GUI Regression Automation

Cem Kaner J.D., Ph.D.

 www.kaner.com. 408-244-7000

Background Notes

In the mass-market software world, most efforts to automate testing that I've seen have been expensive failures. Paradigms that predominate the Information Technology and DoD-driven worlds don't apply well to the mass-market paradigm. Our risks are different. Solutions that are highly efficient for those environments are not at all necessarily good for mass-market products.

The point of an automated testing strategy is to save time and money. Or to find more bugs or harder-to-find bugs. The strategy works if it makes you more effective (you find better and more-reproducible bugs) and more efficient (you find the bugs faster and cheaper).

I wrote a lot of test automation code back in the late 1970's and early 1980's. (That's why WordStar hired me as its Testing Technology Team Leader when I came to California in 1983.) Since 1988, I haven't written a line of code. I've been managing other people, and then consulting, seeing talented folks succeed or fail when confronted with similar problems. I'm not an expert in automated test implementation, but I have strengths in testing project management, and how that applies to test automation projects. That level of analysis--the manager's view of a technology and its risks/benefits as an investment--is the point of this talk.

About Cem Kaner

I teach, consult to software companies, and practice law within the software community. I work to improve customer satisfaction and safety *and* corporate profitability.

As a software developer, I've programmed, done UI design, managed software development projects, software test groups and documentation groups, sold software at Egghead, and consulted to companies to build or rebuild project teams or small departments. I've written the best selling book on software testing and am currently writing two others.

As an attorney, I typically represent individuals and small businesses, whether they are developers or customers. Most clients are involved with software or with writing. I draft and negotiate contracts and advise clients on their rights after a breach of contract or a failure of a relationship. I also do extensive legislative work, participating in multi-year projects that are drafting laws to govern software contracting and software-related aspects of electronic commerce. I've also served as a Deputy District Attorney (full-time *pro bono* assignment) and as an investigator/mediator for a consumer protection agency.

I hold a Ph.D. in Experimental Psychology, with a primary interest in Human Factors (how to redesign systems and machines so that they work better for people.) I hold a B.A. in Arts & Sciences (Math, Philosophy), and a J.D. (law degree). I'm Certified by the American Society for Quality Control in Quality Engineering.

Overview

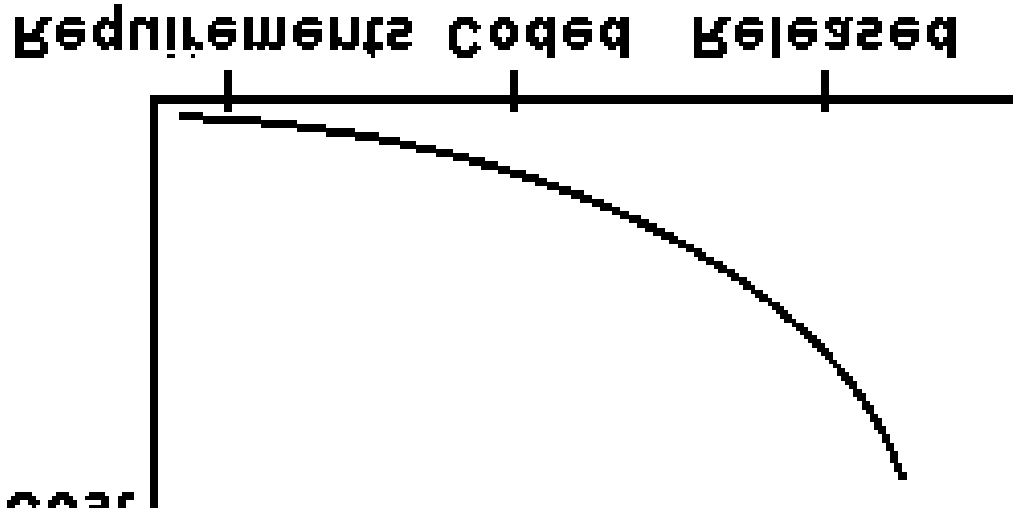
- The regression test paradigm
- Problems with the paradigm
- Typical practices that result in failure.
- Short term gains are possible
- Automation is software development
- Architectural approaches to GUI automation
- LAWST discussions
- Breaking away from the regression paradigm

Regression Test Paradigm

- Create a test case.
- Run it and inspect the output
- If program fails, report bug and try later.
- If program passes, save the resulting outputs.
- In future tests run the program and compare the output to the saved results. Report an exception when the current output and the saved output don’t match.

A Few Problems

- **Test case creation is expensive.**
- **Your most technically skilled staff are tied up in automation**
- **Automation can delay testing, adding even more cost (albeit hidden cost.)**



A Few Problems

- **You are executing weak tests.**
  - » How many bugs do you find with a test that the program has already passed? (IF you do extensive automation, 6-25%)
  - » How much work does it take to do this extensive a level of automation? (50%?)
- But the test case development finds lots of bugs, doesn't it?
  - » Cadence's data supports this, but indirectly illustrates the power of manual testing.

A Few Problems

- **Many groups automate only the easy tests**
  - » often suggested strategy
  - » result is expensive investment in weak testing of superficial issues
- **Management misunderstands the depth of testing**
  - » Is a 5000 test suite big or small?
- **Maintenance can be hugely expensive.**
  - » Recode hundreds of tests to catch up with one coding change
  - » Yet another half-baked programming language with mediocre development tools

A Few Problems

- **What do you have for your next release?**
  - » What *is* your coverage? Do you know what tests you aren't running or what areas of the program aren't covered?
  - » Can you read the code well enough to maintain it?
  - » What weaknesses are there in the scripts?
  - » How do you detect a failure?
  - » ***Will management allow you enough time to test, given that the automation "should" do the testing?***

Typical Practices that Often Result in Failure

- Capture replay
- Script of individual tests
- Part time automation
- “100% automation”
- Automation of easy tests
- No documentation

Short Term Gains are Possible

- Printer compatibility testing
  - » Modem compatibility seems equally obvious.
  - » How do we do video compatibility testing or tests of other devices that require human appraisal?
- Stress testing
  - » some tests can only be done by machines (simulate 100,000 users).
- Performance benchmarking.
- ***Smoke testing.***

Interesting Approaches: Data Driven Architecture

Table example:

- Picture
- Caption
  - » typeface
  - » size
  - » style
  - » placement

Note with this example:

- we never run tests twice
- we automate execution, not evaluation
- we save SOME time
- we focus the tester on design and results, not execution.

Interesting Approaches: Frameworks

Frameworks are code libraries that separate routine calls from designed tests.

- modularity
- reuse of components
- partial salvation from the custom control problem
- independence of application (the test case) from user interface details (execute using keyboard? Mouse? API?)

LAWST

Rather than talking about how we were having problems, could we build a process that captures experience across labs?

- facilitated meeting
- war stories
- discussion
- principles and facts
- votes and arguments

LAWST

Pattern of evolution

- capture-replay (disaster)
- individual cases (disaster)
- complex frameworks (disaster)
- data driven or simpler framework (may be stable)

LAWST

Reset management expectations

- it takes time
- you need people
- benefits are for next release

==========

Should you automate?

- A one-release product?
- A first-release product with a rapidly changing UI?
- A multi-platform product?

LAWST

Localization was far less successful than we expected.

- I think that there are some successes out there, though.
- Think about functionality vs. content. What functionality risks are there? Are they worth adding the localizability complexity needed to make the test cases themselves localizable?

LAWST

Automation is Software Development

- big code base, with features (each test is a feature)
- we would never tolerate design of other software as 50,000 standalone features
- requirements, architecture, standards, documentation, discipline

LAWST

Data-Driven Architecture

- The program's variables are data
- The program's commands are data
- The program's UI is data
- The program's state is data

LAWST

Frameworks

- Define every feature of the application under test
  - » custom controls and kludges
- commands / features of tool
- Small, often reused tasks
- Large, complex chunks
- Utility functions
  - » standardized logger, may not need it.

LAWST

Framework risks

- over-ambitious
- poor communication means non-use
- some products don’t call for this type of investment

OTHER THOUGHTS

Other types of automation and other goals of automation

WHY FOCUS ON REGRESSION?

What Do We Want From Automation?

- Save time and money while you find bugs.
- Automate the test the first time you run it.
- Run scrillions of tests.
- Make it easy to figure out what has and what has not been tested.
- Handle multi-dimensional (multi-variable) issues.
- What else?

Examples of Non-Regression Uses of the GUI Regression Tools

Test execution.

- Exploratory testing
  - » partial automation
  - » full automation with an oracle
  - » (clean room)
- Function equivalence testing
- State transition testing
- Event-log driven testing (I think this is AKA monkeys).

The biggest challenge is the oracle.

Other Things to Automate

Automated testing tools provide special capabilities:

- Analyzing the code for bugs
- Designing test cases
- Automatically creating test cases
- Relatively easy manual creation of test cases
- Executing the tests
- Validating the test results

Many tools offer extra doo-dads such as integration with bug tracking or source control, project management, etc.

No tool offers all these capabilities.

Non-Regression

At LAWST, each of us found that we had our most successful experiences in automation in joint projects with the testing and programming staff.

Don’t get locked into the paradigm, but use the tool if it is useful.

