# Assessment in the Software Testing Course[1]


## Cem Kaner[2]


## Workshop on the Teaching of Software Testing

## February 2003


## Abstract

This report collects my notes on the assessment issues in my software testing course. These notes will serve as the raw data archive for a summary paper submitted for traditional publication. The notes include a description of approach to examinations, sample exam questions, exam study guide, examples of my grading scheme for exams. The notes also include sample assignments (with grading notes).


## Contents



## 1. Background: The Software Testing Course

This report collects my notes on the assessment issues involved in my software testing course. The notes are still somewhat rough. I'll probably edit them one more time before writing a summary paper for publication. These notes will serve as the raw data archive for that summary paper.

The course itself has been in evolution since 1987, when I began "Tester College" while managing the testing group in Electronic Arts' Creativity Division. Hung Quoc Nguyen and I then developed a software testing course for the Silicon Valley chapter of the American Society for Quality in 1994. I have taught that course to working professionals frequently since then, in public courses offered by UC Berkeley Extension and UC Santa Cruz Extension, Software Quality Engineering, Software Test Labs, logiGear, and Satisfice, and in onsite classes at large and small software companies (such as Microsoft, Hewlett-Packard, Intel, Quarterdeck, Compaq, PostalSoft, PowerQuest, Symantec, Rational, Kodak, Gilbarco, Aveo, BMC, IDTS, Tideworks, Wind River, Cigital, and many others.) A recent version of my commercial course notes is available at http://www.testingeducation.org/coursenotes/kaner_cem/cm_200204_blackboxtesting.

I modified the course for academic use in 2000 and have taught the academic course at Florida Tech five times, modifying it (especially in the assessment) each time. A version of my academic course notes is available at http://www.testingeducation.org/coursenotes/kaner_cem/ac_200108_blackboxtesting.

The academic course focuses on black box software testing. I teach a second course in glass box testing that takes the black box course as a prerequisite.

The black box course has five core topic areas, which I prioritize as follows:

- ***Paradigms of software testing:*** a look at 9 dominant styles of black box testing. Students apply several of these to a sample application, such as StarOffice.
- ***Bug advocacy***: effective replication, analysis, and reporting of bugs.
- ***Test documentation***: examples of test documentation components and an overview of requirements analysis to determine what is needed in what context.
- ***Additional test design issues:*** The primary examples are an overview of GUI-level regression testing and design of these tests for maintainability, and all pairs combination testing.
- ***Process and organizational issues:*** We look primarily at the structure and missions of typical software testing groups and the implications for testing of different software lifecycle models. This material is presented primarily to provide context for those students who have no industrial experience, and to provide exposure to alternative contexts for students who have worked only in one or two companies.

Several other topics come and go in the class, depending on student interest, applicability to the sample application that we are testing, and various other factors. These have included state-model-based testing, software test metrics, high volume test automation, status reporting, project planning, quality/cost analysis, failure modes and effects analysis, and finding a job in software testing.

This report focuses on assessment issues, and so I will not further discuss the choice of topics here.

Throughout the course, we apply what we learn to a sample application. So far, we've used the TI Interactive Calculator and the word processing and the presentation modules of *OpenOffice*. Another senior member of our faculty (James Whittaker) also uses sample applications in his testing courses, primarily Microsoft products under development.

I recommend working with the open source products for several reasons:

- The students bug reports are publicly available. This can help them at job interview time (they can point to records of their actual work product). It also encourages them to take the reporting task seriously.
- Students can see the progress of their bugs through the bug reporting system, watching comments develop on their bug reports, reporting fixes, complaining of non-reproducibility, asking for more information, and so on. They get to participate in a series of real-project bug discussions, gaining insight and experience that will be directly applicable on the job.
- The students' work is valued and they get personal feedback. This is not true of all open source projects, but an instructor can get a sense of the feedback style on the project by examining the reports already in the bug tracking database before selecting an application.

## 2. Assessment Methods

Assessment is the course's primary educational tool.

I give lectures and students (academic and commercial) generally like them, but lectures can only transmit so much information, and students forget them anyway.

I use the lectures to provide a structure for the material and to provide real-life examples, compelling or entertaining stories that will help students understand how or why a technique was used in practice, what the effects of different life cycle models can be, and so on. The lectures create contexts for the material.

During the lectures, I also run several discussions focused on hypotheticals or thought experiments. These are effective learning tools for some students.

My expectation is that most students will do most of their learning while doing homework, assignments, and studying for, writing, and reviewing the results of tests and exams.

I encourage students to work together when they do assignments. In general, encouraging collaboration (students co-sign artifacts that they work on together, with the explicit expectation that more co-authors must produce more work), seems to have been effective in eliminating plagiarism. The collaboration is done openly instead of secretly.

However, because 35% of the final grade comes from the assignments, there is an incentive for a weak student to pair with a stronger student in order to cash in on the high marks the stronger student will earn. The first few times that I taught the course, this was a serious problem and at least two students passed the course who probably should not have. I addressed this problem with the following policy, printed in the syllabus (and reviewed with students in the first class):

> To pass the course, you must have a passing average on the mid-term test and the final exam.
>
> - ***Undergraduates (CSE 4431):*** If the average of your mid-term test and your final exam is below 60%, you will fail the course no matter how well you do on the assignments and no matter how many bonus points you have.
> - ***Graduates (SWE 5410):*** If the average of your mid-term test and your final exam is below 68%, you will fail the course no matter how well you do on the assignments and no matter how many bonus points you have.
>
> You can earn grades as follows
>
> - In-class quizzes — up to 5% (1% per quiz, pass/fail grading)
> - Assignments — 30-35% (depends on the number of quizzes)
> - Mid-term test — 25%
> - Final exam — 40%
> - Bonus Assignments (including bug reports) — up to 10%
>   - Bug reports — up to 5%
>
> Total points available — 110%
>
> I don't grade on a curve. If everyone gets 90% or more, everyone gets an A. (B is 80-89; C is 70-79; D is 60-69; F is 0-59).

Since adopting the threshold policy, I haven't seen as much obvious imbalance in effort on the assignments. The student who can't pass the exams can't pass the course.

# 3. Exams

To maximize the educational benefits from the mid-term and final exams, I hand out a pool of questions a few weeks before the exam. The exam questions are selected from the pool. The exam is closed book.

## *3.1 Benefits*

***The most important benefit of this approach is that it allows the students to think through their answers and prepare them carefully.*** The exam is merely a production exercise--the student isn't spending precious time trying to understand each question and think through a strategy for answering it. Because I can assume that the students have thoughtfully developed their answers, I can apply a higher standard when I grade the answers.

***Another important benefit is that it gives the students structure for studying together.*** I encourage students to review and discuss each others' answers.[3] The questions focus their work, give them something explicit to work on.

***A third benefit is that this approach allows me to ask complex questions without unfairly disadvantaging students whose native language is not English.*** People read at different speeds. Students whose English language skills are still under development need extra time to read and comprehend essay questions and to structure their answers. Because they have that extra preparation time, I don't have make allowances for them when I grade.

## *3.2 Source material*

Appendix A lists questions that I've used in the course.

Appendix B is an instruction / suggestion sheet that I give students with the questions. The sheet includes two types of guidance:

- How to study for this type of test
- How to answer questions on this type of test.

Appendix C provides analyses and answers for several of the exam questions.

## *3.3 Risks and Problems*

The primary problem with this course's approach is that many students aren't used to answering essay questions and so they deal with them ineffectively. This problem is not unique to computer science students. For example, teaching first year law students how to answer essay questions is a critical task, repeated in course after course.[4] Many universities publish guidelines for undergraduates on how to study for, and organize, essay questions.[5]

The problems that I note here apply broadly to essay-format exams among graduate students in computer science. For example, at Florida Tech in Fall, 2002, a substantial minority of the population of students writing the Software Engineering Comprehensive Exam failed or nearly failed the exam because of ineffective essay-answering strategies that yielded

- weakly structured answers that missed important points or
- shotgun answers (unfocused, not directly responsive to the question).

### 3.3.1 Weak Structure

Consider the following question as an example:

> Define a scenario test and describe the characteristics of a good scenario test. Imagine developing a set of scenario tests for the Outlining feature of the word processing module of *OpenOffice*. What research would you do in order to develop a series of scenario tests for Outlining? Describe two scenario tests that you would use and explain why each is a good test. Explain how these tests would relate to your research.

This has several components:

- Define a scenario test
- Describe the characteristics of a good scenario test
- What research would you do in order to develop a series of scenario tests for Outlining?
- Describe two scenario tests you would use.
- Explain why each of the two scenario tests is a good test
- Explain how these two scenario tests would relate to your research

A well organized answer will have at least six sections, one for each of the bulleted components. You might have two additional sections, by splitting *Describe two scenario tests you would use* and *Explain why each of the two scenario tests is a good test* into two sections, one for each test.

Without structure, it is easy to miss a section and thereby to lose points.

*Students must learn to focus their answer to match the "call of the question" (the specific issues raised in the question).*

It is a safe bet that a substantial portion of the undergraduate or graduate CS students who attend the software testing class will not yet have sufficiently developed their skills in identifying and responding to the call of the question.

### 3.3.2 Shotgun Answers

A student using a shotgun strategy responds with a core dump of everything that seems to be relevant to the general topic. Much of this information might be correct, but if it is non-responsive to the call of the question, it is irrelevant and I will ignore it. However, to the extent that irrelevant information is incorrect, if I notice an error, I will deduct points for it.

Here's an example of a question that yielded a lot of shotgunning and not enough points:

> Imagine that you are an external test lab, and Sun comes to you with *OpenOffice*. They want you to test the product. When you ask them what test documentation they want, they say that they want something appropriate but they are relying on your expertise. To decide what test documentation to give them, what questions would you ask (up to 7 questions) and for each answer, how would the answer to that question guide you?

In the course, we looked over a long list of requirements-eliciting questions.[6] Students were free to use the ones we discussed or to supply their own.

The question does *not* call for definition / discussion of IEEE 829 or for a list of the common test documentation components. It doesn't call for a description of test matrices or a discussion of how to create them. I got these and much more on a recent exam. Unless this information was couched in terms of a question or the interpretation of the question/answer, it was irrelevant--a waste of the students' time (and in a time-limited exam, a tax on the student's ability to complete the rest of the test in the time available).

### 3.3.3 Time Management

Students *should have* fewer exam time management problems when their exams contain only questions from the study guide. After all, they have (in theory) answered each question and they have a sense of how long each question takes to answer.

In practice, many students run out of time the first time they take a test like this, because they don't realize that time management will be an issue or what to do to manage it. As an example of how to manage a timing problem, if a student develops a "perfect" answer to an essay question, but discovers that it will take an hour to write, she will prioritize and reduce the length of the answer in order to fit it within the time available.

### 3.3.4 Lack of Preparation

I encountered the worst timing problems during my first year of teaching at Florida Tech. I didn't yet have a reputation among the students, and few other instructors gave students a list of study questions *that included all of the questions that would actually appear on the exam.* As a result, students in my first two courses didn't study the particular questions in detail and didn't develop their own answers. These students performed badly on their mid-term exams; many of them didn't come close to finishing the exam because they had to read, comprehend, and plan an answer for each question rather than recognizing the question and starting to write the answer they have already prepared.

### 3.3.5 Weak Group Preparation

The best way to prepare for these tests is for each student to attempt each question on his own. The first attempt should be open book with no time limit. After each student has his own answers, he should compare notes with other students. The diversity of approaches will highlight ambiguities in the question, hidden assumptions on the part of the student, and muddled, disorganized thinking about the structure and call of the question. Independent preparation by several students is essential.

Unfortunately, many students form study groups in which they either:

- Divide up the questions. One or two students attempt to answer each question and then report back to the group. The rest of the students then attempt to memorize the answers.
- Attempt to develop the answers in-group, four or more students arguing and together.

Neither of these approaches works well. There are so many questions in the study list that few (or no) students can effectively memorize all the answers. As a result, I often see answer fragments, relevant material mixed with irrelevant (something memorized for a different question), or answers that have been distorted (such as forgotten words, points made so far out of sequence that they don't make sense, etc.)

The group-think approach works better but often produces weak answers. The group tends to latch onto the first answer that appears to make sense. Or it latches onto the answer advocated by the loudest or most persuasive or most persistent student in the group.

It is much more effective to start from a diverse group of prepared answers, with the people who understand and can explain why they prepared the answers in they way they did.

I tell students this every term, and every term a significant group of students tries the divide-and-(oops)-don't-conquer strategy and the work-only-during-group-study sessions. Most of them learn their lesson the hard way when they write an unsatisfactory mid-term exam.

### 3.3.6 Weak Answers Propagate Through the Group

Sometimes, the entire class answers a question in a way that is obviously (to me) mistaken or otherwise sub-optimal. I've seen several class-specific exam answers like this. By class-specific, I mean that a different class, on encountering the same question, has handled it much better.

### 3.3.7 Failure to Consult Required Readings

I publish my lecture notes on the web, using a tool called *BlackBoard*. Along with my lecture notes, I supply copies of several other articles, some in a folder labeled as *Required Reading* and others in a folder labeled as *Recommended Reading*.

Surprisingly often, students consult the lecture notes and ignore the required readings. I now choose at least one question that relies on the required readings and not on the lecture, just to remind students (reminder-by-consequence) that they are supposed to read the required readings.

A more subtle problem arises when a question can be answered to a mediocre degree from the lecture notes, and much better from the required readings. In that case, the large majority of the class often gives the mediocre answer. It is tempting to the grader to accept the majority product as the right product.

### 3.3.8 Excessively Short Lists are Too Easy to Memorize

One safeguard against students memorizing every answer (relying on other students to generate answers for them) is that there are too many answers to memorize.

### 3.3.9 Excessively Long Lists and Lists Distributed Too Late Motivate Little Studying

If the list goes to students too late (relative to its length), the list is seen as unreasonable -- impossible -- not worth paying careful attention to.

### 3.3.10 Prioritization is not Student-Driven

An issue raised with the assessment approach used in this course is that it is seen as micromanaging the study habits of the students. The instructor boils the course down to a relatively short list of questions and the students study these (and nothing else).

I don't see this as much of a problem. If I include everything that I think is important in the class, then the students study a fairly wide range of material.

Left without guidance, students still prioritize, but their prioritization is based more on rumor and those hints (real or imagined) that they got from the instructor. To some degree, their prioritization is also based on their interests. Students study things that capture their imagination and often learn a great deal from that. The exam structure in this course does not encourage people to take long, fascinating tangents. I try to make up for this, not entirely successfully, with the assignments and bonus point opportunities.

## 4. Assignments

Appendix D provides sample assignments and grading notes.

The intent of the assignments is to give students practice exercises so they can build skills.

Students are encouraged to test in pairs and to edit each other's bug reports before filing them in the sample application's bug database, such as *IssueZilla* for *OpenOffice*.

The assignments are useful as far as they go, but they are inefficient. It takes many students three assignments before they get reasonably competent in the style of domain analysis that I teach. (They hand in Assignment 3 about 5 weeks after the start of classes.)

What we need (and are developing) are many more, simpler homework exercises that can be used for practice. My analogy is to the typical Calculus course. Students do a lot of homework, and learn many concepts quickly, much more quickly than the testing students are picking up concepts in the testing course.

## 5. Bonus Assignments

Students can collect up to 10 bonus points, bringing their maximum possible point count to 110%.

In general, I use bonus assignments to encourage students to improve their communication skills and their system administration skills.

As examples, when we test *OpenOffice*:

- One student takes responsibility for providing installation and update technical support for the class. If anyone has problems installing *OpenOffice*, they go to this student.
- Another student takes responsibility for coaching people in the mechanics of the bug tracking system.
- Another student or two might make a class-long presentation on a topic of interest. For example, a student with relevant work experience gave a presentation on failure modes and effects analysis. Another presentation might profile test tools available at sourceforge.org.

Additionally, students can earn bonus points by reporting bugs and editing bug reports.

Here are the rules I include in the syllabus on bonus point bug reports:

> **Bug Reporting:**
>
> The *OpenOffice* staff are primarily volunteers, like you. ***These people are not to be abused. Bug reports should be written in a respectful tone. If your reports are disrespectful, sarcastic, or in any other way inappropriate in tone, I will refuse to award any bonus points for any of your bug reports.*** Some of these volunteers may reject your reports unreasonably. They might reject perfectly good (bad) bugs. Or they might respond sarcastically or disrespectfully. This happens in industry too. You'll have to learn to deal with it. (But don't respond in kind.)
>
> Submitting bug reports is voluntary, but every report you submit will be reviewed and commented on by the *OpenOffice* staff. This is an important training opportunity.
>
> I will award bonus points (up to 5% of your grade can be for bug reports or bug replications) for bug reports. Here are my standards for awarding bonus points for bugs.

1. Many of the bug reports in the *IssueZilla* database don't meet my standards. Many people who work on open source projects never had training in testing. But you do have that training, and the point of this exercise is to give you experience writing bug reports at a professional level. Therefore I will hold you to a professional standard, not to the standard of a part-time volunteer. I suspect that several bugs that are sent to me will not be awarded bonus points.
2. Reports of already-reported bugs will not qualify for a bonus.
3. I will award UP TO 1 bonus point per bug report (to a maximum total of 5 points across all bugs submitted). If you developed an acceptable bug report as a group of N people, you will get 1/N points each for that bug.
4. I will not base the decision on personal demonstrations of bugs. If the bug report, as written seems unclear, confusing, or insignificant, it does not qualify. The bonus is for the report, not for the bug.
5. I will not review every bug in the database. I will only look at bugs that you ask me to look at. If I have to wade through mediocre or poor bugs from you, I will give up, even if that means skipping potentially worthy ones. Therefore, please exercise care in pointing me to your submissions. Only send me to your good ones.

**Bug Reviews:**

The O-O bug database has many bugs that have not been replicated or analyzed. They need these replicated and, often, explained in more detail.

I will award bonus points (up to 5% of your grade can be for bug reports or bug replications) for bug reviews.

1. Each well-reviewed bug is worth up to one-half bonus point (to a maximum of 5). If you do an acceptable review as a group of N people, you will get 1/2N points each for that bug.
2. I will not review every bug in the database. I will only look at bugs that you ask me to look at. If I have to wade through mediocre or poor reviews from you, I will give up, even if that means skipping potentially worthy ones. Therefore, please exercise care in pointing me to your work. Only send me to your good ones.

Assignment 2 presents bug review standards in more detail.

***Bug reports and bug reviews that are done as part of an assignment are not eligible for bonus credit.***

## 6. Quizzes

I also use occasional quizzes -- unannounced tests that are worth about 1% of the final grade each. I mark them pass/fail.

I use the quiz to focus the students' attention. For example, sometimes I want them to puzzle through a new concept in class or to apply something that we've talked about in the last few lectures. Occasionally, I use a quiz or a brief, hand-in homework assignment, to help me understand what information has been successfully conveyed to most of the class.

## 7. Closing Notes, Including Plans for Change

Overall, I think the approach of using evaluation to drive students' learning experiences has been successful. However, there are some areas for improvement in the course:

- Student performance on mid-terms is unnecessarily weak. The study guide helps those who rely on it, but too many students ignore it until their mid-term wake-up call. Next time that I teach the course, I'll try a lecture well before the midterm on study strategies (we've done that already, even

with a presentation by a previous student), that shows how I grade test questions. I'll probably start with a quiz, using a question from a prior exam, so that students will have thought intensely about the question before I show how I grade it.

- We're experimenting with rubrics. These might help some students improve the structure and approach to their assignments.
- We need more practice materials, a series of exercises that run from simple to complex, that have students work through the routine aspects of each testing technique. My lab has been working on developing these but we don't have a good enough collection yet.
- The course relies on readings and lecture notes. There isn't a course text because I haven't yet found a good course text. Probably next time I teach, I'll require Kaner / Bach / Pettichord's *Lessons Learned in Software Testing* and perhaps Whittaker's *How to Break Software* or Hendrickson's lecture notes on *Bug Hunting.* Over the long term, though, we need to develop a traditional textbook.

## Appendix A: Exam Study Guide Questions

Note: I don't include all of the following questions in every list and I change the list from year to year. I cover different topics from year to year and some of these will be irrelevant in a given year.

It's important to keep the workload manageable. Depending on your school's culture, you might make your list longer or shorter--but don't underestimate your students. Many students will rise to a challenge, especially if they believe you are genuinely interested in their work.

The exams are closed book.

### *Timing, Coverage and Difficulty of the Exam*

The questions in each section below vary in difficulty and length.

In drafting an exam, I answer each question that is a serious candidate for inclusion in the exam and clock my answer. To clock the answer, I write the answer out once, to get my thinking and structure down. Then I write a second draft and time that. (Remember, students have been drafting their answers in the course of studying for the exam, so on the exam, they are generating the Nth draft answer.) I allow students twice as long as it took me to hand write my second draft. For a 75 minute exam, I cumulate questions to total 55-65 minutes, leaving the extra 10-20 minutes for students who write slowly. For example, the exam might include 4 definitions, 4 short answers and 2 long answers. This particular exam offers 100 points worth of questions, but some of my 75 minute exams are out of 95 or 105 -- the total count is less important than the estimated time and difficulty of the complete product.

I rate questions as Easy, Medium, and Hard and drive the difficulty of the exam by the mix of the ratings.

Finally, I pick the questions in a way that reasonably represents what we covered in class. In some cases, the questions rely on explicitly required readings rather than on material we covered in the lecture. In some cases, the questions on the list cover material that I don't actually reach in time for the exam. These questions are excluded from the exam.

### *Ambiguity*

One of the advantages of circulating the questions in advance is that the students can challenge them before the exam. Surprisingly, a question might be perfectly clear to the students in one semester but ambiguous to the students in the next semester.

I encourage students to draw ambiguities to my attention. I resolve the ambiguities by sending an electronic mail message to the students. I may exclude the question from the exam if the correction came too late or the answer to the corrected question is too complex.

### *Part 1: Definitions (5 points each)*

Definitions should take 2-3 minutes each. In drafting the exam, I allow about 3 minutes per definition.

1. **Domain testing**
2. **Equivalence class**
3. **Boundary condition**
4. **Best representative**
5. **Fault vs. failure vs. defect**
6. **Function testing**
7. **Regression testing**
8. **Specification-based testing**
9. **Power of a test**
10. **Public bugs vs private bugs**
11. **Prevention costs**
12. **Appraisal costs**
13. **Internal vs. external failure costs**
14. **Oracle**
15. **Exploratory testing**
16. **Waterfall lifecycle**
17. **Lifecycle model**
18. **Evolutionary development**
19. **Line (or statement) coverage**
20. **Boundary chart**
21. **Software quality**
22. **Black box testing**
23. **Glass box / white box testing**
24. **Risk-based testing**
25. **Corner case**
26. **Finite state machine**
27. **Stochastic testing**
28. **Dumb monkey**
29. **State**
30. **Combination testing**
31. **All-pairs combination testing**
32. **Input constraints**
33. **Storage constraints**
34. **Smart Monkey**
35. **State variable**
36. **Value of a state variable**

37. **Testing project plan**
38. **Test matrix**
39. **Manual test script**
40. **Attribute to be measured**
41. **Surrogate measure**
42. **Defect arrival rate**
43. **Defect arrival rate curve (Weibull distribution)**
44. **Stress testing**
45. **Computation constraints**
46. **Output constraints**

### *Part 2: Short Answers (10 points each)*

Short answers should take about 5 minutes to answer. In planning the timing of the exam, I allow about 6 minutes per short answer question.

1. Give two examples of defects you are likely to discover and five examples of defects that you are unlikely to discover if you focus your testing on line-and-branch coverage.
2. Give three different definitions of "software error." Which do you prefer? Why?
3. Ostrand & Balcer described the category-partition method for designing tests. Their first three steps are:
    (a) Analyze
    (b) Partition, and
    (c) Determine constraints

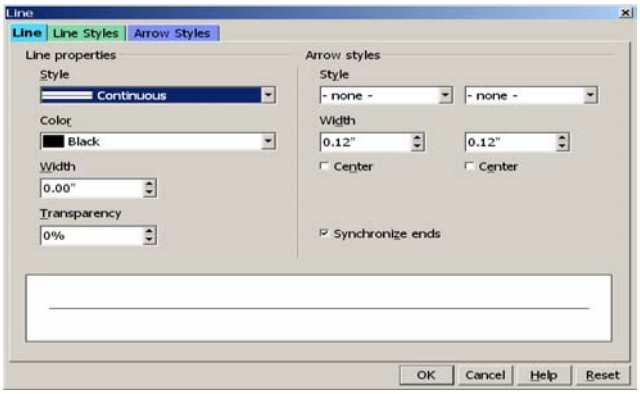
    Describe and explain these steps.
4. A program asks you to enter a password, and then asks you to enter it again. The program compares the two entries and either accepts the password (if they match) or rejects it (if they don't). You can enter letters or digits. How many *valid* entries could you test? (Please show and/or explain your calculations.)
5. A program is structured as follows:
    - It starts with a loop, the index variable can run from 0 to 20. The program can exit the loop normally at any value of the index.
    - Coming out of the loop, there is a case statement that will branch to one of 10 places depending on the value of X. X is a positive, non-zero integer. It can have any value from 1 to MaxInt.
    - In 9 of the 10 cases, the program executes X statements and then goes into another loop. If X is even, the program can exit the loop normally at any value of its index, from 1 to X. If X is odd, the program goes through the loop 666 times and then exits. In the 10th case, the program exits.

    Ignore the possibility of invalid values of the index variable or X. How many paths are there through this program? Please show and/or explain your calculations.
6. Consider a program with two loops, controlled by index variables. The first variable increments (by 1 each iteration) from -3 to 20. The second variable increments (by 2 each iteration) from 10 to 20.

The program can exit from either loop normally at any value of the loop index. (Ignore the possibility of invalid values of the loop index.)

- If these were the only control structures in the program, how many paths are there through the program?
  - If the loops are nested
  - If the loops are in series, one after the other
- If you could control the values of the index variables, what test cases would you run if you were using a domain testing approach?
- Please explain your answers with enough detail that I can understand how you arrived at the numbers.

7. List and briefly explain three strengths of the waterfall lifecycle.
8. List and briefly explain three strengths of the evolutionary lifecycle.
9. Describe the characteristics of a good scenario test.
10. List and explain four claimed strengths of manual scripted tests and four claimed weaknesses.
11. List (and briefly describe) four different missions for a test group. How would your testing strategy differ across the four missions?
12. Distinguish between using code coverage to highlight what has not been tested from using code coverage to measure what has been tested. Describe some benefits and some risks of each type of use. (In total, across the two uses, describe three benefits and three risks.)
13. In lecture, I used a minefield analogy to argue that variable tests are better than repeated tests. Provide five counter-examples, contexts in which we are at least as well off reusing the same old tests.
14. List and describe five different dimensions (different "goodnesses") of "goodness of tests".
15. Describe two difficulties and two advantages of state-machine-model based testing.
16. Can you represent a state machine graphically? If so, how? If not, why not?
17. Explain the relationship between graph traversal and our ability to automate state-model-based tests.
18. Compare and contrast the adjacency and incidence matrices. Why would you use one instead of the other?
19. What does it tell us about the system under test if the model of system (accurately) shows weak connectivity?
20. What is the state explosion problem and what are some of the ways that state-model-based test designers use to cope with this problem?
21. Consider the variable, "synchronize ends" in this dialog from *OpenOffice* Presentation. "Synchronize ends" can be checked or unchecked. Are these two values distinct? Justify your answer.




Part 3: Long Answers (20 points each)

1. Imagine testing a date field. The field is of the form MM/DD/YYYY (two digit month, two digit day, 4 digit year). Do an equivalence class analysis and identify the boundary tests that you would run in order to test the field. (Don't bother with non-numeric values for these fields.)

2. I, J, and K are signed integers. The program calculates

   K = IntegerPartOf (SquareRoot (I*J))

   For this question, consider only cases in which you enter integer values into I and J. Do an equivalence class analysis from the point of view of the effects of I and J (jointly) on the variable K. Identify the boundary tests that you would run (the values you would enter into I and J).

   *NOTE: Variations on this question consider:*

   K = I * J

   K = I / J

3. Ostrand & Balcer described the category-partition method for designing tests. Their first three steps are:

   1. Analyze
   2. Partition, and
   3. Determine constraints

   Apply their method to this function:

   I, J, and K are unsigned integers. The program calculates K = IntegerPartOf (SquareRoot (I*J)).

   For this question, consider only cases in which you enter integer values into I and J.

   Do an equivalence class analysis from the point of view of the effects of I and J (jointly) on the variable K.

4. The Spring and Fall changes between Standard and Daylight Savings time creates an interesting problem for telephone bills. *Focus your thinking on the complications arising from the daylight savings time transitions.* Create a table that shows risks, equivalence classes, boundary cases, and expected results for a long distance telephone service that bills calls at a flat rate of $0.05 per minute. Assume that the chargeable time of a call begins when the called party answers, and ends when the calling party disconnects.

5. Describe a traceability matrix.
   - How would you build a traceability matrix for the word processor in *OpenOffice*?
   - What is the traceability matrix used for?

- What are the advantages and risks associated with driving your testing using a traceability matrix?
- Give examples of advantages and risks that you would expect to deal with if you used a traceability matrix for the word processor. Answer this in terms of two of the main features of the word processor. You can choose which two features.

6. What is regression testing? What are some benefits and some risks associated with regression testing? Under what circumstances would you use regression tests?
7. Why is it important to design maintainability into automated regression tests? Describe some design (of the test code) choices that will usually make automated regression tests more maintainable and explain (briefly) why each choice increases maintainability.
8. Suppose that you find a reproducible failure that doesn't look very serious.
   - Describe three tactics for testing whether the defect is more serious than it first appeared.
   - As a particular example, suppose that the display got a little corrupted (stray dots on the screen, an unexpected font change, that kind of stuff) in *OpenOffice*'s word processor when you drag the mouse across the screen. Describe three follow-up tests that you would run, one for each of the tactics that you listed above.
9. Imagine testing a file name field. For example, go to an Open File dialog, you can enter something into the field. Do a domain testing analysis: List a risk, equivalence classes appropriate to that risk, and best representatives of the equivalence classes. For each test case (use a best representative), briefly explain why this is a best representative. Keep doing this until you have listed 12 best-representative test cases.
10. Suppose that you had access to the *OpenOffice* source code and the time / opportunity to revise it. Suppose that you had access to the source code and that you decided to do a diagnostics-based high volume automated test strategy to test *OpenOffice* Presentation's treatment of copying and pasting of slides.
    - What diagnostics would you add to the code, and why?
    - Describe 3 potential defects, *defects that you could reasonably imagine would be in the software that handles copy/paste of slide*, that would be easier to find using a diagnostics-based strategy than by using a lower-volume strategy such as exploratory testing, spec-based testing, or domain testing.
11. Consider testing the *OpenOffice* function that lets you save a document in HTML format.
    - How would you develop a list of risks for this capability? (If you are talking to people, who would you ask and what would you ask them?) (If you are consulting books or records or databases, what are you consulting and what information are you looking for in it?)
    - Why is this a good approach for building a list of risks?
    - List 10 risks associated with this function.
    - For each risk, briefly (very briefly) describe a test that could determine whether there was an actual defect.

    *Note: In practice, I probably won't ask for 10 risks on the exam. On the exam, I'll ask for fewer, perhaps five or seven. The review questions ask for more because I would rather have students think of a longer list and then give their best several items on the exam. If I told them to prepare a list of seven and then asked for seven on the exam, many students would struggle to remember the last two or three examples. If they prepare for 10, they are more likely to be able to give 7 on the exam than if I ask them to prepare for 7.*
12. Consider testing the *OpenOffice* function that lets you enter data into a table in the word processor.

- How would you develop a list of risks for this capability? (If you are talking to people, who would you ask and what would you ask them?) (If you are consulting books or records or databases, what are you consulting and what information are you looking for in it?)
- Why is this a good approach for building a list of risks?
- List 10 risks associated with this function.
- For each risk, briefly (very briefly) describe a test that could determine whether there was an actual defect.

13. Consider testing the *OpenOffice* function that lets you enter data into a spreadsheet on a Presentation slide.
    - How would you develop a list of risks for this capability? (If you are talking to people, who would you ask and what would you ask them?) (If you are consulting books or records or databases, what are you consulting and what information are you looking for in it?)
    - Why is this a good approach for building a list of risks?
    - List 10 risks associated with this function.
    - For each risk, briefly (very briefly) describe a test that could determine whether there was an actual defect.
14. Imagine that you were testing the feature, Save With Password in the *OpenOffice* word processor.
    - Explain how you would develop a set of scenario tests that test this feature.
    - Describe a scenario test that you would use to test this feature.
    - Explain why this is a particularly good scenario test.
15. Imagine that you were testing the feature, Save With Password in the *OpenOffice* word processor.
    - Explain how you would develop a set of soap operas that test this feature.
    - Describe one test that might qualify as a soap opera.
    - Explain why this is a good soap opera test.
16. Imagine that you were testing the feature, Insert Object in the *OpenOffice* Presentation module.
    - Explain how you would develop a set of scenario tests for this feature.
    - Describe a scenario test that you would use to test this feature.
    - Explain why this is a particularly good scenario test.
17. Define a scenario test and describe the characteristics of a good scenario test. Imagine developing a set of scenario tests for the Outlining feature of the word processing module of *OpenOffice*. What research would you do in order to develop a series of scenario tests for Outlining? Describe two scenario tests that you would use and explain why each is a good test. Explain how these tests would relate to your research.
18. (The following statement is not true, but pretend it is for exam purposes.) Sun has just announced that they will include email support in Release 2.0 of the StarOffice product, which they will ship in November, 2003. They announce that in the first implementation, the lists of new and saved messages will be displayed in spreadsheet format, based on the existing spreadsheet code.
    - List and briefly explain 5 risk factors that you would expect to find associated with the spreadsheet interface to the email database. (Refer to Amland's paper for discussion of risk factors.)
    - For each risk factor, predict 2 defects that could arise in the spreadsheet interface part of the 2.0 project. By "predict", I mean name and describe the potential defect, and explain why that particular risk factor makes this defect more likely.

19. We are going to do some configuration testing on the *OpenOffice* word processor. We want to test it on
    - Windows 95, 98, and 2000 (the latest service pack level of each)
    - Printing to an HP inkjet, a LexMark inkjet, and a Xerox laser printer
    - Connected to the web with a dial-up modem (28k), a DSL modem, and a cable modem
    - With a 640x480 display and a 1024x768 display
        - How many combinations are there of these variables?
        - Explain what an all-pairs combinations table is
        - Create an all-pairs combinations table. (Show at least some of your work.)
        - Explain why you think this table is correct.
20. Imagine that you are an external test lab, and Sun comes to you with *OpenOffice*. They want you to test the product. When you ask them what test documentation they want, they say that they want something appropriate but they are relying on your expertise. To decide what test documentation to give them, what questions would you ask (up to 7 questions) and for each answer, how would the answer to that question guide you?
21. (The following statement is not true, but pretend it is for exam purposes.) *OpenOffice*.Org has just announced that they will include built-in support for digital cameras and digital video recorders in the Word Processing module. They announce that the word processor will have features that allow users to download single images and digitally stored movies from supported cameras and recorders. Users will be able to place a picture or movie in a document and view the picture or a movie while editing the document and they will be able to see the picture and the starting frame of the movie on the document printout. Users will be able to edit the pictures or movies, such as by cropping them, stretching or resizing them, changing their colors, or imposing text or other graphics on the image.
    - List and briefly explain 5 risk factors that you would expect to find associated with the new support for digital pictures and video. (Refer to Amland's paper for discussion of risk factors.)
    - For each risk factor, predict 2 defects that could arise in the support for digital pictures and video. By "predict", I mean name and describe the potential defect, and explain why that particular risk factor makes this defect more likely.
22. The oracle problem is the problem of finding a method that lets you determine whether a program passed or failed a test.

    Suppose that you were doing automated testing of spell-checking in the *OpenOffice* word processor.

    Describe three different oracles that you could use or create to determine whether this feature was working. For each of these oracles,
    - identify a bug that would be easy to detect using the oracle. Why would this bug be easy to detect with this oracle? and
    - identify another bug that your oracle would be more likely to miss. Why would this bug be harder to detect with this oracle?
23. You are using a high-volume random testing strategy for the *OpenOffice* word processing program. You will evaluate results by using an oracle.
    - Consider testing the spell-checking feature using oracles. How would you create an oracle (or group of oracles)? What would the oracle(s) do?
    - Now consider the placement of footnotes at the bottom of the page. How would you create an oracle (or group of oracles) for this? What would the oracle(s) do?

- Which oracle would be more challenging to create or use, and why?

24. You are using a high-volume random testing strategy for the *OpenOffice* Presentation program will evaluate results by using an oracle.
    - Consider inserting a spreadsheet into a slide. When you enter values into the spreadsheet, you can insert functions into the cells of the spreadsheet. Think about testing those functions using oracles. How would you create an oracle (or group of oracles)? What would the oracle(s) do?
    - Consider entering a chart into a slide. Once you have entered data into the chart, *OpenOffice* draws the chart. Think about testing the chart as drawn to determine whether it properly shows the chart data. How would you create an oracle (or group of oracles)? What would the oracle(s) do?
    - Which oracle would be more challenging to create or use, and why?
25. Imagine that you were testing the spellchecking feature of the *OpenOffice* Word Processor. Describe four examples of each of the following types of attacks that you could make on this feature, and for each one, explain why your example is a good attack of that kind.
    - Input constraint attacks
    - Output constraint attacks
    - Storage constraint attacks
    - Computation constraint attacks.

    (Notes for you while you study. Refer to Jorgensen / Whittaker's paper on how to break software. Don't give me two examples of what is essentially the same attack. In the exam, I will not ask for all 16 examples, but I might ask for 4 examples of one type or two examples of two types, etc.)
26. Imagine that you were testing the *OpenOffice* word processor feature that lets you save a document in HTML format.

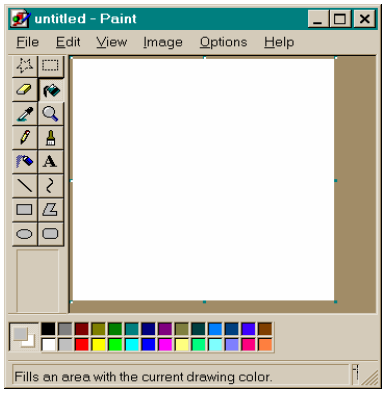
    Describe four examples of each of the following types of attacks that you could make on this feature, and for each one, explain why your example is a good attack of that kind.
    - Input constraint attacks
    - Output constraint attacks
    - Storage constraint attacks
    - Computation constraint attacks.

    (Notes for you while you study. Refer to Jorgensen / Whittaker's paper on how to break software. Don't give me two examples of what is essentially the same attack. In the exam, I will not ask for all 16 examples, but I might ask for 4 examples of one type or two examples of two types, etc.)
27. What is the Defect Arrival Rate? Some authors model the defect arrival rate using a Weibull probability distribution. Describe this curve and briefly explain three of the claimed strengths and three of the claimed weaknesses or risks of using this curve.
28. The following group of slides are from Windows Paint 95. Please don't spend your time replicating the steps or the bug. (You're welcome to do so if you are curious, but I will design my marking scheme to not give extra credit for that extra work.)

    Treat the steps that follow as fully reproducible. If you go back to ANY step, you can reproduce it.

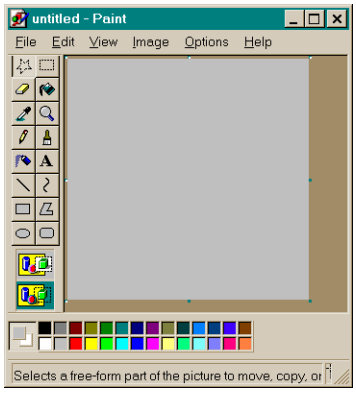
    For those of you who aren't familiar with paint programs, the essential idea is that you lay down dots. For example, when you draw a circle, the result is a set of dots, not an object. If you were using a draw program, you could draw the circle and then later select the circle, move it, cut it, etc. In a paint program, you cannot select the circle once you've drawn it. You can select an area that includes the dots that make up the circle, but that area is simply a bitmap and none of the dots in it have any relationship to any of the others.

***I <u>strongly</u> suggest that you <u>do this question last</u> because it can run you out of time if you have not thought it through carefully in advance.***



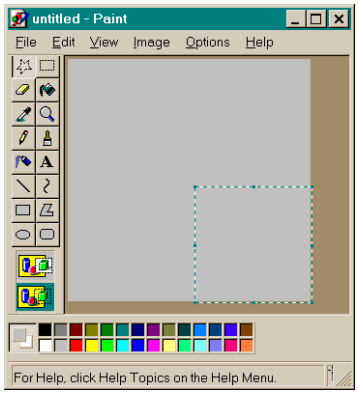

Here's the opening screen. The background is white. The first thing that we'll do is select the Paint Can

We'll use this to lay down a layer of grey paint on top of the background. Then, when we cut or move an area, we'll see the white background behind what was moved.



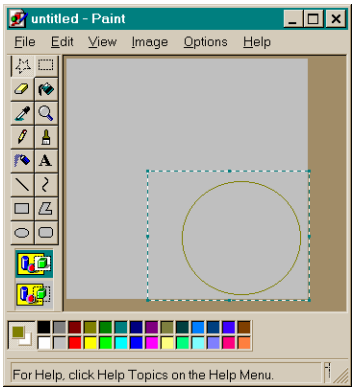

Here's the screen again, but the background has been painted gray.

The star in the upper left corner is a freehand selection tool. After you click on it, you can trace around any part of the picture. The tracing selects that part of the picture. Then you can cut it, copy it, move it, etc.



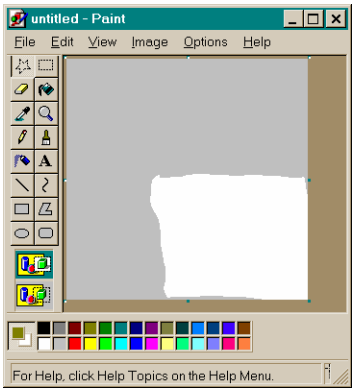

This shows an area selected with the freehand selection tool. The bottom right corner is selected. (The dashed line surrounds the selected area.)

*NOTE: The actual area selected might not be perfectly rectangular. The freehand tool shows a rectangle that is just big enough to enclose the selected area. For our purposes, this is not a bug. This is a design decision by Microsoft.*



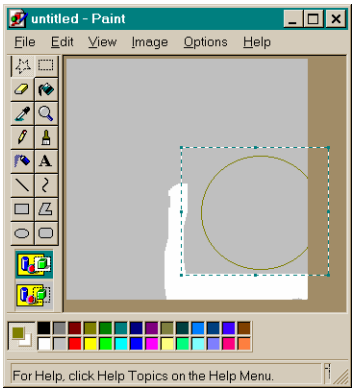

Next, we'll draw a circle (so you can see what's selected), then use the freehand select tool to select the area around it.

When you use the freehand selection tool, you select an area by moving the mouse. The real area selected is not a perfect rectangle. The rectangle just shows us where the selected area is.

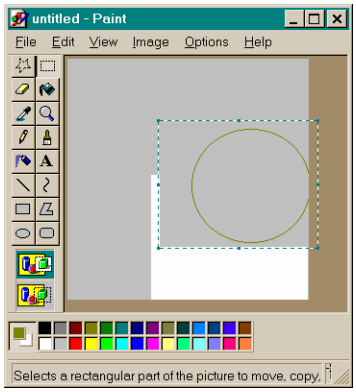



Now we cut the selection. (To do this, press Ctrl-X.)

The jagged border shows exactly the area that was selected.




Next, exit the program, restart it, color the background grey, draw the circle, select the area around the circle and drag it up and to the right.




This time, we'll try the Rectangular Selection tool.

With this one, if you move the mouse to select an area, the area that is actually selected is the smallest rectangle that encloses the path that your mouse drew.

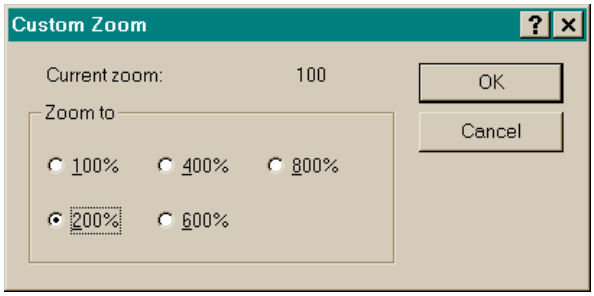
So, exit the program, start it up, color the background, draw a circle, click the Rectangular Selection tool, select the area around the circle and move it up. It works.

This works.

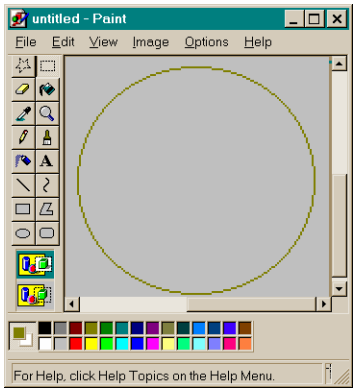
Well, this was just too boring, because everything is working. When you don't find a bug while testing a feature, one tactic is to keep testing the feature but combine it with some other test.

In this case, we'll try Zooming the image. When you zoom 200%, the picture itself doesn't change size, but the display doubles in size. Every dot is displayed as twice as tall and twice as wide.

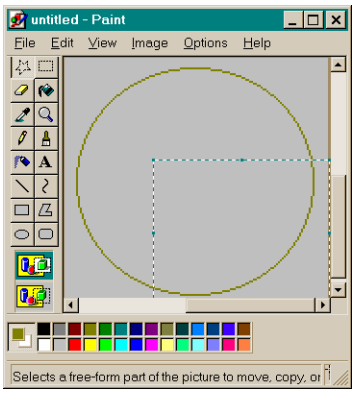
So we exit the program, start it up, color the background grey, draw the circle, and then . . .




Bring up the Custom Zoom dialog, and select 200% zoom, click OK.



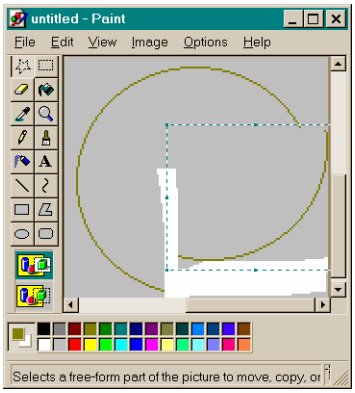
It worked. The paint area is displayed twice as tall and twice as wide. We're looking at the bottom right corner. To see the rest, we could move the scroll bars up or left.




So, we select part of the circle using the freehand selection tool. We'll try the move and cut features.

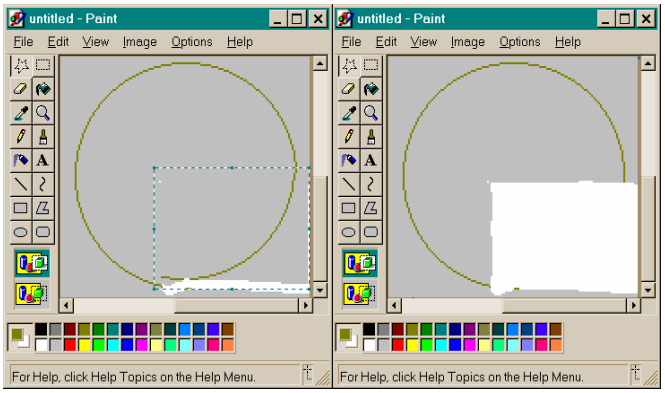
Cutting fails.

When we try to cut the selection, the dashed line disappears, but nothing goes away.



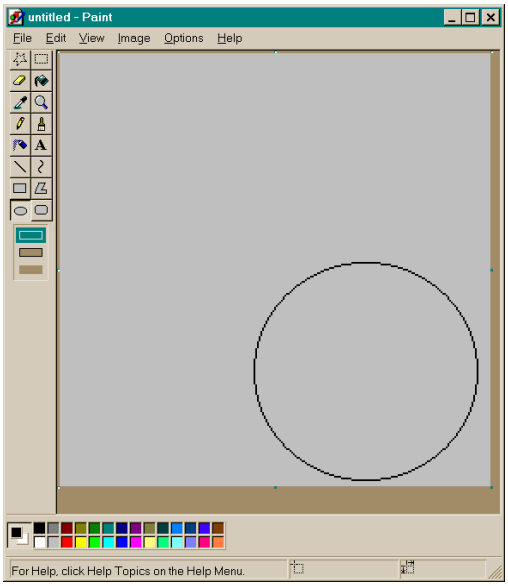

Exit the program, start again, color the background, draw the circle, zoom to 200%, select the area.

Drag the area up and to the right. It works.

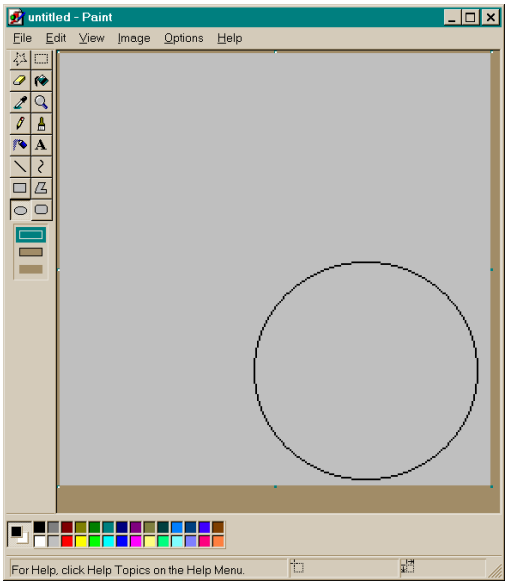



Exit the program, start again, color the background, draw the circle, zoom to 200%, select the area.

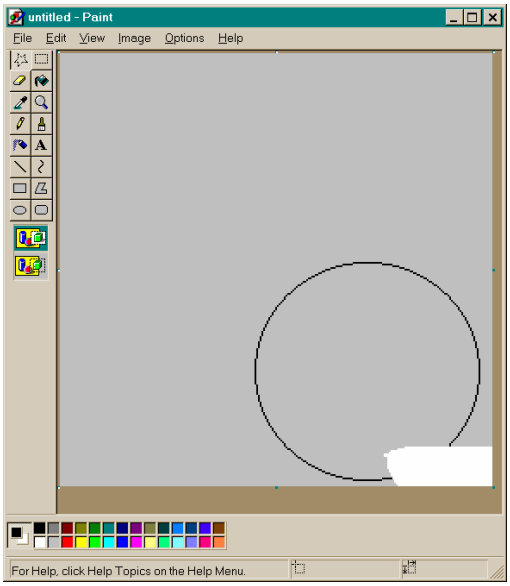
Now try this. Select the area and move it a bit. THEN press Ctrl-C to cut. This time, cutting works.




Exit the program, start again, color the background, draw the circle, zoom to 200%, and this time, *grow the window* so you can see the whole drawing area.




Now, select the circle. That seems to work.




But when you press Ctrl-X to cut the circle, the program cuts the wrong area.

Now, write a bug report. I want two sections:

- The Problem summary (or title)
- The Problem Description (how to reproduce the problem)

Additionally, please describe three follow-up tests that you would run with this bug

# APPENDIX B: The Study Guide that I Give Students

- **Your test will be sampled from a list of questions that I give you in advance. Those questions include definitions, short answer and long answer questions. Solutions might require short essays, mathematical derivations, or code fragments.**
- **You may not use any reference materials during the test (closed book test).**
- **I recommend that you study with one or more partners. 3-5 people is a good sized group. 8 is too many.**
- **The best way to prepare for these tests is to attempt each question on your own. Your first attempt for each question should be open with no time limit. Check the lecture notes AND the required readings.**
- ***AFTER* you have tried your own answers, compare notes with your friends.**
- **Working with others will help you discover and work through ambiguities *before* you take the test. If a question is unclear, send me a note before the test. If you tell me early enough, I can fix it. If a question takes too long to answer, send me a note about that too.**
- **When you write the test, keep in mind that I am reading your answer with the goal of finding reasons to give you points:**
    - **If the question contains multiple parts, give a separate answer for each part.**
    - **If a question asks about "some", that means at least two. I normally expect three items in response to a "some". Similarly if the question asks for a list, I am expecting a list of at least three.**
    - **Be aware that different words in questions have different meanings. For example, each of the following words calls for a different answer: identify, list, define, describe, explain, compare, contrast. If I ask you to list and describe some things, give me a brief identification (such as a name) of each and then a description for each one.**
    - **If you find it hard to define or describe something, try writing your answer around an example.**
    - **If you are asked to describe the relationship among things, you might find it easiest to work from a chart or a picture. You are not required to use a diagram or chart (unless I ask for one), but feel free to use one if it will help you get across your answer.**
    - **If I ask you to describe or define something that is primarily visual (such as a table or a graph), your answer will probably be easier to write and understand if you draw an example of what you are defining or describing.**
- **If I ask you for the result of a calculation, such as the number of paths through a loop, show your calculations or explain them. Let me understand how you arrived at the answer.**
- **If I ask you to analyze something according to the method described in a particular paper or by a particular person, I expect you to do it their way. If I ask you to describe their way, do so. If I ask you to apply their way, you don't have to describe it in detail, but you have to do the things they would do in the order they would do them, and you have to use their vocabulary to describe what you are doing.**
- **The test is time-limited—75 minutes. Plan to spend no more than 4 minutes on any definition, no more than 10 minutes on any short answer, and no more than 15 minutes on any long answer. Spend less on most answers. Suppose the test has 4 definitions (20 points), 2 short answers (20 points), and 3 long answers (60 points). You should plan to spend, on average, about 3 minutes per definition, about 8 minutes per short answer, and about 12 minutes per long answer (total = 64 minutes). Use the remaining 11 minutes to check your work.**
- **Pick the order of your answers. If you spend too long on definitions, start writing your long answers first. If you are nervous, start with the questions you find easiest to answer.**

**Study Guide Suggestions -- Page 2**

- **Be aware of some factors that, in general, bias markers. These are generalizations, based on research results. I try, of course, to be unbiased, but it's a good idea to keep these in mind with ANY grader for ANY exam:**
  - ***Exams that are hard to read tend to get lower grades*. Suggestions: Write in high contrast ink (such as black, medium). Write in fairly large letters. Skip every second line. Don't write on the back of the page. If your writing is illegible, print. *If I can't read something you wrote, I will ignore it.***
  - ***Start a new question on a new page*. More generally, *leave lots of space on the page.* This gives you room to supplement or correct your answer later (when you reread the exam before handing it in) and it gives me room to write comments on the answer, and it makes the answer easier to read.**
  - ***Answers that are well-organized are more credible*. Suggestions: If the question has multiple parts, start each part on a new line, and identify each part at its start. In a list, start each list item on a new line—maybe bullet the list. For example, consider the question: "What is the difference between black box and white box testing? Describe the advantages and disadvantages of each." You could organize this with five headings:**
    - **Difference between black and white**
    - **Advantages of black box**
    - **Disadvantages of black box**
    - **Advantages of white box**
    - **Disadvantages of white box**
  - ***Don't answer what has not been asked*. For example, if I ask you to define one thing, don't define that and then give me the definition of something related to it. If you do, (a) I won't give you extra credit, (b) I'll think that you don't know the difference between the two things, and (c) if you make a mistake, I'll take off points.**
  - ***Give the number of items requested.* For example, if I ask for two scenario tests, don't give one or three. If you give one, you miss points. If you give three, I will either grade the first two and ignore the third (this is my normal approach) or grade the first two that I happen to read (whatever their order on the page) and ignore the third. I will never read the full list and grade what I think are the best two out of three.**

**Additional points to consider.**

- **Beware of simply memorizing some points off a slide. If I think you are giving me a memorized list without understanding what you are writing, I will ruthlessly mark you down for memorization errors. In general, if you are repeating a set of bullet points, write enough detail for them that I can tell that you understand them.**
- **Use a good pen. Lawyers and others who do lots of handwriting buy expensive fountain pens for a reason. The pen glides across the page, requiring minimal pressure to leave ink. If you use a fountain pen, I suggest a medium point (write large) to avoid clogging. Also try gel pens or rollerballs. Get one that you can write with easily, to avoid writer's cramp. Basic ballpoints are very hard on you. Look at how tightly you hold it and feel how hard you press on the page.**

# Appendix C Grading the Exam

Here are examples of how I have graded exam questions. These notes are a bit cleaned up, to make them understandable to another reader. However, I've left in several discussions that I inserted in my grading notes at the time that I graded the question.

- Most of these are about how to grade the question
- Others are about what will have to be taught in the future, or provided in reference materials, to help students achieve better grades next time.
- I often include reference material or lists in my grading notes. These are often there simply to jog my memory. Sometimes, however, they are there to remind me of what I have to treat as acceptable. The problem is that students, relying on reputable sources (including other courses) will give answers that, in my humble opinion, are mistaken or ridiculous. I will accept many of these answers even though I disagree with them, but to maintain grading consistency, I need a list of what I will accept and what I will not accept. Otherwise, my tolerance of some of these answers will vary too much with my mood.

The critical features of my grading structure (what you'll always see in my working notes) are:

- Table format, with one column for each point-deserving type of information. (One row per student.) For all but the most trivial questions, I actually fill in the table for each question. This gives me detailed information about each student's performance on each question, which I use to good advantage when a student comes to me to question her grade.
- For most questions, including all complex questions, I show the points available within the column on the column itself.
- An outline of a sufficient answer.
- On a complex question, I will often paste in a list or discussion from lecture notes or one of the readings.

Suppose that a question is worth 20 points.

- In the columns, I might allow up to 30 points. This is primarily because different people legitimately approach the same question in different ways and so my grading structure has to allow for this.
- No matter what the student's total point count is, I sometimes reserve the 20$^{th}$ point for style and organization. That is, you can get 19/20 based on the standard point count, but I won't give a perfect grade unless I think the answer is well written. This is an especially common decision when the total of available points is beyond 20, and so a disorganized answer that covers lots of ground would otherwise get a perfect grade.
- Also, some questions are just too hard. I might allow 1 or 2 points merely for attempting the question.
- One of my frequent columns is "clue." This is a source of discretionary points. I define the discretionary rule on a question-by-question basis. Some examples:
    - If I allow up to 1 point for Clue, the default is 0 points. If the student's answer shows more insight than I think is reflected in the point count for the answer, I will raise the grade by 0.5 or 1 points.
    - If I allow up to 2 points for Clue, the default is probably 0 points, but the most common score is probably 1 point.

## Definitions 5 points each

| ***Power of a test*** | Probability notion | If bug is there | Ability to reveal bug | Stat comparison | Grade |
|---|---|---|---|---|---|
| | | | | | |

Power of a test.

If two tests are potentially capable of exposing the same type of defect, one test is more powerful if it is more likely to expose the defect.

- likelihood of revealing (or ability to reveal) a defect if the defect is there <I give a max of 3.5/5 if the answer doesn't point out that this depends on whether the bug is there.>
- if two tests can reveal the same defect equally well, the more powerful test can also reveal other defects
- analogous to the concept of statistical power

| ***Exploratory testing*** | Test and learn in parallel | about product, risks, market, test methods | Don't follow pre-existing detailed plan | Whittaker's attacks | Other | Final / 5 |
|---|---|---|---|---|---|---|
| | | | | | | |

***Exploratory Testing*** involves simultaneous learning, testing, evaluating, planning, and reporting.

Alternative answer: simultaneous testing and learning, plus mention of Whittaker's attacks. We discuss *How to Break Software* in class. In a different class, the appropriate source of examples might come from Hendrickson's *Bug Hunting* slides, etc.

| ***Storage constraints*** | | | | | | |
|---|---|---|---|---|---|---|

[Note to reader: the concept of "storage constraints" comes from a paper by Whittaker & Jorgenson, that was expanded in Whittaker's book, *How to Break Software.* We look at four fundamental types of constraints, input constraints, output constraints, storage constraints, and computational constraints. Many bugs are caused by a programmer's failure to consider the possibility of violation of one of these types of constraints.]

I'd like to see something about data structures. I expect to see discussion of limitation of storage in terms of the types of data that are stored or the place of storage, not input/output/computation overflows. The attacks on storage are attacks on the data structure, *how* the data is stored.

***Future notes: We need the additional details available Why Software Fails for this to be a good question.***

From James Whittaker and Alan Jorgensen's (2000) paper, *How to Break Software*

> "Data is the lifeblood of software; if you manage to corrupt it, the software will eventually have to use the bad data and what happens then may not be pretty. It is worthwhile to understand how and where data values are established.
>
> "Essentially, data is stored either by reading input and then storing it internally or by storing the result of some internal computation. By supplying input and forcing computation, we enable data flow through the application under test. The attacks on data follow this simple fact as outlined in attacks 12-14. However, without access to the source code, many testers do not bother to consider these attacks. We believe, though, that useful testing can be done even though specifics of the data implementation are hidden. We like to tell our students to practice "looking through the interface." In other words, take note of what data is being stored while the software system is in use. If data is entered on one screen and visible on

another, then it is being stored. Information that is available at any time is being stored.

"Some data is easy to see. A table structure in a word processor is one such example in which not only the data but the general storage mechanism is displayed on the screen. Some data is hidden behind the interface and requires analysis to discover its properties.

"Once the nature of the data being stored is understood, try to put yourself in the position of the programmer and think of the possible data structures that might be used to store such data. The more that programming and data structures are understood, the easier it will be to execute the following attacks. The more completely you understand the data you are testing, the more successful the attacks will be at finding bugs.

**"Twelfth attack: Apply inputs using a variety of initial conditions**

Inputs are often applicable in a variety of circumstances. Saving a file, for example, can be performed when changes have been made, and it can also be performed when no changes have been made. Testers are wise to apply each input in a number of different circumstances to account for the many such interactions that users will encounter when using the application.

**"Thirteenth attack: Force a data structure to store too many/too few values**

"There is an upper limit on the size of all data structures. Some data structures can grow to fill the capacity of machine memory or hard disk space and others have a fixed upper limit. For example, a running monthly sales average might be stored in an array bounded at 12 or fewer entries, one for each month of the year.

"If you can detect the limits on a data structure, try to force too many values into the structure. If the number is particularly large, the developer may have been sloppy and not programmed an error case for overflow.

"Special attention should be paid to structures whose limits fall on the boundary of data types 255, 1023, 32767 and so on. Such limits are often imposed simply by declaration of the structure's size and very often lack an overflow error case.

"Underflow is also a possibility and should be tested as well. This is an easy case, requiring only that we delete one more element than we add. Try deleting when the structure is empty, then try adding an element and deleting two elements and so on. Give up if the application handles 3 or 4 such attempts.

**"Fourteenth attack: Investigate alternate ways to modify internal data constraints**

"The phrase "the right hand knoweth not what the left hand doeth" describes this class of bugs. The idea is simple and developers leave themselves wide open to this attack; in most programs there are lots of ways to do almost anything. What this means to testers is that the same function can be invoked from numerous entry points, each of which must ensure that the initial conditions of the function are met.

"An excellent example of this is the crashing bug one of our students found in PowerPoint, regarding the size of a tabular data structure. The act of creating the table is constrained to 25×25 as the maximum size. However, one can create such a table, then add rows and columns to it from another location in the program—crashing the application. The right hand knew better than to allow a 26×26 table but the left hand wasn't aware of the rule."

| ***Equivalence class*** | | | | | |
|---|---|---|---|---|---|
| | | | | | |

Two tests are members of the same equivalence class if you expect the same results from each. Tests are equivalent with respect to a theory of error. Two tests might be equivalent relative to one potential failure and entirely different with respect to a different potential failure.

## Short Answers 10 points each

| | Nested analysis | Series analysis | First variable -3,0,20 | Second variable 10,20 | Used invalid values | Total |
|---|---|---|---|---|---|---|
| <Points> | 4 | 3 | 2 | 2 | ignore | 10 |
| | | | | | | |

**Consider a program with two loops, controlled by index variables. The first variable increments (by 1 each iteration) from -3 to 20. The second variable increments (by 2 each iteration) from 10 to 20. The program can exit from either loop normally at any value of the loop index. (Ignore the possibility of invalid values of the loop index.)**

- **If these were the only control structures in the program, how many paths are there through the program?**
  - **If the loops are nested**
  - **If the loops are in series, one after the other**
- **If you could control the values of the index variables, what test cases would you run if you were using a domain testing approach?**
- **Please explain your answers with enough detail that I can understand how you arrived at the numbers.**

***Analysis***

- The first variable has 24 possible values (-3, -2, -1, 0, 1, ..., 20)
- Second variable has 6 possible values (10, 12, 14, 16, 18, 20)

a) Analysis if loops are nested.

Suppose loop 1 was 1, 2, 3, Suppose loop 2 was 4,5

Loop 1 with loop 2 inside it =

2 one through loop (1,4) (1,5)

4 twice through loop (1,4,2,4), (1,4,2,4,5), (1,4,5,2,4), (1,4,5,2,4,5)

8 three through loop (1,4,2,4,3,4), (1,4,2,4,3,45), (1,4,2,45,3,4), (1,4,2,45,2,45), (1,45,2,4,3,4), (1,45,2,4,3,45), (1,45,2,45,3,4), (1,45,2,45,3,45)

Illustrates the general rule:

If N1 = number of values of the outer loop and N2 = number of values of the inner loop,

Number of paths = sum (i=1 to N1) N2^i

Example 2 + 2*2 + 2*2*2 for our sample loop

The sum is therefore (sum)(i=1-to-24) 6^i

| | |
|---|---|
| 1 | 6 |
| 2 | 36 |
| 3 | 216 |
| 4 | 1296 |
| 5 | 7776 |

| | |
|---|---|
| 6 | 46656 |
| 7 | 279936 |
| 8 | 1679616 |
| 9 | 10077696 |
| 10 | 60466176 |
| 11 | 362797056 |
| 12 | 2176782336 |
| 13 | 13060694016 |
| 14 | 78364164096 |
| 15 | 4.70185E+11 |
| 16 | 2.82111E+12 |
| 17 | 1.69267E+13 |
| 18 | 1.0156E+14 |
| 19 | 6.0936E+14 |
| 20 | 3.65616E+15 |
| 21 | 2.1937E+16 |
| 22 | 1.31622E+17 |
| 23 | 7.8973E+17 |
| 24 | 4.73838E+18 |
| | 5.68606E+18 |

b) Analysis if the loops are in series

- Total = 24 x 6 = 144 paths
- First variable -3, 0, 20 (Ignore the possibility of invalid values of the loop index)
- Second variable 10, 20

[Note to the reader: Students who handled the nested analysis well handled everything else well. In contrast, some students who correctly counted the variables' values, figured out the series, and seemed to handle the material reasonably comfortably, blew the nested analysis.

- If the nested analysis is done well, it probably deserves more than 4 points out of 10 -- but the student doesn't need the points.
- If the nested analysis is not done well, the 4 point allowance serves as a cap on the damage this part of the question can do to the grade for the full question.]

| | Not tested | Measure | Benefit N | Risk N | Benefit M | Risk M | |
|---|---|---|---|---|---|---|---|
| | 2 | 2 | 1 each | 1 each | 1 each | 1 each | |
| | | | | | | | |

**Distinguish between using code coverage to highlight what has not been tested from using code coverage to measure what has been tested. Describe some benefits and some risks of each type of use. (In total, across the two uses, describe three benefits and three risks.)**

---

***Grading notes:***

*Emphasis on what has not been tested:* you are looking for such things as blind spots in the testing, or reality check on the process or on the projected ship date. There is no necessary claim that this is a valid progress measure. You are merely identifying a set of tasks that have not been done.

*Emphasis on measurement:* Assumption is that coverage is a valid measure of testing process. You are looking for status check or productivity of staff member or group or nearness to completion.

*Benefits and risks of highlighting the negative:*

*Benefits:*

- reveal problems with the testing process
- reveal weaknesses or blind spots of the testing strategy
- reveal the overall utility of a collection of testing artifacts (no point maintaining a large test suite that achieves only 2% coverage)
- reveal impossibility of a ship date

*Risks:*

- blaming tone
- might persuade managers to rely on this, in a way that encourages them to use coverage as a measurement of progress later.

*Benefits and risks of measurement:*

*Benefits:*

- for exam purposes, I will accept the notion that we can check nearness to completion of testing with this measure
- can note progress against a plan
- can report results in a way managers are used to hearing
- one factor in a ship decision

*Risks:*

- encourages people to do things that are counted rather than things that are more likely to reveal problems
- discourages people from tests (e.g. configuration tests) that are not counted
- gives mgmt the false perception of progress because it omits key tests that are not counted.
- Encourages premature release of the product
- Discriminates against testers who do tests that are "redundant" under this measure

[References: Marick's writings on coverage, such as Classic Testing Mistakes and How to Misuse Code Coverage.]

| 7 ***Month Field*** | Boundary chart * | Realize Inter-depence | 28 day month | 29 day leap year | 30 day | 31 day | Month Range | Year Range | Invalid pairings | Invalid max min | Invalid chars (shotgun penalty -- they should **not** appear) | Tests of full field (all 3 values in 1 test) | Overall analysis | Total / 24 | Grade / 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Points | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | -2 | 4 | 2 | 24 | 10* |

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | |

* = discretionary boost of up to one point allowed for "Clue", should be rarely given.

**Imagine testing a date field. The field is of the form MM/DD/YYYY (two digit month, two digit day, 4 digit year). Do an equivalence class analysis and identify the boundary tests that you would run in order to test the field. (Don't bother with non-numeric values for these fields.)**

***Grading notes--***

- "overall analysis"--refers to the discussion and presentation of the analysis.
- A boundary chart is not compulsory, but some organized presentation of the material is. These 2 points are for the presentation of the answer
- Interdependence: the valid days will differ depending on which month and whether we are in leap year or not.
- Invalid pairings: there must be tests of invalid combinations, such as Feb 29 in a non-leap-year or June 31. [If the student shows no thinking about invalid combinations, deduct additional points from "overall analysis"]
- There's a 10% grading penalty for wasting space on non-numeric values. These don't belong in the answer (read the question), so the student is writing a shotgun answer. I don't always have the opportunity to penalize for defocused shotguns, but this is such an obvious situation that I am glad to take advantage of it. It lets me make a point about sticking to the call of the question when I review mid-term test results.

Equivalence classes

- valid days. There are 4 equivalence classes for days.
- All are 0-x, where x=28, 29, 30, 31 depending on the months and leap year
    1. month = 28 day
    2. month = 29 day leap year
    3. month = 30 day
    4. month = 31 day
- valid months 1-12
- years 0-9999 or some other plausible range

List tests

(a) {28 day month} {0,1,28,29} {0,2000,9999,10000, leap year}

(b) {29 day month} {0,1,29,30} {0,2000,9999,10000, leap year}

(c) {30 day month} (0,1,30,31} {0,2000,9999,10000, leap year}

(d) {31 day month} {0,1,31,32} {0,2000,9999,10000, leap year}

In other words, for tests of type (a), pick a 28 day month (February) and test with one of the numbers in the set {0, 1, 28, 29} and with one of the numbers in the set {0,2000,9999,10000, any leap year}.

| ***Minefield question*** | E1 | E2 | E3 | E4 | E5 | Total |
|---|---|---|---|---|---|---|

| | | | | | | |
|---|---|---|---|---|---|---|
| | 2 | 2 | 2 | 2 | 2 | 10 |
| | | | | | | |

**8. In lecture, I used a minefield analogy to argue that variable tests are better than repeated tests. Provide five counter-examples, contexts in which we are at least as well off reusing the same old tests.**

<(I'm crediting repeated tests across different configurations, even though the thing that varies here is the configuration (it is a varying test in this sense)>

The following quotes are from a discussion on Software-Testing mailing list. They were not read in class, but they provide arguments that were seen as appropriate by two senior members of the field. Students should get credit for any of these.

Examples from James Bach:

> "You might rationally repeat tests...
>
> "1. if there is a substantially greater probability of a problem happening in an area that is exercised by the tests, compared to other areas. The distribution of problems across a product space is not necessarily uniform.
>
> "2. if any problem that could be discovered by those tests is likely to have substantially more importance than problems in other areas. The distribution of the importance of product behavior is not necessarily uniform.
>
> "3. if they have *some* value and are sufficiently inexpensive compared to the cost of new and different tests. New tests may still be vitally important for the test effort, however.
>
> "4. if the tests you repeat represent the only tests that seem worth doing. This is the virus scanner argument: maybe a repeated virus scan is okay, instead of constantly changing virus tests. However, sometimes we introduce variation because we aren't sure what tests truly are worth doing.
>
> "5. if variation of tests is specifically prohibited by contract or regulation. In other words, the point of your testing may not be to find problems.
>
> "6. if the repeated tests comprise a performance standard that gets its value by comparison with previous executions of the same exact tests. When historical test data is used as an oracle, then you must take care that the tests you perform are comparable to the historical data. Holding tests constant may not the only way to make results comparable, but it can be the best choice available.
>
> "7. when the discovery of bugs is probablistic, perhaps due to important variables involved that you can't control in your tests. Performing a test that is, to you, exactly the same as a test you've performed before, may result in discovery of a bug that was always there but not revealed until the uncontrolled variables line up in a certain way. This is the same reason that a gambler at a slot machine has for playing again after losing the first time."

From Ross Collard, supplementing Bach:

1, Regression Testing is Insurance

James Bach's original posting (attached) discussed whether it is better to vary or repeat the same test cases. It was not directly about regression testing, but regression testing is related and I would like to address the implications of his posting for regression testing.

Many people think that regression testing is over-rated, and the minefield analogy could be used to help make the case for this position. One implication of the minefield is: "Don't re-test the same conditions."

I don't agree that regression testing is over-rated. Large organizations like Microsoft and Cisco mindlessly re-run the same regression test cases night after night, often at the rate of tens of thousands of test cases per night.

Almost all of these test cases almost always pass -- usually well over 99% pass with reasonably stable and mature systems. The biggest category of regression test case failures is generally the repeats -- the ones we already know about, because they failed before for minor reasons and we are being leisurely in getting around to fixing them. So these test case failures do not provide any new information.

Places like Cisco have made their automated regression testing fast enough and cheap enough that even if the pass rate is 100%, they have not paid too much to gain a sense of confidence. (A caution -- many observers would say that this is a dangerous and false sense of confidence if the regression testing is not highly competent.)

In the movie "Groundhog Day", Bill Murray wakes up each morning, only to have to re-live the same day over and over. Bill could have been a regression tester, because the heart of regression testing is repetition; re-running the same test cases from version to version of a system. As Yogi Berra said: "It's deja vu all over again."

2. Regression Testing is Distinct from Modification Testing

It helps to ensure we are using words in the same way here.

Localized change testing or modification testing addresses what has changed. This testing is narrow in focus, based on the change requests, problem reports, programmers' impact assessments, before / after comparisons (the diff or delta files) from source code control tools, or other sources of information.

3. Regional Impact Testing

After the specific change has been checked, i.e, the system behavior conforms to the expected new behavior described in the change request, or the problem as reported in the problem report has been fixed, some people perform what they call regional impact testing. (Note -- there is usually more than one change request or problem report included in a new system build. Localized testing is done for each one of them.)

Regional impact testing goes beyond the localized change and seeks to test in the perceived high-impact region around the change. This requires that the testers have a reasonable chance of identifying the high-impact region, which usually requires a gray box view of the system architecture -- what connects to what internally.

As an example of regional impact testing, consider two system features which externally appear to be unrelated. If one feature is changed, there is no particular reason to re-test the other. Internally, though, let's say these features are data-coupled, i.e., share some common data. This means that the features interact through the shared data and could interfere with each other. The first feature could run first, corrupt the shared data, and cause problems for the other feature.

Regression testing, at least in this view, is different and follows the localized modification testing and regional impact testing. In other words, the modification itself has already been checked prior to the regression testing.

4. The Mines Move

A significant observation, in my opinion, was stated by Kamesh Pemmaraju in his posting.

He made the point that the minefield analogy is inexact (like all analogies). The locations of the hidden mines are not static. Every night after we have cleared (or more likely, partly cleared) the minefield the enemy is back in there seeding the field with new mines. (With apologies to software engineers -- we know (hope?) their primary objective is not to seed mines.)

As Kamesh said: "In a dynamic environment, new mines are planted and old mines (that were cleared earlier) re-appear and these active mines may now occur in the paths that were traversed before."

5. Changes Often Introduce Bugs

We have ample evidence that changes can introduce bugs, and these bugs are scattered in patterns that do not respect the parts of the system we have already tested. In other words, it is not hard to inadvertently break something which previously was working.

According to Watts Humphrey (I think) of the SEI, the probability of a software engineer inadvertently introducing a defect with a modification is 20% to 50%. To be fair, most of these new defects are trivial, and about two-thirds of them are seen and fixed / removed immediately by the software engineer before they are seen by the system testers.

Capers Jones of Software Productivity Research estimates that for every hundred Y2K fixes, seven new defects were introduced. Y2K fixes, while numerous, were considered very straightforward and low risk, perhaps 1 or 2 on a scale from 1 to 10 of the difficulty of software fixes.

IBM has reported that 9% of all modifications to its MVS mainframe operating system introduce new defects -- and that is just the ones they know about.

For an Alcatel subsidiary which makes high-speed backbone switches for the Internet, the number of modifications which introduce new bugs is over 20%. (I don't have permission to reveal the subsidiary's name.)

We also have the notorious example of DSC Communications, where an inadvertent one-character bug in a three-line-of-code change to an existing two million LOC system came close to putting the company out of business. They did not catch the side effect (the inserted bug) because they did not adequately regression test. Or at least, that was a question raised in a congressional inquiry into the damages the side effect caused.

6. Variations and Equivalence

I do not necessarily disagree with James Bach's heuristic: "It's better to vary tests than to repeat the same tests." As he said, the reason for raising the issue is to help ensure we think through the advantages and disadvantages of varying test cases in a particular situation.

Several people in prior postings pointed out that the number of test cases run is typically only a small sample of all possible conditions, it is better to vary the sample on re-runs.

However the whole idea of equivalence classes (sets of test cases grouped together based on commonality vs. variances, where if the system works for one test case it likely works for all in the set), reduces the importance of varying the test cases.

In theory, this "if it works for one, it works for them all" claim means that there is no point to variations within an equivalence class.

Of course, since the definitions of equivalence classes always have some assumptions and uncertainties, and our equivalence class groupings are usually imperfect, there is still a significant value to varying the test cases nevertheless.

So while deciding whether to vary the test cases is worthwhile, and how, I do not see these as the most important issues.

The more important point in my opinion is to reasonably extensively re-test existing features and characteristics that should not have been affected by a change -- in other words, whether to regression test at all, regardless of how much the test cases are varied or held constant.

Many, perhaps most, organizations make a change, then test the change itself and a little bit around the change, and do little or no regression testing.

7. Partial Regression Testing

Many test groups do not run a full regression test on every build or version of a system, because it is prohibitively expensive and time-consuming. Ideally, the regression testing would be so fast and cheap that the testers can mindlessly re-run all the test cases, but this is usually not the situation.

The idea in partial regression testing is either (a) to draw a boundary around a change to a system, and to test only within that boundary, or (b) to take a subset of an entire regression test case library, based on intelligent selection criteria for the situation, and re-run only this subset

The assumption behind the first idea is that the system is decoupled -- any change introduced within the boundary cannot adversely affect anything outside the boundary, i.e., in the untouched remainder of the system.

Some purists think that "partial regression" is a contradiction in terms: a regression test means a complete re-test and thus cannot be partial. Despite this quibble, the concept of a partial regression test can help to determine the appropriate limits for a regression test.

8. Re-Test Coverage Guidelines

To what extent should existing features, which should not have themselves been changed, be re-tested after a modification?

Coverage guidelines embody a strategy for determining how much regression testing to do, based on our best guess at the trade-offs of costs, benefits and risks in a particular testing situation.

To implement these guidelines, each test case in the regression test case library has to be categorized and tagged by its category (a test case can belong to multiple categories). Examples of categories include (a) heavily used features and (b) unusually complex uses of a feature.

For a low-to-moderate risk, low-to-moderate complexity system, fairly typical guidelines for the recommended degree of re-testing after a change, (i.e., not including the testing of the change itself), organized by category of regression test cases, are as follows:

| Regression Test Category | Degree of Coverage (*) |
|---|---|
| [(*) Percentage of all regression test cases in each category which are re-run.] | |
| 1.Smoke test | 100% |

2. Test cases which have failed in the past

| | |
|---|---|
| a. Critical errors | 100% |
| b. Moderate or minor errors | 10% to 20% |

3. Test cases for basic functionality

| | |
|---|---|
| a. Area(s) most impacted by changes in this release | 25% to 50% |
| b. Positive remaining test cases | 5% to 10% |
| c. Negative (robustness) remaining test cases | 10% to 20% |

| | |
|---|---|
| 4. Test cases for complex features | 25% to 50% |
| 5. Test cases for frequently or heavily used features | 25% to 50% |
| 6. Test cases for business-critical features | 50% to 100% |
| 7. Bad fix test cases | 100% |

The overall percentage coverage (percentage of all the test cases in the test case library which are included in this particular regression test run) might be 25%-40%, as a weighted average of the categories listed above.

9. Mutations

Douglas Hoffman discussed automated test case mutations in his STAR presentation.

Earlier, I mentioned extracting subsets of test cases from an existing (and probably large) library of regression test cases. These test cases do not have to physically all exist -- there could be a test data generator available to generate variations on demand. So for partial regression testing the test cases effectively can be extracted from a virtual set.

(Mutation analysis, which seems to have fallen into disuse in its initial use, originally generated mutations for a different purpose - to examine the efficacy of test cases.)

10. Repeatability vs. Rotation

If only a subset of the test cases in a particular category within the regression test case library are going to be selected for re-testing, there is the option of selecting and using the exact same test cases from cycle to cycle of regression testing, or rotating the subset of test cases run in each cycle.

Let's assume that of all the existing positive test cases for basic functionality of a low-risk, low-complexity system, the target coverage has been set to 10%. In other words, if there is a total of 150 test cases in this category in the test case library, 15 of them will be included in this regression test.

The question is: on subsequent re-testing of future versions of the system, should the same subset of let's say 15 out of 150 test cases from a given category be re-used, or should the membership of this subset be rotated (varied) from test cycle to test cycle?

This is simply a re-statement of James Bach's original question.

There are two schools of thought on this. One opinion is that the randomly selected test cases should not be rotated but remain constant, because this way there is before-and-after comparability of the same test results from test cycle to test cycle.

With the first option, 15 test cases are selected and run in every regression cycle. The remaining 135 test cases are never utilized., at least not in this series of regression test cycles.

This first option, to re-run the exact same test cases from cycle to cycle, has the virtue of repeatability. The selected test cases are run routinely in every cycle, so that we can compare the results of these same test cases across all the cycles of regression testing.

The same test cases should always produce the same results over time, unless there is a deliberate reason that we should expect a change in results. The discrepancy in results, if it occurs, is what we are interested in. (Comparator tools are cheap and simple, but great for this mindless re-checking.)

As a few people such as Rex Black mentioned, this repetition is sometimes contractually required, especially in areas regulated by U. S. government agencies such as the DOD, FDA, FAA and NRA.

In 5 cycles of regression testing, however, the total coverage does not exceed 10% because the same test cases are always being run.

The other school of opinion is that the members of the subset of test cases taken from each category should be rotated from test cycle to test cycle, in order to provide a broader test coverage over time.

The second option, to rotate the subset of test cases within the same category from test cycle to test cycle, has the virtue of providing broader coverage -- a wider range of the test cases within the category are executed over a series of regression test cycles. However, we lose 100% repeatability -- not all the selected test cases are run in every regression cycle.

With this second option, a different subset of 15 test cases out of the total of 150 are selected and run in each regression cycle. Over a duration of 5 test cycles, a total of 75 different test cases are executed, for a total coverage of 50%, but none of them are repeated.

11. The Cost of Generating Variations of Outcomes

Deliberate and random or periodic variations are fairly easy to build into regression test case and can be triggered automatically, and generating variations of test cases can be automated too.

Now I want to be a hypocrite and disagree with myself.

It is easy to generate variations of the input data values and initial conditions.

It is frequently much harder to generate the correct variations of the expected outcomes of these test cases.

Imagine, for example, we have to test a system which computes taxes for the U.S. Internal Revenue Service from input tax returns.

Producing the input variations is dead easy -- we can use a test data generator to bury us in test cases within minutes.

But trying to determine whether we got the right results (the computed amounts of tax due) is the killer. For this we would need an oracle, which in the general case would have to incorporate all the U.S. tax code -- the oracle would be bigger than the system we are trying to test.

12. Parallel or Volume Testing

In some ways our great-grandparents who tested mainframe systems were way ahead of the game. They used unplanned, uncontrolled variations as part of a popular mainframe testing technique called parallel testing

In a parallel test, which is also called a volume test, a large volume of test cases are "pumped through" the system or the feature being tested, in a before-and-after comparison of the new system version with the prior version. The term back-to-back testing is also used, to describe the situation where the same set of test cases is executed with two versions of the same system. Apart from what is expected to have changed from version to version, everything else should be the same in the new version as in the old one.

Unlike a planned, detailed test, usually the parallel test cases are deliberately not pre-defined individually nor pre-filtered. The test cases are usually extracted in bulk wholesale mode from a passing (large) stream of live data. This live data stream is continuously changing. After the extracted test data is used in one before-and-after comparison of two system versions, it is typically thrown away and a fresh (and different) extract is used the next time.

A pre-defined set of test cases may reflect biases and contain gaps. The hope of parallel testing is that, with a sufficiently large volume of these test transactions, all significant conditions and combinations of conditions will be checked.

This idea is a little like throwing mud at a politician: with sufficient volume, some must stick. In the words of Lenin: "Quantity has a quality of its own." (He was commenting on the use of tanks in ground warfare.)

Unfortunately, parallel testing carries its own significant dangers, the biggest of which is the hidden carry-forward of pre-existing bugs. The before and after versions of the system contain the same hidden bug and so both behave the same way. New England Tel (now part of Verizon) faced a huge liability and litigation for years of small over-billings which added up to large sums, based on a hidden bug which passed years' worth of parallel testing.

To be fair, the hidden carry-forward of pre-existing bugs is a danger which can occur with any type of before-and-after results comparison, including regression testing where the expected results have not been independently computed.

13. Varying the Number of Test Cases per Cycle

For most test projects, it is unrealistic to assume that exactly the same number of test cases will be executed on each build.

As the trust in each subsequent build increases, the testers may be willing to decrease the number of regression test cases for each build. More likely, not all features will be ready to test in the earlier the test cycles, so that not all test cases could be run even if we wanted to. All the test cases are not likely to be ready at the beginning, even if all the features are available to test. A full regression test may deliberately not be run on each build, either because of the frequent turnaround (the quick replacement of the build by the next new version does not allow enough time), or because the regression testing is too expensive. And some builds may be skipped, as the testers may choose to wait for a later build instead of taking the build which is immediately available.

In addition, the testers usually gain more confidence from build to build, as (a) the testers' understanding of what they have increases, and (b) the versions become progressively cleaner.

So far, my discussion has assumed that the number of test cases to re-run is fixed from build to build. This variation in the number of test cases run from cycle to cycle may influence the mix between repeatability vs. rotation.

As the systenm under trest stabilizes / matures, some regression test groups increase the proportion of test cases which are repeated (fixed), but I have not figured out as yet whether this is a good idea or not.

14. Determining the Best Re-Test Strategy

So do we repeat, rotate or use some mix of the two?

I have not done any kind of survey, but in my experience most regression test organizations repeat anywhere from 75% to 100% of the same test cases, and rotate 0% to 25%.

I suspect the amount of rotation (variation) has generally been way too low.

We should repeat test cases only in areas where:

(a) The probabilities of new breakages appear to be high. For example, areas where prior failures rates have been high are good candidates for intense re-testing.

(b) The costs of failure are so high we are not willing to take any chances.

(c) We want comparable results from test cycle to test cycle.

(d) The cost of variations (e.g., independently computing the new expected outcomes) is too high to be effective.

Ross Collard

| ***Three different missions*** | Mission 1 | Strategy 2 | Mission 1 | Strategy 2 | Mission 1 | Strategy 2 | Clue 1 | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | |

**List three different missions for a test group. How would your testing strategy differ across the three missions?**

Grading Notes:

A plausible relationship between the mission and the strategy / activity is sufficient for full credit, but the strategy / activity description has to go beyond a statement of fulfilling the mission. For example, if the mission is "Find defects", the testing strategy has to say something more than "find defects" such as concentrating on stress tests, concentrating on high risk areas of the product, etc.

Varying definitions of test groups (from course slides)

- Find defects
- Maximize bug count
- Block premature product releases
- Help managers make ship / no-ship decisions
- Assess quality
- 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

- Assure quality

"Clue" is discretionary, typical value is 0.5. If the mission strategies are weak but I give them a full credit, I'll not award the clue point (it was already awarded).

| ***Goodness of tests*** | D1<br>2.5 | D2<br>2.5 | D3<br>2.5 | D4<br>2.5 | total |
|---|---|---|---|---|---|
| | | | | | |

**List and describe four different dimensions (different "goodnesses") of "goodness of tests".**

Grading Notes:

1 point for the item and 1.5 for the description

List from course:

- More powerful
- More credible
- Provides better support for troubleshooting
- Representative of a broader group of tests
- Is representative of events more likely to be encountered by the customer
- Is more likely to help the tester or developer develop an insight into the program
- Is easier to automate, easier to evaluate, more feasible, lower opportunity cost

## Long Answers 20 points each

| 11. Scenario | Explain how develop | Describe | Explain why good | total |
|---|---|---|---|---|
| | 2 pts each good idea<br>-2 if not for set<br>max 8 | Max 5<br>(well described, good scenario) | Tie facts of test to stated elements, 2 per element<br>Max 8 | |
| | | | | |

**Imagine that you were testing the feature, *Save With Password* in the *OpenOffice* word processor.**

- **Explain how you would develop a set of scenario tests that test this feature.**
- **Describe a scenario test that you would use to test this feature.**
- **Explain why this is a particularly good scenario test.**

**Grading Notes**

Scenario tests:

- **Explain how you would develop a set of scenario tests for this feature. I expect a range of possible ideas**
    - Research
        - Customers
        - Competitors
        - In-house documentation for tasks
    - Implementation

  Note that we're talking about a SET, not just one. If the explanation is appropriate only for a single scenario test (not for a set), deduct points.
- **Describe a scenario test you would use. I evaluate it against**
    - Realistic
    - Complex
    - Unambiguous
    - Persuasive / credible to stakeholder
- **Why is THIS a particularly good test**
    - Tie the facts to the elements

| Name | How many Combs 2 | What is all Pairs table 4 | Create Table 10 | Why Correct 4 | Total 20 |
|---|---|---|---|---|---|
| | | | | | |

**We are going to do some configuration testing on the *OpenOffice* word processor. We want to test it on**

- **Windows 95, 98, and 2000 (the latest service pack level of each)**
- **Printing to an HP inkjet, a LexMark inkjet, and a Xerox laser printer**
- **Connected to the web with a dial-up modem (28k), a DSL modem, and a cable modem**
- **With a 640x480 display and a 1024x768 display**
    - **How many combinations are there of these variables?**
    - **Explain what an all-pairs combinations table is**
    - **Create an all-pairs combinations table**
    - **Explain why you think this table is correct.**

***Grading notes--***

- Combinations 3x3x3x2=54
- "explain" and "why correct?" are essentially the same question. The second one is an opportunity for the student to look back and check the work as she starts writing her criterion.
- Students who blow the combination chart (by missing all pairs) are capped at 50%. This looks harsh, but this is a very easy table and the students have had it for plenty of time. They shouldn't get this wrong.

| ***Test Plan*** | Strategy | Q1 | Guide 1 | Q2 | guide 2 | Q3 | Guide 3 | Q4 | Guide 4 | Q5 | Guide 5 | Q6 | Guide 6 | Total / 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | |

**Imagine that you are an external test lab, and Sun comes to you with *OpenOffice*. They want you to test the product. How will you decide what test documentation to give them? (Suppose that when you ask them what test documentation they want, they say that they want something appropriate but they are relying on your expertise.) To decide what to give them, what questions would you ask (4 to 6 questions) and how would the answers to those questions guide you?**

***Grading notes-***

***[Note: I have refined the wording of this question since grading the exam in which this question appeared. This analysis will be different next time because you won't be asked for an overall strategy. "How will you decide what test documentation to give them?" is deleted.]***

With "typical" points and 4 questions, the student gets 16 plus up to 3 for strategy.

With "typical" points and 6 questions offered, the student can get 24 plus up to 3 for strategy,

Maximum points possible are 39 (1 per question, 5 per guide across 6 questions, plus up to 3 for strategy)

I reserve discretion over "A" and may slightly raise or lower an answer if it is in the 18-20 total range.

***Strategy:*** How will you decide what test documentation to give them? *An answer that says, I'll ask them questions, is worth 0 points because I've said to ask questions.* On the other hand, if they add extra research ideas beyond asking questions, they can have 1 (typical) to 3 (professional-level) points.

***Questions:***

- The question alone gets one point for itself. They got a list of questions in the course, there is nothing original here, just very simple memory work.
- The question alone gets no guidance points, zero. Guidance should take the form of a specific statement of impact (of the answer) on the content or structure of the test documentation. An exceptionally insightful and useful guidance answer can earn 5 points. An adequate (the typical) answer earns 3 points. A weak answer earns 0-2. *For examples of impact discussions, see the test documentation chapter in Lessons Learned in Software Testing. This was required reading in the course.*
- Some people misread the question as calling for an aggregate judgment on the value of the answers. I added back up to 6 points for the aggregate evaluation.

See course slides on requirements questions:

- *Is test documentation a <u>product</u> or <u>tool</u>?*

- *Is software quality driven by legal issues or by market forces?*
- *How quickly is the design changing?*
- *How quickly does the specification change to reflect design change?*
- *Is testing approach oriented toward proving conformance to specs or nonconformance with customer expectations?*
- *Does your testing style rely more on already-defined tests or on exploration?*
- *Should test docs focus on what to test (objectives) or on how to test for it (procedures)?*
- *Should control of the project by the test docs come early, late, or never?*
- *Who are the primary readers of these test documents and how important are they?*
- *How much traceability do you need? What docs are you tracing back to and who controls them?*
- *To what extent should test docs support tracking and reporting of project status and testing progress?*
- *How well should docs support delegation of work to new testers?*
- *What are your assumptions about the skills and knowledge of new testers?*
- *Is test doc set a process model, a product model, or a defect finder?*
- *A test suite should provide prevention, detection, and prediction. Which is the most important for this project?*
- *How maintainable are the test docs (and their test cases)? And, how well do they ensure that test changes will follow code changes?*
- *Will the test docs help us identify (and revise/restructure in face of) a permanent shift in the risk profile of the program? Should docs (be) automatically created as a byproduct of the test automation code?*

*Additionally, we might see the Phoenix questions (these are listed in Thinkertoys) or other context-free questions (see Gause & Weinberg, Exploring Requirements).*

| ***Oracle*** | Hyphenation<br>8 | Footnotes<br>8 | Compare<br>6 | Total<br>20 |
|---|---|---|---|---|
| | | | | |

**You are using a high-volume random testing strategy for the *OpenOffice* word processing program. You will evaluate results by using an oracle.**

- **Consider testing the hyphenation feature using oracles. How would you create an oracle (or group of oracles)? What would the oracle(s) do?**
- **Now consider the placement of footnotes at the bottom of the page. How would you create an oracle (or group of oracles) for this? What would the oracle(s) do?**
- **Which oracle would be more challenging to create or use, and why?**

***Note: If you don't understand hyphenation, substitute "spell checking" for "hyphenation in this question.***

***Grading notes.***

In 2002, I applied gentle grading because we didn't spend enough time on this in class for a detailed answer. Additionally, just before the exam, it became clear that several foreign students were befuddled about hyphenation. So, I sent out a note allowing spell checking instead, and included this as one of the easy questions on the exam.

Hyphenation:

- How would you create an oracle?
    - Use of prior version is worth 4 points (of 8). The problem is that there's no reason to believe the prior version works. Add points for discussion of this issue.
    - Compare to competitor is worth 4-8 points depending on whether the answer deals with the question of how we know the competitor works
    - We could run both word perfect and word in parallel and raise the flag if they disagree with our result
    - We could build random sentences from a small vocabulary of words that have known hyphenation characteristics, then check whether they were hyphenated properly against a list
    - We might run only a partial oracle that looks at hyphenation under some simple rules.
- What would it do?

Spell checking

- How
- What would it do

Footnotes

- There are several things to check here--placement on the page, agreement between the reference mark (e.g. footnote number) in the body and the one in the footnote, formatting of the footnote, formatting of the reference mark, break of the long footnotes across pages, formatting of tables in footnotes, etc.
- How
    - Use of prior version is worth 3 points (of 7). The problem is that there's no reason to believe the prior version works. Add points for discussion of this issue.
    - Compare to competitor is worth 3-7 points (of 7) depending on whether the answer deals with the question of how we know the competitor works
    - We could run both word perfect and word in parallel and raise the flag if they disagree with our result
    - We might run a partial oracle that looks only at some of the footnoting issues.
    - If you write your own, how do you know it works and how do you decide what to include. Are you designing it in a way that makes it likely to be suitable for high-volume work?
- What would it do

Which oracle is more challenging and why

- Footnoting is much more challenging because there are so many variables in play. Consider the problem of placement at the bottom of the page, carrying long notes across pages, formatting of tables and pictures inside footnotes, etc.
- However, a *reasoned* argument in favor of the other (hyphenation or spellcheck) will be accepted to the extent that it is credibly argued.

| ***follow-up testing*** | Steps<br>3 | Options<br>3 | Configs<br>3 | Generality<br>(3) | Other<br>(3) | Example1<br>4 | example2<br>4 | Example3<br>4 | Total<br>20 |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |

**Suppose that you find a reproducible failure that doesn't look very serious.**

- **Describe three tactics for testing whether the defect is more serious than it first appeared.**
- **As a particular example, suppose that the display got a little corrupted (stray dots on the screen, an unexpected font change, that kind of stuff) in *OpenOffice*'s word processor when you drag the mouse across the screen. Describe three follow-up tests that you would run, one for each of the tactics that you listed above.**

***Grading Notes***

Describe three tactics for testing: If you list an item without describing it, only 1 point.

Each description is worth 3 points, each example is 4 points. That totals 21, but from 19 to 20 is my discretion

My examples of follow-up tests

- Tests related to my steps
  - Enter more data into the table
  - Enter data into the table lots of times (repeat same entries)
- Tests related to the structure of the situation
  - Vary the size of the table
  - Vary the contents of the table
  - Vary the color, alignment, font, line width, etc of the table entries
- Tests related to the failure
  - ?? what else causes mouse droppings ??
  - print preview the screen
- Tests related to the persistent variables / options
  - Location of the program within the window
  - Whether the program is maximized
  - Default cell format, such as alignment within the cells, font, line width, etc.
- Tests related to the hardware
  - Different video resolution
  - Different monitors

- Different video cards
- Different OS (check one of the ports, is this unique? If not, then anything specific to windows might be irrelevant)
- Different mouse / mouse driver
- Different memory

# Appendix D: Sample Assignments

## *Assignment 1: Create a first test case chart*

**In lecture, we brainstormed answers to the following:**

> **The program reads three integer values from a card. The three values are interpreted as representing the lengths of the sides of a triangle. The program prints a message that states whether the triangle is scalene, isosceles, or equilateral.**
>
> » From Glen Myers, *The Art of Software Testing*

- ***Write a set of test cases that would adequately test this program.***
- ***Please write your name on your answer so that I can return it to you. Hand it in when you are done.***

Let's take one more crack at Myers' exercise. I'd like a table like this from you that lists 5 good test cases:

| Test | Test Case | Risk | Why this test is powerful | Expected Result |
|---|---|---|---|---|
| **1** | | | | |
| **2** | | | | |
| **3** | | | | |
| **4** | | | | |
| **5** | | | | |

Where the columns are defined as:

- Test—fill in the test case number
- Test case—be specific about the inputs that you'll feed the program. For example, 1,1,2
- Risk—be specific about the error you are trying to detect. For example (from 1,1,2), the risk is that the program might accept a "triangle" that has side 1 + side 2 = side 3.
- Why this test is powerful—A test is powerful, compared to other tests, if it is more likely to expose a failure than they are. If you test for a specific type of failure (the risk), use a powerful test – one that is at least as likely to expose that failure as any other. To fill in this part of the answer, explain why you think the test you chose is powerful. It might be helpful to provide examples of similar tests that are less powerful than this one. If you think that this test is equivalent to many others (no more, no less powerful), say "Equivalent" and list 2 or 3 other tests that you think are equivalent to this one. This is an easy answer—but if I can easily spot a better test against the same risk, you won't get credit for saying "equivalent".
- Expected result—What should the program do? You might respond, "Error Message."

**Assignment Instructions**

- Feel free to work in groups
- If you work with others, make sure to name them.
- If you work in a group, it is OK for the group to hand in one collective answer that you all co-sign. (Just provide your name, I don't need your student number.) However, if you co-sign it, you must be able to able to explain EVERY test case that you have on the page.
- I expect better work from a group than from individuals and I will mark accordingly.
- If you work in a group, I expect 5 tests from each of you. So, if there are two of you in the group, the assignment should contain 10 tests.
- Please don't submit 100 tests. Pick 5 good ones per person. Prioritizing among possible tests is one of the important skills of good testers.

**Grading Rubric for the Triangle problem**

This rubric is given to the students. For each test, the first column indicates the points to be awarded if you give an answer that has the characteristics listed in the second column. Each test can be awarded up to 10 points.

**Test Case:**

| | |
|---|---|
| **0** | Values for the sides of a triangle not provided or incorrect. |
| **1** | Values for the sides of the triangle provided and correct. |

**Risk:**

| | |
|---|---|
| **0** | Implausible or generic to the point of no value. Not a risk. |
| **1** | Generic Risk statement, Vague not related to power. |
| **2** | Less clear or less plausible but related to power. |
| **3** | Clear statement of plausible risk not related to power or<br>Less clear or less plausible risk but related to power. |
| **4** | Clear statement of plausible risk related to power. |

**Power:**

| | |
|---|---|
| **0** | No clear linkage between the power discussion and the risk. |
| **1** | Shows that the test can detect the error identified in the risk. |
| **2** | The test can detect an error, indicate how or why and provide some comparison to other tests. Some indication that this is a good test. |
| **3** | Good comparison examples but weak explanation otherwise. A sufficiently strong explanation but without examples. Must indicate this is better than the others. |
| **4** | State the principles under which this is more powerful and gives a persuasive example. |

**Expected Result:**

| | |
|---|---|
| **0** | Expected result not provided or incorrect. |
| **1** | Expected result provided and correct. |

## *Assignment 2: Replicate and Edit Bugs*

The purpose of this assignment is to give you experience editing bugs written by other people. This task will give you practice thinking about what a professional report should be, before you start entering your own reports into this public system.

- Work with *OpenOffice* Writer, the word processor.
- Read the instructions at http//qa.*OpenOffice*.org/helping.html, and make sure to use the oooqa keyword appropriately. Read the bug entry guidelines at http://www.*OpenOffice*.org/bugs/bug_writing_guidelines.html.
- Find 5 bug reports in IssueZilla about problems with *OpenOffice* Writer that appear to have not yet been independently verified. These are listed in the database as "unconfirmed" or "new". As of 9/1/2002, there are 927 such reports associated with the "word processor" component. To find lots of bugs, use the search at http://www.*OpenOffice*.org/issues/query.cgi rather than at http://qa.*OpenOffice*.org/issuelinks.html.
- For each report, review and replicate the bug, and add comments as appropriate to the report on issuezilla.
- Send me an email with the bug numbers and for each bug, with comments on what was done well, what was done poorly and what was missing that should have been there in the bug report.

**Assignment Procedure**

For each bug:

- Review the report for clarity and tone (see "first impressions", next slide).
    - Send comments on clarity and tone in the notes you send me (but don't make these comments on the bug report itself)
- Attempt to replicate the bug.
    - Send comments to me on the replication steps (were the ones in the report clear and accurate), your overall impressions of the bug report as a procedure description, and describe any follow-up tests that you would recommend.
- You may edit the bug report yourself, primarily in the following ways.
    - Add a comment indicating that you successfully replicated the bug on XXX configuration in YYY build.
    - Add a comment describing a simpler set of replication steps (if you have a simpler set). Make sure these are clear and accurate.
    - Add a comment describing why this bug would be important to customers (this is only needed if the bug looks minor or like it won't be fixed. It is only useful if you clearly know what you are talking about, your tone is respectful).
    - Your comments should NEVER appear critical or disrespectful of the original report or of the person who wrote it. You are adding information, not criticizing what was there.
- If you edit the report in the database, **never change what the reporter has actually written**. You are not changing his work, you are adding comments to it at the end of the report
- Your comments should have your name and the comment date, usually at the start of the comment, for example: "(Cem Kaner, 12/14/01) Here is an alternative set of replication steps:")
- Send me an email, telling me that you have reviewed the report and made changes.

**A Checklist for Editing Bugs**

The bug editor should check the bug report for the following characteristics:

A. First impressions—when you first read the report:

1. Is the summary short (about 50-70 characters) and descriptive? (see the slide: *Important Parts of the Report: Problem Summaries*)
2. Can you understand the report? As you read the description, do you understand what the reporter did? Can you envision what the program did in response? Do you understand what the failure was?
3. Is it obvious where to start (what state to bring the program to, to replicate the bug)?
4. Is it obvious what files to use (if any)? Is it obvious what you would type?
5. Is the replication sequence provided as a numbered set of steps, which tell you exactly what to do and, when useful, what you will see?
6. Does the report include unnecessary information, personal opinions or anecdotes that seem out of place?
7. Is the tone of the report insulting? Are any words in the report potentially insulting?
8. Does the report seem too long? Too short? Does it seem to have a lot of unnecessary steps? (This is your first impression—you might be mistaken. After all, you haven't replicated it yet. But does it LOOK like there's a lot of excess in the report?)
9. Does the report seem overly general ("Insert a file and you will see" – what file? What kind of file? Is there an example, like "Insert a file like blah.foo or blah2.fee"?)

B. When you replicate the report:

10. Can you replicate the bug?
11. Did you need additional information or steps?
12. Did you get lost or wonder whether you had done a step correctly? Would additional feedback (like, "the program will respond like this...") have helped?
13. Did you have to guess about what to do next?
14. Did you have to change your configuration or environment in any way that wasn't specified in the report?
15. Did some steps appear unnecessary? Were they unnecessary?
16. Did the description accurately describe the failure?
17. Did the summary accurate describe the failure?
18. Does the description include non-factual information (such as the tester's guesses about the underlying fault) and if so, does this information seem credible and useful or not?

C. Closing impressions:

19. Does the description include non-factual information (such as the tester's guesses about the underlying fault) and if so, does this information seem credible and useful or not? (The report need not include information like this. But it should not include non-credible or non-useful speculation.)
20. Does the description include statements about why this bug would be important to the customer or to someone else? (The report need not include such information, but if it does, it should be credible, accurate, and useful.)

D. Follow-up tests:

21. Are there follow-up tests that you would run on this report if you had the time? (There are notes on follow-up testing in the course slides 105-117)?
22. What would you hope to learn from these tests?
23. How important would these tests be?

24. You will probably NOT have time to run many follow-up tests yourself. Don't take the time to run more than 1 or 3 such tests.
25. Are some tests so obvious that you feel the reporter should run them before resubmitting the bug? Can you briefly describe them to the reporter?
26. Some obvious style issues that call for follow-up tests—if the report describes a corner case without apparently having checked non-extreme values. Or the report relies on other specific values, with no indication about whether the program just fails on those or on anything in the same class (what is the class?) Or the report is so general that you doubt that it is accurate ("Insert any file at this point" – really? Any file? Any type of file? Any size? Maybe this is accurate, but are there examples or other reasons for you to believe this generalization is credible?)

**GRADING NOTES FOR THE BUG EDITING ASSIGNMENT**

Two components for grading the papers –

1) Comments at Issuezilla (the *OpenOffice* database)
2) Editor's report submitted to us.

I allocated 14 points possible for each bug, but totalled out of 10. That is, if you got a 3/14 for the bug, your score was changed to 3/10. Similarly, 14/14 became 10/10. There were 10 points available for each bug.

**COMMENTS ON THE BUG REPORTS THEMSELVES, FILED IN ISSUEZILLA**

[NOTE: This was prepared as feedback for students but can be easily turned into a rubric.]

The content of your comments has to vary depending on the problem. The key thing is that the follow-up report has to be useful to the reader.

For example, a simple failure to replicate might be sufficient (though it is rarely useful unless it includes a discussion of what was attempted.) Sometimes, detailed follow-up steps that simplify or extend the report are valuable.

**This is worth up to 7 points out of 10**

| | Subcomponents of the comments at Issuezilla | Points possible |
|---|---|---|
| 1 | Report states configuration and build | + Up to 1 |
| 2 | If the report is disrespectful in tone, zero the grade for the report. | 0 for the report |
| 3 | If you clearly report a simpler set of replication steps | + Up to 5 |
| 4 | If you clearly report a good follow-up test | + Up to 5 |
| 5 | A follow-up test or discussion that indicates that you don't understand the bug is not worth much. | + Up to 1 |
| 6 | If there is enough effort and enough usable information in the follow up test. | + Up to 3 |
| 7 | If you make a good argument regarding importance (pro or con) | + Up to 5 |

| | | |
|---|---|---|
| 8 | If the bug is in fact not reproducible, and the report demonstrates that you credibly tested for reproducibility | + Up to 5 |
| 9 | Nonreproducible bug on alternate configuration without discussion | - 1 |
| 10 | Nonreproducible bug on alternate configuration that was already dismissed | - 2 |

**REPORT TO US**

**This is worth up to 7 points out of 10**

Here, you evaluated the report rather than trying to improve it. I wanted to see details that suggested that you had insight into what makes bug reports good or bad, effective or ineffective. I did not expect you to walk through every item in the checklist and tell me something for each item (too much work, most of it would have wasted your time). Instead, I expected that you would raise a few issues of interest and handle them reasonably well. For different reports, you might raise very different issues.

1) I was interested in comments on:

   a) What was done well.
   b) What was done poorly.
   c) What was missing that should have been there.

2) In the assignment, the checklist suggested a wide range of possible comments, on

   d) First impressions
   e) Replication
   f) Closing impressions
   g) Follow-up tests

   The comments did not have to be ordered in any particular way but they should have addressed the issues raised in the assignment checklist in a sensible order. We credited them as follows:

   Individual issue discussions are worth up to 3 points, but are normally worth 0.5 or 1 point (typically 1 point if well done). An exceptional discussion that goes to the heart of the quality of the report or suggests what should have been done in a clear and accurate way is worth 2 points. An exceptional and extended (long) discussion that goes to the heart of the quality of the report AND includes follow-up test discussion or suggests (well) what should should have been done is worth 3 points.

3) The primary basis of the evaluation in this section is insight into the quality of the bug report. If the student has mechanically gone through the list of questions, without showing any insight, the max point count is 5. If we see insight, the max point count is 7.
   A discussion that shows that the tester did not understand the bug under consideration is worth at most 5, and was often worth less.

| Bug number | Comments at issuzilla | Editor's report | Total points |
|---|---|---|---|
| **1** | **7** | **7** | **14 ≡ 10** |
| **2** | **7** | **7** | **14 ≡ 10** |
| **3** | **7** | **7** | **14 ≡ 10** |
| **4** | **7** | **7** | **14 ≡ 10** |
| **5** | **7** | **7** | **14 ≡ 10** |
| **GRAND TOTAL** | | | **50** |

### *Assignment 3: Domain Testing & Bug Reporting*

1. Create between 10 and 20 *domain tests*. You can stop at 10 if you find (and write up) 2 bugs. You can stop at 15 tests if you find (and write up) 1 bug.
2. Work in the *Word Processing* part of *OpenOffice*.
3. Pick a function associated with Word Processing. Please run all of your tests on the same function. (If several students are working together, you can pick one function per student.)
4. Pick one (1) input, output, or intermediate result variable
   - Identify the variable. Stick with that one variable throughout testing.
   - Run a mainstream test (a test that is designed to exercise the function without stressing it). You do tests like this first in order to learn more about the function and the variable's role in that function.
5. Identify risks associated with that variable
6. For each risk, design a test specifically for that risk that is designed to maximize the chance of finding a defect associated with this risk.
7. Explain what makes this a powerful test. It might help you to compare it to a less powerful alternative.
8. What is the expected result for the high-power test?
9. What result did you get?
10. Report your results in a table format that has the following columns:
    1. Feature or function
    2. Variable name or description
    3. Risk
    4. Test
    5. What makes this test powerful
    6. Expected result
    7. Obtained result
11. If you find bugs, write up bug reports and enter them into Issuezilla.
12. *I strongly recommend that you pair up with someone and have them replicate your bug and evaluate a draft version of your report before you submit it to Issuezilla. I will evaluate your report against a professional standard of quality (essentially, the same evaluation that you just did in Assignment 2).*
13. Write a summary report that explains what you believe you now know and don't know about the function, based on your testing. (If your group tested several functions, write up a summary report for each.)

**Notes (that I'll use for grading) on Exercise 3**

- It's important to answer every section:
  - The table needs 7 columns
  - There should be 10-20 tests and 0-2 bug reports
  - There should be a summary report that explains what you know about the function under test.
- It's important to show the domain analysis (or its results)
  - Use boundary values

- Identify them as bounds and equivalence classes or identify the different sections of the space as you partition it. You might find it useful to start with a boundary analysis (and table).
- Be specific about risk
- Be specific about power (compare to others of the same equivalence class)

### *Assignment 4 Exploratory Attacks*

- This is the fourth of five assignments.
- This is a good assignment for discovering bugs. Remember that if you want bonus points, the bugs MUST be in the database (and I must be notified of it) by December 6 (CSE 4431) or December 8 (SWE 5410). Please feel free to take a bug to a replicator as soon as you have written it up. Enter the bug into IssueZilla when you feel it is good enough to enter (whether you have taken it to a replicator or not). Then send me a note with the bug report or with a pointer to it.
- Conduct at least 4 tests of the *OpenOffice* Word Processing feature, involving one attack from each class of attack:
  - Input constraints
  - Output constraints
  - Storage constraints
  - Computation
- Do NOT do these tests on an embedded spreadsheet in *OpenOffice*.
- For each test,
  - explain why your test is a particularly powerful example of that kind of attack. (That is, explain why this test is better than other, similar tests that you could derive from the same type of attack.)
  - Explain why your attack is a member of the class (input / output / storage / computation) that claim for it.
  - If you find a bug, please report it in the bug tracking database.
- I encourage you to do this assignment with a (one) partner. Creative testing works better in pairs.
- The usual collaboration rules apply--if two of you work together, you should hand in eight tests, two from each category.

## *Assignment 5 Test Automation Requirements*

- Do this question in collaboration with one other student.
- Imagine that you are on the *OpenOffice* testing team as a full-time staff member. You are asked to use a GUI automation tool, such as WinRunner, Silk, or QA Robot, to automate some or all of the testing of *OpenOffice* Word.
- Read "Avoiding Shelfware", "Architectures of Test Automation" and the other papers on test automation on the Blackboard site.
- Consider any ten of the "Twenty-Seven Questions About Requirements" discussed in Avoiding Shelfware. To the best of your ability (sometimes you will make and state reasonable assumptions rather than doing extensive research), answer those ten questions and for each one, explain how that answer would influence your decisions as to what to automate, how to automate, and when in the project to automate it. (Note: you should consider 10 questions whether you work alone or with one other student.)

[1] This work was supported by National Science Foundation grant EIA-0113539 ITR/SY+PE "Improving the Education of Software Testers.".

[2] Cem Kaner is a Professor of Software Engineering at the Florida Institute of Technology in Melbourne Florida. The author acknowledges the assistance of Ajay Jha, Becky Fiedler and Pat McGee in preparing parts of this material and the longer term contributions of James Bach.

[3] "Encouraging students to study together" includes setting up two to three study sessions before each test--I supply coffee and chocolates or pay for breakfast at the local café while the students study together. I'm available to facilitate the discussion when students get stuck or have apparently irreconcilably conflicting views, but I don't provide the answers. Along with getting the students to work together--something that many computer science students are not used to doing, this has a positive effect on morale.

[4] There are plenty of online resources for law students who are learning how to write essay exams, such as

- Martha Peters, "A General Plan for Exams," University of Iowa College of Law Academic Achievement Program, http://www.uiowa.edu/~aap001/examwrite.html, viewed 2/3/03.
- Carolyn Nygren, "Legal Learning for Bar Candidates -- Bar Exam", http://www.findlaw.com/studyskills/3_bar_candidates.html, viewed 2/3/03.
- Gregory Berry, "Rules of Effective Examsmanship for Law Students," School of Law, Howard University, http://www.law.howard.edu/faculty/pages/berry/advice/examtips.htm, viewed 2/3/03.

In addition, many books coach law students on exam preparation and writing skills.

[5] Two examples of University website guides to essay questions are:

- Writing@CSU Writing Guide: "Answering Exam Questions," http://writing.colostate.edu/references/processes/exams, viewed 2/3/03.
- University of Durham Undergraduate Information Site, "Advice on Answering Exam Questions," http://www.dur.ac.uk/biological.sciences/Undergraduate/ugexampage2.htm

These are tip-of-the-iceberg examples. Searches on www.dogpile.com or www.google.com on phrases like "essay exam strategy" and "call of the question" yield thousands of links.

[6] See Cem Kaner, James Bach & Bret Pettichord, *Lessons Learned in Software Testing*, Wiley, 2002, chapter 6.