Software Test Automation: A Real-World Problem

Cem Kaner, Ph.D., J.D.

This Talk

The most widely used class of automated testing tools leads senior software testers into software development blunders that a first year programming student shouldn't make.

We can learn (or relearn) interesting basic lessons about software development from the problems that accompany these tools.

Testing is an Important Task

"Failed ERP Gamble Haunts Hershey"

Computerworld, November 1, 1999.

"A $112 million ERP project has blown up in the face of Hershey Foods."

"Last week, when Hershey announced a 19% drop in third-quarter profits, CEO Kenneth Wolfe said the system fixes are taking longer than expected and are requiring more extensive changes."

"They've missed Halloween, they're probably going to miss Christmas, and they might even start missing Easter."

"The company recently . . . developed a list of changes . . . Wolfe said,

'But they need to be tested

before we can put them in,

and we can't get that done.' "

Testing is a Huge Task

- Complete testing is impossible
- Testing eats ½ or more of the development budget of several projects.
- On new releases of some mature mass-market products, the software publisher is spending 4 weeks on testing and fixing for every week of new enhancements (new features, UI redesign, etc.)
- Companies that develop application software typically have specialized software testing groups. These testers typically do manual testing (relatively little automation), so their work is labor-intense and slow.
- Automation is desirable, but carries serious risks of its own.

The GUI Regression Test Paradigm

This is the most common style of automated testing.

- 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 the future:
    - » 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.

Most commonly, we run these tests on the finished program, testing underlying functions by issuing commands through the program’s GUI. Therefore, these are typically called GUI-level test tools.

Common Problems: Bad Design & Programming Practices

Worst case (very common)

- The Capture / Replay tool:
  - » Tests can be created quickly and easily by non-programmers.

Results:

- Embedded constants
- No modularity
- No source control
- No documentation
- No requirements analysis
- No wonder we fail.

(We <u>are</u> writing applications. Windows NT 4 had 6 million lines of code, and 12 million lines of test code.)

Common Problems

Various problems such as:

- Underestimated cost
- Your most technically skilled staff are tied up in automation

But especially,

Low power of regression testing

And,

Low attention to maintainability.

19 Common Mistakes

- **Don’t underestimate the cost of automation.**
- **Don’t underestimate the need for staff training.**
- **Don’t expect to be more productive over the short term.**
- **Don’t spend so much time and effort on regression testing.**
- **Don’t use instability of the code as an excuse.**
- **Don’t put off finding bugs in order to write test cases.**
- **Don’t write simplistic test cases.**
- **Don’t shoot for “100% automation.”**
- **Don’t use capture/replay to create tests.**
- **Don’t write isolated scripts in your spare time.**
- **Don’t create test scripts that won’t be easy to maintain over the long term.**
- **Don’t make the code machine-specific.**
- **Don’t fail to treat this as a genuine programming project.**
- **Don’t “forget” to document your work.**
- **Don’t deal unthinkingly with ancestral code.**
- **Don’t give the high-skill work to outsiders.**
- **Don’t insist that all of your testers be programmers.**
- **Don’t put up with bugs and crappy support for the test tool.**
- **Don’t forget to clear up the fantasies that have been spoonfed to your management.**

Requirements Analysis

What Are the Requirements For a Successful Test Automation?”

- At two meetings of the Los Altos Workshops on Software Testing, we pooled our requirements-oriented questions resulting in the following (next slide) list of 27 interesting questions.
- Requirements: circumstances or preferences that will drive the design.
- HERE’S ONE KEY EXAMPLE:
  - » Will the user interface of the application be stable or not?

27 Requirements Questions

- **Will the user interface of the application be stable or not?**
- **To what extent are oracles available?**
- **To what extent are you looking for delayed-fuse bugs (memory leaks, wild pointers, etc.)?**
- **Does your management expect to recover its investment in automation within a certain period of time? How long is that period and how easily can you influence these expectations?**
- **Are you testing your own company's code or the code of a client? Does the client want (is the client willing to pay for) reusable test cases or will it be satisfied with bug reports and status reports?**
- **Do you expect this product to sell through multiple versions?**
- **Do you anticipate that the product will be stable when released, or do you expect to have to test Release N.01, N.02, N.03 and other bug fix releases on an urgent basis after shipment?**
- **Do you anticipate that the product will be translated to other languages? Will it be recompiled or relinked after translation (do you need to do a full test of the program after translation)? How many translations and localizations?**
- **Does your company make several products that can be tested in similar ways? Is there an opportunity for amortizing the cost of tool development across several projects?**
- **How varied are the configurations (combinations of operating system version, hardware, and drivers) in your market? (To what extent do you need to test compatibility with them?)**

27 Requirements Questions

- What level of source control has been applied to the code under test? To what extent can old, defective code accidentally come back into a build?.How frequently do you receive new builds of the software?
- Are new builds well tested (integration tests) by the developers before they get to the tester?
- To what extent have the programming staff used custom controls?
- How likely is it that the next version of your testing tool will have changes in its command syntax and command set?
- What are the logging/reporting capabilities of your tool? Do you have to build these in?
- To what extent does the tool make it easy for you to recover from errors (in the product under test), prepare the product for further testing, and re-synchronize the product and the test (get them operating at the same state in the same program).
- (In general, what kind of functionality will you have to add to the tool to make it usable?)
- Is the quality of your product driven primarily by regulatory or liability considerations or by market forces (competition)?
- Is your company subject to a legal requirement that test cases be demonstrable?
- Will you have to be able to trace test cases back to customer requirements and to show that each requirement has associated test cases?

27 Requirements Questions

- **Is your company subject to audits or inspections by organizations that prefer to see extensive regression testing?**
- **If you are doing custom programming, is there a contract that specifies the acceptance tests? Can you automate these and use them as regression tests?**
- **What are the skills of your current staff?**
- **Do you have to make it possible for non-programmers to create automated test cases?**
- **To what extent are cooperative programmers available within the programming team to provide automation support such as event logs, more unique or informative error messages, and hooks for making function calls below the UI level?**
- **What kinds of tests are really *hard* in your application? How would automation make these tests easier to conduct?**

Table-Driven, Interpreted Architecture

To improve maintainability in the face of a constantly changing user interface, split the design of the test cases from the automation of the features.

- Describe the test cases as data that can be fed to the program
- Describe the methods to set the value of each feature.

Here's an example (next slide):

The Calendar Example

The Calendar Example

The Calendar Example

Think About:

- **Automated testing is software development.**
- **We decided to do table-driven programming because we realized that maintainability was a key requirement for us. *Requirements drive design.***
- **Our choice of architecture had a huge effect on everything else we did. This particular choice resulted in a relatively small program and a big data table. To add new tests, we added a new line to the table. To update our model of the program, we changed the test code itself. There were lots of other possible architectures. Each one would have carried its own strengths and weaknesses. *Architecture drives implementation and maintenance.***

Think About:

- **The larger the project, the more important it is to think through the requirements and design early.**

Los Altos Workshop on Software Testing (LAWST)

Much of this material was developed at the first 3 meetings of LAWST. These are non-profit (no charge, invitation-only) meetings of experienced consultants and practitioners, in which we share good practices and lessons learned on tightly defined issues. LAWST 1-3 participants were:

Chris Agruss,Tom Arnold, James Bach, Richard Bender, Jim Brooks, Karla Fisher, Chip Groder, Elizabeth Hendrickson, Doug Hoffman, Keith Hooper, III, Bob Johnson, Cem Kaner (host / founder of LAWST), Brian Lawrence (facilitator & co-host of LAWST), Tom Lindemuth, Brian Marick, Thanga Meenakshi, Noel Nyman, Jeffery E. Payne, Bret Pettichord, Drew Pritsker, Johanna Rothman, Jane Stepak, Melora Svoboda, Jeremy White, and Rodney Wilson.