The Context-Driven Approach to Software Testing

Cem Kaner, J.D., Ph.D.
Professor of Computer Sciences
Florida Institute of Technology
Star East
May 17, 2002

Context: “It Depends”

- “It depends” is a frustrating answer
  - But we don’t (or shouldn’t) respect consultants who tell us to do the same things under all circumstances.
  - Complex circumstances require tailored responses.
  - The challenge is to figure out the relevant factors and how to deal with them in appropriate ways.

The Seven Basic Principles of the Context-Driven School

- The value of any practice depends on its context.
- There are good practices in context, but there are no best practices.
- People, working together, are the most important part of any project's context.
- Projects unfold over time in ways that are often not predictable.
- The product is a solution. If the problem isn't solved, the product doesn't work.
- Good software testing is a challenging intellectual process.
- Only through judgment and skill, exercised cooperatively throughout the entire project, are we able to do the right things at the right times to effectively test our products.

Example: Test Group Missions

- Find defects
- Maximize bug count
- Support a unit testing effort
- Block premature product releases
- Help managers make ship / no-ship decisions
- Help stakeholders translate their requirements into acceptance tests
- Minimize technical support costs
- Conform to regulations
- Minimize safety-related lawsuit risk
- Assess conformance to specification
- Find safe scenarios for use of the product (find ways to get it to work, in spite of the bugs)
- Verify correctness of the product
- Assess quality
- Assure quality

Example: Test Documentation

- How should we document our testing?
  - Common testing templates, based on IEEE Standard 829, call for very extensive documentation.
  - Should we follow the standard and generate all the paper? Ignore the standard?
  - Or should we decide on our project's requirements, and adopt a documentation approach that meets our actual needs (which may or may not involve 829)?

Example: Test Documentation

- Common documentation requirements questions:
  - Will the documentation be a product or a tool?
  - Do you expect to need test documentation in lawsuits?
  - How quickly is the design changing?
  - How many tests are you trying to document?
  - Will you check test results against an oracle or against a planned result?

Example: GUI Regression Automation

- 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? How easily can you influence these expectations?
- For more, see "Architectures of Test Automation" www.kaner.com/testarch.html and "Avoiding Shelfware" http://www.kaner.com/pdfs/shelfwar.pdf.

Best Practices?

- Rather than accepting an alleged best practice:
  - Ask when and why a given practice would be beneficial?
  - What risks and benefits are associated with it?
  - What skills, documents, development processes, and other resources are required to enable the process?

A Best Practice?

- Should we *really* document every test case in writing and include an expected result?
  - When / why is this beneficial?
    -
  - What are the risks and benefits?
    -
  - What do you need to enable it?
    -
  - When is this inappropriate?
    -

Bad Practices?

- Sometimes, people will suggest practices that seem "obviously" bad.
  - You might (perhaps reasonably) conclude that the person is an idiot.
  - You might (more productively) ask why this practice seems useful to this person.
    - When would it be beneficial?
    - What are its risks and benefits?
    - What do you need to enable it?
    - When is it inappropriate?

An Obviously Bad Practice

- Let's do personnel evaluation of testers by counting their bug reports.
  - When / why is this beneficial?
    -
  - What are the risks and benefits?
    -
  - What do you need to enable it?
    -
  - When is this inappropriate?
    -

We Must Reappraise

- Over the last 30 years, the context of our work has shifted dramatically
  - When I started programming (1967), 10,000 lines of code was a lot
  - In early 1980s, 100,000 lines was a lot
  - Now, we routinely crank out multi-million line systems
- Yet, many of the practices and approaches we recommend to testers today, and see on tester certification exams, are unchanged from the 1980's.

Reappraisal (Some Thoughts)

- Study / train skills rather than practices.
- Build skills and experience in multiple areas, not just testing.
- Look for opportunities to collaborate technically with programmers (e.g. on testability or unit testing).
- Learn a wide range of testing techniques. Select and combine them appropriately for the project at hand. (see *Paradigms of Software Testing*, www.kaner.com/pdfs/slides/paradigm.pdf)
- Appraise the techniques you use (and the documentation you create) against the need to test massive systems. You may need to lighten your processes.

The Seven Basic Principles of the Context-Driven School

- The value of any practice depends on its context.
- There are no best practices.
- People, working together, are the most important part of any project's context.
- Projects unfold over time in ways that are often not predictable.
- The product is a solution. If the problem isn't solved, the product doesn't work.
- Good software testing is a challenging intellectual process.
- Only through judgment and skill, exercised cooperatively throughout the entire project, are we able to do the right things at the right times to effectively test our products.