Test Automation: Why Context Matters

Jonathan Kohl
jonathan@kohl.ca
www.kohl.ca/blog

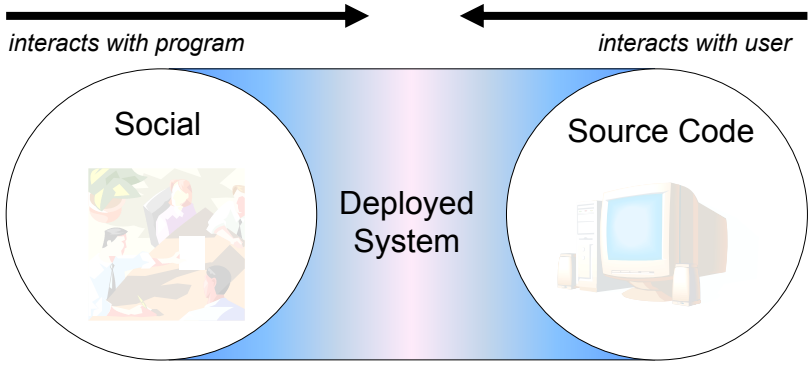
::kohl concepts

Test Automation Contexts

- What is a context in testing?
  - Broad contexts provide some perspective on test automation
- Why does it matter?
  - Test automation is about managing trade-offs—The more we know about the context we are working in, the more informed we are when developing an automation strategy
  - We need to be aware of what we're putting our faith in when we use one testing method over another

::kohl concepts

Contexts Overlap in a System



::kohl concepts

Who is the user?

If the user is a:

- **Business user**—the *social* context in which the software is used is important
- **Machine**—the context in which the software is used is the deployed *system environment*
- **Programmer**—the context used is the *source code*

::kohl concepts

Social Context

- When thinking about the social context, think of what the people need, not necessarily what the software does
- People are not typically motivated by software, they are motivated by a perceived need
- In a workplace, people are motivated by social interactions, which are influenced by business goals

::kohl concepts

Social Context - Characteristics

- **Ambiguity**—The language that business rules are expressed in can be ambiguous
- **Unpredictable behaviour**—Humans are cognitive beings that react individually to stimulus or respond to environments differently
- **Visual orientation**—Most testing is “black-box” testing through the user interface
- **Testers—**Testing in this context is typically done by QA, software testers or domain experts

::kohl concepts

Social Context – Testing Goals

- Test in a deployed system
- Emulate user interactions
- Simulate production conditions in a realistic environment

::kohl concepts

Social Context - Automation

Challenges

- Not all tests can be run, let alone automated (see Cem Kaner's "*The Impossibility of Complete Testing*")
- There are some tasks a human is better suited to than a computer and vice-versa
- Visual aspects are difficult to test by a machine
  - Humans are better at pattern matching
  - Machines can't investigate strange behaviour
- Many important bugs are found because a tester has a hunch; computers can't do that because they aren't intelligent

::kohl concepts

Social Context - Automation

Challenges (continued):

- Tests are more complex at the user interface and are often developed by testers with less programming expertise
- The user interfaces can change frequently, which increases maintenance
- Tests that attempt to deal with the user are actually an emulation of user actions: automated tests only run in a system context, not in the social context

::kohl concepts

Social Context - Automation

Possible trade-offs:

- In some cases, a computer excels at testing, in others, a human does a better job
- Time vs. investigative testing—Manual testing is time consuming, but can be richer and more fruitful; while machines can run tests very quickly, they can't see suspicious behavior and investigate it
- Using a tool vs. replacing testers with tools
  - Cem Kaner more accurately describes automation using the term "Computer Assisted Testing" rather than test automation
  - Test automation does not mean we have intelligent robots doing our work for us

::kohl concepts

Code Context

- Test-Driven Development has popularized testing in the code context
- Developers are becoming “test-infected”
- Developers now have many automated unit testing automation frameworks to use
- Many are pushing the boundaries and moving into the system and social contexts

::kohl concepts

Code Context - Characteristics

- **Predictability**—Computers can repeat the identical steps over and over, without distractions or variation
- **Certainty**—Actions are consistent rather than influenced by changing motivations and business rules
- **Proximity to the source code**—This is where the source code lives; testing closer to the source can help us find problems faster
- **Isolation**—Tests can be run to high degrees of isolation
- **Technical Expertise**—Developers have programming expertise, and write tests in the language of the application
- **Low-level view**—Most testing is “white-box” testing

::kohl concepts

Code Context - Challenges

- Not testing in the user environment means that program usability can be overlooked
- Over-reliance on testing in this context can mask problems in a user or other exposed interface
- Not testing in a system context can mean that unanticipated problems occur when the software is deployed due to differences between the development and system environments
- Development of a large number of tests over time can stifle design decisions and slow builds down

::kohl concepts

System Context

- The finished software is deployed here
- Automation relies on testable interfaces that hook into a deployed system
- Sometimes called “Grey-box” testing because it doesn’t rely on the user interface, but it is a step away from the source code
- Tests deployed software without using a GUI (ex. load testing and other tools use a protocol as a testable interface)
- Tests devices whose user is another machine (ex. embedded devices, drivers, adapters)

::kohl concepts

System Context

- You can use a testable interface that is more stable than the GUI for automation. Since test cases are less fragile, this can be a good complement to manual testing
- You can facilitate simulation of conditions that the software might encounter in the field
- Tests run faster in the system context than in a visual environment, which works well for creating test data

::kohl concepts

System Context - Challenges

- Moving from doing unit tests in a code context to testing in a system context can be challenging:
  - The number of possible tests can increase
  - More time may be required to setup, build and deploy software and run the tests
- Often requires a special testing interface that also needs to be maintained

::kohl concepts

System Context - Challenges

- Can require different skills than testing in the social or code context does:
  - A lack of a GUI makes it a challenge for conventional testers but does allow for component isolation
  - Testing in a deployed system is slower, and takes more time and work to set up, but it is closer to the live system where the software will be used

::kohl concepts

System Context - Challenges

- Moving from a social context to the less visual system context can also be challenging
- Instead of thinking of human users, testers need to think more like a machine, requiring:
  - A higher technical skills threshold
  - Different testing techniques

::kohl concepts

Comparing Contexts - Goals

- Test automation in a social context often seeks to emulate a user's actions
  - Example: Automating regression tests to free up tester resources
  - *What can you potentially lose when you focus on test automation in this way?*
- Test automation in a code context often seeks to address design considerations and "change detection"
  - Tests drive the design of the software (in Test-Driven Development)
  - The resulting suite of automated tests can help developers when refactoring code

Comparing Contexts - Goals

- Test automation in a system context can seek to find a balance between user interaction and the behaviour of the code
  - Code might behave differently in a deployed environment because code influenced by other machines or components within a system can behave in unpredictable ways
- When the user is a machine, the testable interface often occurs in a system context

Context Confusion

- Terminology can be understood differently depending on the context, causing confusion when people from different contexts work together
- While the terminology and expressions are similar, they may have different implications in different contexts

::kohl concepts

Context Confusion

- In TDD circles, you often hear the expressions: *"100% automation"* and *"Automate all tests"*
  - In a code context, this means that during development, you automate all tests. An automated test suite is a by-product of development.
  - In the code context, this is useful, and necessary. However, this doesn't always make sense in a social context, where there is an almost infinite number of possible tests. It is impossible to run, let alone automate all possible tests. Furthermore, some tests are better done by a human.

::kohl concepts

Context Confusion

- Social Context automation experts say, “Don’t automate something you don’t understand”
  - Automated tests against a GUI usually require significant effort
  - Tests can be complex and require detailed design considerations
  - It is easy to end up with a mess of un-maintainable code
- TDD developers say, “If I don’t understand the code, I write a test”
  - Writing a test is a great way to figure out how the code works.
  - Tests are often small:
    Example: `assert_equal("1000", some_method(a, b))`
- Even though these statements sound contradictory, they are both correct. *In their own context.*

::kohl concepts

Context Trade-Offs

- Code context:
  - Developers understand that testing at the GUI is very difficult
  - They often say, “Make the GUI so thin we don’t have to test it”
  - This can work if you design projects like this, but what if your project wasn’t designed this way?
- Social context:
  - Testers are often in situations where automation isn’t being done in the source code
  - They try to do the best they can automating through the GUI, but often end up with tests that don’t have component isolation
- *What would happen if testers and developers worked together?*
  - See Mike Kelly’s article “How to Win Friends and Automate Testing” in the May 2005 issue of *Better Software* magazine

::kohl concepts

Dealing with Trade-Offs

- As tests move more towards the user and get more complex, look at how testers from the social context manage complexity
  - Use heuristics (*rules of thumb*, such as equivalence partitioning) to deal with the massive amount of possible tests. (See writings by James Bach, Cem Kaner and Michael Bolton for more on heuristics.)

Dealing with Trade-Offs

- As tests move more towards the code, find out how isolation, design decisions and change detection can help with test automation projects
  - Use developer skills to help with test design and development

Dealing with Trade-Offs

- There is often an inverse relationship between the context and the size and complexity of a test
- Moving toward the social context, tests are generally larger and more complex
- However, if a test case is any larger than a few lines of code, it should make you nervous. Why?
  - Tests should be as simple as possible. The more complex the test, the greater the chance of having bugs in the test code.
  - There's nothing worse than working with a buggy test automation system.

Moving Forward

How to identify contexts:

- Create a model of the system under test
- Identify your users
- Identify trade-offs when thinking about automation:
  - *What do I gain from automating this test?*
  - *What might I lose by automating this test?*
- Choose when to automate
- Know why you automate

Set Goals and Measure

- Once you create a model of your system, set goals for your automation efforts
- Periodically check to see if you've met the goals, and adjust accordingly
- Make goals product-centric
- Be careful to not measure success measuring:
  - Number of tests automated
  - Lines of code in automation suite
  - Purchasing automation software
- Without setting and measuring the right goals, it's easy to put in a lot of effort and see few results.
- The right kinds goals are aligned with the big picture. Your *product* should see improvements.

::kohl concepts

Resources – General Testing

- The Impossibility of Complete Testing, Cem Kaner
  www.kaner.com/pdfs/impossible.pdf
- Heuristic Risk-Based Testing, James Bach
  www.satisfice.com/articles/hrbt.pdf

::kohl concepts

Resources – Test Automation

- Kaner, Cem. *GUI Regression Automation.* www.kaner.com/pdfs/gui_regression_automation.pdf
- Bach, James. *Test Automation Snake Oil.*
- http://www.satisfice.com/articles/test_automation_snake_oil.pdf
- Pettichord, Bret. *Homebrew Test Automation.* www.io.com/%7Ewazmo/papers/homebrew_test_automation_200409.pdf
- Pettichord, Bret. *Deconstructing GUI Test Automation.* www.io.com/%7Ewazmo/papers/deconstructing_gui_test_automation.pdf
- Pettichord, Bret. *Design For Testability* www.io.com/%7Ewazmo/papers/design_for_testability_PNSQC.pdf
- Hunt and Thomas. *Pragmatic Unit Testing.* www.pragmaticprogrammer.com/starter_kit/ut/index.html

::kohl concepts

Resources - TDD

- www.testdriven.com
- Beck, Kent. *Test Driven Development: By Example.* Addison Wesley, 2002.
- William Wake's site: http://xp123.com/xplor/#Programming%20-%20Test-First

::kohl concepts