming to do manually. In addition, a manual approach might not always be effective in finding certain classes of defects. Test automation offers a possibility to perform these types of testing effectively. Once automated tests have been developed, they can be run quickly and repeatedly. Many times, this can be a cost-effective method for regression testing of software products that have a long maintenance life. Even minor patches over the lifetime of the application can cause existing features to break which were working at an earlier point in time.

There are many approaches to test automation, however below are the general approaches used widely:

- **Code-driven testing**. The public (usually) interfaces to classes, modules or libraries are tested with a

variety of input arguments to validate that the results that are returned are correct.

- **Graphical user interface testing**. A testing framework generates user interface events such as keystrokes and mouse clicks, and observes the changes that result in the user interface, to validate that the observable behavior of the program is correct.
- **API driven testing**. A testing framework that uses a programming interface to the application to validate the behaviour under test. Typically API driven testing bypasses application user interface altogether.

Test automation tools can be expensive, and are usually employed in combination with manual testing. Test automation can be made cost-effective in the long term, especially when used repeatedly in regression testing.

In automated testing the Test Engineer or Software quality assurance person must have software coding ability, since the test cases are written in the form of source code which, when run, produce output according to the assertions that are a part of it.

One way to generate test cases automatically is model-based testing through use of a model of the system for test case generation, but research continues into a variety of alternative methodologies for doing so. In some cases, the model-based approach enables non-technical users to create automated business test cases in plain English so that no programming of any kind is needed in order to configure them for multiple operating systems, browsers, and smart devices.[2]

What to automate, when to automate, or even whether one really needs automation are crucial decisions which the testing (or development) team must make. Selecting the correct features of the product for automation largely determines the success of the automation. Automating unstable features or features that are undergoing changes should be avoided.[3]

Code-driven testing

A growing trend in software development is the use of testing frameworks such as the xUnit frameworks (for example, JUnit and NUnit) that allow the execution of unit tests to determine whether various sections of the code are acting as expected under various circumstances. Test cases describe tests that need to be run on the program to verify that the program runs as expected.

Code driven test automation is a key feature of agile software development, where it is known as test-driven development (TDD). Unit tests are written to define the functionality *before* the code is written. However, these unit tests evolve and are extended as coding progresses, issues are discovered and the code is subjected to refactoring .[4] Only when all the tests for all the demanded features pass is the code considered complete. Proponents argue that it produces software that is both more reliable and less costly than code that is tested by manual exploration. It is considered more reliable because the code coverage is better, and because it is run constantly during development rather than once at the end of a waterfall development cycle. The developer discovers defects immediately upon making a change, when it is least expensive to fix. Finally, code refactoring is safer; transforming the code into a simpler form with less code duplication, but equivalent behavior, is much less likely to introduce new defects.

Graphical User Interface (GUI) testing

Many test automation tools provide record and playback features that allow users to interactively record user actions and replay them back any number of times, comparing actual results to those expected. The advantage of this approach is that it requires little or no software development. This approach can be applied to any application that has a graphical user interface. However, reliance on these features poses major reliability and maintainability problems. Relabelling a button or moving it to another part of the window may require the test to be re-recorded. Record and playback also often adds irrelevant activities or incorrectly records some activities.

A variation on this type of tool is for testing of web sites. Here, the "interface" is the web page. This type of tool also requires little or no software development. However, such a framework utilizes entirely different techniques because it is reading HTML instead of observing window events.

Another variation is scriptless test automation that does not use record and playback, but instead builds a model of the Application Under Test (AUT) and then enables the tester to create test cases by simply editing in test parameters and conditions. This requires no scripting skills, but has all the power and flexibility of a scripted approach. Test-case maintenance seems to be easy, as there is no code to maintain and as the AUT changes the software objects can simply be re-learned or added. It can be applied to any GUI-based software application. The problem is the model of the AUT is actually implemented using test scripts, which have to be constantly maintained whenever there's change to the AUT.

API driven testing

API driven testing is also being widely used by software testers as it's becoming tricky to create and maintain GUI-based automation testing.

Programmers or testers write scripts using a programming or scripting language that calls interface exposed by the application under test. These interfaces are custom built or commonly available interfaces like COM, HTTP, command line interface. The test scripts created are executed using an automation framework or a programming language to compare test results with expected behaviour of the application.

What to test

Testing tools can help automate tasks such as product installation, test data creation, GUI interaction, problem detection (consider parsing or polling agents equipped with oracles), defect logging, etc., without necessarily automating tests in an end-to-end fashion.

One must keep satisfying popular requirements when thinking of test automation:

- Platform and OS independence
- Data driven capability (Input Data, Output Data, Metadata)
- Customizable Reporting (DB Data Base Access, Crystal Reports)
- Easy debugging and logging
- Version control friendly – minimal binary files

- Extensible & Customizable (Open APIs to be able to integrate with other tools)
- Common Driver (For example, in the Java development ecosystem, that means Ant or Maven and the popular IDEs). This enables tests to integrate with the developers' workflows.
- Support unattended test runs for integration with build processes and batch runs. Continuous integration servers require this.
- Email Notifications like bounce messages
- Support distributed execution environment (distributed test bed)
- Distributed application support (distributed SUT)

Framework approach in automation

A test automation framework is an integrated system that sets the rules of automation of a specific product. This system integrates the function libraries, test data sources, object details and various reusable modules. These components act as small building blocks which need to be assembled to represent a business process. The framework provides the basis of test automation and simplifies the automation effort.

The main advantage of a framework of assumptions, concepts and tools that provide support for automated software testing is the low cost for maintenance. If there is change to any test case then only the test case file needs to be updated and the driver Script and startup script will remain the same. Ideally, there is no need to update the scripts in case of changes to the application.

Choosing the right framework/scripting technique helps in maintaining lower costs. The costs associated with test scripting are due to development and maintenance efforts. The approach of scripting used during test automation has effect on costs.

Various framework/scripting techniques are generally used:

1. Linear (procedural code, possibly generated by tools like those that use record and playback)
2. Structured (uses control structures - typically ‘if-else’, ‘switch’, ‘for’, ‘while’ conditions/ statements)
3. Data-driven (data is persisted outside of tests in a database, spreadsheet, or other mechanism)
4. Keyword-driven
5. Hybrid (two or more of the patterns above are used)
6. Agile automation framework

The Testing framework is responsible for:[5]

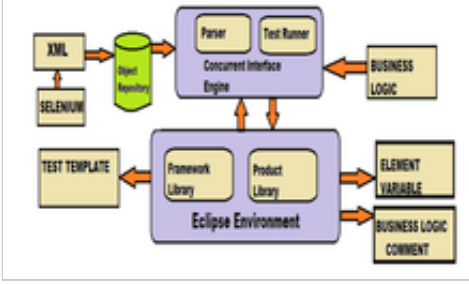
1. defining the format in which to express expectations
2. creating a mechanism to hook into or drive the application under test
3. executing the tests
4. reporting results

Test automation interface

Test automation interface are platforms that provide a single workspace for incorporating multiple testing tools and frameworks for System/Integration testing of application under test. The goal of Test Automation Interface is to simplify the process of mapping tests to business criteria without coding coming in the way of the process. Test automation interface are expected to improve the efficiency and flexibility of maintaining test scripts.[6]

Test Automation Interface consists of the following core modules:

- Interface Engine
- Interface Environment
- Object Repository



Test Automation Interface Model

Interface engine

Interface engines are built on top of Interface Environment.
Interface engine consists of a parser and a test runner. The parser is present to parse the object files coming from the object repository into the test specific scripting language. The test runner executes the test scripts using a test harness.[6]

Interface environment

Interface environment consists of Product/Project Library and Framework Library. Framework Library have modules related with the overall test suite while the Product/Project Library have modules specific to the application under test.[6]

Object repository

Object repositories are a collection of UI/Application object data recorded by the testing tool while exploring the application under test.[6]

Defining boundaries between automation framework and a testing tool

Tools are specifically designed to target some particular test environment, such as Windows and web automation tools, etc. Tools serve as a driving agent for an automation process. However, an automation framework is not a tool to perform a specific task, but rather an infrastructure that provides the solution where different tools can do their job in a unified manner. This provides a common platform for the automation engineer.

There are various types of frameworks. They are categorized on the basis of the automation component they leverage. These are:

1. Data-driven testing
2. Modularity-driven testing

3. Keyword-driven testing
4. Hybrid testing
5. Model-based testing
6. Code driven testing
7. Behavior driven testing

See also

- Software Testing portal
- List of GUI testing tools
- List of web testing tools
- Software testing
- System testing
- Unit test

References

1. Kolawa, Adam; Huizinga, Dorota (2007). *Automated Defect Prevention: Best Practices in Software Management*. Wiley-IEEE Computer Society Press. p. 74. ISBN 0-470-04212-5.
2. "Proceedings from the 5th International Conference on Software Testing and Validation (ICST). Software Competence Center Hagenberg. "Test Design: Lessons Learned and Practical Implications." (http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4578383).
3. Brian Marick. "When Should a Test Be Automated?" (http://www.stickyminds.com/sitewide.asp?Function=edetail&ObjectType=ART&ObjectId=2010). StickyMinds.com. Retrieved 2009-08-20.
4. *Learning Test-Driven Development by Counting Lines*; Bas Vodde & Lasse Koskela; IEEE Software Vol. 24, Issue 3, 2007
5. "Selenium Meet-Up 4/20/2010 Elisabeth Hendrickson on Robot Framework 1of2" (http://www.youtube.com/watch?v=qf2i-xQ3LoY). Retrieved 2010-09-26.
6. "Conquest: Interface for Test Automation Design" (http://www.qualitycow.com/Docs/ConquestInterface.pdf). Retrieved 2011-12-11.

- Elfriede Dustin et al. (1999). *Automated Software Testing*. Addison Wesley. ISBN 0-201-43287-0.
- Elfriede Dustin et al. *Implementing Automated Software Testing*. Addison Wesley. ISBN 978-0-321-58051-1.
- Mark Fewster & Dorothy Graham (1999). *Software Test Automation*. ACM Press/Addison-Wesley. ISBN 978-0-201-33140-0.
- Roman Savenkov: *How to Become a Software Tester*. Roman Savenkov Consulting, 2008, ISBN 978-0-615-23372-7
- Hong Zhu et al. (2008). *AST '08: Proceedings of the 3rd International Workshop on Automation of Software Test* (http://portal.acm.org/citation.cfm?id=1370042#). ACM Press. ISBN 978-1-60558-030-2.
- Mosley, Daniel J.; Posey, Bruce. *Just Enough Software Test Automation* (http://www.amazon.com/Just-Enough-

Software-Test-Automation/dp/0130084689/ref=sr_1_5?s=books&ie=UTF8&qid=1337627825&sr=1-5). ISBN 0130084689.
- Hayes, Linda G., "Automated Testing Handbook", Software Testing Institute, 2nd Edition, March 2004
- Kaner, Cem, "Architectures of Test Automation (http://www.kaner.com/pdfs/testarch.pdf)", August 2000

External links

- Practical Experience in Automated Testing (http://www.methodsandtools.com/archive/archive.php?id=33)
- Test Automation: Delivering Business Value (http://www.applabs.com/internal/app_whitepaper_test_automation_delivering_business_value_1v00.pdf)
- Test Automation Snake Oil (http://www.satisfice.com/articles/test_automation_snake_oil.pdf) by James Bach
- When Should a Test Be Automated? (http://www.stickyminds.com/r.asp?F=DART_2010) by Brian Marick
- Guidelines for Test Automation framework (http://web.archive.org/web/20110707113430/http://info.allianceglobalservices.com/Portals/30827/docs/test%20automation%20framework%20and%20guidelines.pdf)
- Advanced Test Automation (http://www.testars.com/docs/5GTA.pdf)

Categories: Software testing | Automation | Graphical user interface testing

---