Avoiding Shelfware: A Manager's View of Automated GUI Testing

Cem Kaner J.D., Ph.D., ASQ-CQE

Overview

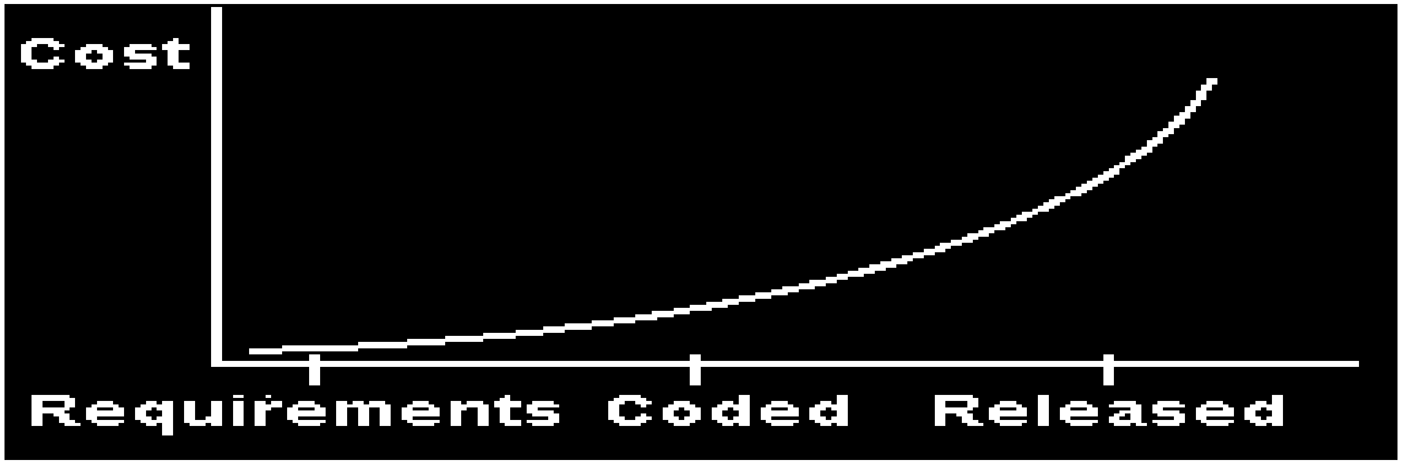
- The GUI regression test paradigm
- 19 common mistakes
- 27 requirements questions
- Planning for short-term ROI
- Successful architectures
- Conclusions from LAWST

GUI 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.**

Common Mistakes and Problems

- **Underestimated cost**
- **Delayed testing, adding even more cost (albeit hidden cost.)**



Common Mistakes and Problems

- **Low power of regression testing**

Common Mistakes and Problems

- **Failure to recognize that we are writing applications.**
- **Adoption of appalling design and programming practices.**
  - » **Embedded constants**
  - » **No modularity**
  - » **No source control**
  - » **No documentation**
  - » **No requirements analysis**
  - » **No wonder we fail.**

Common Mistakes and Problems

- **Maintainability is a core issue because our main payback is usually in the next release, not this one.**

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

- Requirement: “Anything that drives design choices.”
- 27 questions on later slides

- HERE’S ONE KEY EXAMPLE:
  - » Will the user interface of the application be stable or not?

Short Term ROI

- Smoke testing
- Configuration testing
- Stress testing
- Performance benchmarking
- Other tests that extend your reach

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 compability 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-synchronizethe 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?

Six Successful Architectures

1. Quick & dirty
2. Data-driven
3. Framework
4. Application-independent data-driven
5. Real-time simulator with event logs
6. Equivalence testing: Random tests using an oracle

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**

Data-Driven Architecture

<<<Big Graphic Goes Here >>>CaptionTall rowboundingbox

Data-Driven Architecture

	Caption location	Caption typeface	Caption style	Caption Graphic (CG)	CG format	CG size	Bounding box width
1	Top	Times	Normal	Yes	PCX	Large	3 pt.
2	Right	Arial	Italic	No			2 pt.
3	Left	Courier	Bold	No			1 pt.

Data Driven Architecture

Note with this example:

- **we never ran tests twice**
- **we automated execution, not evaluation**
- **we saved SOME time**
- **we focused the tester on design and results, not execution.**

Frameworks

Code libraries:

- **modularity**
- **reuse of components**
- **hide design evolution of UI or tool commands**
- **partial salvation from the custom control problem**
- **important utilities such as error recovery**

Application-Independent Example

Numeric Input Field													
		Nothing	LB of value	UB of value	LB of value - 1	UB of value + 1	0	Negative	LB number of digits or chars	UB number of digits or chars	Empty field (clear the default value)	Outside of UB number of digits or chars	Non-digits

Two More Non-Regression Architectures

- Simulator with event logs
- Random testing using an oracle.

Think About:

- Automation is software development.
- Regression automation is expensive and inefficient.
- Automation need not be regression--you can run new tests instead of old ones.
- Maintainability is essential.
- Set management expectations with care.

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.