

Black Box Software Testing

(Professional Seminar)

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

Professor of Computer Sciences
Florida Institute of Technology

Section:25

Questioning Strategies and the Satisfice Model

Summer, 2002

Contact Information:

kaner@kaner.com

www.kaner.com (testing website)

www.badsoftware.com (legal website)

I grant permission to make digital or hard copies of this work for personal or classroom use, with or without fee, provided that (a) copies are not made or distributed for profit or commercial advantage, (b) copies bear this notice and full citation on the first page, and if you distribute the work in portions, the notice and citation must appear on the first page of each portion, (c) each page bear the notice "Copyright (c) Cem Kaner" or if you changed the page, "Adapted from Notes Provided by Cem Kaner". Abstracting with credit is permitted. The proper citation for this work is Cem Kaner, *A Course in Black Box Software Testing (Professional Version)*, Summer-2002, www.testing-education.org. To copy otherwise, to republish or post on servers, or to distribute to lists requires prior specific permission and a fee. Request permission to republish from kaner@kaner.com.

Testers' Questions: Does Your Car Work?

- How do you know your car works?
- Are there situations in which your car would stop working?
- Who else uses your car? Do they use it differently than you, so that it might work for you but fail for them?
- What facts would cause you to believe that your car *doesn't* work?
- In what ways could your car *not* work, yet seem to you that it does?
- In what ways could your car work, yet seem to you that it *doesn't*?
- Do you know enough about cars to answer these questions?
- Have you observed your car enough, *today*, to answer them?
- Under what circumstances would these questions matter?

Questioning

Requirements analysis requires information gathering

- Read books on consulting
- Gause & Weinberg, *Exploring Requirements* is an essential source on context-free questioning

There are many types of questions:

- Open vs. closed
- Hypothetical vs. behavioral
- Opinion vs. factual
- Historical vs. predictive
- Context-dependent and context-free

The classic context-free questions

The traditional newspaper reporters' questions are:

- Who
- What
- When
- Where
- How
- Why

For example, *Who will use this feature? What does this user want to do with it? Who else will use it? Why? Who will choose not to use it? What do they lose? What else does this user want to do in conjunction with this feature? Who is not allowed to use this product or feature, why, and what security is in place to prevent them?*

We use these in conjunction with questions that come out of the testing model (see below). The model gives us a starting place. We expand it by asking each of these questions as a follow-up to the initial question.

Context-Free Questions: Defining the Problem

Based on: *The CIA's Phoenix Checklists (Thinkertoys, p. 140) and Bach's Evaluation Strategies (Rapid Testing Course notes)*

- Why is it necessary to solve the problem?
- What benefits will you receive by solving the problem?
- What is the unknown?
- What is it that you don't yet understand?
- What is the information that you have?
- What is the source of this problem? (Specs? Field experience? An individual stakeholder's preference?)
- Who are the stakeholders?
- How does it relate to which stakeholders?
- What isn't the problem?
- Is the information sufficient? Or is it insufficient? Or redundant? Or contradictory?
- Should you draw a diagram of the problem? A figure?

Context-Free Questions: Defining the Problem

- Where are the boundaries of the problem?
- What product elements does it apply to?
- How does this problem relate to the quality criteria?
- Can you separate the various parts of the problem? Can you write them down? What are the relationships of the parts of the problem?
- What are the constants (things that can't be changed) of the problem?
- What are your critical assumptions about this problem?
- Have you seen this problem before?
- Have you seen this problem in a slightly different form?
- Do you know a related problem?
- Try to think of a familiar problem having the same or a similar unknown.
- Suppose you find a problem related to yours that has already been solved. Can you use it? Can you use its method?
- Can you restate your problem? How many different ways can you restate it? More general? More specific? Can the rules be changed?
- What are the best, worst, and most probable cases you can imagine?

Context-Free Questions

Context-free process questions

- Who is the client?
- What is a successful solution worth to this client?
- What is the real (underlying) reason for wanting to solve this problem?
- Who can help solve the problem?
- How much time is available to solve the problem?

Context-free product questions

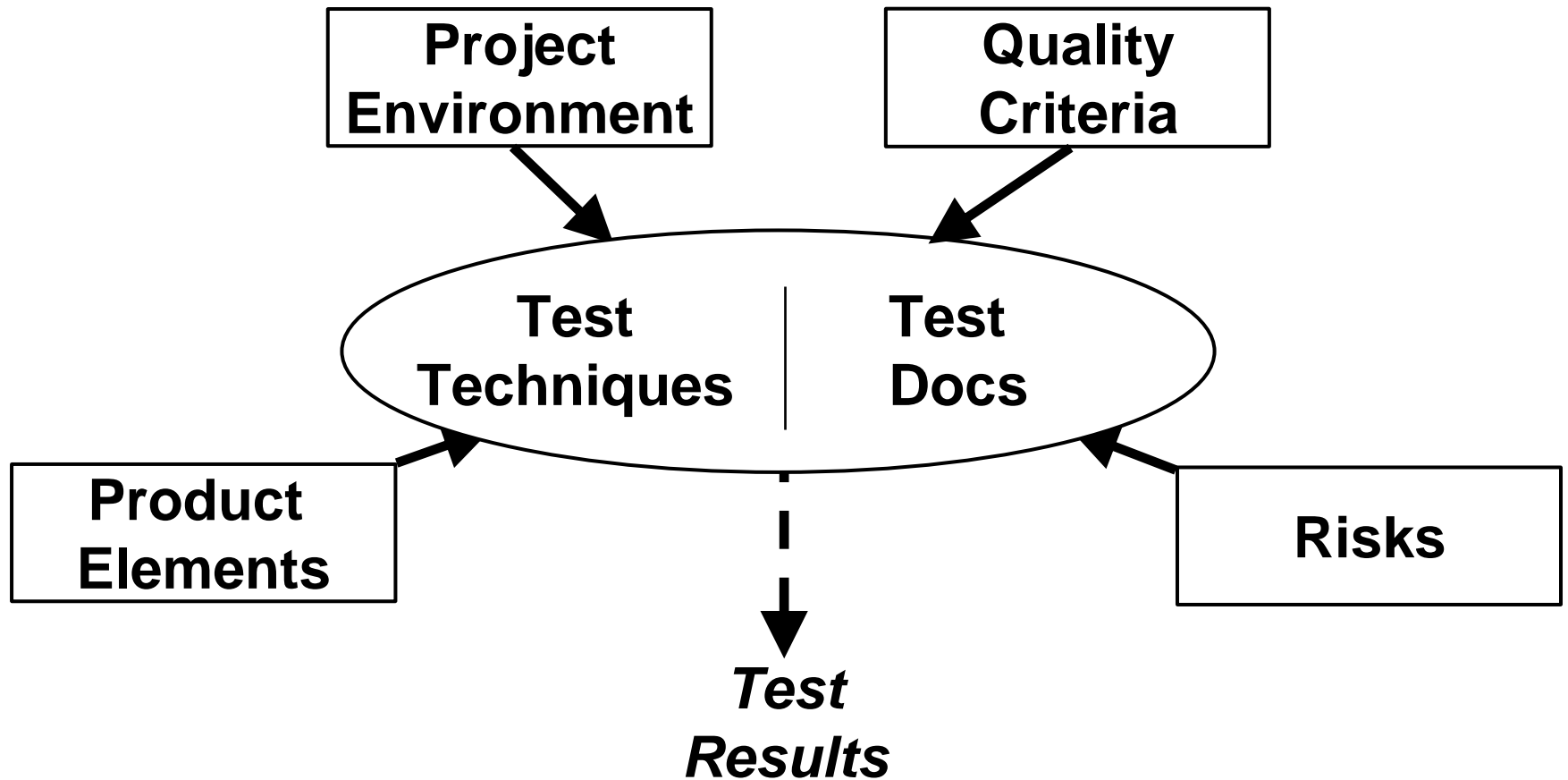
- What problems could this product create?
- What kind of precision is required / desired for this product?

Metaquestions (when interviewing someone for info)

- Am I asking too many questions?
- Do my questions seem relevant?
- Are you the right person to answer these questions?
- Is there anyone else who can provide additional information?
- Is there anything else I should be asking?
- Is there anything you want to ask me?
- May I return to you with more questions later?

A sample of additional questions based on Gause & Weinberg's Exploring Requirements p. 59-64

A Model of Software Testing



Developing Questions

By Using the Testing Model

There are several ways to use the testing model.

- When you run out of testing ideas, walk the charts looking for a project / product / quality factor / risk that you haven't based a test on recently.
- Randomly combine project / product / quality factors / risks – make up a test case that is influenced by the selected product factor, that tests the selected product element against the selected quality criterion or risk.
- Analyze a specification, operating on the assumption that every statement defines a project factor, a product factor, a quality criterion, or identifies a risk.

Project Environment Factors:

- **Stakeholders**
- **Processes**
- **Staff**
- **Schedules**
- **Equipment**
- **Tools & Test Materials**
- **Information**
- **Items Under Test**
- **Logistics**
- **Budget**
- **Deliverables**

These aspects of the environment constrain and enable the testing project

Project Factors

Stakeholders:

Anyone who is a client of the main project

Anyone who is a client of the testing project

Includes customers (purchasers), end users, tech support, programmers, project mgr, doc group, etc.

Processes:

The tasks and events that comprise the main project

How the overall project is run

The tasks and events that comprise the test project

How the testing project is run

Staff:

Everyone who helps develop the product

Sources of information and assistance

Everyone who will perform or support testing

Special talents or experiences of team members

Size of the group

Extent to which they are focused or are multi-tasking

Organization: collaboration & coordination of the staff

Is there an independent test lab?

Project Factors

Schedules: *The sequence, duration and synchronization of events*

- When will testing start and how long is it expected to take?
- When will specific product elements be available to test?
- When will devices or tools be available to support testing?

Equipment: *Hardware required for testing*

- What devices do we need to test the product with? Do we have them?

Tools & Test Materials: *Software required or desired for testing.*

- Automation: Are such tools available? Do we want to use them? Do we have them? Do we understand them?
- Probes or diagnostics to help observe the product under test?
- Matrices, checklists, other testing documentation?

Information: *(As needed for testing) about the project or product.*

- Specifications, requirements documents, other reference materials to help us determine pass/fail or to credibly challenge odd behaviour.
 - »What is the availability of these documents?
 - »What is the volatility of these documents?

Project Factors

Items Under Test: *Anything that will be tested*

- For each product element:
 - » Is it available (or when will it be)?
 - » Is it volatile (and what is the change process)?
 - » Is it testable?

Logistics: *Facilities and support needed for organizing and conducting the testing*

- Do we have the supplies / physical space, power, light / security systems (if needed) / procedures for getting more?

Budget: *Money and other resources for testing*

- Can we afford the staff, space, training, tools, supplies, etc.?

Deliverables: *The observable products of the test project*

- Such as bug reports, summary reports, test documentation, master disk.
 - » What are you supposed to create and can you do it?
- Will we archive the items under test and other products of testing?

Product Elements: A product is...

An experience or solution provided to a customer.

Everything that comes in the box, plus the box!

*Functions and data, executed on a platform,
that serve a purpose for a user.*

- 1 A software product is much more than code.
- 2 It involves a purpose, platform, and user.
- 3 It consists of many interdependent *elements*.

Product Elements:

Structures: *Everything that comprises the physical product*

- Code: the code structures that comprise the product, from executables to individual routines
- Interfaces: points of connection and communication between subsystems
- Hardware: hardware components integral to the product
- Non-executable files: any files other than programs, such as text files, sample data, help files, etc.
- Alternate Media: anything beyond software and hardware, such as paper documents, web links and content, packaging, license agreements, etc.

Product Elements:

Functions: *Everything that the product does.*

- User Interface: functions that mediate the exchange of data with the user
- System Interface: functions that exchange data with something other than the user, such as with other programs, hard disk, network, printer, etc.
- Application: functions that define or distinguish the product or fulfill core requirements
- Error Handling: functions that detect and recover from errors, including error messages
- Testability: functions provided to help test the product, such as diagnostics, log files, asserts, test menus, etc.

Temporal relationships: *How the program functions over time*

- Sequential operation: state-to-state transitions
- Data: changes in variables over time
- System interactions: such as synchronization or ordering of events in distributed systems

Product Elements:

Data: *Everything that the product processes*

- Input: data that is processed by the product
- Output: data that results from processing by the product
- Preset: data supplied as part of the product or otherwise built into it, such as prefab databases, default values, etc.
- Persistent: data stored internally and expected to persist over multiple operations. This includes modes or states of the product, such as options settings, view modes, contents of documents, etc.
- Temporal: data based on time, such as date stamps or number of events recorded in a unit of time

Product Elements:

Platform: *Everything on which the product depends*

- External Hardware: components and configurations that are not part of the shipping product, but are required (or optional) in order for the product to work. Includes CPU's, memory, keyboards, peripheral boards, etc.
- External Software: software components and configurations that are not a part of the shipping product, but are required (or optional) in order for the product to work. Includes operating systems, concurrently executing applications, drivers, fonts, etc.

Operations: *How the product will be used*

- Usage Profile: the pattern of usage, over time, including patterns of data that the product will typically process in the field. This varies by user and type of user.
- Environment: the physical environment in which the product will be operated, including such elements as light, noise, and distractions.

Product Elements: Coverage

Product coverage is the proportion of the product that has been tested.

There are as many kinds of coverage as there are ways to model the product.

- Structural
- Functional
- Temporal
- Data
- Platform
- Operations

*See Software Negligence
& Testing Coverage at
www.kaner.com for 101
examples of coverage
“measures.”*

Quality Criteria

Capability

Reliability

Usability

Performance

Installability

Compatibility

Supportability

Testability

Maintainability

Portability

Localizability

Efficiency

*Quality is value to some
person*
-- Jerry Weinberg

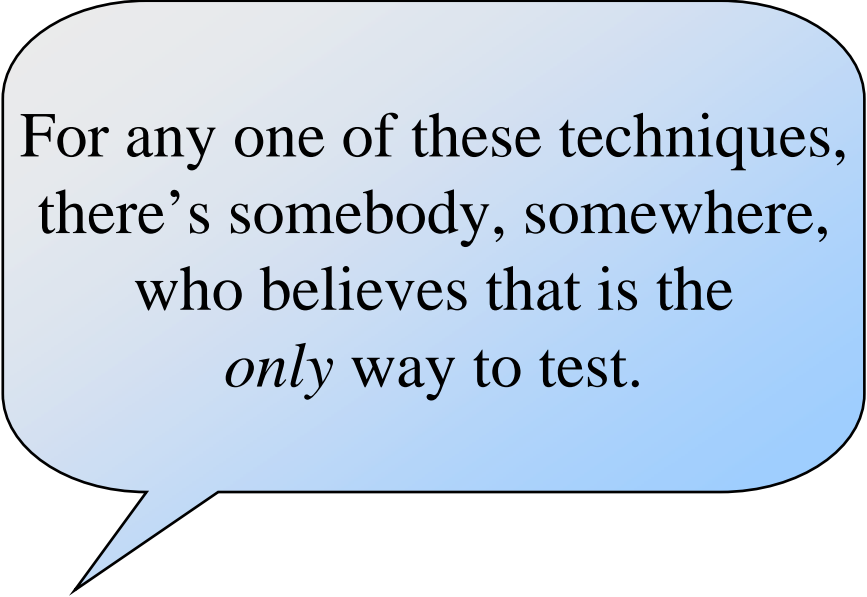
Test Techniques

- Analyze the situation.
- Model the test space.
- Select what to cover.
- Determine test oracles.
- Configure the test system.
- Operate the test system.
- Observe the test system.
- Evaluate the test results.

***A test technique
is a recipe
for performing
these tasks in order to
reveal something
worth reporting***

General Test Techniques

- **Function testing**
- **Domain testing**
- **Stress testing**
- **Flow testing**
- **User testing**
- **Regression testing**
- **Risk testing**
- **Claims testing**
- **Random Testing**



For any one of these techniques,
there's somebody, somewhere,
who believes that is the
only way to test.

Test Strategy

“How we plan to cover the product so as to develop an adequate assessment of quality.”

A good test strategy is:

- *Diversified*
- *Specific*
- *Practical*
- *Defensible*

Test Strategy

- Makes use of test techniques.
- May be expressed by test procedures and cases.
- Not to be confused with test *logistics*, which involve the details of bringing resources to bear on the test strategy at the right time and place.
- You don't have to know the entire strategy in advance. The strategy can change as you learn more about the product and its problems.

Test Cases/Procedures

- **Test cases and procedures should manifest the test strategy.**
- **If your strategy is to “execute the test suite I got from Joe Third-Party”, how does that answer the prime strategic questions:**
 - How will you *cover the product* and *assess quality*?
 - How is that *practical* and *justified* with respect to the *specifics* of this project and product?
- **If you don't know, then your real strategy is that you're trusting things to work out.**

Diverse Half-Measures

- There is no single technique that finds all bugs.
- We can't do any technique perfectly.
- We can't do all conceivable techniques.

Use “**diverse half-measures**”-- lots of different points of view, approaches, techniques, even if no one strategy is performed completely.

*Heuristics from James Bach's Test Plan Evaluation Model,
www.satisfice.com*

Heuristic	Basis for the Heuristic
1. Testing should be optimized to find important problems fast, rather than attempting to find all problems with equal urgency.	The later in the project that a problem is found, the greater the risk that it will not be safely fixed in time to ship. The sooner a problem is found after it is created, the lesser the risk of a bad fix.
2. Test strategy should focus most effort on areas of potential technical risk, while still putting some effort into low risk areas just in case the risk analysis is wrong.	Complete testing is impossible, and we can never know if our perception of technical risk is completely accurate.
3. Test strategy should address test platform configuration, how the product will be operated, how the product will be observed, and how observations will be used to evaluate the product.	Sloppiness or neglect within any of these four basic testing activities will increase the likelihood that important problems will go undetected.

Heuristics for Test Plan Evaluation

Heuristic	Basis for the Heuristic
<p>4. Test strategy should be diversified in terms of test techniques and perspectives. Methods of evaluating test coverage should take into account multiple dimensions of coverage, including structural, functional, data, platform, operations, and requirements.</p>	<p>No single test technique can reveal all important problems in a linear fashion. We can never know for sure if we have found all the problems that matter. Diversification minimizes the risk that the test strategy will be blind to certain kinds of problems.</p> <p><i>Use diverse half-measures to go after low-hanging fruit.</i></p>
<p>5. The test strategy should specify how test data will be designed and generated.</p>	<p>It is common for the test strategy to be organized around functionality or code, leaving it to the testers to concoct test data on the fly. Often that indicates that the strategy is too focused on validating capability and not focused enough on reliability.</p>

Heuristics for Test Plan Evaluation

Heuristic	Basis for the Heuristic
<p>6. Not all testing should be pre-specified in detail. The test strategy should incorporate reasonable variation and make use of the testers' ability to use situational reasoning to focus on important, but unanticipated problems.</p>	<p>A rigid test strategy may make it more likely that a particular subset of problems will be uncovered, but in a complex system it reduces the likelihood that <i>all</i> important problems will be uncovered. Reasonable variability in testing, such as that which results from interactive, exploratory testing, increases incidental test coverage, without substantially sacrificing essential coverage.</p>
<p>7. It is important to test against implied requirements—the full extent of what the requirements mean, not just what they say.</p>	<p>Testing only against explicit written requirements will not reveal all important problems, since defined requirements are generally incomplete and natural language is inherently ambiguous.</p>

Heuristics for Test Plan Evaluation

Heuristic	Basis for the Heuristic
8. The test project should promote collaboration with all other functions of the project, especially developers, technical support, and technical writing. Whenever possible, testers should also collaborate with actual customers and users, in order to better understand their requirements.	Other teams and stakeholders often have information about product problems or potential problems that can be of use to the test team. Their perspective may help the testers make a better analysis of risk. Testers may also have information that is of use to them.
9. The test project should consult with development to help them build a more testable product.	The likelihood that a test strategy will serve its purpose is profoundly affected by the testability of the product.

Heuristics for Test Plan Evaluation

Heuristic	Basis for the Heuristic
10. A test plan should highlight the non-routine, project-specific aspects of the test strategy and test project.	Virtually every software project worth doing involves special technical challenges that a good test effort must take into account. A completely generic test plan usually indicates a weak test planning process. It could also indicate that the test plan is nothing but unchanged boilerplate.

Heuristics for Test Plan Evaluation

Heuristic	Basis for the Heuristic
11. The test project should use humans for what humans do well and use automation for what automation does well. Manual testing should allow for improvisation and on the spot critical thinking, while automated testing should be used for tests that require high repeatability, high speed, and no judgment.	Many test projects suffer under the false belief that human testers are effective when they use exactly specified test scripts, or that test automation duplicates the value of human cognition in the test execution process. Manual and automated testing are not two forms of the same thing. They are two entirely different classes of test technique.

Heuristics for Test Plan Evaluation

Heuristic	Basis for the Heuristic
12. The test schedule should be represented and justified in such a way as to highlight any dependencies on the progress of development, the testability of the product, time required to report problems, and the project team's assessment of risk.	A monolithic test schedule in a test plan often indicates the false belief that testing is an independent activity. The test schedule can stand alone only to the extent that the product is highly testable, development is complete, and the test process is not interrupted by the frequent need to report problems.
13. The test process should be kept off of the critical path to the extent possible. This can be done by testing in parallel with development work, and finding problems worth fixing faster than the developers fix them.	This is important in order to deflect pressure to truncate the testing process.

Heuristics for Test Plan Evaluation

Heuristic	Basis for the Heuristic
<p>14. The feedback loop between testers and developers should be as tight as possible. Test cycles should be designed to provide rapid feedback to developers about recent additions and changes they have made before a full regression test is commenced. Whenever possible testers and developers should work physically near each other.</p>	<p>This is important in order to maximize the efficiency and speed of quality improvement. It also helps keep testing off of the critical path.</p>

Heuristics for Test Plan Evaluation

Heuristic	Basis for the Heuristic
15. The test project should employ channels of information about quality other than formal testing in order to help evaluate and adjust the test project. Examples of these channels are inspections, field testing, or informal testing by people outside of the test team.	By examining product quality information gathered through various means beyond the test team, blind spots in the formal test strategy can be uncovered.
16. All documentation related to the test strategy, including test cases and procedures, should be undergo review by someone other than the person who wrote them. The review process used should be commensurate with the criticality of the document.	Tunnel-vision is the great occupational hazard of testing. Review not only helps to reveal blind spots in test design, but it can also help promote dialog and peer education about test practices.

Evaluating Your Plan: Context Free Questions

**Based on: *The CIA's Phoenix Checklists (Thinkertoys, p. 140)*
*and Bach's Evaluation Strategies (Rapid Testing Course notes)***

- Can you solve the whole problem? Part of the problem?
- What would you like the resolution to be? Can you picture it?
- How much of the unknown can you determine?
- What reference data are you using (if any)?
- What product output will you evaluate?
- How will you do the evaluation?
- Can you derive something useful from the information you have?
- Have you used all the information?
- Have you taken into account all essential notions in the problem?
- Can you separate the steps in the problem-solving process? Can you determine the correctness of each step?
- What creative thinking techniques can you use to generate ideas? How many different techniques?
- Can you see the result? How many different kinds of results can you see?
- How many different ways have you tried to solve the problem?

Evaluating Your Plan: Context Free Questions

- What have others done?
- Can you intuit the solution? Can you check the results?
- What should be done?
- How should it be done?
- Where should it be done?
- When should it be done?
- Who should do it?
- What do you need to do at this time?
- Who will be responsible for what?
- Can you use this problem to solve some other problem?
- What is the unique set of qualities that makes this problem what it is and none other?
- What milestones can best mark your progress?
- How will you know when you are successful?
- How conclusive and specific is your answer?

Exercise

Pretend that you are a member of the test team for Star Office. Here are some (fictitious) details:

- The code is developed remotely (much of it is developed by programmers in another country).
- You have access to the source code and many bug reports from users because the program is open source.
- This is a significant update to the program.
 - » Lots of bugs will be fixed.
 - » The word processor has been substantially revised so that it interprets and formats bullets, numbered lists, and table layouts more compatibly with MS Word. This is in response to customer complaints that the formatting is often incorrect or inconsistent when one creates a Word file or a StarOffice document with these features, and then use StarOffice to read a Word file or Word to read an exported StarOffice file.

Exercise (continued)

- This is a group exercise. Divide the 4 categories of factors among yourselves:
 - » Project factors
 - » Product elements
 - » Quality criteria
 - » Risks (for the relevant list, see the Risk-based testing section (section 14) earlier in the course)
- If possible, work in pairs at flipcharts.
- Divide the category you are working on into its subcategories and work on one at a time (maybe allocate one flipchart page each).
 - » For example, Stakeholders, Processes, and Staff are subcategories in the Project Factors list.

Exercise (Continued)

On your flipcharts,

- **list what you know for each subcategory that you are working on.**
- **In many subcategories, you won't know much. Along with writing down what you do know, write down questions that call for relevant information that you don't yet know.**
- **You might learn that some tasks or information or decisions are time-critical. You need to do them or get them right away. If so, write them on a separate sheet or in a different bright color and start to deal with them before the end of today.**

Exercise

- After working in separated pairs for a while, come back and compare notes.
- These notes provide you with a first draft test plan, first draft task list, and a lot of information about the project, more than you'd expect.

