# Testing Undocumented Software Under Impossible Deadlines

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

**Star 99 East**

# *Introduction*

I didn't come up with the idea for this talk--the STAR folk suggested it as something that I might be able to provide a few insights into. After all, this *is* a chronic complaint among testers. . . .

Well, here goes. How *should* you deal with a situation in which the software comes to you undocumented on a tight deadline?

***I think that the answer depends on the causes of the situation that you're in.*** Your best solution might be technical *or* political *or* resume-ical. It depends on how you and the company got into this mess. If it is a mess.

# *What Caused This Situation?*

**So, let's start with some questions about causation:**

- Why is the software undocumented?
- Why do you have so little time?
- What are the quality issues for this customer?

# *Why Do You Have So Little Time?*

**A Few Possibilities**

- Time-to-release drives competition
- Cash flow drives release date
- Key fiscal milestone drives release date
- Executive belief that you'll never be satisfied, so your schedule input is irrelevant
- Executive belief that testing group is out of control and needs to be controlled by tight budgets
- Executive belief that you have the wrong priorities (e.g. paperwork rather than bugs)
- Executive belief that testing is irrelevant
- Structural lack of concern about the end customer, or
- Maybe you have an appropriate amount of time.

# *Quality Issues For This Customer?*

- In-house, “In-house” outsourced
- External, custom
- External, large package, packaged
- External, mass-market, packaged
- *What is quality in this market? What are their costs of failure?*
  - User groups, Software stores, Sales calls, Support calls

**Over the long term, a good understanding of quality in the market, and as it affects your company’s costs, will buy you credibility and authority.**

# *Political Approaches: Buying Time*

Many of the ideas in this section will help you if you're dealing with a ***single project*** that is out of control.

If your problem is structural (reflects policy or standard practice of the company), then some of these ideas will be counter-productive.

# *Buying Time: Delaying a Premature Release -2*

## Take the high road

- "I want to see knives in the chests, not in the backs."
  - *– Trip Hawkins, founding President, Electronic Arts*
- Communicate honestly.
- Avoid springing surprises on people.
- Never sabotage the project.
- Don't become **The Adversary**: If you are nasty and personal, you will become a tool, to be used by someone else. And you will be disposable.

# *Buying Time: Delaying a Premature Release -3*

- Search for show-stoppers. If you can, dedicate a specialist within your group to this task.
- Circulate deferred bug lists broadly.
- Consider writing mid-project and late-in-project *assessments of:*
  - extent of testing (by area)
  - extent of defects (by area, with predictions about level of bugs left)
  - deferred bugs
  - likely out of box experience or reviewers' experience

# *Buying Time: Delaying a Premature Release -4*

- Do regular, effective status reporting. List:
  - your group's issues (deliverables due to you, things slowing or blocking your progress, etc., areas in which you are behind or ahead of plan)
  - project issues
  - bug statistics
- Find allies *(think in terms of quality-related costs)*

# *Signing Off on the Release?*

- Don't sign off.
- Don't accept the authority to sign off.
- Don't pretend to be "QA" when you (obviously) are not.
- Position yourself as a technical information provider.
  - Publish pre  release reports that provide data on the stability of the product and the extent of completed testing
  - Publish a report that lays out the likely issues that will be raised by magazine reviewers (or other third parities) when they see the product
  - Let the people who want to ship prematurely own their responsibility. Your responsibility is to provide them with the information they need to make that decision.

# *Technical Approaches*

- Find reference docs (you can get lots of data, even if you can't get specs.)
- Re-use materials across projects.
- *Plan for* exploratory testing.
- Do *cost-effective* automation.
- Use tools to find bugs faster
  - web page syntax checkers, spiders, etc.
- Facilitate inspections
  - *(Get those bugs out before they reach you.)*
- Cover the obvious legal issues.

# *Finding Reference Docs*

- Whatever specs exist
- Software change memos that come with each new internal version of the program
- User manual draft (and previous version's manual)
- Product literature
- Published style guide and UI standards
- Published standards (such as C-language)
- 3rd party product compatibility test suites
- Published regulations
- Internal memos (e.g. project mgr. to engineers, describing the feature definitions)
- Marketing presentations, selling the concept of the product to management
- Bug reports (responses to them)
- Reverse engineer the program.
- Interview people, such as
  - development lead
  - tech writer
  - customer service
  - subject matter experts
  - project manager
- Look at header files, source code, database table definitions
- Specs and bug lists for all 3rd party tools that you use
- Prototypes, and lab notes on the prototypes
- industry experts
- Interview development staff from the last version.
- Look at customer call records from the previous version. What bugs were found in the field?
- Usability test results
- Beta test results
- Ziff-Davis SOS CD and other tech support CD's, for bugs in your product and common bugs in your niche or on your platform
- BugNet magazine / web site for common bugs
- News Groups, CompuServe Fora, etc., looking for reports of bugs in your product and other products, and for discussions of how some features are supposed (by some) to work.
- Localization guide (probably one that is published, for localizing products on your platform.)
- Get lists of compatible equipment and environments from Marketing in theory, at least.)
- Look at compatible products, to find their failures (then look for these in your product), how they designed features that you don't understand, and how they explain their design. See listserv's, NEWS, BugNet, etc.
- Exact comparisons with products you emulate
- Content reference materials (e.g. an atlas to check your on-line geography program)

# *Reusable Test Matrix*

| 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) | Maximum possible number of chars | Non-digits |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |

# *Group Management Approaches*

- **Can you add head count?**
  - Will they let you?
  - Can you absorb them?
    - Do a work flow analysis to figure out what parts of each task could be split off and delegated to a newcomer.
- **Stay organized**
  - Use functional outlines
  - Or use a detailed project plan
  - Create and publish explicit coverage reports
- **Get critical processes under control**
  - Example: release management