



Expert Testing...Real World Solutions

PSD Testing Computer Software Course

Contents

Introductions

Please briefly introduce yourself.

How long have you worked in

- software testing?
- programming?
- technical/customer support?

Have you ever tested software?

On which platforms?

What two things do you most want to walk away with from this course?

Course Objectives

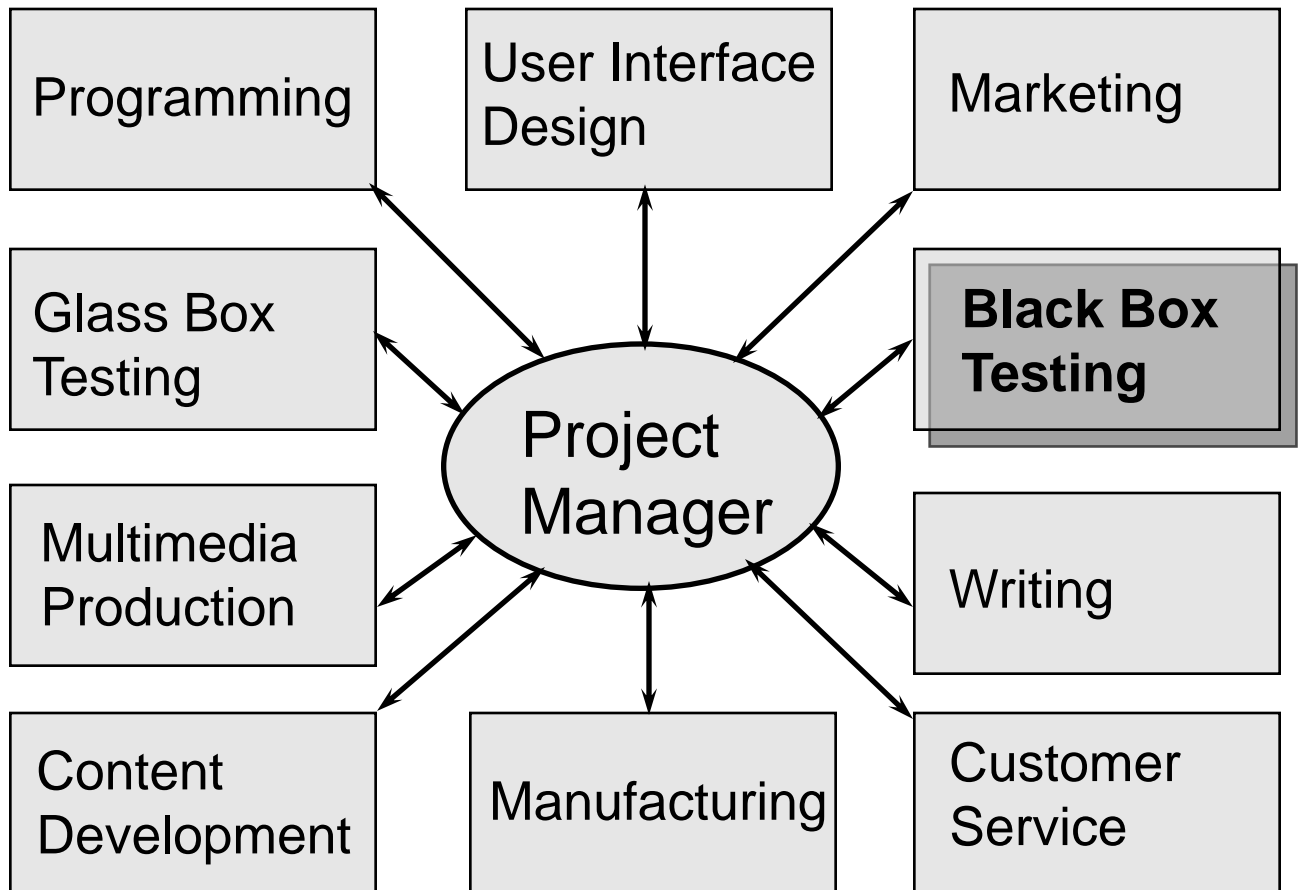
- Learn an organized, pragmatic approach to black box software testing.
- Develop practical testing and test result reporting skills.
- Improve effectiveness in testing, test planning, and project management.

Testing Computer Software

Part 1

An Overview of Product Development

Product Development Organization



During development, several groups (or people) act as service providers to the Project Manager. They often play other roles beyond this, but during development, it pays to look at them as members of a managed group that seeks to ship a high quality, marketable product on time.

Product Development Organization

Project Manager: Responsible for shipping a salable product on time and within budget.

Architect: Design the product's overall (code & data) structure, relationships among components, and relationships with other expected parts of the customers' system.

User Interface Designers: Design the program to be usable and useful.

Programmers: Design and write/test the code.

Glass Box Testers: Use knowledge of the internals of the code to drive the testing. Use code inspections and other code-aware methods to find errors, along with glass box test techniques.

Tech Writers: Write the user manual, help, etc.

Content Developers: There may be several different groups here, animating, filming, writing and performing music, writing, etc.

Product Development Organization

Multimedia Producer: Responsible for managing content development, selecting content to include with the product, negotiating rights, etc.

Marketing: Manage the product's overall profit/loss plan. Evaluate competitors, appraise sub-markets (e.g. Should we be compatible with a certain printer?), design packaging and advertisements, run trade shows, etc.

Black Box Testers: Test from the customer's perspective.

Customer Service: Help evaluate the effect of design decisions, bugs, and existing problems on customers.

Manufacturing: Buy materials. Help everyone control their contribution to the cost of goods. Drive the late-stage schedule. Produce the product.

Development Styles

A software lifecycle model is a definition of the process which is used to develop software.

The lifecycle model defines the phases of the development process, the order they occur in, the degree to which they may overlap, and the entry and exit criteria for each phase.

Development Styles

Waterfall- Document driven.

Concept – Requirements- Architecture-
Design- Code and DeBug- Test – Ship

When it's done well- Documents and reviews
at each stage.

Interactive or Evolutionary-

Start by designing and implementing the most
prominent parts of the program.

Test it, get feedback then add to it and refine.

Test it, give feedback.... Get the picture?

Code and Fix- quickest, not usually formal, not much
project planning, just what it says...

You code your general idea and test it. Do that
until you like what you get then ship it.

I've worked on projects where the spec was
written by the developer after the code was
written. Only to satisfy his manager.

Design-to-schedule – you keep developing till it's time
to ship (or money runs out). Then it is done.

Development Styles

Rapid Prototyping

A rapid prototype is a simplified and untested equivalent of the actual application, performing all the basic functions specified for the final product (Howell, 1992).

This is not a diagrammatic approximation or representation, which tends to be looked at as an abstract thing, but an actual implementation of the specifications for the application. These prototypes can be realistically tested and assessed and rapidly changed in an iterative manner until consensus is reached.

The volume of actual code produced in scripts is relatively low and emphasis is put in user-interface design, link structure design and definition of contents entry as the different multimedia components. These characteristics along with widespread availability of authoring tools, make it possible for rapid development and testing of prototypes.

Rapid-prototyping strategy has short feedback loops through cycles of development and evaluation.

Waterfall and RAD

Development Style Examples

The Development Style is effected by Company
or Product Goals.

Is the Project:

Requirements constrained

Schedule constrained

Cost constrained

Development Style Examples

- 1 A Word Processor for Teenagers with lots of Templates and Graphics = Back to School Product

Quality concern: Drop-dead release date- can ship with defects, keep expenses low.

- 2 A Professional Desktop publishing application or an Internet Browser application in very competitive markets.

Quality concern: Feature set, reliability or Performance with low defect, schedule second, cost third.

- 3 Internal Company Application- not for public release.

Quality Concern: Feature set first, then cost, third schedule

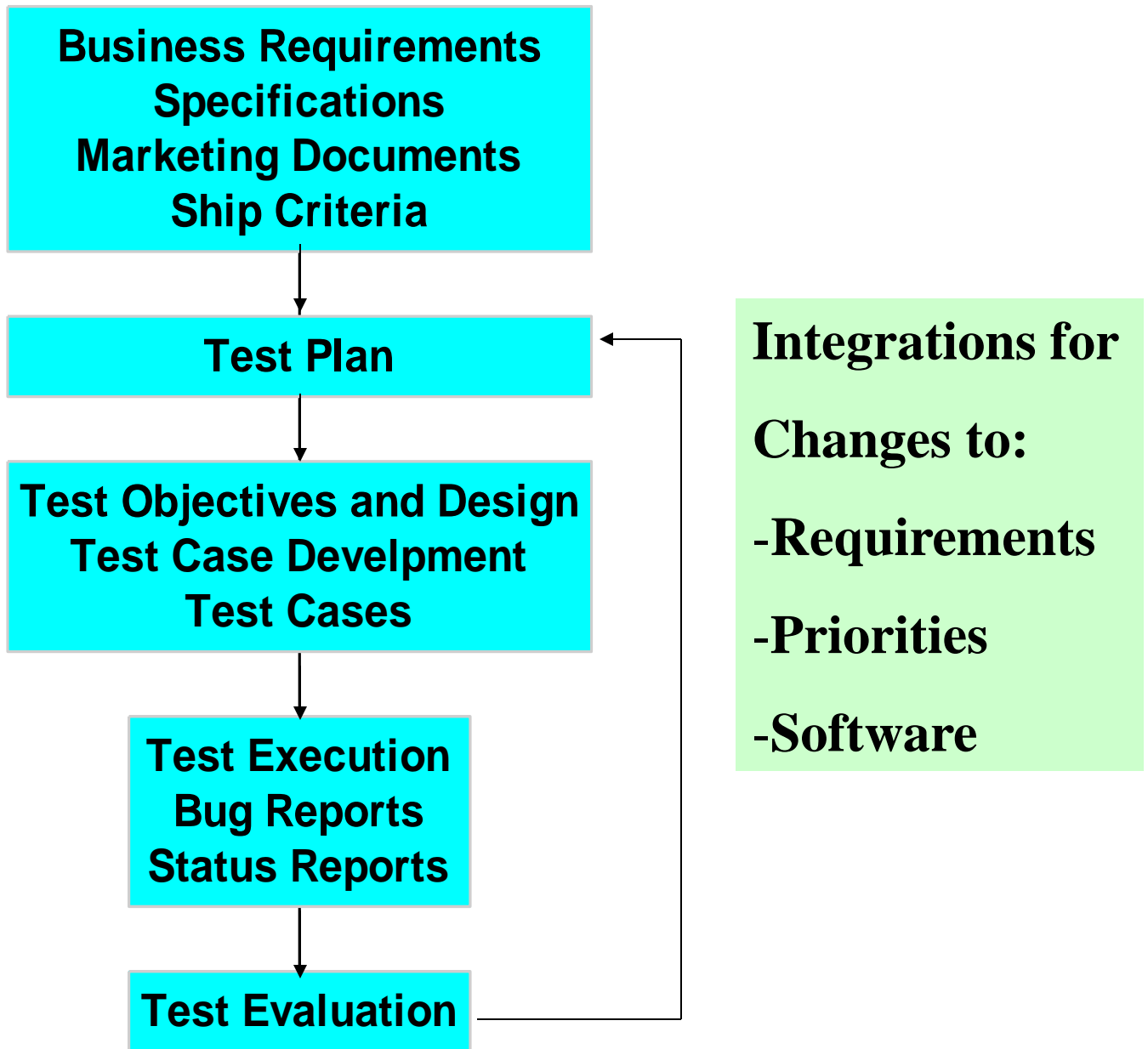
- 4 Software for Military, Bank- Sold but only to be used only internally.

Quality Concern: Very low defects- use their quality standard not yours! Will hold payment if it's not exactly what they want, how they want it.

- 5 Small Company Development team for company's only product that makes everyone in the company's salary.

Quality Concern: Feature set, schedule and cost- low defects marginally next.

Development Lifecycle for Test Group



Requirements

What are Requirements?

Who writes them?

Where do we get them?

Are they worth anything?

Requirements - the customer's problem or desires

Specification - what you intend to make to solve
the problem or meet the desire

Design - how the thing works

For example:

Requirement: the room should be warm

Specification: The room will be kept between 25
and 28 degrees.

Design: Heat pump, augmented by electric heater

Detailed Design: ductwork diagrams, etc

Requirements

Sometimes Requirements are written documents.
Sometimes all we get from a Team are
Storyboards or a Prototype.

The formats can be very different. What they look
like is unique to every company and writer.
They are hopefully full of useful information.

- Target Platform
- System Requirements
- A description of the functionality
- Intended Use
- Expected Problem Areas

We'll do an example of Requirements later in the
course.

Product Reference Sources

- User Documentation
- Specification
- Requirement
- Product Proposal
- Memos, e-mails, notes and other correspondences
- Generate your own reference -
Formulate questions and ask
 - If possible, conduct your own research before asking the questions.

Typical Project

Product Development Milestones

- Product Design and Specification Complete
- First Functionality
- Pre-Alpha or Alpha Candidate
- Alpha
- Pre-Beta or Beta Candidate
- Beta
- User Interface Freeze
- Pre-Final or Golden Master Candidate (GMC)
- Final Test
- Release or Golden Master

These milestones are defined quite differently by different development groups.
Testing Computer Software, pages 266 - 302

Product Development Milestones

Product Design & Specification Complete

It has been determined what is to be built, what it will look like, what benefits it will provide, and how it will work.

First Functionality

The program has just enough capability that you can try it out.

Pre-Alpha or Alpha Candidate

The alpha milestone will be hit in a few days or weeks. Before the program can be declared “alpha” the Testing Group has to check that it meets the alpha criteria.

Alpha

A significant (and agreed-upon) portion, if not all, of the product (including documentation, code, additional art or other content, etc.) has been completed. The product is ready for first in-house use.

Product Development Milestones

Pre-Beta or Beta Candidate

A *beta candidate* has been submitted. If it meets the beta criteria (as verified by the Testing Group), the software is beta.

Beta

Most, if not all, of the product is complete. Some companies send “beta copies” of the product to customers at this milestone.

User Interface Freeze

Every aspect of the user interface of the software has been frozen. Some companies except error messages and help screens.

Pre-Final or GMC

A *final candidate* has been submitted. If all the requirements have been met, it is final.

Final Test

Last wave of testing before it gets inflicted on customers.

Release or Golden Master

Variation in Milestone Definitions

Alpha definitions:

- a. All code complete (but not necessarily working).
- b. Core functionality complete (but not necessarily useful).

Beta definitions:

- a. Reasonably predictable as X weeks (e.g. 6) before shipment.
 - All known Level 1 (crash or data loss) errors corrected.
 - No known bugs look so complex that they might not be fixed before shipment.
 - X% (such as 90%) of all Level 2 (non-crash but serious errors that should be fixed) bugs reported have been closed and the total number of remaining bugs is less than K.
- b. All product components (code and non-code) are available, functional, and installable.
- c. Customer useful. All critical features coded. No known bugs that could damage an external beta user's data or embarrass the publisher.

The point is not to decide which definition is correct. The point is to understand which definition is in use.

Notes

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Testing Computer Software

Part 2

An Overview of the Testing Organization

Role of the Testing Group

Quality Assurance?

- Many testing groups call themselves “Quality Assurance.” This can give the wrong idea.
- A QA group has the resources and authority to set standards and to solve problems. This group owns the quality of the project, and thus the satisfaction of the company’s customers.

For example, the *real* head of QA can send programmers to courses in effective software engineering. The role includes prevention as well as late-stage damage control.

- Normally, the *real* head of QA in a company is the President or a Senior Vice-President.

Testing Computer Software, pages 345 – 349

Read Juran’s comments on the Taylor theory of management, referenced in “Negotiating Testing Resources” in the Appendix

Role of the Testing Group

The Testing Group provides important technical services, including:

- finding and reporting bugs.
- identifying weak areas of the program.
- identifying high risk areas of a project.
- explaining findings in ways that help
 - engineering solve or fix the problems.
 - the customer service staff help customers.
 - management make sound business decisions about each bug.

Typical Responsibilities of a Tester

Find Problems

- Find bugs.
- Find design issues.
- Find more efficient ways to find bugs.

Communicate Problems

- Report bugs and design issues.
- Report on testing progress.
- Evaluate and report the program's stability.

More Senior Testers Manage/Supervise Testing Projects

- Prepare test plans and schedules.
- Estimate testing tasks, resources, time and budget.
- Measure and report testing progress against milestones.
- Teach other testers to find bugs.

More on Responsibility and Career Paths for Testers

See the American Society for Quality's *Body of Knowledge for the Certified Software Quality Engineer* in the Appendix.

- <http://www.asqc.org/standcert/cert.html>
- <http://www.asqc.org/standcert/certification/csqe.html>

Notes

[illegible]

Testing Computer Software

Part 3

Themes of the Course

Themes of the Course

- Management (not Testing) heads up QA.
- Many development methods work.
- The program doesn't work. Therefore, your task is to find errors.
- Be methodical.
- Complete testing requires a nearly infinite series of tests.
- Look for powerful representatives of classes of tests.
- Cost-justify your processes.
- Think in terms of quality costs.
- Communication must be effective.
- Change is normal.
- Test planning can be evolutionary.

Management Heads Up QA

- A product's head of QA is the person who makes the decisions that determine its quality. This happens long before the start of testing.
- Test a lousy program forever and you'll end up with an expensive, well-tested, lousy program.
- Your task is to find and clearly report software errors.

Testing Computer Software, Chapter 15

Many Development Methods Work

- There are many different successful approaches to managing projects.
- Companies and project managers have their own styles.
 - Some are much less formal than others.
 - Some are quite formal, but aren't in the traditional "waterfall" mold.
- You will want to form your own opinions and develop your own approaches.

Testing Computer Software, page 255

The Program Doesn't Work

- Nobody would pay you to test if their program didn't have bugs.
- All programs have bugs.
- Any change to a program can cause new bugs, and any aspect of the program can be broken.
- You DON'T "verify that the program is working." You FIND bugs.

If you set your mind to show that a program works correctly, you'll be more likely to miss problems than if you want and expect the program to fail.

Be Methodical

- Black box testing isn't just banging away at the keyboard.
- To have any hope of doing an efficient job, you must be methodical:
 - Break the testing project down into tasks and subtasks so that you have a good idea of what has to be done and what has been done.
 - Track what has been done so that people don't repeat the same tasks and you know what tasks are left.
 - Prioritize the tasks.

Testing Computer Software, page 216

Complete Testing is Impossible

- There are a nearly infinite number of paths through any non-trivial program.
- There are a virtually infinite set of combinations of data that you can feed the program.

You can't test them all.

- Therefore, your task is to find bugs -- not to find *all* the bugs.
- You want to
 - find as many bugs as possible
 - find the most serious bugs
 - find bugs as early as possible
- Your challenges will require judgment, trade-offs, and efficiency.

Testing Computer Software, pages 17-22

Use Powerful Representatives of Tests

Develop powerful tests by analyzing

- equivalence classes
- boundary conditions
- input combinations
- data/functionality relationships

Cost-Justify Your Processes

- Every task you do costs time and money.
- You are often asked to do several tasks that don't directly help you find and report more bugs.

Every minute that you spend on one of these tasks is a minute that you aren't finding bugs, reporting bugs, or finding a more efficient way to find or report bugs.

- Use your creativity to develop efficient ways to improve your productivity.

Think in Terms of Quality Costs

Testing accounts for only *some* of the money that your company spends on quality. Quality-related costs include:

- Prevention costs: everything the company spends to prevent software and other product related errors.
- Appraisal costs: all testing costs, including the cost of everything else the company does to look for errors.
- Internal failure costs: the costs of coping with errors discovered during development and testing.
- External failure costs: the costs of coping with errors discovered, typically by your customers, after the product is released.

Analyzing situations and problems in these terms will give you a broader understanding, and will often help you make your arguments more effectively or find allies for your position.

Communication Must be Effective

- Your bug reports advise people of difficult situations. The problems you report can affect
 - the project schedule
 - the company's cash flow
- The clearer your reports are, the more likely it will be that the company will make reasonable business decisions based on them.
- Persuasive and technical writing, oral argument, face-to-face negotiation, and diplomacy are core skills for your job.

Change is Normal

- Project requirements change, as do design and specifications.
- The market changes.
- Development groups come to understand the product more thoroughly as it is being built.
- Some companies accept late changes into their products as a matter of course.

As a new tester, you might decide quickly that this is a bad, amateurish way to do business.

It might be, and it might not be.

Take steps to make yourself more effective in dealing with late changes.

Test Planning Can Be Evolutionary

- If you write your full test plan, develop all the cases, and then test, much of the planning and development might be made obsolete by program *design changes* before you run a single test on the changed part of the program.
- Develop your test plan gradually, in parallel with the testing effort.

Testing Computer Software, Chapters 1 & 13

Notes

[illegible]

Testing Computer Software

Part 4

Costs Associated with Testing and Quality

Software Development Tradeoffs

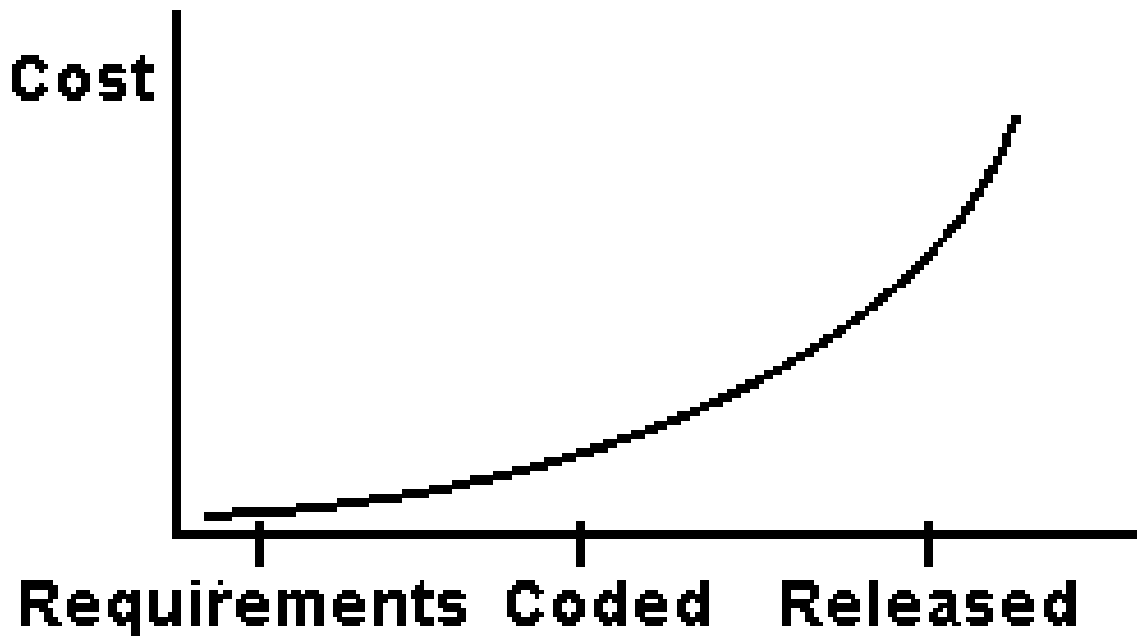
There are at least three factors project managers consider in making tradeoffs throughout the product's development.

- Quality
 - Features
 - Reliability
 - Usability
- Schedule (calendar time)
- Resource
 - Availability
 - Cost (money)

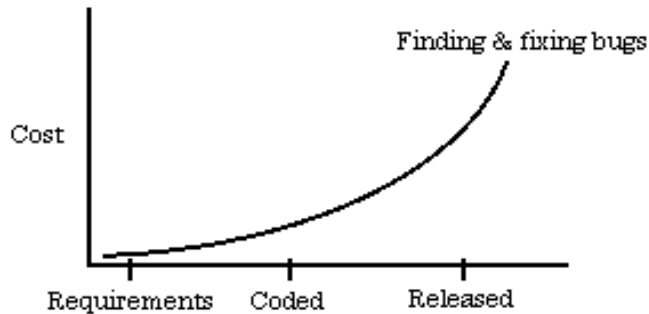
Different development methodologies impose different constraints on the project manager's freedom to make these tradeoffs.

Cost of Finding and Fixing Software Errors

Bugs cost more later
(so find them sooner)



Cost of Finding and Fixing Software Errors



These costs escalate because people are affected by bugs, and are more severely affected as the product gets closer to release. We all know the obvious:

- Bugs in requirements can be fixed without anything having to be recoded.
- Programmers who find their own bugs can fix them without taking time to file bug reports or explain them to someone else.
- It is hugely expensive to deal with bugs in the field (in customers' hands).

Along with this, there are many effects on other stakeholders in the company. For example, consider the marketing assistant who spends days trying to create a demo, but can't because of bugs.

Quality Cost Analysis

Quality Cost Measurement is a cost control system used to identify opportunities for reducing controllable quality-related costs

A key goal of the quality engineer is to help the company minimize its cost of quality.

Refer to “Quality Cost Analysis: Benefits & Risks” in the Appendix.

Quality Cost Analysis

The **cost of quality** is the total amount the company spends to achieve and cope with the quality of its product.

This includes the company's investments in improving quality and its expenses arising from inadequate quality.

Quality-Related Costs

Prevention	Appraisal
All cost of preventing software errors, documentation errors, and any other sources of customer dissatisfaction.	All costs of all types of inspection (testing).
Internal failure	External failure
ALL costs of coping with errors discovered during development.	All costs of coping with errors discovered, typically by your customers, after the product is released.

Categories of Quality Costs

Prevention	Appraisal
Staff training Requirements analysis Early prototyping Fault tolerant design Defensive programming Usability analysis Clear specification Accurate internal documentation Pre-purchase evaluation of the reliability of development tools	Design review Code inspection Glass box testing Black box testing Beta testing Test automation Usability testing Pre-release out-of-box testing by customer service staff
Internal failure	External failure
Bug fixes Regression testing Wasted in-house user time Wasted tester time Wasted writer time Wasted marketer time Wasted advertisements Direct cost of late shipment Opportunity cost of late shipment	Technical support calls Answer books (for Support) Investigating complaints Refunds and recalls Interim bug fix releases Shipping updates product Warranty, liability costs PR to soften bad reviews Lost sales Lost customer goodwill Supporting multiple versions in the field Reseller discounts to keep them selling the product

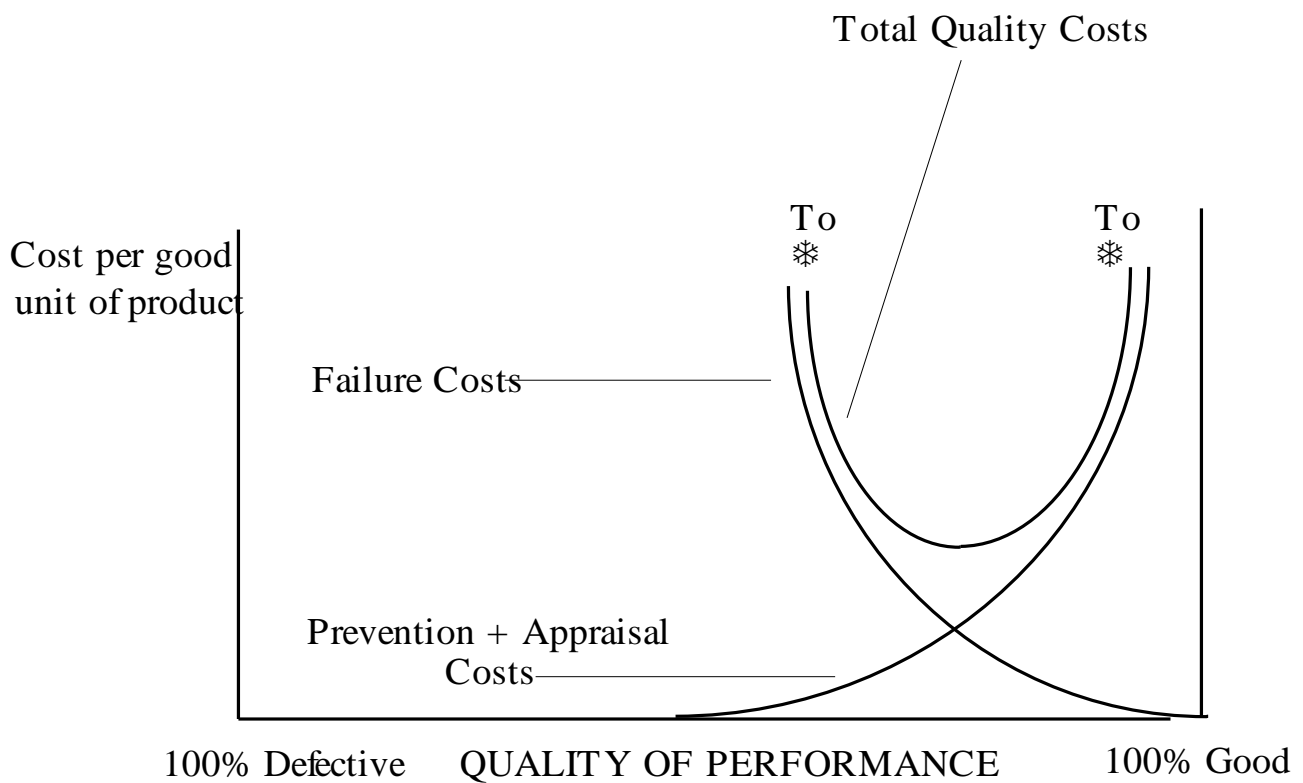
Fundamentals of Quality Cost

An Example of the Theoretical Model of Optimum Quality Cost

	\$ Scenario 1	%	\$ Scenario 2	%	\$ Scenario 3	%
Prevention	\$ 10,000.00	3%	\$ -	0%	\$ 20,000.00	6%
Training	\$ 10,000.00	3%	\$ -	0%	\$ 20,000.00	6%
Appraisal	\$ 130,000.00	36%	\$ 30,000.00	5%	\$ 90,000.00	26%
Testing	\$ 130,000.00	36%	\$ 30,000.00	5%	\$ 90,000.00	26%
Total Prevention + Appraisal	\$ 140,000.00	38%	\$ 30,000.00	5%	\$ 110,000.00	32%
Internal Failure	\$ 150,000.00	41%	\$ 40,000.00	7%	\$ 142,000.00	42%
Bug fixing	\$ 120,000.00	33%	\$ 30,000.00	5%	\$ 115,000.00	34%
Regression test on fixes	\$ 30,000.00	8%	\$ 10,000.00	2%	\$ 27,000.00	8%
External Failure	\$ 75,000.00	21%	\$ 495,000.00	88%	\$ 90,000.00	26%
Technical Support	\$ 60,000.00	16%	\$ 450,000.00	80%	\$ 70,000.00	20%
Returns	\$ 15,000.00	4%	\$ 45,000.00	8%	\$ 20,000.00	6%
Total Failure	\$ 225,000.00	62%	\$ 535,000.00	95%	\$ 232,000.00	68%
TOTAL QUALITY COST	\$ 365,000.00	100%	\$ 565,000.00	100%	\$ 342,000.00	100%

Fundamentals of Quality Cost

Theoretical Model of Optimum Quality Cost



Quality Cost Analysis: Risks of this Approach

Quality Cost analysis provides several benefits:

- New opportunities to find / create common ground with other groups in the company.
- Analysis tool for budgeting and planning quality-related activities.
- Excellent communication tool for senior management.

But there are some risks too. We might unwittingly open ourselves up to customer dissatisfaction and perhaps, legal problems:

- Don't think you've solved customer dissatisfaction problems by driving down support costs.
- Don't consider only your own external failure costs. Know the costs to your customer.

Reducing External Failure Costs: Call Avoidance

Prevention (improve quality)

Diversion (to non-human sources of support)

Minimization (of staff time spent on direct support)

Evasion (delay responding due to minimization or charge for support)

Unfortunately, companies can limit back-end failure costs without improving customer satisfaction.

Notes

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Testing Computer Software

Part 5

An Example Test Series: The Program's First Tests

Example Test Series

The program's specification:

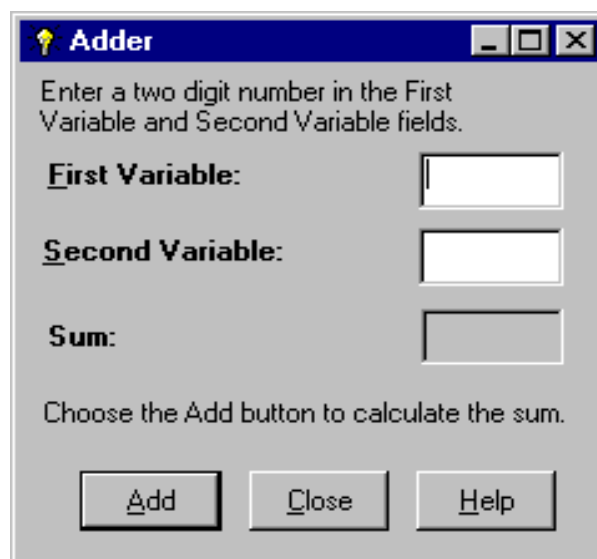
- This program is designed to add two numbers, which you will enter
- Each number should be one or two digits
- The program will echo your entries, then print the sum.
- Press `Add` to do the calculation.

The First Cycle of Testing

1. Start with an obvious and simple test.
2. Make some notes about what else needs testing.
3. Look for boundary conditions.
4. Check the valid cases and see what happens.
5. Do some random testing.
6. Summarize what you know about the program and its problems.

Analyzing an Error Exercise 1 (1)

Launch the Adder application



The screenshot shows a classic Windows-style application window titled "Adder". The window has a blue title bar with a yellow lightbulb icon on the left and standard minimize, maximize, and close buttons on the right. The main content area is light gray and contains the following elements:

- Instructional text: "Enter a two digit number in the First Variable and Second Variable fields."
- Input fields: Three white rectangular text boxes with thin gray borders.
- Labels: The labels "First Variable:", "Second Variable:", and "Sum:" are positioned to the left of their respective input fields. The "First Variable:" and "Second Variable:" labels have an underline under the first letter.
- Bottom instruction: "Choose the Add button to calculate the sum."
- Buttons: Three buttons are located at the bottom of the window. The first button is labeled "Add" with an underline under the 'A'. The second button is labeled "Close" with an underline under the 'C'. The third button is labeled "Help" with an underline under the 'H'.

1. Start with an Obvious Test

The Problem Report Form

YOUR COMPANY'S NAME _____ CONFIDENTIAL _____ PROBLEM REPORT # _____

PROGRAM _____ RELEASE _____ VERSION _____

CONFIGURATION _____

REPORT TYPE (1-6) _____ SEVERITY (1-3) _____ ATTACHMENTS (Y/N) _____

1 - Coding error 4 - Documentation 1 - Fatal If yes, describe _____

2 - Design Issue 5 - Hardware 2 - Serious _____

3 - Suggestion 6 - Query 3 - Minor _____

PROBLEM SUMMARY _____

CAN YOU REPRODUCE THE PROBLEM (Y/N/S/U) _____

PROBLEM AND HOW TO REPRODUCE IT _____

SUGGESTED FIX (optional) _____

REPORTED BY _____

DATE ____/____/____

2. What Else Needs Testing?

What?

Why?

Testing Computer Software, pages 4-5

3. Look for Boundaries

- Two tests belong to the same ***equivalence class*** if you expect the same result (pass/fail) of each. Testing multiple members of the same equivalence class is, by definition, redundant testing.
- ***Boundaries*** mark the point or zone of transition from one equivalence class to another. The program is more likely to fail at a boundary, so these are the best members of equivalence classes to use.

Equivalence Class & Boundary Analysis

There are $199 \times 199 = 39,601$ test cases for valid values:

- definitely valid: 0 to 99
- might be valid: -99 to -1

There are infinitely many invalid cases:

- 100 and above
- -100 and below
- anything non-numeric

Equivalence Classes

Variable	Valid Case Equivalence Classes	Invalid Case Equivalence Classes	Notes
First number	-99 to 99	> 99 < -99 non-number expressions	max bounds min bounds See ASCII bounds in the next section. That yield interesting (invalid) results. 0 is always interesting. Other sources?
Second number	same as first	same as first	same
Sum	-198 to 198		

The simplest way to build an equivalence class analysis over time is to put the information gathered into a table. The table should eventually contain all variables. This means all input variables, all output variables, and any observable intermediate variables. In constructing this table, List all (or many) of the variables first, filling in information about them as it is obtained.

Boundary Analysis Table

Variable	Valid Case Equivalence Classes	Invalid Case Equivalence Classes	Boundaries and Special Cases	Notes
First number	-99 to 99	> 99 < -99 non-number expressions	99, 100 -99, -100 / ; null entry 0	max bounds min bounds See ASCII bounds in the test case design section. That yield interesting (invalid) results. 0 is always interesting. Are there other sources of data for this variable? Ways to feed it bad data?
Second number	same as first	same as first	same	same
Sum	-198 to 198			

The boundary analysis table is an improvement on the equivalence class table. There is one additional column -- Boundaries and Special Cases. These are the specific values to be used as test cases.

4. Tests of Valid Input

Test Case	Expected Results	Notes

5. Exploratory, Random and Ad Hoc Testing

This is not systematic but it is intense testing. The goal is to *probe* for weak *areas* of the program. Walk quickly through all areas of the program. Check various error conditions, such as:

- Out of boundary
- Null input and overflow (storage of the sum)
- Non-numeric
- Hardware and memory error handling

6. Summarize What You Know and What You'd Check Next

What types of tests would you run next time?

For example:

- Check underlying keyboard boundaries, editing keys
- Look for device-related failures
- Look for output-related failures
- Look for maximum number of digits failures
- Think about what keys users might use along with numbers

The Second Cycle of Testing

- 1) Before doing any testing, review the responses to the problem reports to see what has to be done and what can't be done yet.
- 2) Review comments on the problems that won't be fixed. They may suggest the need for further tests.
- 3) Pull out the notes from last time, add the new notes to them, and start testing.

Notes

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Testing Computer Software

Part 6

The Impossibility of Complete Testing and the Problems of Coverage

The Impossibility of Complete Testing

If you test completely, then at the end of testing, there cannot be any undiscovered errors.

This is impossible because:

- 1) The domain of possible inputs is too large.
- 2) There are too many combinations of data to test.
- 3) There are too many possible paths through the program to test.
- 4) And then there are the user interface errors, the configuration/compatibility failures, and dozens of other dimensions of analysis.

1. Too Many Inputs

The domain of possible inputs is too large

- Test all valid inputs
- Test all invalid inputs
- Test all edited inputs
- Test all variations on input timing

2. Too Many Combinations

Variables interact.

- For example, a program crashes when attempting to print preview a high resolution (say, 800x600 dpi) image on a high resolution screen. The option selections for printer resolution and screen resolution are interacting.
- For example, a program fails when the sum of a series of variables is too large.

Suppose there are N variables. Suppose the number of choices for the variables are V_1 , V_2 , through V_N . The total number of possible combinations is $V_1 \times V_2 \times \cdots \times V_N$. This is huge.

We saw 39,601 combinations of just two variables whose values could range only between -99 and 99.

Here's a case that isn't so trivial. **There are 318,979,564,000 possible combinations of the first four moves in chess.**

3. Too Many Paths

Here's an example that shows that there are too many paths to test even in a fairly simple program. This one is from Myers, The Art of Software Testing.

The program starts at A.

From A it can go to B or C

From B it goes to X

From C it can go to D or E

From D it can go to F or G

From F or from G it goes to X

From E it can go to H or I

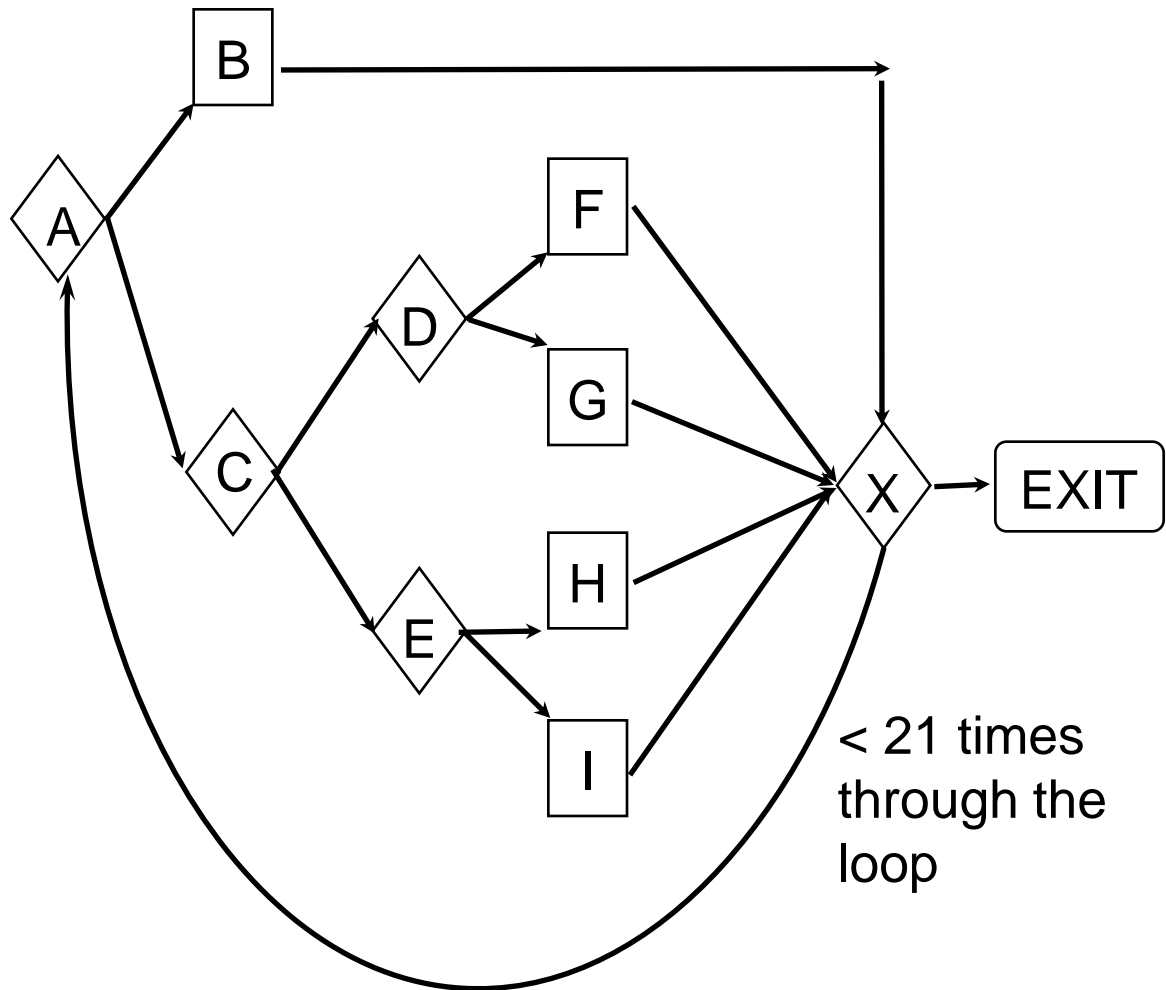
From H or from I it goes to X

From X the program can go to EXIT or back to A. It can go back to A no more than 20 times.

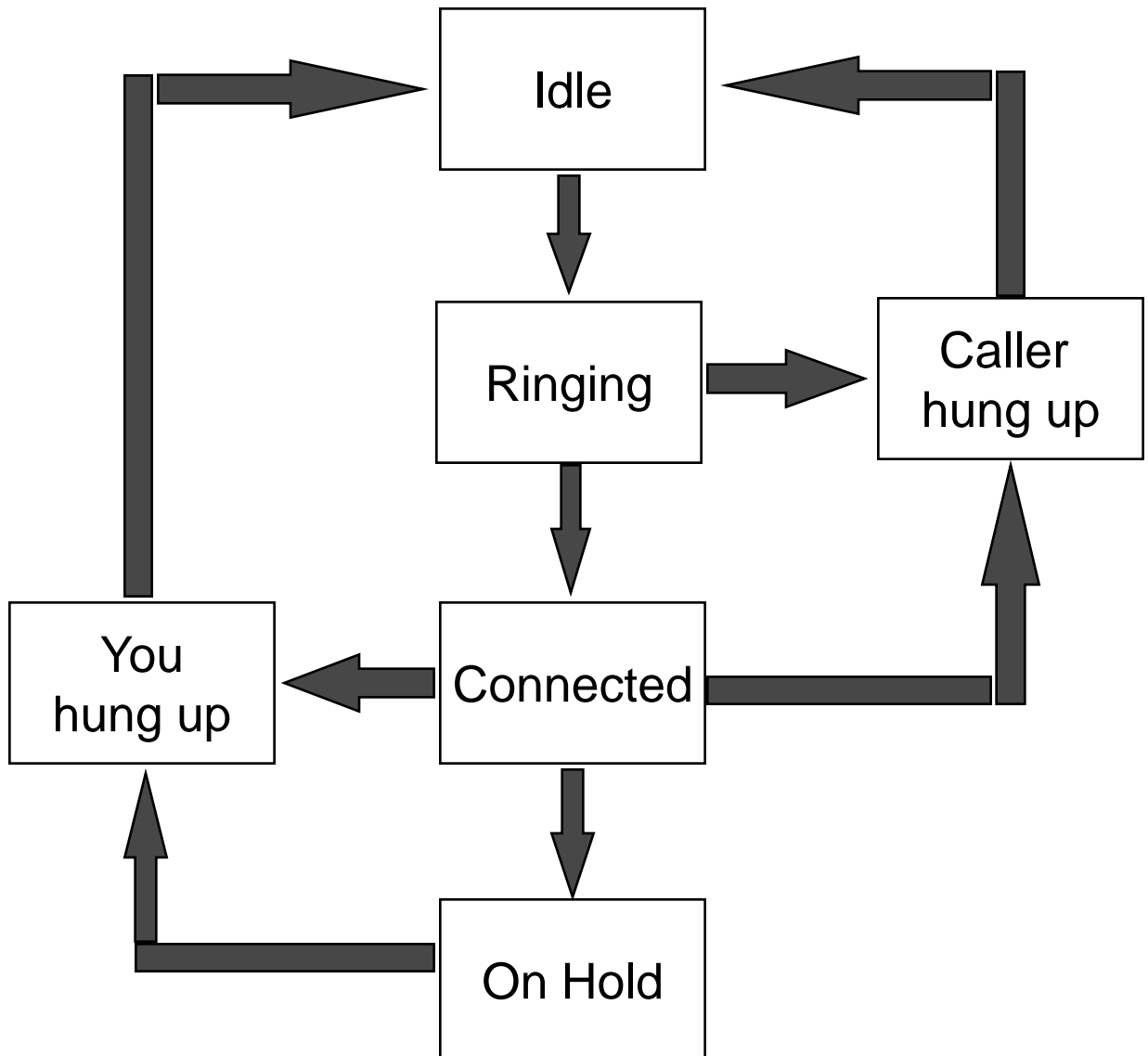
- One path is ABX-Exit. There are 5 ways to get to X and then to the EXIT in one pass.
- Another path is ABX-ACDFX-Exit. There are 5 ways to get to X the first time, 5 more to get back to X the second time, so 25 cases like this.

3. Too Many Paths (cont.)

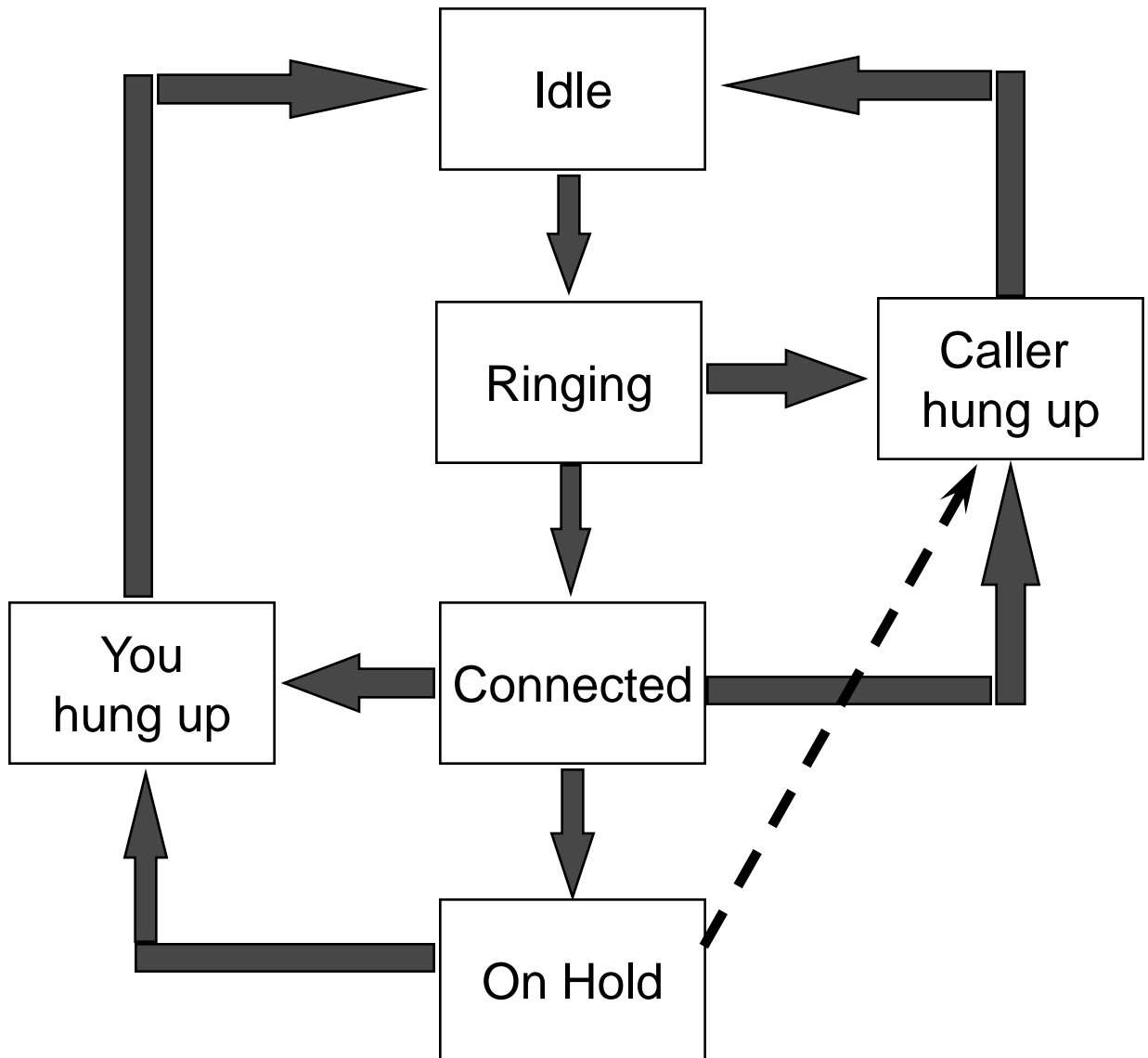
There are $5^1 + 5^2 + \dots + 5^{19} + 5^{20} = 10^{14} = 100$ trillion paths through the program to test or approximately one billion years to try every path (assuming one could write, execute and verify a test case every five minutes).



3. Too Many Paths (cont.)



3. Too Many Paths (cont.)



Way Too Many Paths + Combos

Suppose there are N variables. Suppose the number of choices for the variables are V_1, V_2, \dots, V_N . The total number of possible combinations is $V_1 \times V_2 \times \dots \times V_N$. (assuming the choices are independent.)

Suppose there are M binary decisions. The total number of choices is therefore 2^M (assuming the decisions are independent).

Suppose there are O loops, and the maximum number of times that loop i can loop is L_i . There are up to $L_1 \times L_2 \times \dots \times L_O$ combinations of different number of times through the loops.

So we have up to

$$V_1 \times V_2 \times \dots \times V_N \times 2^M \times L_1 \times L_2 \times \dots \times L_O$$

distinct test cases.

Way Too Many Paths + Combos (cont.)

To put these crazy numbers in perspective, if you could run one test per minute, and you tested 24 hours/day, with no vacations, you could run one billion test cases in 1902.6 years.

To test the first four moves of chess (318,979,564,000 test cases) would take a mere 606,886.5 years.

Therefore, any set of tests we run is a tiny sample of the universe of tests that we could run if we had infinite time. We are always working against insufficient time, looking for powerful samples.

The Problem of Coverage

Coverage measures of the amount of testing done of a certain type. Since testing is done to find bugs, coverage is a measure of your effort to detect a certain class of potential errors:

- **100% line coverage** means that you tested for every bug that can be revealed by simple execution of a line of code.
- **100% branch coverage** means you will find every error that can be revealed by testing each branch.
- **100% coverage** means that you tested for every possible error. This is obviously impossible.

***So what kind(s) and level(s) of coverage are considered appropriate?
There is no magic answer.***

The Problem of Coverage

IEEE Unit Testing Standard is

- 100% Statement Coverage
- 100% Branch Execution

(IEEE Std. 982.1-1988, § 4.17, “Minimal Unit Test Case Determination”)

Most companies do not achieve this.

Several people seem to believe that complete statement and branch coverage means complete testing. (Or at least, sufficient testing.)

Line & Path Coverage Are Not Complete

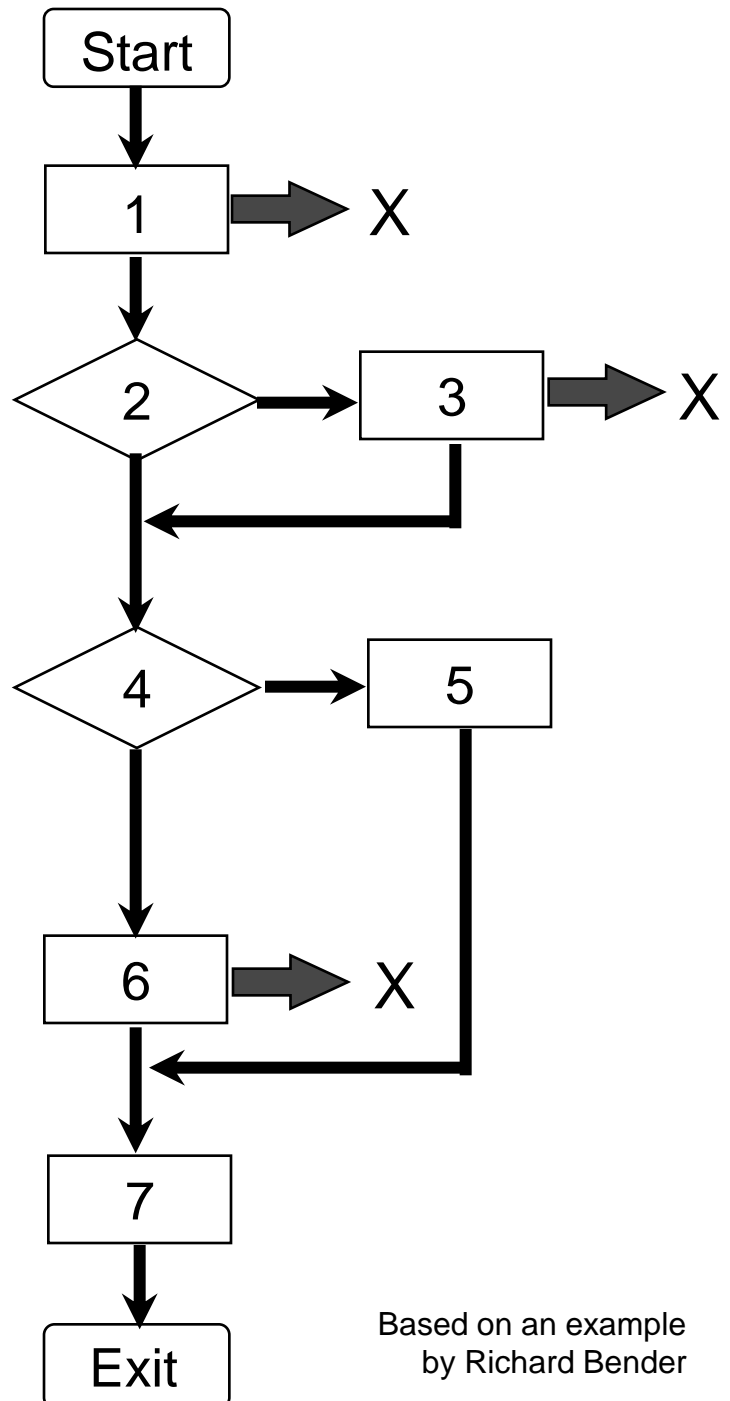
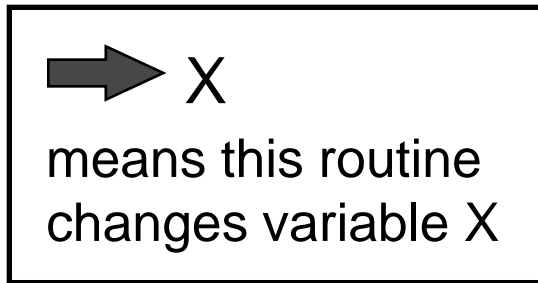
All you are testing is the flowchart.

You are not testing:

- Data flow
- Tables that determine control flow in table-driven code
- Side effects of interrupts, or interaction with background tasks
- Special values, such as boundary cases. These might or might not be tested.
- Unexpected values (e.g. divide by zero)
- User interface errors
- Timing-related bugs
- Compliance with contracts, regulations, or other requirements
- Configuration/compatibility failures
- Volume, load, hardware faults

Brian Marick, author of [The Craft of Software Testing](#), teaches the limitations of these simple coverage measures particularly well.

One Last Example: Data Flows



1 (x) 2 3 (x) 4 5 7
1 (x) 2 4 6 (x) 7
*Now we have 100%
branch coverage, but
where is 1(x) 7?*
1 (x) 2 4 5 7

Based on an example
by Richard Bender

More on Coverage

Line and branch coverage are easy to measure, but they are not the only available measures of the extent of testing.

Read the discussion of 101 different coverage measures, “Software Negligence and Testing Coverage” in the Appendix.

Notes

[illegible]

Testing Computer Software

Part 7

Objectives of Testing

Objectives of Testing

Your objective should not be to verify that the program works correctly because:

- if you can't test the program completely, you can't verify that it works correctly.
- the program *doesn't* work correctly, so you can't verify that it does.
- then as a tester, you fail to reach your goal every time you find an error.
- you'll be more likely to miss problems than if you want and expect the program to fail.

The Objective of Testing a Program is to Find Problems

Finding problems is the core of your work. You should want to find as many problems as possible. The more serious the problem, the better.

A test that reveals a problem is a success. A test that did not reveal a problem is (often) a waste of time.

The Purpose of Finding Problems is to Get Them Fixed

The point of the exercise is quality improvement.

The best tester isn't the one who finds the most bugs or who embarrasses the most programmers.

The best tester is the one who gets the most bugs fixed.

A Different Take on Coverage: Public vs. Private Bugs

A programmer's **public bug rate** includes all bugs left in the code when it is given to someone else (such as a tester.) Rates of one bug per hundred statements, or even three hundred, are not unusual.

A programmer's **private bug rate** includes all bugs ever created in the code, including the ones fixed before passing the program to testing.

Estimates of private bug rates have ranged from 15 to 150 bugs per 100 statements. Therefore, programmers must be finding and fixing between 80% and 99.3% of their own bugs before their code goes to test. (Even the sloppy ones find and fix a lot of their own bugs.)

What does this tell us?

It says that we are looking into the programmer's blind spots. Merely repeating the same types of tests that the programmers did will not yield more bugs. That is one of the reasons why an alternative approach is so valuable.

Notes

[illegible]

Tips On Hitting The Wall

What do you do when you've been testing the product for a while and you finally hit the wall?

A few suggestions:

- Manual-driven testing for fresh perspective
- Rotate program areas
- Bring in users for observation
- Bring in users and have them test
- Pull out top bugs, see where the weak areas of the program are, and re-attack
- Create complex documents (use competitors' demos)
- Automation

Notes

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Testing Computer Software

Part 8

Getting Started

Common Terminology

Test Case: A test that (ideally) executes a single well defined test objective (i.e., a specific behavior of a feature under a specific condition).

Test Script: A particular set of step-by-step instructions describing how a test case is executed. A test script may contain one or more test cases.

Test Suite: A collection of test scripts or test cases used for validating bug fixes (or finding new bugs) within a logical or physical area of the tested product. For example, an acceptance test suite contains all the test cases used to validate that the software has met a certain predefined acceptance criteria; a regression suite contains all the test cases used to validate that all previously fixed bugs are still intact.

Common Terminology (cont.)

Test Specification: A set of test cases, input, and/or conditions to be used in testing a particular or an entire set of features in an application. A test specification often includes descriptions of the expected results.

Test Requirement: A document describing the items and features to be tested under a required condition.

Common Terminology (cont.)

Test Type: A specific logical testing objective in which a collection of tests attempts to expose errors or verify the correctness of the application's behavior.

Read “Test Types and Their Place in the Software Development Cycle” in the Appendix.

Starting a List of Black Box Test Types...

Acceptance	Security	Smoke
Regression	Database	Quick Look
Ad Hoc		End-to-End
Random	Configuration	Shotgun
Unstructured	Compatibility	Sanity
Gorilla/Guerilla		
	UI	User Scenario
Load	Usability	Real World
Volume	Documentation	Out of Box
Stress	Help	
Performance	Collateral	Memory
Boundary/Limit		
	Internationalization	
Installation	Localization	And last...
Forced Error Handling		Functionality

Test Types

TIME →

Begin Alpha Testing Phase				
Begin Beta Testing Phase		Begin Final Testing Phase		
			Golden Master	
				Ship
Alpha Phase	Beta Phase	Final Phase		
TYPES OF TESTS RECOMMENDED				
Configuration Compatibility *	Real World User-Level Test	Install/Uninstall Test		
Task-Oriented Functional Test	Unstructured Test	Configuration Compatibility ^		
Boundary Test	Task-Oriented Functional Test	Real World User-Level Test		
Stress (Memory-related) ^^	Forced-Error Test	Unstructured Test		
Unstructured Test	Full Configuration Compatibility Test			
	Volume Test			
	Stress Test			
	Install/Uninstall Test			
	Performance Test			

Common Terminology (cont.)

Test Condition: A condition in which an application under test operates.

- Application specific condition

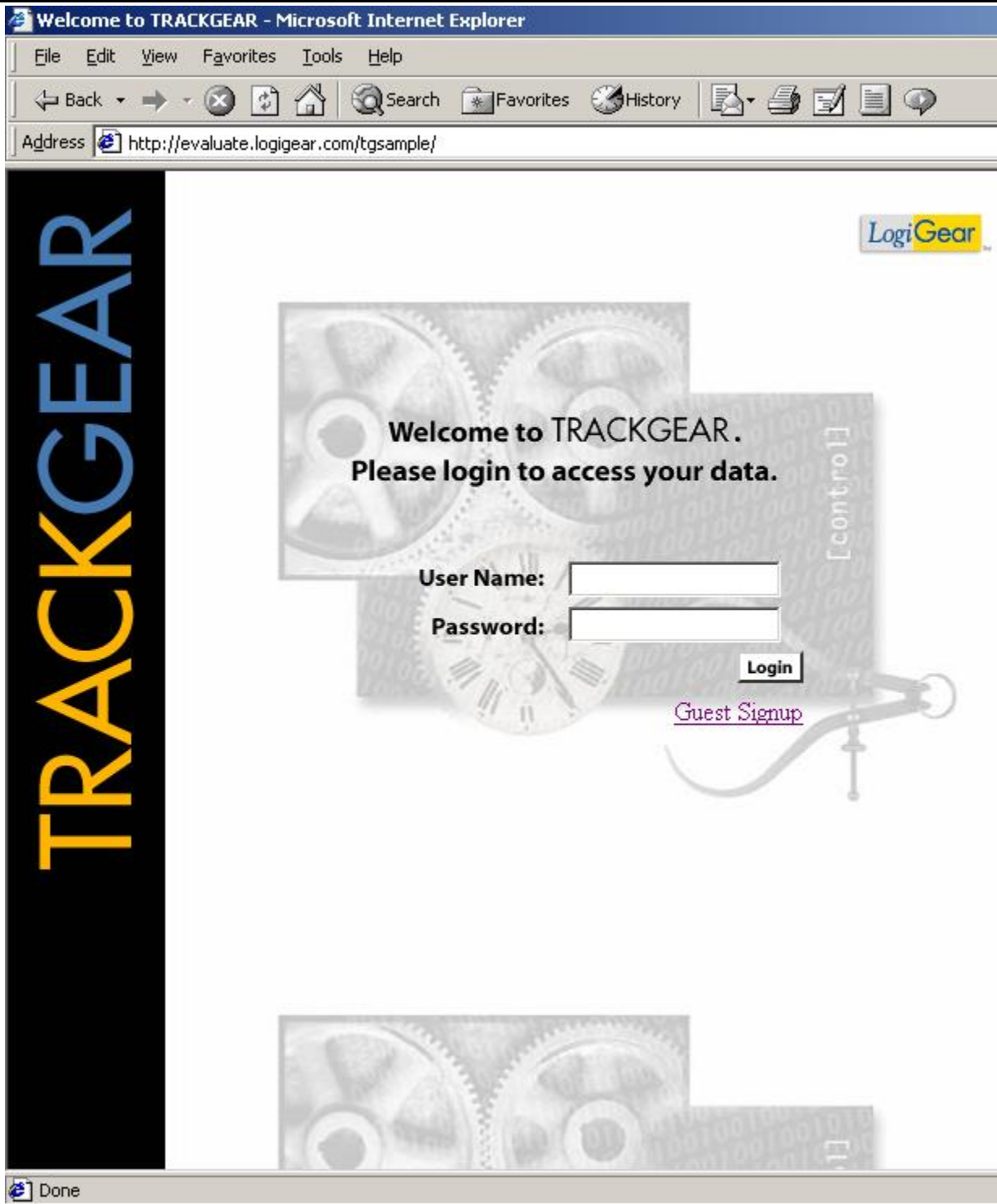
For example, run the same spell-check test in Page View mode will crash the application (It does not crash in Normal View mode).

- Environmental specific condition
 - Dynamic environment (resource such as RAM or disk space)
 - Static environment (compatibility)

Frequently Asked Questions

- What types of tests do I do?
- What reference do I use in testing the product?
- How do I test?
- How do I know if there is an error?
- How do I report an error?

Testing Example



Start with the Requirements

User Name:

- 4-8 Characters
- Mix of Alpha-Numeric
- Unique Error Messages.

Password:

- 4 – 8 characters
- Must be mix of Alpha-Numeric
- Mix of Upper and Lower case
- Cannot be the same as User Name.

Functional Error Handling:

- 3 failed attempts and you are blocked out of the system. System Admin must let you back in.

[illegible]

Requirements Example

Control	ID/Name	Type	minimum value	maximum value	Valid Values	Invalid values	Detection Rules	
ID	user_name	HTML Test Box	4	8	[A-Z],[a-z],[0-9],_		none	
Password	password	HTML Test Box	4	8	[A-Z],[a-z],[0-9],_,-,.,!,@,\$,^,(*),*		none	
Login		Image						
Guest Signup		Link						
Hidden form field	company							
Hidden form field	page							

Requirements Example Part 2

Testing Requirements October 8, 2001

For the purposes of Launch 1.0 testing:

System Requirements

Browser

- Microsoft Internet Explorer 4.0 and higher
- Settings required:
- Javascript enabled
- Cookies enabled
- Volume enabled
- SSL enabled

Platform

- Windows 98 & 2000

Plug-Ins

- Flash 4.0 and higher

Media Player

- Microsoft Windows Media Player 7.0

Hardware

- At least a Pentium II, 300 MHZ processor
- 64 MB or more RAM memory
- At least 800X600 monitor size, 16 bit color
- At least 1 GB of free space on hard drive

* NOTE: Will require limited testing of "failed" system requirements, i.e. Netscape, Mac, Real Media Player, etc.

Connection Speed

Bandwidth tested under following connection rates

- Less than 128 kbps
- Between 128 kbps and 300 kbps
- Over 300 kbps

Let's Add more Requirement- Add more tests.

What tests would you add with the addition of the other requirements?

Notes

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Testing Computer Software

Part 9

Software Errors

Quality and Software Errors

How do *you* define quality?

What do *you* think is the proper definition of a software error?

What is Quality?

- Fitness for use (Dr. Joseph M. Juran).
- The totality of features and characteristics of a product that bear on its ability to satisfy a given need (American Society for Quality Control).
- Conformance with requirements (Philip Cosby).
- The total composite product and service characteristics of marketing, engineering, manufacturing and maintenance through which the product and service in use will meet expectations of the customer (Armand V. Feigenbaum).

What is Quality? (cont.)

Juran distinguishes between Customer Satisfiers and Dissatisfiers as key dimensions of quality:

Satisfiers:

- right features
- adequate instruction

Dissatisfiers:

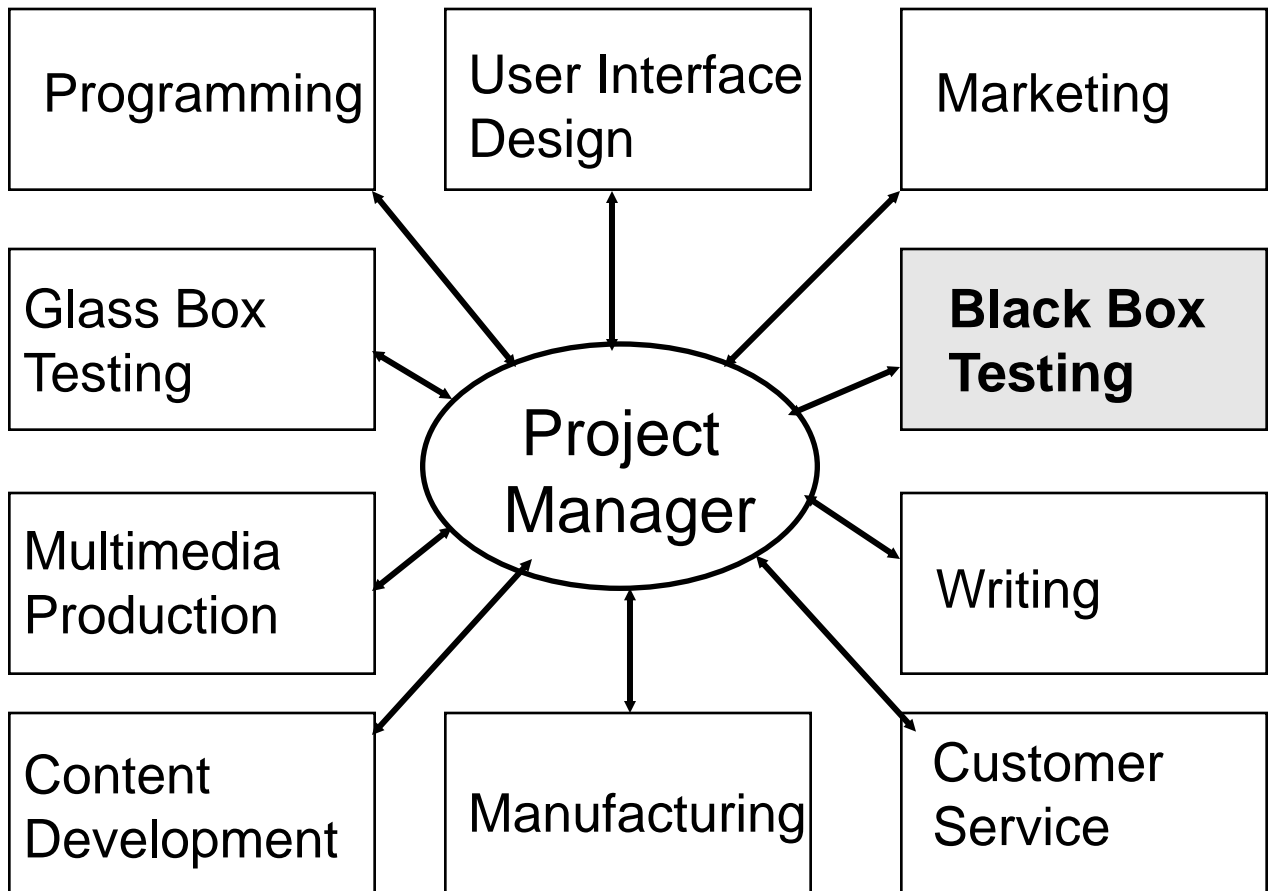
- unreliable
- hard to use
- too slow
- incompatible with the customer's equipment

What is Quality? (cont.)

We measure quality by evaluating the following three factors:

- The right features
- Reliability
- Usability (easy-to-use and reasonable performance)

Quality is Multidimensional



When you sit in a project team meeting, discussing a bug, a new feature, or some other issue in the project, you must understand that each person in the room has a different vision of what a “quality” product would be. Fixing bugs is just one issue. The next page gives some examples.

Quality is Multidimensional (cont.)

Programmers: *Great code is maintainable, well documented, easy to understand, well organized, fast, and compact.*

Tech Writers: *A high quality program is easily explainable. Aspects of the design that are confusing, unnecessarily inconsistent, or hard to describe are marks of bad quality.*

Marketing: *Customer satisfiers are the things that drive people to buy the product and to tell their friends about it.*

Customer Service: *Good products are supportable. They have been designed to help people solve their own problems or to get help quickly.*

Localization Manager: *A good product is easy to translate and modify to make it suitable for another country and culture.*

What is a Software Error?

A software error is present when the program does not do what its user reasonably expects it to do.

It is fair and reasonable to report any deviation from high quality as a software error.

The existence of software errors reflects an impediment on the quality of the product, but does not necessarily imply that the developers are incompetent.

Testing Computer Software, Chapter 4

Why are there Errors?

Is your programmer sloppy?

- New testers (and several old testers) often fall into a trap. They blame programmers for bugs and conclude that the programmers that they are working with are incompetent, uncaring, or unprofessional. This is counterproductive. It leads to infighting instead of communication, and it leads to squabbling over bugs instead of research and bug fixing. Additionally, programmers actually find and fix the large majority of their own bugs.

Why does software have bugs?

- Bugs come into the code for many reasons. It is worth considering some of the common causes because they vary a lot across companies. You will learn to vary your strategic approaches as you learn your companies' weaknesses.

Why are there Errors? (cont.)

Bugs exist due to many reasons:

- The major cause of error is that *programmers deal with tasks that aren't fully understood or fully defined.*

If you graduated from a Computer Science program, how much training did you have in task analysis? Requirements definition? Usability analysis? Negotiation and clear communication of negotiated agreements? Not much!

- *Late design changes* result in last minute code changes, which are likely to have errors.

Why are there Errors? (cont.)

Bugs exist due to many reasons:

- *Failure to use source control* tools creates characteristic bugs. For example, if a bug goes away, comes back, goes away, comes back, goes away, comes back, then ask how the programming staff makes sure it's linking the most recent version of each module when it builds a version for you to test.
- Some *third-party components introduce bugs*. Your program might rely on a large suite of small components that display a specific type of object, filter data in a special way, drive a specific printer, etc. Many of these tools, bought from tool vendors or hardware vendors, are surprisingly buggy.

Why are there Errors? (cont.)

Bugs exist due to many reasons:

- Similarly, some third party hardware, or its drivers, are non-standard and don't respond properly to standard system calls.
Incompatibility with hardware is often cited as the largest single source of customer complaints into technical support groups.
- Some programmers (on some platforms) work with *poor development tools*, buggy compilers, style checkers, debuggers, profilers, etc. make it too easy to get bugs or too hard to find bugs.
- When one programmer tries to fix bugs, or otherwise modify another programmer's code, there is a lot of room for *miscommunication* and error.
- Some companies *drive their programmers too hard*. They don't have enough time to design, bulletproof, or test their code.
- And, sometimes *people just make mistakes*.

Types of Errors

You will report all of these types of problems, but it is important to keep straight in your mind, and on the bug report, which type you are reporting.

- **Coding Error:** The program doesn't do what the programmer would expect it to do.
- **Design Issue:** It's doing what the programmer intended, but a reasonable customer would be confused or unhappy with it.
- **Requirements Issue:** The program is well designed and well implemented, but it won't meet one of the customer's requirements.
- **Documentation / Code Mismatch:** Report this to the programmer (via a bug report) and to the writer (usually via a memo or a comment on the manuscript).
- **Specification / Code Mismatch:** Sometimes the spec is right; sometimes the code is right and the spec should be changed.

13 Types of Software Errors

User Interface

Error Handling

Boundary-Related

Calculation

Initial and Later States

Control Flow

Handling or Interpreting Data

Race Conditions

Load Conditions

Hardware/Environment Compatibility

Source, Version, and ID Control

Testing

Documentation

Testing Computer Software, Chapter 4 & its Appendix A

Why Categorize Errors

Categorization helps you condense a huge list of problems into a conceptually manageable group of ideas. The list is valuable:

- Training value for new members of the project
- Areas to consider during early design discussions/review
- Test plan audit (including self-audit)
- Suggest new test cases

Any approach to categorizing errors is arbitrary. Our approach was to try to group errors together if they would require similar thinking processes to plan and find.

Testing Computer Software

Part 10

How to Report Software Errors

Testing Computer Software, pages 65-76, 91-97

Reporting Errors

**Bug reports are your
primary work product.**

Reporting Errors (cont.)

As soon as you run into a problem in the software, fill out a Problem Report form.

- Explain how to reproduce the problem.
- Analyze the error so you can describe it in a minimum number of steps.
- Write a report that is complete, easy to understand, and **non-antagonistic**.
- If a sample test file is absolutely essential to reproducing a problem, reference it and attach the test file to the report.

Characteristics of a Useful Problem Report

- Written
- Uniquely numbered
- **Simple** (non-compound - one bug per report)
- Understandable
- Reproducible
- **Non-judgmental**

The Problem Report Form

TRACKGEAR
Submit
New

Submit New Report

Save Save & Clone

Config. Info

PROJECT: TGSample BUILD: 1-alpha_01 Module: Unassigned

Config ID: Unassigned Attachment...

Error Type: Compatibility-SW Keyword: Browser- NN4x Reproducible: Yes

Severity: 2-Medium Frequency: 2-Medium Priority: 2-Medium

Summary

SUMMARY:

Adding Project Member page does not refresh properly to show new members.

Body

STEPS:

1. Log in as Admin.
2. Click the Project button in the Navigation bar.
3. Select the 'Project Members' radio button; click Go.
4. Add a new project member.

Notes & Comments:

Result: New member does not appear in the 'Project Members' list until the page is manually refreshed (F5).

Assigned: Auto Assigned Stopper: 4-GMC

Save Save & Clone

Report Content

The Problem Report Form

Two key components of the bug report are:

- Report Content;
- Tracking (including Decision Making and Processing) Management.

The effective bug reporting skills you want to develop, are for the most part in the Report Content component.

Report Content

Problem Summary:

This one-line description of the problem is the most important part of the report.

- The project manager will use it when reviewing the list of bugs that haven't been fixed.
- Executives will read it when reviewing the list of bugs that won't be fixed. They might only spend additional time on bugs with "interesting" summaries.

The ideal summary tells the reader what the bug is, what caused the bug, what part of the program it's from and what its worst consequence is. It runs from 8 to 15 words long. You might not fit all this information in 15 words, but you might fit in the most important parts.

Report Content (cont.)

Problem Summary:

We use the following syntax for writing the problem summary:

Symptom + Action + Operating Condition

Report Content (cont.)

Problem Description and Steps to Reproduce:

- First, describe the problem. What is the bug? Don't rely on the summary to do this -- some reports will print this field without the summary.
- Next, go through the steps that you use to recreate this bug. Start from a known place (e.g. boot the program) and then describe each step until you hit the bug.
- Describe the erroneous behavior and if necessary, explain what should have happened. (Why is this a bug? Be clear.)
- If you expect the reader to have any trouble reproducing the bug (special circumstances are required), be clear about them.

Bug Reporting Exercise 1

See Exercises

Report Content (cont.)

Comments:

In many of the best databases, there are free-form comments fields that will take comments from anyone on the project.

- You will get comments (especially questions) from the project manager, the programmer and tech support. Other groups in your company might also be allowed to enter their comments.
- This field is especially valuable for recording progress and issues with difficult or politically charged bugs.
- Be cautious about your wording. Just like e-mail and usenet postings, it is easy to read a joke or a remark as a flame. Never flame.

A Few More Fields

Reproducible?:

You may or may not have this on your form, but you should always provide this information.

- Never say it is reproducible unless you have recreated the bug. (Always try to recreate the bug before writing the report.)
- If you have tried and tried but you can not recreate the bug, say No. Then explain what steps you tried in your attempt to recreate it.
- If the bug appears sporadically and you do not yet know why, say “sometimes” and explain.

A Few More Fields (cont.)

Severity:

You will have to rate the bug's seriousness. Many companies use a three-level rating:

- 1 - Critical: This means fatal to the release (unacceptable to ship)
- 2 - Serious: It's a bad bug, but it doesn't, for example, cause data loss or a program crash.
- 3 - Minor: It's a bug, but it's not a big deal.

Your company's definitions may be a bit different from these. And you might have a five-level rating instead of three -- check with your manager.

Many companies sort their summary reports by severity, so you want to fill in this field thoughtfully.

A Few More Fields (cont.)

Frequency vs. Priority:

With some bug tracking systems, you also have to rate the bug frequency. **Frequency** is usually graded by assessing the following three characteristics:

- How easy is it for the user to encounter the bug
- How frequent would the user encounter the bug
- How often the buggy feature is used

Many companies use a three-level rating:

- 1 - Always
- 2 - Often
- 3 - Seldom

Priority rating is either automatically generated by the bug tracking system by assessing the Severity and the Frequency ratings or assigned only by the project manager. This is the *fix-priority* that everyone who is responsible for working on the bug will go by.

Priority--Severity vs. Frequency

	Grade	Value
Severity	1	1
	2	3
	3	5
Frequency	1	1
	2	3
	3	5

$$\text{Priority} = (\text{SeverityValue} + \text{FrequencyValue}) / 2$$

$$S1F1 = (1+1) / 2 = 1$$

$$S1F2 = (1+3) / 2 = 2$$

$$S2F1 = (3+1) / 2 = 2$$

$$S1F3 = (1+5) / 2 = 3$$

$$S3F1 = (5+1) / 2 = 3$$

$$S2F2 = (3+3) / 2 = 3$$

$$S2F3 = (3+5) / 2 = 4$$

$$S3F2 = (5+3) / 2 = 4$$

$$S3F3 = (5+5) / 2 = 5$$

Bug Reporting Exercise 1.1

See Exercises

A Few More Fields (cont.)

Keyword (Functional Area):

You may have to categorize the bug according to its functional area.

The tracking system should include a list of the possible keywords. Read the list and ask questions in order to learn what each category includes.

It is important to categorize these bugs consistently. They'll often be assigned out and summary-reported by category. Burying a bug in the wrong category can lead to its never getting fixed.

If you're creating the list of functional areas, keep it short, perhaps 20 areas.

A Few More Fields (cont.)

Resolution

The project manager has the privilege to assign most of the resolutions in this field. Common resolutions include:

- **New:** The newly filed bug needs to be reviewed by the project manager.
- **Proposed Feature Change:** The reporter proposed (perhaps) a better way to implement the feature.
- **To Be Reassigned:** The assigned person recommends the bug to be reassigned to someone else in the same group or in a different group.
- **To Be Fixed:** The bug is being worked on.
- **To Be Investigated:** The assigned person or group needs to do further investigating to support the decision making process.
- **Postpone:** Postpone working on this bug to a later time but it is desirable to fix the bug in this release.

A Few More Fields (cont.)

- **Need More Info:** The programmer needs more info from you. She has probably asked a question in the comment field.
- **Can Not Reproduce:** The programmer can't make the problem happen again. You have to add more details, reset the resolution to *To Be Fixed*, and notify the programmer.
- **Fixed:** The programmer says it's fixed. Now you do your regression testing.
- **Not a Problem:** The program works as it's supposed to.
- **Duplicate:** This is just a repeat of another bug report (XREF it on this report.)
- **Deferred:** This bug will be fixed in a later release, not now.
- **Withdrawn:** The tester who reported this bug is withdrawing the report.

Notes

This image shows a single sheet of white paper with horizontal blue or grey ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Testing Computer Software

Part 11

Analyzing Software Errors

Testing Computer Software, Chapter 5.

Why Analyze a Reproducible Bug?

Analyze bugs in order to:

- make your communication effective;
 - Make sure you are reporting what you think you are reporting.
 - Make sure that questionable side issues are thoroughly investigated.
 - Create accountability.
- support the making of business decisions;
- avoid wasting the time of the programming and management staff;
- find more bugs.

Analyzing a Reproducible Error

Start by making sure the error is reproducible.

- 1) Describe how to get the program into a known state. For example, starting the program takes it into a known (just booted) state.
Describe the state clearly, so that anyone familiar with the program will know what to do to get the program into that state.
- 2) Specify an exact series of steps that expose the problem.
- 3) Test your steps to make sure that you can reproduce the problem if you do exactly (and only) what it says in the bug report.

Analyzing a Reproducible Error (cont.)

Begin analyzing.

A clear, reproducible set of steps may be all you need to file the report. But you may be able to improve the report in four ways:

- 1) You might be able to *simplify* the report by **eliminating unnecessary or irrelevant steps**.
- 2) You might be able to *simplify* the report by **splitting it into two reports**.
- 3) You might be able to *strengthen* the report by **showing that it is more serious than it first appears**.
- 4) You might be able to *strengthen* the report by **showing that it is more general than it first appears**.

1. Eliminate Unnecessary Steps

Sometimes it is not immediately obvious what steps can be dropped from a long sequence of steps in a bug.

Look for critical steps -- Sometimes the first symptoms of an error are subtle.

Look carefully for any hint of an error as you take each step -- A few examples of potentially erroneous behavior:

- Error messages (you got a message 10 minutes ago. The program didn't fully recover from the error, and the problem you see now is caused by that poor recovery.)
- Processing delays
- Blinking screen or a flash
- Jumping cursor or multiple cursors
- Misaligned text or slightly distorted graphics
- Characters doubled or omitted
- *In-use* light on when the device is not in use

1. Eliminate Unnecessary Steps (cont.)

You made a list of all the steps that you took to show the error. You are now trying to shorten the list.

- If you've found what looks like a critical step, try to eliminate almost everything else from the bug report. Go directly from that step to the last one (or few) that shows the bug.
- In general, try taking out individual steps or small groups of steps. Can you show the bug with the others?

2. Split the Report in Two

When you see two related problems, you *might* report them together on the same report as long as you show that there are two of them.

For example, if in the previous Microsoft Paint exercise, you vary step 5 and get a different symptom, you might report 5a and 5b together.

5a Attempt to delete (Ctrl-X or Del) the selected area. RESULT -- nothing is deleted. OR

5b Expand the Paint window so that the entire image is visible. Attempt to delete the selected area. RESULT -- a different area is deleted.

But if this lengthens the report or makes it at all confusing, write two reports instead. If you would have to run two tests to verify that a bug has been fixed, it's fair to report the two symptoms that you have to test on two separate bug reports.

When you report related problems, it's a courtesy to cross-reference them. For example:

Related bug -- see Report # xxx

3. Show that it is More Serious

Look for follow-up errors: Keep using the program after you get this problem. Does anything else happen? Sometimes a modest-looking bug can lead to a system crash or corrupted data.

Look for nastier variants: Vary the conditions under which you got the bug. For example, try to get the bug while the program is doing a background save. Does that cause data loss or corruption along with this failure?

Check for this error in previous program

versions: In many projects, an old bug (from a previous shipping release of the program) might not be taken very seriously if there weren't lots of customer complaints. (If you know it's an old bug, check its complaint history.) But if the bug is new (especially in a maintenance release that is just getting rid of a few bugs) then the project manager might take it very seriously even if it's minor.

Look for nastier configurations: Sometimes a bug will show itself as more serious if you run the program with less memory, a higher resolution output, more graphics on a page, etc.

4. Show that it is More General

Look for alternative paths to the same problem:

For example, the Paint bug is described in terms of a zoom level of 200%. The program allows other settings, such as 400%. Try them. Then you can write a report that says there's a problem for any zoom level greater than 100%.

Look for configuration dependence: In the case of a zoom bug, you might suspect some tie to the video driver, the video card manufacturer or the video resolution. Check these -- try to reproduce the bug on a different machine with a different video card, in a different video resolution.

As a matter of general good practice, it pays to replicate every bug on a second machine.

Analyzing an Error Exercise 1

See Exercises

Analyzing an Error Exercise 1.1

See Exercises

Notes

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Testing Computer Software

Part 12

Reproducing Software Errors

Non-Reproducible Errors

Bugs don't just miraculously happen and then go away. A failure to replicate may or may not indicate sloppiness. It may instead reflect a failure dimension that you're not thinking about.

Nancy Leveson (Safeware) draws useful distinctions between errors, conditions, and failures.

- An **error** is a design flaw or a deviation from a desired or intended state.
- An error will not yield a failure without the **conditions** that trigger it. Example, if the program yields $2+2=5$ on the tenth time you use it, you will not see the error before or after the tenth use.
- The **failure** is the program's actual incorrect or missing behavior under the error-triggering conditions.

Non-Reproducible Errors

When we find a bug, we are looking at a *failure*, a set of symptoms of an underlying *error*. We hypothesize the cause, then we try to recreate the *conditions* that make the error visible. *Sometimes, these conditions are subtle.*

- Always report non-reproducible errors, but describe your steps and observations precisely. Programmers will often figure them out.
- Every attempt to replicate involves a theory.
- The analyzing process: **Hypothesize, test, observe and evaluate.**

Why is this Bug Hard to Reproduce?

- Memory dependent
- Memory corruption
 - Run-time errors
 - Predicated on corrupted data
- Configuration dependent
 - Software
 - Hardware
- Timing related
- Initialization
- Data flow dependent
- Control flow dependent
- Error condition dependent
- Multi-threading dependent
- Special cases
 - Algorithm
 - Dates

Making Errors Reproducible

- Write down everything you remember about what you did the first time.
- Note which things you are sure of and which are good guesses.
- Note what else you did before starting on the series of steps that led to this bug.
- Review similar problem reports you've come across before.
- Use tools such as capture/replay program, debugger, debug-logger, videotape, or monitoring utilities that can help you identify things that you did before running into the bug.
- Talk to the programmer and/or read the code.

Testing Computer Software

Part 15

Introduction to Black Box, Glass Box, and Gray Box Testing

Black Gray White

	Who does it?	Tool Use?	What is it?
Black			
Gray			
Glass Clear White			

Black-box Testing

Testing based on expected behavior at the external interface

Dick Bender

Focus on input, outputs, and an “externally derived” theory of operation

Cem Kaner

Testing based on external attributes and behaviors

Doug Hoffman

What is Black Box Testing?

Black box tests execute the running program, without reference to the underlying code. This is testing from the customer's view rather than from the programmer's.

Testing Computer Software, pages 41-57, 210-213

Common Types of Black Box Testing

- Acceptance (into-testing) Test
- Feature-Level Test
- System-Level Test
- Regression Test
- Configuration & Compatibility Test
- Documentation and Online Help Test
- Utilities/Tool Kits and Collateral Test
- Install/Uninstall Test
- Integrity & Release Testing
- Import / Export
- User Interface Test
- Usability Testing
- Performance Test
- Benchmark Test
- Acceptance by the Customer
- Beta Testing
- Staked Application Test
- Initial Stability Assessment

Testing Computer Software, pages 41-57, 210-213
and “Test Types & Their Place in the Software
Development Cycle” in the Appendix.

Glass-Box Testing

Structural testing based on physical logic flow and data flow in the code

Dick Bender

Focus on code, data structures and architectures

Cem Kaner

Tests design based on the knowledge of the code to exercising the code

Doug Hoffman

Glass Box Testing

In glass box testing, the tester (usually the programmer) uses her knowledge of the source code to create test cases.

Testing Computer Software, pages 41-49

Fundamental Benefits of Glass Box Testing

- Highly efficient
- Focused concentration on specific modules
- Isolate genuine boundaries
- Isolate all branches
- Isolate transitions between algorithms
- Finer grain measurement (e.g. coverage)
- Able to assess code coverage

Important Methods of Glass Box Testing

- Compilers
- Debuggers
- Assertion checking
- Code path tracing
- Coverage analyzers
- Basic unit testing, using stubs & drivers, especially for driving modules through boundaries and error conditions.
- Data flow testing (based on data flow tracing)
- Inspections and code walk-through (these are not “testing” because you don’t *execute* the code, but they are important ways to find bugs).

Some Problems that Glass Box Testing Often Miss

- Timing-related bugs
- Side effects of interrupts
- Unexpected error conditions
- Special data conditions
- Interaction with background tasks
- Invalid onscreen information
- User interface inconsistency
- User interface everything else
- Failure to comply with contract or regulation
- Configuration/compatibility failures
- Volume, load, hardware faults

Testing Computer Software, page 211

Gray-Box Testing

Using inferred or incomplete structural or design information to expand or focus black-box testing

Dick Bender

Testing involving inputs, outputs, but test design is educated by information about the code and the program operation of a kind that would normally be out of scope of the view of the tester

Cem Kaner

Gray-Box Testing

Test design based on the knowledge of algorithms, internal states, architecture, and other high-level descriptions or program behaviors

Doug Hoffman

Gray-box testing consists of methods and tools derived from the knowledge of the application internals and the environment with which it interacts, that can be applied in black-box testing to enhance testing productivity, bug finding and bug analyzing efficiency

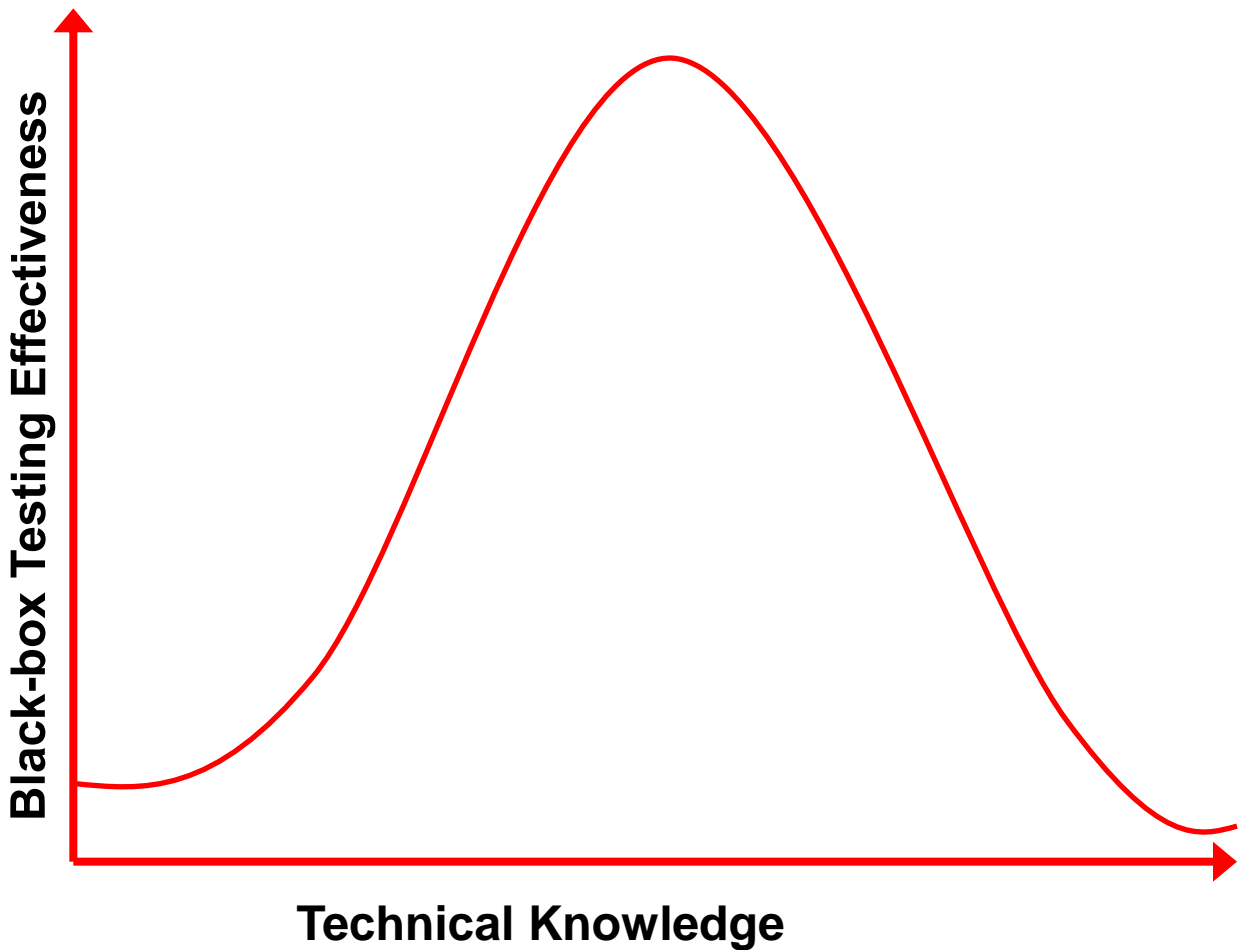
Hung Nguyen

Gray Box Testing

Gray box methods make internal information about the program visible to a black box tester.

- Complexity metrics or other code-derived metrics
- Standards compliance
- Event logs
- Memory meters
- Memory run-time errors detector
- Resource meters

Testing Effectiveness



Is There a Bug?

“Zen Testing”, Dick Bender (LAWST IX)

...looking for something that's not there.

Gray Box Testing

The focus is on gray-box testing

- Black-box testing focuses on user (end feature) context
- White-box testing focuses on developer (code) context
- *Gray-box testing focuses on high-level design, environment, and interoperability context*

Starting a List of Ad Hoc Tests

Do Detective work.

What is going on here?

What COULD a user possibly do?

What is going on at the system level?

How are the systems connected?

Starting a List of Ad Hoc Tests

Test Case development:

1) Analyze Feature:

what is goal of dialog box/ HTML page/frame/
functional area/ etc. (for expected results)

2) What are objects

3) what are states and properties of Objects (valid/
invalid input)

4) what is going on I cannot see. (API calls,
cookies set. registry keys read, clock read, page
size read, etc.)

5) what is error handling for this area

6) what would/ could/ might a user do?

ADD- intended use

Notes

[illegible]

Testing Computer Software

Part 17

Test Planning Materials

Test Planning Materials

- Outlines & Lists
- Tables
- Matrices

Test Planning Materials

Outlines and Lists

An **outline** organizes information into a hierarchy of lists and sublists.

Outline Example

Microsoft Notepad menu list in an outline format

File

- New
- Open...
- Save
- Save As...
- Page Setup...
- Print...
- Exit

Edit

- Undo
- Cut
- Copy
- Paste
- Delete
- Select All
- Time/Date
- Word Wrap

Search

- Find...
- Find Next

Help

- Help Topics
- About Notepad

Test Planning Materials

Tables

A **table** organizes information in two dimensions showing relationships between variables.

Table Example 1

The program's specification:

- The program is designed to add two numbers, which you will enter
- Each number should be one or two digits
- The program will echo your entries, then print the sum. Press `Enter` after each number
- To start the program, type `ADDER`

Testing Computer Software, Chapter 1

Table Example 1 (cont.)

<i>What you do</i>	<i>What happens</i>
Type ADDER and press the Enter key	The screen blanks. You see a question mark at the top of the screen.
Press 2	A 2 appears after the question mark
Press Enter	A question mark appears on the next line
Press 3	3 appears after the second question mark
Press Enter	A 5 appears on the third line. A couple of lines below it is a question mark.

```
C:>ADDER
```

```
? 2
```

```
? 3
```

```
5
```

```
? _
```

Table Example 1 (cont.)

A Tabular Format for Listing Equivalence Classes

Variable	Valid Case Equivalence Classes	Invalid Case Equivalence Classes	Boundaries and Special Cases	Notes
First number	-99 to 99	> 99 < -99 non-number expressions	99, 100 -99, -100 / ; null entry 0	max bounds min bounds See ASCII bounds in the next section. That yield interesting (invalid) results. 0 is always interesting. Are there other sources of data for this variable? Ways to feed it bad data?
Second number	same as first	same as first	same	same
Sum	-198 to 198			

Table Example 2

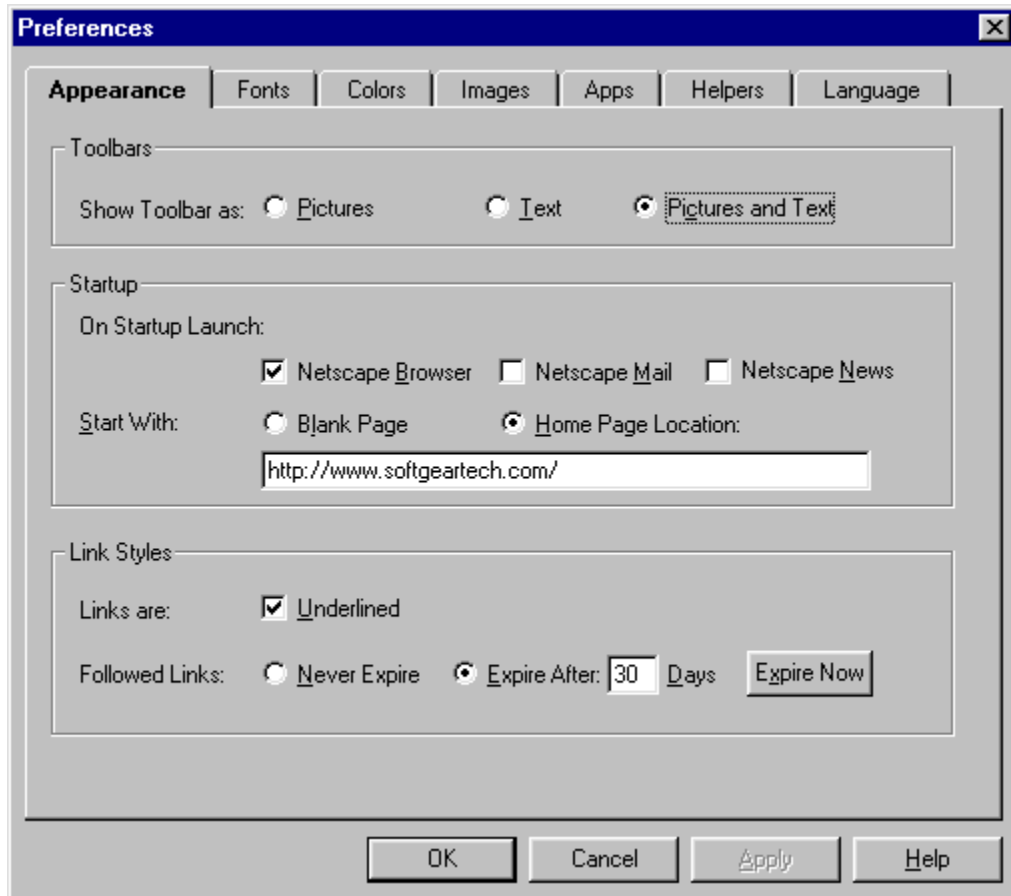


Table Example 2 (cont.)

A Tabular Format for Input Combinations

Test #	Toolbars PTB	On Startup, Browser Y/N	On Startup, Mail Y/N	On Startup, News Y/N	Start With BVE	Links DU	Followed NE
1	P	Y	Y	Y	B	D	N
2	P	Y	N	N	V	D	E
3	P	N	Y	Y	E	U	E
4	P	N	N	N	B	U	N
5	T	Y	Y	N	V	U	N
6	T	Y	N	Y	E	U	E
7	T	N	Y	N	B	D	E
8	T	N	N	Y	V	D	N
9	B	Y	Y	Y	E	D	N
10	B	Y	N	N	B	U	E
11	B	N	Y	Y	V	D	E
12	B	N	N	N	E	U	N

Test Planning Materials

Matrices

A **matrix** is a special type of table used for data collection.

Matrix Example

Input Dialog Test Matrix																									
Additional Instructions:																									
	Type in each of the entry fields, one at the time																								
	Nothing																								
	Valid value																								
	At LB of value																								
	At UB of value																								
	At LB of value - 1																								
	At UB of value + 1																								
	Outside of LB of value																								
	Outside of UB of value																								
	0																								
	Negative																								
	At LB number of digits or chars																								
	At UB number of digits or chars																								
	Empty field (clear the default value)																								
	Outside of UB number of digits or chars																								
	Non-digits																								
	Space																								
	Non-printing char (e.g., Ctrl+char)																								
	DOS filename reserved chars (e.g., "\ * . :")																								
	Upper ASCII (128-254)																								
	Upper case chars																								
	Lower case chars																								
	Modifiers (e.g., Ctrl, Alt, Shift-Ctrl, etc.)																								
	Function key (F2, F3, F4, etc.)																								

Table vs. Matrix Example 1

This is a simple Find dialog box. It takes three inputs:

- Find what: a text string (“UPPERCASE”, “lowercase” and “MixedCase”)
- Match whole word only: ON or OFF
- Match case: ON or OFF

Simplify this by considering only three values for the text string, “UPPERCASE” and “lowercase” and “MixedCase”. Combine the inputs.

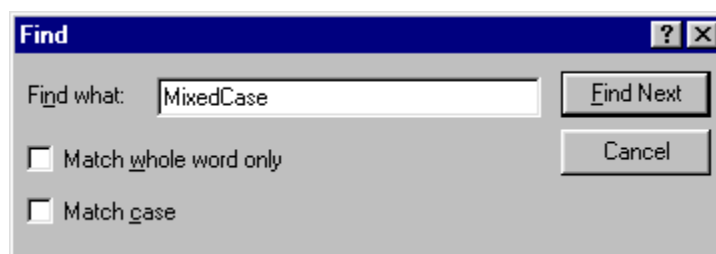
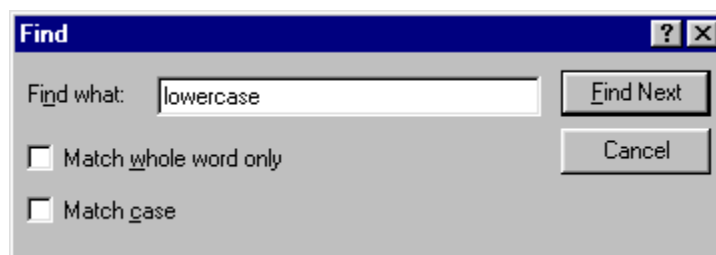
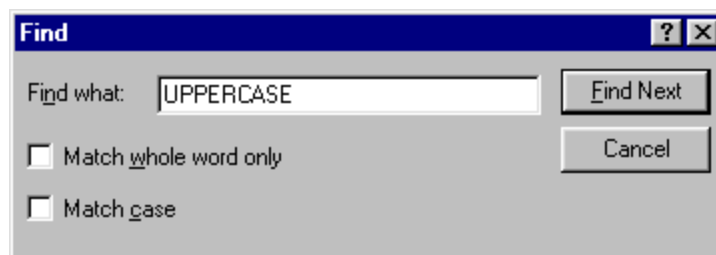
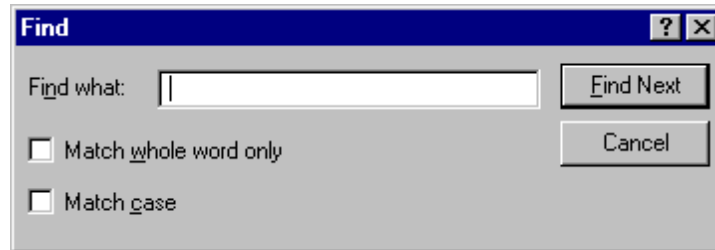


Table vs. Matrix Example 2

	PROPERTY											
CONTROL	1	2	3	4	5	6	7	8	9	10	11	12
Match case	OFF	OFF	ON	ON	OFF	OFF	ON	ON	OFF	OFF	ON	ON
Match whole	OFF	ON	OFF	ON	OFF	ON	OFF	ON	OFF	ON	OFF	ON
Find What	"UPPERCASE"	"UPPERCASE"	"UPPERCASE"	"UPPERCASE"	"lowercase"	"lowercase"	"lowercase"	"lowercase"	"MixedCase"	"MixedCase"	"MixedCase"	"MixedCase"

- Use a table to list all combinations of the three variables.
- There are 12 combinations altogether.

Table vs. Matrix Example 3

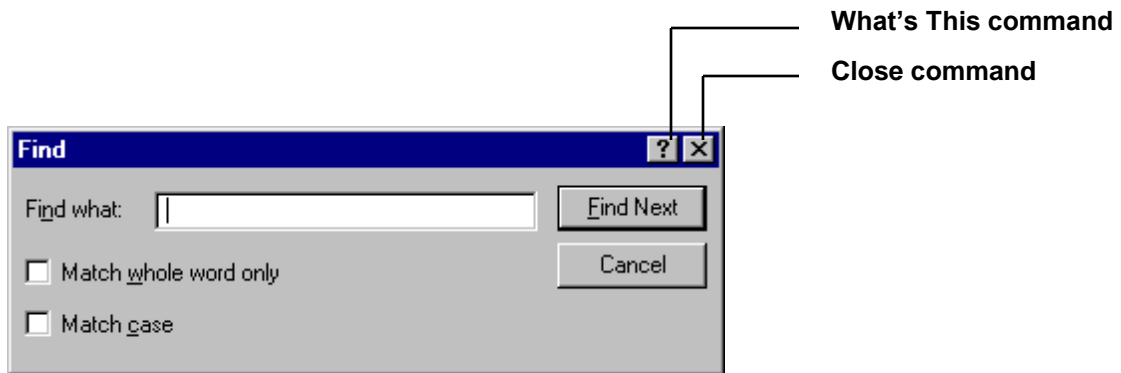


	PROPERTY											
CONTROL	1	2	3	4	5	6	7	8	9	10	11	12
Match case	OFF	OFF	ON	ON	OFF	OFF	ON	ON	OFF	OFF	ON	ON
Match whole	OFF	ON	OFF	ON	OFF	ON	OFF	ON	OFF	ON	OFF	ON
Find What	"UPPERCASE"	"UPPERCASE"	"UPPERCASE"	"UPPERCASE"	"lowercase"	"lowercase"	"lowercase"	"lowercase"	"MixedCase"	"MixedCase"	"MixedCase"	"MixedCase"

To test the Find functionality:

- Create a text file with the following strings in it: "UPPERCASE", "lowercase" and "MixedCase";
- Set the three inputs according to each of the 12 combinations shown in the table, one at the time;
- Choose the Find or Find Next button and validate the results.

Table vs. Matrix Example 4



- For each input combination, verify the behavior of each of the four command buttons: Find Next, Cancel, What's This and Close.
- Create a matrix for this set of tasks.

Table vs. Matrix Example 5

	PROPERTY											
CONTROL	1	2	3	4	5	6	7	8	9	10	11	12
Match case	OFF	OFF	ON	ON	OFF	OFF	ON	ON	OFF	OFF	ON	ON
Match whole	OFF	ON	OFF	ON	OFF	ON	OFF	ON	OFF	ON	OFF	ON
Find What	"UPPERCASE"	"UPPERCASE"	"UPPERCASE"	"UPPERCASE"	"lowercase"	"lowercase"	"lowercase"	"lowercase"	"MixedCase"	"MixedCase"	"MixedCase"	"MixedCase"
[Find Next]												
Cancel												
[?]												
[X]												

To test the behavior of every command buttons:

- Create a text file with the following strings in it: "UPPERCASE", "lowercase" and "MixedCase";
- Set the three inputs according to each of the 12 combinations, one at the time;
- For each input combination, choose each command button, one at the time and verify the behavior;
- When the test is complete, put an X in the matrix cell to indicate so.

Table vs. Matrix Example 6

	PROPERTY											
CONTROL	1	2	3	4	5	6	7	8	9	10	11	12
Match case	OFF	OFF	ON	ON	OFF	OFF	ON	ON	OFF	OFF	ON	ON
Match whole	OFF	ON	OFF	ON	OFF	ON	OFF	ON	OFF	ON	OFF	ON
Find What	"UPPERCASE"	"UPPERCASE"	"UPPERCASE"	"UPPERCASE"	"lowercase"	"lowercase"	"lowercase"	"lowercase"	"MixedCase"	"MixedCase"	"MixedCase"	"MixedCase"
[Find Next]	X	X	X	X	X	X	X	X	X	X	X	X
Cancel	X	X	X	X	X	X	X	X	X	X	X	X
[?]	X	X	X	X	X	X	X	X	X	X	X	X
[X]	X	X	X	X	X	X	X	X	X	X	X	X

- The matrix above indicates that all 48 tests have been completed.

Table/Matrix Exercise

See Exercises

Notes

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Testing Computer Software

Part 18

Black Box Test Case Design

Good Test Case Design

An excellent test case satisfies the following criteria:

- Reasonable probability of catching an error
- Neither too simple nor too complex
- Not redundant with other tests
(think of equivalence classes)
- Best of its breed
(think of boundary conditions)
- Makes program failures obvious

Neither Too Simple Nor Too Complex

- Simple test cases (e.g., Functional Simple Acceptance Test) are only good for validation.
- Complex test cases (lack of design behind them) can be time-consuming to reproduce and analyze.
- Transition from simple cases to complex cases.
- Automation tools can bias your development toward overly simple or complex tests.

Testing Computer Software, pages 125, 241, 289, 433

Make Program Failures Obvious

Important failures have been missed because they weren't noticed after they were found.

Some strategies:

- 1) Show expected results.
- 2) Keep the output simple and well formatted.
- 3) If possible,
 - Automate comparison against known good output.
 - Only print failures.
 - Log failures to a separate file.

Testing Computer Software, pages 125, 160, 161-164

Examples from Printer Testing

Normal. *Italics*. **Bold**. ***Bold Italics***. Normal. *Italics*. **Bold**. ***Bold Italics***.
Normal. *Italics*. **Bold**. ***Bold Italics***. Normal. *Italics*. **Bold**. ***Bold Italics***.
Normal. *Italics*. **Bold**. ***Bold Italics***. Normal. *Italics*. **Bold**. ***Bold Italics***.
Normal. *Italics*. **Bold**. ***Bold Italics***. Normal. *Italics*. **Bold**. ***Bold Italics***.

Frame graphical output

Put a one-pixel wide frame around the graphic and/or around the edge of the printable page. Off-by-one-pixel errors will often erase one of these borders.

For color output, create color sample charts.

Print a color wheel or some other standard progression (light to dark or rainbow progression) that is visually obvious. Name the colors. Have a comparison page handy.

Draw regular line-art that can show distortions

Draw circles and perfect squares. Look for bowing, stretching, dropout. Use some dashed lines, to see if dashes are drawn unequally long.

Test Case Design Methodologies

Equivalence Partitioning

Boundary Value Analysis

State Transition Testing

Cause-Effect Graphing

Syntax Testing

Statement Testing

Branch/Decision Testing

Data Flow Testing

Branch Condition Testing

Branch Condition Combination Testing

Modified Condition Decision Testing

LCSAJ Testing ((Linear Code Sequence and Jump)

Random Testing

From British Computer Society list at www.bcs.com

Notes

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Test Case Design

Part 18.1

Equivalence Class Partitioning & Boundary Analysis

Equivalence Class & Boundary Analysis

Equivalent Class Partitioning and Boundary Condition Analysis

- Equivalence Class—Two tests belong to the same equivalence class if the expected result of each is the same. Executing multiple test cases of the same equivalence class is by definition, redundant testing.
- Boundaries—Mark the point or zone of transition from one equivalence class to another. The program is more susceptible to errors at the boundary conditions. Therefore, these are powerful sets of test cases within the equivalent class to use.
- Generally, each class is partitioned by the boundary values. Nevertheless, not all equivalence classes have boundaries. For example, the browser equivalence classes consist of Netscape Navigator class and Microsoft Internet Explorer class.

It is important to recognize that two tests are equivalent only with respect to a specific risk.

For example, is $64+63$ equivalent to $64+64$?

***Read Testing Computer Software, 2nd Edition, Wiley
for more information***

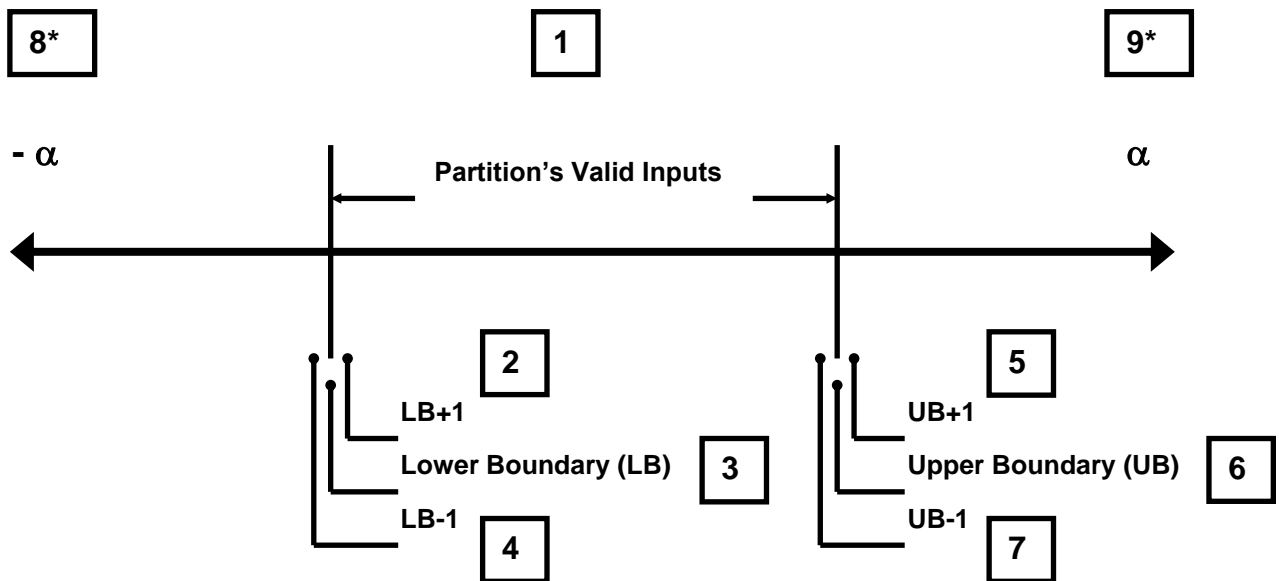
Equivalence Class & Boundary Analysis (cont.)

Two tests are equivalent if you expect the same result from each. Here are some examples:

- Ranges of numbers (such as all the numbers between 10 and 99)
- Membership in groups (dates, time, country names, etc.)
- Invalid inputs
- Equivalent output events (variation of inputs that produces the same outputs)
- Equivalent operating environments
- The number of times you've done something
- The number of records in the database (or how many other equivalent objects)
- Equivalent sums or other arithmetic results
- Equivalent number of items entered (such as the number of characters entered into a field)
- Equivalent space (on a page or on a screen) required
- Equivalent amounts of memory, disk space, or other resources available to the program

Equivalence Class & Boundary Analysis

* Smallest/Largest Possible Values Allowed via UI



- For each equivalence class partition, we'll have at most, 9 test cases to execute.
- It is essential to understand that each identified equivalence class represents a specific risk that it may pose.

Equivalence Class & Boundary Analysis

General Steps

- Identify the classes
- Identify the boundaries
- Identify the expected output(s) for valid input(s)
- Identify the expected error-handling (EH) for invalid inputs
- Generate a table of test cases (maximum, 9 test cases for each partition of each class).

Equivalence Class & Boundary Analysis

Welcome to TRACKGEAR - Microsoft Internet Explorer

File Edit View >> Links >> Address Go <>

TRACKGEAR

Submit

New

Find

EasyFind

QuickFind

FormFind

CustomFind

QuickFind

PROJECT: TG2

Find bug # 95

Find All bugs with Status Open

Done Local intranet

Valid Values: 1 to 9999
or 1 to 4-digit value

Equivalence Class & Boundary Analysis

Test Case	Input	Output
Value Class Partition		
1	Any Valid Input	Functional Result
2	2	Functional Result
3	1	Functional Result
4	0	Error Handling
5	10000	Error Handling
6	9999	Functional Result
7	9998	Functional Result
8	-99999...	Error Handling
9	99999...	Error Handling
Number of Character Class Partition		
1	Any Valid Input	Functional Result
2	2	Functional Result
3	1	Functional Result
4	NULL	Error Handling
5	5	Error Handling
6	4	Functional Result
7	3	Functional Result
8	N/A	N/A
9	99999...	Error Handling

Boundaries May Not Be Immediately Obvious

Boundaries in the encoding of keystrokes

<u>Character</u>	<u>ASCII Code</u>
/	47
0	48
1	49
2	50
3	51
4	52
5	53
6	54
7	55
8	56
9	57
:	58
A	65
b	98

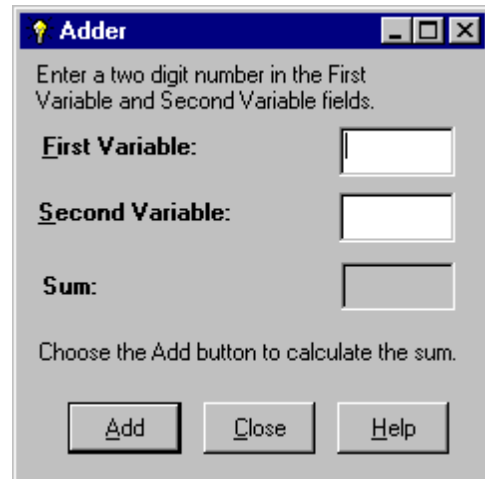
Character Groupings

Low ASCII	These are the Ctrl keys. Interesting ones include 000-null, 007-beep, 008-BS, 009-Tab, 010-LF, 011-VT/home, 012-FF, 013CR, 026-EOF (end of file, very nasty sometimes), 027-ESC	
Non-alphanumeric, standard, printing ASCII characters. We often lump these together even though they're in four distinct groups.	Low non-alphanumeric (ASCII codes 32-47)	space ! " # \$ % & ' () * + , - . /
	Intermediate non-alphanumeric (ASCII 58-64)	: ; < = > ? @
	More intermediates (ASCII 91-96)	[\] ^ _ `
	Top of standard ASCII (ASCII 123-126)	{ } ~
Digits	(ASCII 48-57)	0 1 2 3 4 5 6 7 8 9
Upper and lower case alpha	(ASCII 65-90)	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
	(ASCII 97-122)	a b c d e f g h i j k l m n o p q r s t u v w x y z
Upper ASCII	(ASCII 128-254)	These subdivide further depending on the character set, the user's language, and the application.
Modifier keys	These keys include (depending on the keyboard) Alt, Right-Alt, Shift, Control, Command, Option, Left-Amiga, Right-Amiga, Open-Apple, etc. They generally have no effect when pressed alone, but when pressed in conjunction with some other key, they create a	
	It often pays to test all the "interesting" standard values, plus a sample of others.	
	It is often best to assign a separate chart column to each modifier key, i.e. one column for Ctrl, one for Shift, etc.	
Function keys	Test them alone and in combination with the modifier keys.	
Cursor keys	Test them alone and in combination with the modifier keys. It's common for every modifier key to have a different effect on cursor keys.	
Numeric keypad keys	These are not necessarily equivalent to number keys elsewhere on the keyboard.	
European keyboards	The left and right Alt keys often have different effects on non-English keyboards. Also, these keyboards provide dead keys -- you press a dead key to specify an accent, then press a letter key and (if it's a valid character) the computer displays that let	

Testing Computer Software pages 231, 240-241.

Equivalence Class & Boundary Analysis (cont.)

The Adder GUI
application example:



Specification--Input Value Validation

- There are $199 \times 199 = 39,601$ test cases
 - definitely valid: 0 to 99
 - might be valid: -99 to -1
- There are infinitely many invalid test cases
 - 100 and above
 - -100 and below
 - anything non-numeric
- Look for boundary conditions

Exploratory--Input Value Validation: Integer Boundary Condition Test Cases:

- For each variable (First Variable, Second Variable and Sum Result): 32,767; $32,767 + 1$; -32,768; $-32,768 + (-1)$

Equivalence Class & Boundary Analysis (cont.)

- Other Equivalence Classes and Special Cases
 - Negative values
 - Number of digit or characters
 - 0
 - Non-printable characters
 - Upper ASCII (128-254)
 - O/S reserved characters
 - Space
 - Nothing
 - etc.
- See Input Dialog Test Matrix

Input Dialog Test Matrix

Input Dialog Test Matrix																									
Additional Instructions:																									
	Type in each of the entry fields, one at the time																								
	Nothing																								
	Valid value																								
	At LB of value																								
	At UB of value																								
	At LB of value - 1																								
	At UB of value + 1																								
	Outside of LB of value																								
	Outside of UB of value																								
	0																								
	Negative																								
	At LB number of digits or chars																								
	At UB number of digits or chars																								
	Empty field (clear the default value)																								
	Outside of UB number of digits or chars																								
	Non-digits																								
	Space																								
	Non-printing char (e.g., Ctrl+char)																								
	DOS filename reserved chars (e.g., "\ * . :")																								
	Upper ASCII (128-254)																								
	Upper case chars																								
	Lower case chars																								
	Modifiers (e.g., Ctrl, Alt, Shift-Ctrl, etc.)																								
	Function key (F2, F3, F4, etc.)																								

Brainstorm--Equivalence Class

See Exercises

Notes

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Test Case Design

Part 18.2

State Transitioning Test Cases

State Transitioning

State Transitioning

- Involves an analysis of the relationship among the states and the events or actions that cause the transitions from one state to another.

General Steps

- Identify all supported states
- For each test case define
 - **The starting state**
 - **The input events that cause the transitions**
 - **The output results of each transition**
 - **The end state**
- Tests should cover both positive and negative cases.

State Transitioning

Navigation Command

Current View Mode

Report: 1 of 44

Navigation Command: View, Edit View, Full View

Current View Mode: View, Edit View, Full View

Report #:	201	Reporter:	johnf in QA	Created:
Project:	TG2	Build:	43-a.12.13.99	Module:
Status:	Open	Resolution:	To be Fixed	Modified:
Config ID:	Unassigned	Attachment...		
Error Type:	UI	Keyword:	Unassigned	Reproducible:
Severity:	1-High	Frequency:	1-Every Time	Priority:

Summary: Deleted data can be edited. Edit Profile can be accessed with no profiles available.

Steps (READ-ONLY):

```
http://jupiter/john
1) Login as admin
```

Notes & Comments (READ-ONLY):

```
The old deleted profile info appear
except the Miscellaneous field data
Old data should not reappear and u
```

State Transitioning

Navigation Command

Current View Mode

TRACKGEAR

Submit

New

Find

EasyFind

QuickFind

FormFind

CustomFind

Metrics

Distribution

Trend

Setup

Password

Preferences

Configs

Logout

Help

View

Edit View

Full View

NOTES & COMMENTS:

Notice that the Resolutions are displayed split, above and inside the Unused Resolution area. [johnf 3/14/00 3:55:44 PM] See also # 678. [joz 3:55:26 PM] Not fixed, now, the Resolution are displayed in the right side of the Unused Resolution Area [ioanai 3/20/00 4:44:24 PM] For Net displayed above the Unused Resolution area [johnf 3/24/00 4:56:38 PM] still

Report Number: 726

Project: TG2	Severity: Medium	Reported By: johnf in QA	Created: 3/15/00 1:42:13
Status: Open	Frequency: Every Time	Fixed By: johnsoni in Developer	Fixed: 3/23/00 6:39:12
Resolution: To be Fixed	Priority: Low	Regressed By: johnf in QA	Regressed: 3/24/00 4:57:06
Config ID: PC-NT4.0	Stopper: GMC	Assigned To: johnsoni in Developer	Last Modified: 3/24/00 4:57:14
Build Reported: 03.15.00	Module: Preferences	Error Type: UI	Keyword: IE- 5.01
Build to Regress: 03.24.00	Regressed Build: 03.24.00	Reproducible: Yes	

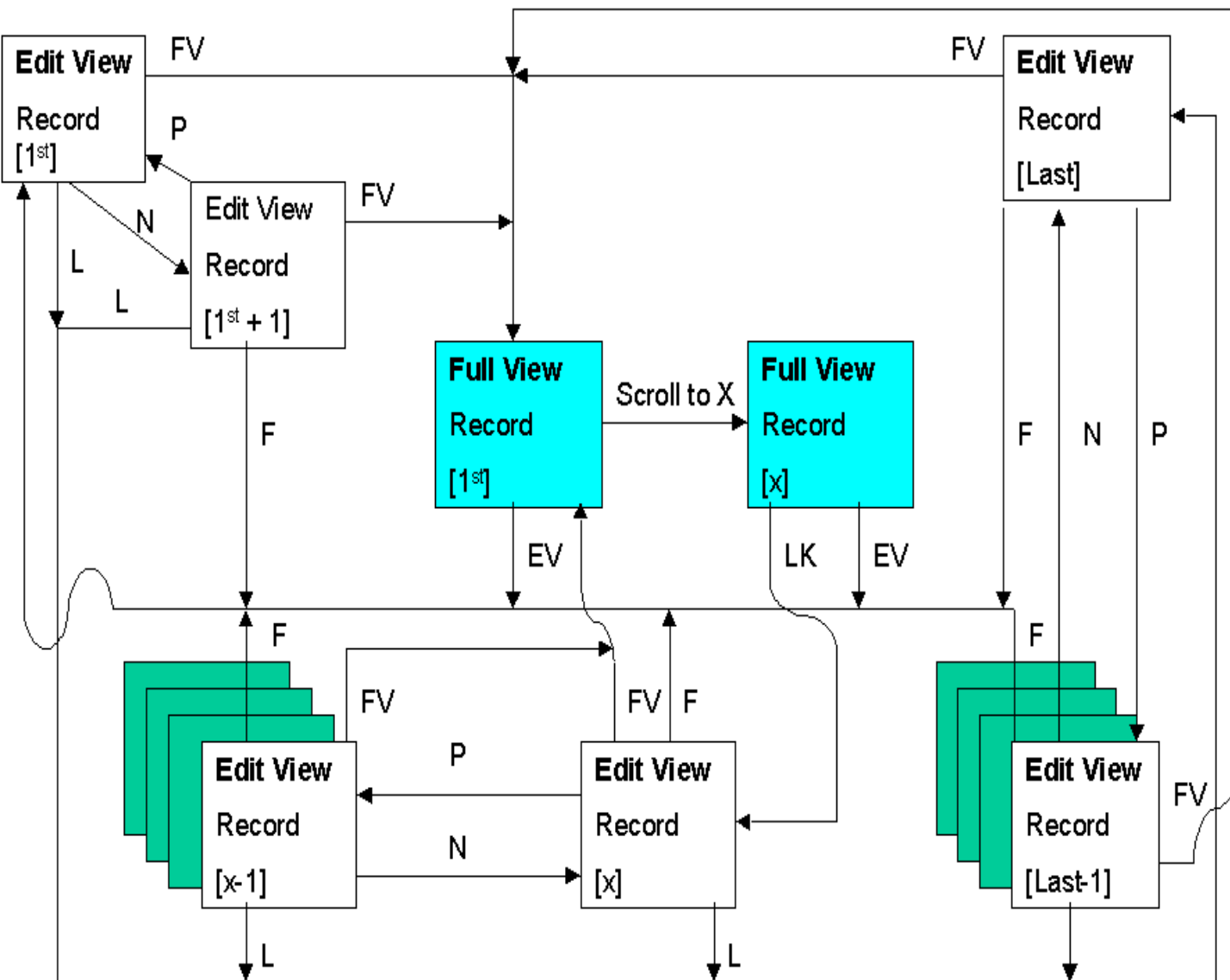
Attachment:

Summary

http://aquarius/bts/bin/treeform.asp?templateid=1&Search=Openme&ProjectView=TG2&reportno=

Local intranet

State Transitioning



State Transitioning

Code	VIEW MODE										NAVIGATION COMMAND					
0	Edit View-Record [1st]	Edit View displaying the 1st record									F		First			
1	Edit View-Record [1st+1]	Edit View displaying the 2nd record									P		Previous			
2	Edit View-Record [x]	Edit View displaying record [x]									N		Next			
3	Edit View-Record [x-1]	Edit View displaying record [x-1]									L		Last			
4	Edit View-Record [Last]	Edit View displaying the last record									FV		Full View			
5	Edit View-Record [Last-1]	Edit View displaying the next to last record														
6	Full View-Record [1st]	Full View displaying the 1st record									EV		Edit View			
7	Full View-Record [x]	Full View displaying record [x]									LK		Record ID Link			
	Test Case No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	Start View Mode	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
	Navigation Command (Input)	N	L	FV	F	P	L	FV	F	P	L	FV	F	N	L	FV
	End View Mode	1	4	6	0	0	4	6	0	3	4	6	0	2	4	6
	Test Case No.	16	17	18	19	20	21	22	23	24	25					
	Start View Mode	4	4	4	5	5	5	5	6	7	7					
	Navigation Command (Input)	F	P	FV	F	N	L	FV	EV	EV	LK					
	End View Mode	0	5	6	0	4	4	6	0	0	2					

Test Case Design

Part 18.3

Standardized Test Cases Using Test Matrices

Test Case Design

We've already talked about boundary analysis. This is one of the critical ways to analyze single variables.

Here we look at patterns of tests on single variables or on single operations.

In the next section, we'll look at relationships among variables.

Test Matrices

Many tests deal with essentially the same problem, so they follow essentially the same pattern.

- For example, for most input fields, you'll do a series of the same tests, checking how the field handles boundaries, unexpected characters, function keys, etc.
- As another example, for most files, you'll run essentially the same tests on file handling.

The matrix is a concise way of showing the repeating tests.

- Put the objects that you are testing on the rows.
- Show the tests on the columns.
- Check off the completed tests in the cells.

The following are examples:

Test Matrix 1

Input Dialog Test Matrix																									
Additional Instructions:																									
	Type in each of the entry fields, one at the time																								
	Nothing																								
	Valid value																								
	At LB of value																								
	At UB of value																								
	At LB of value - 1																								
	At UB of value + 1																								
	Outside of LB of value																								
	Outside of UB of value																								
	0																								
	Negative																								
	At LB number of digits or chars																								
	At UB number of digits or chars																								
	Empty field (clear the default value)																								
	Outside of UB number of digits or chars																								
	Non-digits																								
	Space																								
	Non-printing char (e.g., Ctrl+char)																								
	DOS filename reserved chars (e.g., "\ * . :")																								
	Upper ASCII (128-254)																								
	Upper case chars																								
	Lower case chars																								
	Modifiers (e.g., Ctrl, Alt, Shift-Ctrl, etc.)																								
	Function key (F2, F3, F4, etc.)																								

Test Matrix 2

File I/O Matrix -- Error Handling														
Additional Instructions:														
	FILE OPERATION:													
	- full local disk													
	- almost full local disk													
	- write protected local disk													
	- damaged (I/O error) local disk													
	- remove local disk from drive after opening file													
	- timeout waiting for local disk to come back online													
	- keyboard and mouse I/O during save to local disk													
	- other interrupt during save to local drive													
	- power out during save to local drive													
	- full network disk													
	- almost full network disk													
	- write protected network disk													
	- damaged (I/O error) network disk													
	- remove network disk after opening file													
	- timeout waiting for network disk													
	- keyboard / mouse I/O during save to network disk													
	- other interrupt during save to network drive													
	- power out during save to local drive													
	- local power out during save to network													
	- network power during save to network													

Test Matrix 3

File I/O Matrix

Additional Instructions:

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Standardized Test Cases Exercise

See Exercises

Notes

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Test Case Design

Part 18.4

Some Simple Approaches for Testing Data and Functionality Relationships

Testing Data Field Relationships

Equivalence class and boundary analysis look at each variable (field) in isolation.

This is important, but we also have to look at the relationship between different variables.

Cause-Effect Graphing is one approach to this, but it can be complicated.

Here is a simpler approach that is still quite valuable:

Tabular Format for Data Relationships

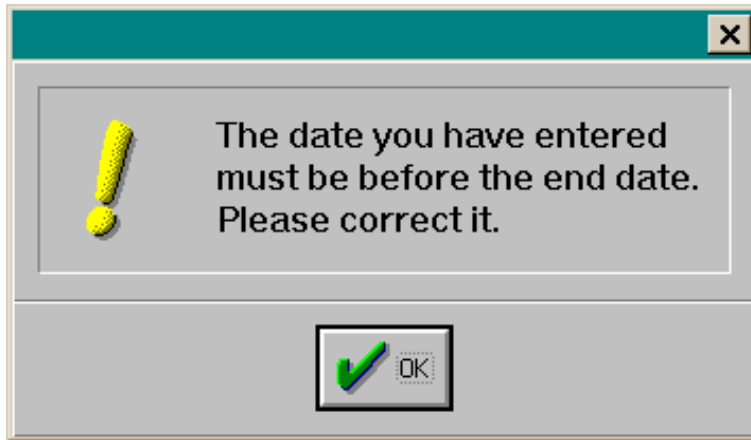
The screenshot shows the TSTimer software window. The menu bar includes File, Edit, Slips, Names, Search, Special, and Help. The main window is divided into several sections. At the top, there's a 'New slip' section with 'All' and 'Find' buttons. Below that, there's a section for 'Time Slip' and 'Expense Slip' with a 'Value' field showing '\$4,475.30'. The main data entry area includes fields for 'Consultant' (John C.), 'Client' (softGear Tech), and 'Activity' (Research), each with a dropdown arrow and a numeric field (1, 1, and 13 respectively). Below these is a 'Description' text area containing 'Researched the Win 95 documentation'. There's a 'Reference' section with two empty text boxes. The 'Start Date' is 5/4/96 and 'End Date' is 5/1/96. 'Time estimated' is 0:00:00 and 'Time spent' is 1:41:11. There are 'on' and 'off' buttons. The 'Billing Status' is 'Billable' and 'Repeat' is unchecked. 'Add to Flat Fee' is unchecked. The 'Rate' is 'Client' with a value of 1 and a rate of 2685.18 (hourly). The 'Override' checkbox is checked. At the bottom, it says 'Active slips 0'.

Field	Value
Consultant	John C.
Client	softGear Tech
Activity	Research
Description	Researched the Win 95 documentation
Start Date	5/4/96
End Date	5/1/96
Time estimated	0:00:00
Time spent	1:41:11
Billing Status	Billable
Rate	Client
Rate Value	1
Rate Amount	2685.18 (hourly)

Look at this record, from the Timeslips Deluxe time and billing database. In this dialog box, click the arrow next to the Consultant field to edit the Consultant record (name, billing info, etc.) or enter a new one.

If you edit here, will the changes *carry over* to every other display of this Consultant record? Also, note that the End Date for this task is *before* the Start Date. That is not possible.

Tabular Format for Data Relationships



The program checks the End Date against the Start Date and rejects this pair as impossible because the task can't end before it starts.

The value of End Date is *constrained by* Start Date, **because** End Date can't be earlier than Start Date.

The value of Start Date *constrains* End Date, **because** End Date can't be earlier than Start Date.

A Tabular Format for Data Relationships

	INPUT	OUTPUT		RESTRICTIONS	
Data Field	Entry Source	Display	Print	Constrained by	Constraint
Start Date					End Date
End Date				Start Date	

This table shows how the variables are related. Here are the columns:

Field: Create a row for each field (Consultant, End Date, and Start Date are examples of fields.)

Entry Source: What dialog boxes can you use to enter data into this field? Can you import data into this field? Can data be calculated into this field? List every way to fill the field -- every screen, etc.

Display: List every dialog box, error message window, etc., that can display the value of this field. When you re-enter a value into this field, will the new entry show up in each screen that displays the field? (Not always -- sometimes the program makes local copies of variables and fails to update them.)

A Tabular Format for Data Relationships

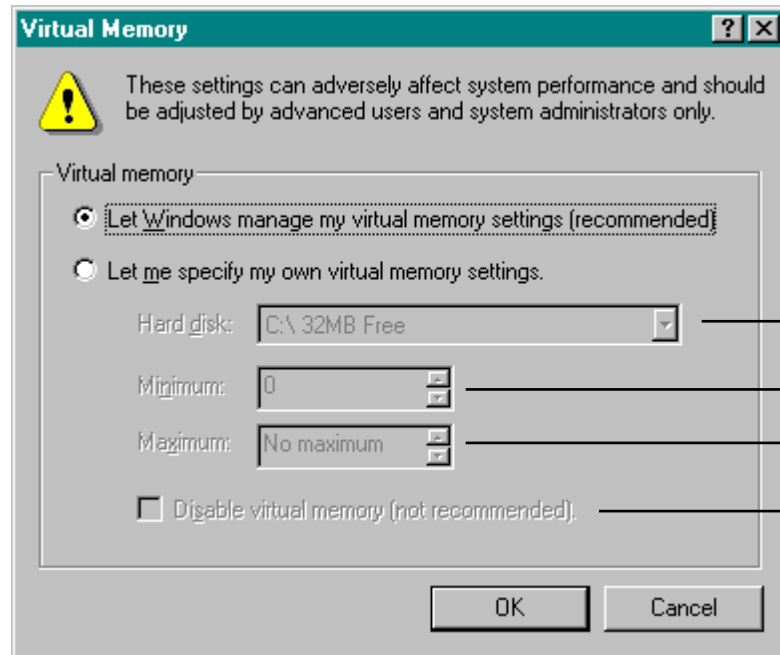
Print: List all the reports that print the value of this field (and any other functions that print the value).

Constrained By: List every variable that constrains the value of this variable. (What if you enter a legal value into this variable, then change the value of a constraining variable to something that is incompatible with this variable's value?)

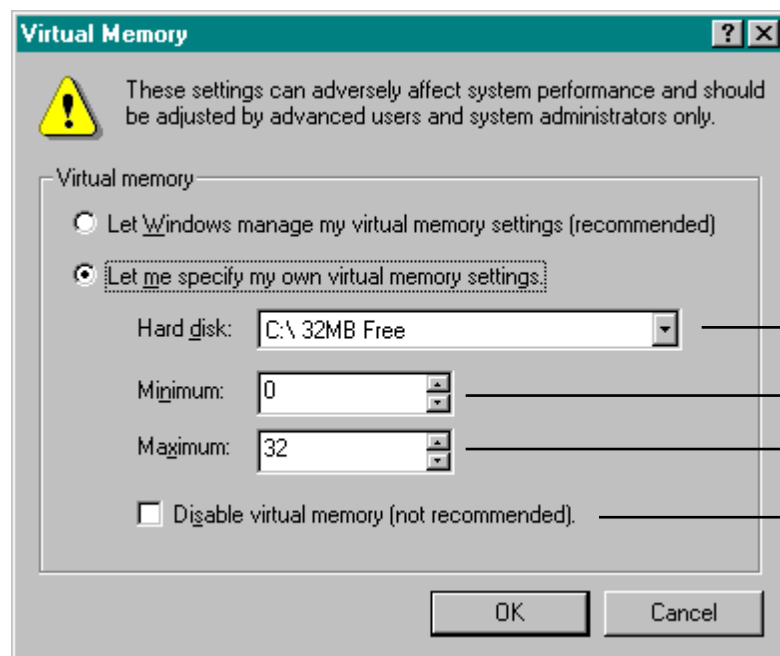
Constraint: List every variable that this one constrains.

Testing Functionality Relationships

The “Let Windows manage my virtual memory settings” radio button constrains the availability of the features in the “Let me specify my own virtual memory settings” group.



Inactive Features



Active Features

Notes

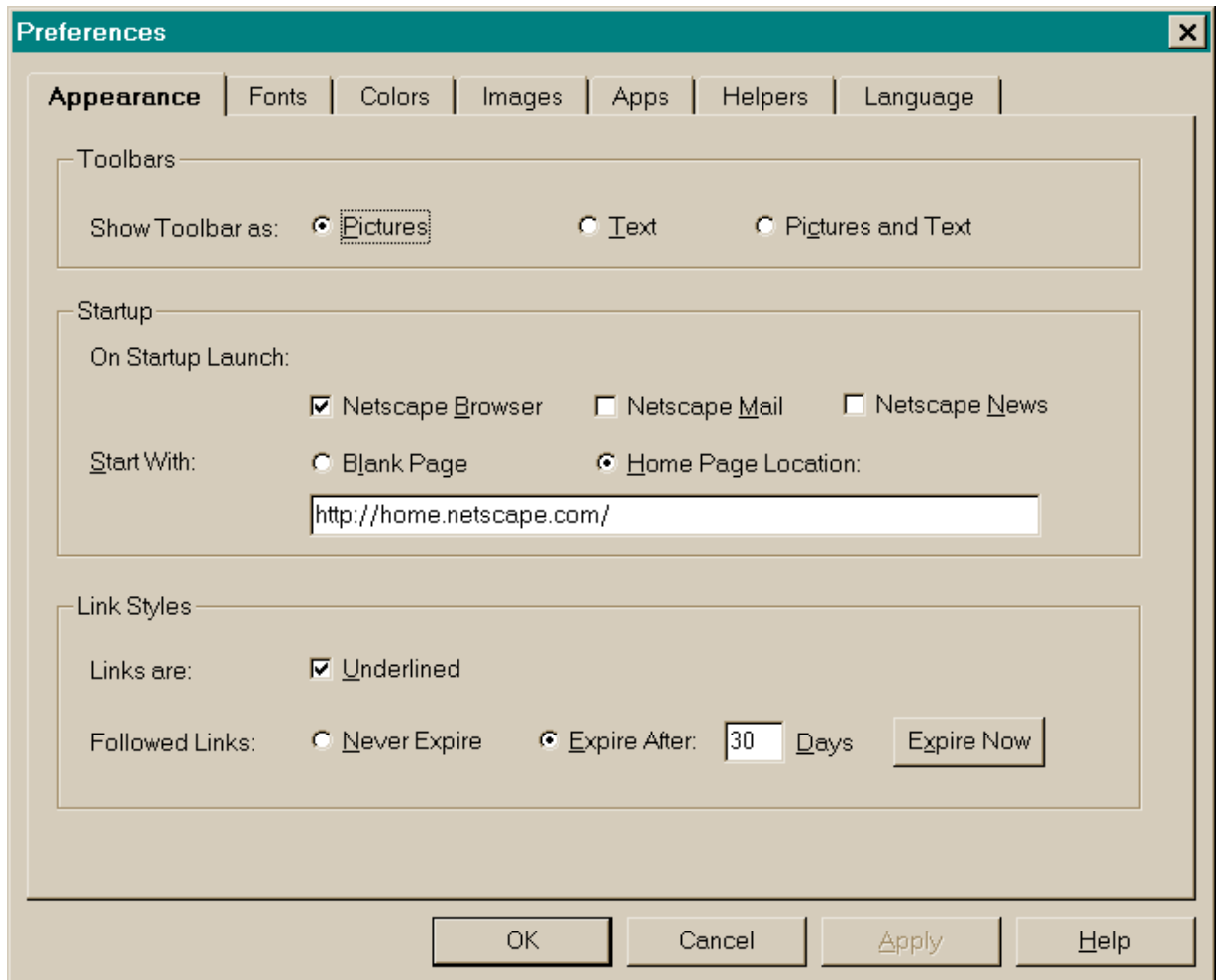
This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Test Case Design

Part 18.5

Testing Variables in Combination

Testing Variables in Combination



The Netscape Preferences dialog.

Testing Variables in Combination (cont.)

If we just look at the Appearance tab of the Netscape Preferences dialog, we see the following variables:

- Toolbars -- 3 choices (P, T, B)
(pictures, text or both)
- On Startup Launch -- 3 choices (B, M, N)
(browser, mail, news). Each of these is an independent binary choice.
- Start With -- 3 choices (B,V,E)
(blank page, home page names a valid file, home page name is empty)
(Many more cases are possible, but let's keep this simple and ignore that for a few slides)
- Links -- 2 choices (N, U)
(No Underlined or Underlined)
- Followed Links -- 2 choices (N,E)
(never expire, expire after 30 days)
(Many more cases are possible)

Testing Variables in Combination (cont.)

We can create $3 \times 2 \times 2 \times 2 \times 3 \times 2 \times 2 = 288$ different test cases by testing these variables in combination. Here are some examples from a combination table:

Test #	Toolbars PTB	On Startup, Browser Y/N	On Startup, Mail Y/N	On Startup, News Y/N	Start With BVE	Links NU	Followed NE
1	P	Y	Y	Y	B	N	N
2	P	Y	Y	N	B	N	E
3	P	Y	N	Y	B	U	N
4	P	Y	N	N	B	U	E
5	P	Y	Y	Y	V	N	N
6	P	Y	Y	N	V	N	E
7	P	N	N	Y	V	U	N
8	P	N	N	N	V	U	E
9	P	N	Y	Y	E	N	N
10	P	N	Y	N	E	N	E
11	P	N	N	Y	E	U	N
12	P	N	N	N	E	U	E

The following are all 288 of the test cases. Every value of every variable is combined with every combination of the other variables.

Testing Variables in Combination (cont.)

1	PYYYBNN	PNYYBNN	TTYBNN	TNYYBNN	BYYYBNN	BNYYBNN
2	PYYYVNE	PNYYVNE	TTYVNE	TNYYVNE	BYYYVNE	BNYYVNE
3	PYYYENN	PNYYENN	TTYENN	TNYYENN	BYYYENN	BNYYENN
4	PYYYBUE	PNYYBUE	TTYBUE	TNYYBUE	BYYYBUE	BNYYBUE
5	PYYYVUN	PNYYVUN	TTYVUN	TNYYVUN	BYYYVUN	BNYYVUN
6	PYYYEUE	PNYYEUE	TTYEUE	TNYYEUE	BYYYEUE	BNYYEUE
7	PYYNBNN	PNYNBNN	TYNBNN	TNYNBNN	BYYNBNN	BNYNBNN
8	PYYNVNE	PNYNVNE	TYNVNE	TNYNVNE	BYYNVNE	BNYNVNE
9	PYYNENN	PNYNENN	TYNENN	TNYNENN	BYYNENN	BNYNENN
10	PYYNBUE	PNYNBUE	TYNBUE	TNYNBUE	BYYNBUE	BNYNBUE
11	PYYNVUN	PNYNVUN	TYNVUN	TNYNVUN	BYYNVUN	BNYNVUN
12	PYYNEUE	PNYNEUE	TYNEUE	TNYNEUE	BYYNEUE	BNYNEUE
13	PYYYBNN	PNYYBNN	TTYBNN	TNYYBNN	BYYYBNN	BNYYBNN
14	PYYYVNE	PNYYVNE	TTYVNE	TNYYVNE	BYYYVNE	BNYYVNE
15	PYYYENN	PNYYENN	TTYENN	TNYYENN	BYYYENN	BNYYENN
16	PYYYBUE	PNYYBUE	TTYBUE	TNYYBUE	BYYYBUE	BNYYBUE
17	PYYYVUN	PNYYVUN	TTYVUN	TNYYVUN	BYYYVUN	BNYYVUN
18	PYYYEUE	PNYYEUE	TTYEUE	TNYYEUE	BYYYEUE	BNYYEUE
19	PYYNBNN	PNYNBNN	TYNBNN	TNYNBNN	BYYNBNN	BNYNBNN
20	PYYNVNE	PNYNVNE	TYNVNE	TNYNVNE	BYYNVNE	BNYNVNE
21	PYYNENN	PNYNENN	TYNENN	TNYNENN	BYYNENN	BNYNENN
22	PYYNBUE	PNYNBUE	TYNBUE	TNYNBUE	BYYNBUE	BNYNBUE
23	PYYNVUN	PNYNVUN	TYNVUN	TNYNVUN	BYYNVUN	BNYNVUN
24	PYYNEUE	PNYNEUE	TYNEUE	TNYNEUE	BYYNEUE	BNYNEUE
25	PYNYBNN	PNNYBNN	TYNYBNN	TNNYBNN	BYNYBNN	BNNYBNN
26	PYNYVNE	PNNYVNE	TYNYVNE	TNNYVNE	BYNYVNE	BNNYVNE
27	PYNYENN	PNNYENN	TYNYENN	TNNYENN	BYNYENN	BNNYENN
28	PYNYBUE	PNNYBUE	TYNYBUE	TNNYBUE	BYNYBUE	BNNYBUE
29	PYNYVUN	PNNYVUN	TYNYVUN	TNNYVUN	BYNYVUN	BNNYVUN
30	PYNYEUE	PNNYEUE	TYNYEUE	TNNYEUE	BYNYEUE	BNNYEUE
31	PYNNBNN	PNNNBNN	TYNNBNN	TNNNBNN	BYNNBNN	BNNNBNN
32	PYNNVNE	PNNNVNE	TYNNVNE	TNNNVNE	BYNNVNE	BNNNVNE
33	PYNNENN	PNNNENN	TYNNENN	TNNNENN	BYNNENN	BNNNENN
34	PYNNBUE	PNNNBUE	TYNNBUE	TNNNBUE	BYNNBUE	BNNNBUE
35	PYNNVUN	PNNNVUN	TYNNVUN	TNNNVUN	BYNNVUN	BNNNVUN
36	PYNNNEUE	PNNNEUE	TYNNNEUE	TNNNEUE	BYNNNEUE	BNNNEUE
37	PYNYBNN	PNNYBNN	TYNYBNN	TNNYBNN	BYNYBNN	BNNYBNN
38	PYNYVNE	PNNYVNE	TYNYVNE	TNNYVNE	BYNYVNE	BNNYVNE
39	PYNYENN	PNNYENN	TYNYENN	TNNYENN	BYNYENN	BNNYENN
40	PYNYBUE	PNNYBUE	TYNYBUE	TNNYBUE	BYNYBUE	BNNYBUE
41	PYNYVUN	PNNYVUN	TYNYVUN	TNNYVUN	BYNYVUN	BNNYVUN
42	PYNYEUE	PNNYEUE	TYNYEUE	TNNYEUE	BYNYEUE	BNNYEUE
43	PYNNBNN	PNNNBNN	TYNNBNN	TNNNBNN	BYNNBNN	BNNNBNN
44	PYNNVNE	PNNNVNE	TYNNVNE	TNNNVNE	BYNNVNE	BNNNVNE
45	PYNNENN	PNNNENN	TYNNENN	TNNNENN	BYNNENN	BNNNENN
46	PYNNBUE	PNNNBUE	TYNNBUE	TNNNBUE	BYNNBUE	BNNNBUE
47	PYNNVUN	PNNNVUN	TYNNVUN	TNNNVUN	BYNNVUN	BNNNVUN
48	PYNNNEUE	PNNNEUE	TYNNNEUE	TNNNEUE	BYNNNEUE	BNNNEUE

Testing Variables in Combination (cont.)

Instead of thinking about all possible ways of combining all of the variables, suppose that we combined them so that only every possible *pair* of values was covered.

Test #	Toolbars PTB	On Startup, Browser Y/N	On Startup, Mail Y/N	On Startup, News Y/N	Start With BVE	Links NU	Followed NE
1	P	Y	Y	Y	B	N	N
2	P	Y	N	N	V	N	E
3	P	N	Y	Y	E	U	E
4	P	N	N	N	B	U	N
5	T	Y	Y	N	V	U	N
6	T	Y	N	Y	E	U	E
7	T	N	Y	N	B	N	E
8	T	N	N	Y	V	N	N
9	B	Y	Y	Y	E	N	N
10	B	Y	N	N	B	U	E
11	B	N	Y	Y	V	N	E
12	B	N	N	N	E	U	N

Testing Variables in Combination (cont.)

This deceptively simple idea comes to us from Bellcore (www.bellcore.com), and was recently lectured on by Siddhartha Dalal.

Read the entire paper at:

<http://www.argreenhouse.com/papers/gcp/AETGieee97.shtml>

Test #	Toolbars PTB	On Startup, Browser Y/N	On Startup, Mail Y/N	On Startup, News Y/N	Start With BVE	Links NU	Followed NE
1	1	1	1	1	1	1	1
2	1	1	2	2	2	1	2
3	1	2	1	1	3	2	2
4	1	2	2	2	1	2	1
5	2	1	1	2	2	2	1
6	2	1	2	1	3	2	2
7	2	2	1	2	1	1	2
8	2	2	2	1	2	1	1
9	3	1	1	1	3	1	1
10	3	1	2	2	1	2	2
11	3	2	1	1	2	1	2
12	3	2	2	2	3	2	1

Some people find it easier to see the pair-wise combinations if shown them with numbers.

Testing Variables in Combination (cont.)

I simplified the combinations by simplifying the choices for two of the fields. In the Start With field, for example, I used either a valid home page name or a blank name. Some other test cases that could go into this field are:

- file name (name.htm instead of using http:// to define a protocol) on the local drive, the local network drive, or the remote drive
- maximum length file names, maximum length paths
- invalid file names and paths
- etc.

For combination testing, select a few of these that look like they might interact with other variables. Test the rest independently.

Similarly for the Expire After field. This lets you enter the number of days to store links. If you use more than one value, use boundary cases, not all the numbers in the range.

Combination Test Case Design

1

Combination Test Case Design

Group A

- ☒ Option1
- ☐ Option2
- ☐ Option3

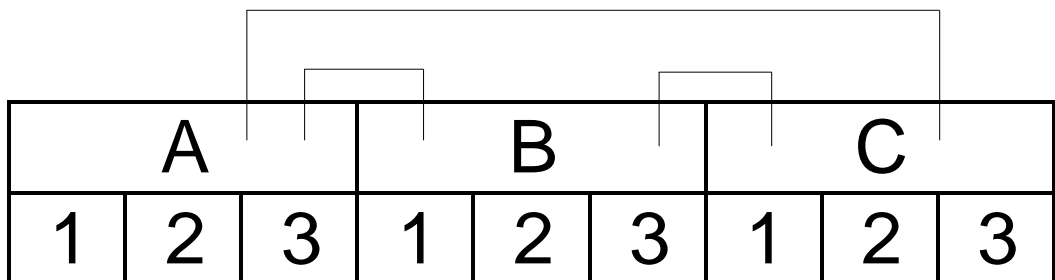
Group B

- ☒ Option1
- ☐ Option2
- ☐ Option3

Group C

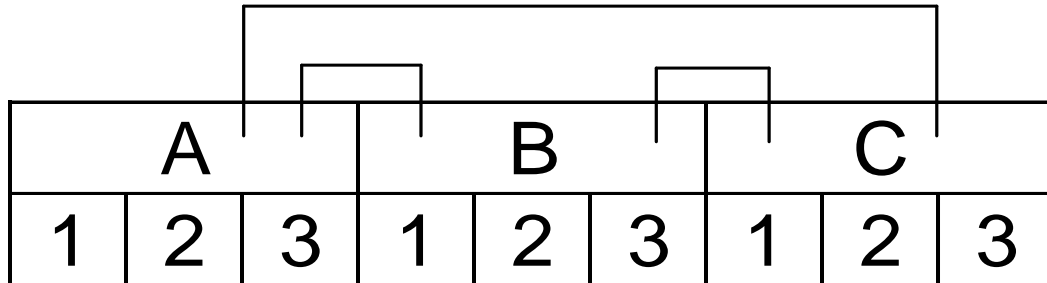
- ☒ Option1
- ☐ Option2
- ☐ Option3

OK Cancel



Combination Test Case Design

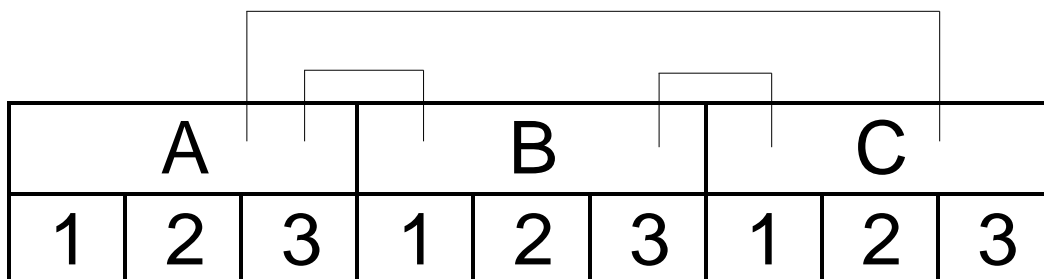
2



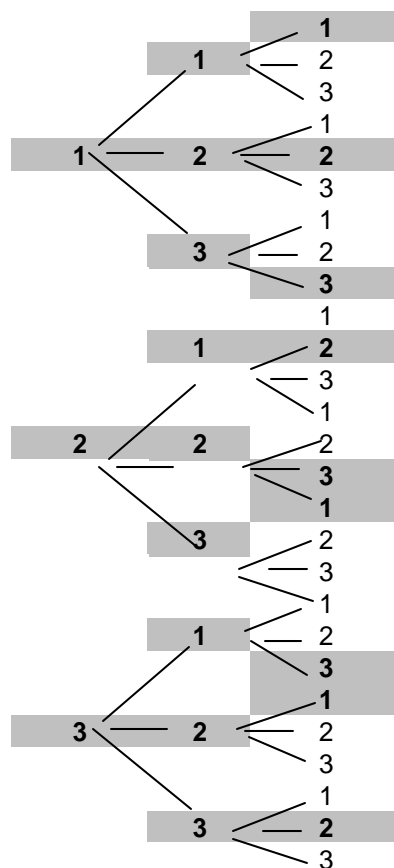
Case	A	B	C
1	1	1	1
2	1	1	2
3	1	1	3
4	1	2	1
5	1	2	2
6	1	2	3
7	1	3	1
8	1	3	2
9	1	3	3
10	2	1	1
11	2	1	2
12	2	1	3
13	2	2	1
14	2	2	2
15	2	2	3
16	2	3	1
17	2	3	2
18	2	3	3
19	3	1	1
20	3	1	2
21	3	1	3
22	3	2	1
23	3	2	2
24	3	2	3
25	3	3	1
26	3	3	2
27	3	3	3

Combination Test Case Design

3

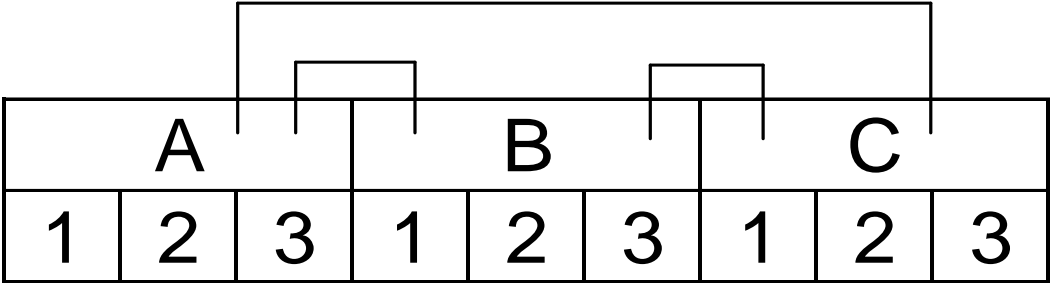


Case	A	B	C
1	1	1	1
2	1	1	2
3	1	1	3
4	1	2	1
5	1	2	2
6	1	2	3
7	1	3	1
8	1	3	2
9	1	3	3
10	2	1	1
11	2	1	2
12	2	1	3
13	2	2	1
14	2	2	2
15	2	2	3
16	2	3	1
17	2	3	2
18	2	3	3
19	3	1	1
20	3	1	2
21	3	1	3
22	3	2	1
23	3	2	2
24	3	2	3
25	3	3	1
26	3	3	2
27	3	3	3



Combination Test Case Design

A



1			1			1		
1				2			2	
1					3			3
	2		1				2	
	2			2				3
	2				3	1		
		3	1					3
		3		2		1		
		3			3		2	

Case	A	B	C
1	1	1	1
2	1	2	2
3	1	3	3
4	2	1	2
5	2	2	3
6	2	3	1
7	3	1	3
8	3	2	1
9	3	3	2

Combination Test Case Design

5

Combination Test Case Design

Group A

- ☒ Option1
- ☐ Option2
- ☐ Option3

Group B

- ☒ Option1
- ☐ Option2
- ☐ Option3

Group C

- ☒ Option1
- ☐ Option2
- ☐ Option3

OK Cancel

Here are your
Final Test Cases:

Case	A	B	C
1	1	1	1
2	1	2	2
3	1	3	3
4	2	1	2
5	2	2	3
6	2	3	1
7	3	1	3
8	3	2	1
9	3	3	2

Combinatorial Design Exercise

See Exercises

Notes

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Testing Computer Software

Part 23

A Survey of Software Testing Tools

Software Testing Tools

- Basic testing environment
- Analyzers
 - Static analyzers
 - Dynamic analyzers
- Other testing support tools and utilities
- Information -- Source materials and training
- Other productivity tools and helps

Software Testing Tools

		Black	Grey	Glass
Basic environment				
	Computer or workstation	x	x	x
	Word processor	x	x	x
	Outline processor	x	x	x
	Text editor	x	x	x
	Spreadsheet	x	x	x
	Problem tracking system	x	x	x
	Flow -charting program	x	x	x
	Project management program	x	x	x
Static Analyzer				
	Standard compliance-checker		x	x
	Complexity analyzer		x	x
	Dataflow analyzer			x
	Source code documentor			x
Dynamic Analyzer				
	Coverage analyzer		x	x
	Tracer -- Debugger		x	x
	Profiler			x
	Assertion checker			x
	Memory validity checker		x	x
Other testing support				
	Automated test case generator		x	x
	Automated acceptance/regression tester	x		
	Automated exploratory tester	x		x
	Version control	x	x	x
	File utilities	x	x	x
	String utilities	x	x	x
	Diagnostic utilities	x	x	x
	O/S (configuration file) loader	x	x	
	Memory utilities	x	x	
	Macro/batch command language	x	x	x
	Screen capture utilities	x		
	Virus checker	x	x	x
	Stress inducer	x		
	Error interpreter	x	x	x
	Resource editor		x	x
	Resource checker	x	x	x
	Source browser	x	x	x
	Events watcher	x	x	x
Information -- Source Materials & Training				
	The application under test	x	x	x
	Source materials & training	x	x	x
	Testing software library	x	x	x
	On-line services -- The Internet	x	x	x
	Tool catalogs	x	x	x
Other productivity tools & helps				
	Database program	x	x	x
	Presentation Graphics program	x	x	x
	Electronic mail	x	x	x
	Contact/Calendar/To-do manager	x	x	x
	Project notebook	x	x	x
	The programmer	x	x	x

Tools and Utilities Providers

- Testing tool vendors
- Development tool vendors
- Utilities publishers
- Shareware and freeware distributors
- Your internal development groups

Other Sources of Information

Brian Marick's Testing Tools and Suppliers List

<http://www.stlabs.com/marick/faqs/tools.htm>

SQE Single Source--Software Testing Publication

<http://www.sqe.com/single/testing.htm>

Development Tool Mail Catalogs

- Programmer's Paradise
 - 1-800-445-7899
 - <http://www.pparadise.com>
- The Programmer's SuperShop
 - 1-800-421-8006
 - <http://www.supershops.com>

Notes

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Testing Computer Software

Part 25

The Test Plan

Lead Software Test Projects with Confidence

Part 3

The Test Plan

Test Plans

“Test Plan links the Specifications to the Item to be tested.”

“A translation document linking design spec to dynamic test.”

*Ould & Unwin Testing in Software Development
British Computer Society 1986 p. 14*

Test Plans

There are many Test Plan paradigms. No one is any better than the next. The goal, for me, is that they are meaningful and they do not take you too much away from the testing project itself.

Test Plans

Master Test Plan

Unit Test Plan

Integration Test Plan

System Test Plan

Acceptance Test Plan

OR

Test Plan
Who, What, Why,
When, Where, How

Test Cases

OR

Test Plan
and
Test Cases

OR

Testing Organization

- Policies
- Procedures
- Processes
- Bug Tracking Process

1-Page
Test Plan

Test Cases

Test Plans

Some Examples of Test Plan types:

- Unit Test Plan
- Integration Test Plan
- Acceptance Test Plan
- User Trial Test Plan
- Final & GM Test Plan
- Master Test Plan
- Public Beta Test Plan

Test Plans

Chapter 8

General Discussion

Test Plans

“A Test Plan is a valuable tool to the extent that it helps you manage your testing project and find bugs. Beyond that, it is a diversion of resources.”

Testing Computer Software p. 205

Test Planning

A Test Plan tells anyone familiar to the project exactly what the test team is doing, exactly what they are not doing and how it will all get done.

Test Planning

Do people read these things?

Who reads them?

What do they look for?

Who reviews It?

How long should it take to write?

What format?

Test Planning

What do they do with Test Plans at your Company?

Test Planning

Test Plan Review:

Who has input?

Are they Signed?

Signatures make the test plan a
'contract' with agreements, obligations
and responsibilities.

Updating the Plan

when?

how often?

Republish and circulate Addendum when there are
functional, schedule or resource changes or you
pass a major milestone.

Test Plan Templates

There are many templates and Guides available on the Net and in Published Documentation.

Templates

- softGear technology's template (see the Appendix)
- CMM/SEI - Software Engineering Institute at Carnegie Mellon Univ.
- ANSI
- ISO 9000
- Your own company may have a template

Test Plan Templates

Templates

- 829-1983 IEEE Standard for Software Test Documentation
- 1008-1987 IEEE Standard for Software Unit Testing
- 1012-1986 IEEE Standard for Software Verification & Validation Plans
- 1059-1993 IEEE Guide for Software Verification & Validation Plans
- A straightforward template with explanations. Nothing fancy.
http://www.pogner.demon.co.uk/mil_498/stp-did.htm
- Guidelines and Boilerplate. Just the basics.
<http://www2.ics.hawaii.edu/~johnson/FTR/Weiss/weiss-test>
- Software Test Plan from Department of Defense. Very, very thorough.
<http://www-mel.nrlmry.navy.mil/docs/stp05.html>

Test Planning

Chapter 9

Making a Test Plan Meaningful

Test Planning

Why write a Test Plan?

What can you do to make your exercise of writing a test plan worthwhile?

Test Planning

The best way to make a test plan useful is
make it worth reading!

What does it say inside?

Test Planning

Let's do another Brainstorm exercise:

What are the Contents of a Test Plan?

Notes

[illegible]

Test Planning part 2

Possible Test Plan contents:

Schedules

Functions to test, functions not to test

Milestone criteria

Test cases

Methodology

Objective

Overview

Purpose

Goals of testing

Test Inputs

Expected Actual Outputs

What Metrics you will track and for what reason

Reporting mechanism:

- Bug Tracking w/ priorities, etc.....

- Weekly coverage reporting

- Final Report

Begin Test Criteria

Release criteria

Stop testing criteria

Test Planning

Build Process and Cycle

Test Types

Outside or Inside Beta Test Planning- time required, goals, finding people, management, goals, results

Automation

Acceptance tests

Tests during what cycle

Smoke Test- Exercise Entire System- if you can automate it can be the acceptance test.

Benchmarking- comparing your product to others in class

Performance testing

Review Product/code specifications

Software audit, review and inspection

Making the build/ Build Engineering

File compare, list of files

Test Procedure Specification- test scripts, test cases,

Risk

Exclusions

Test Planning

Take information analyzed at the beginning of the project and, if it's necessary information to publish: drop it into the plan.

More importantly- anything that you know will need agreement from the entire team or to be referred to in later phases of development needs to be in the plan.

Test Planning

The contents of a Meaningful Test Plan gets the whole team on the same page. The whole Project Team will understand the test team's priorities and risks to the schedule.

It lays out the risk for the project team and company to see and deal with early in the project.

Test Planning

Meaningful Test Plans will have very clear and explicit sections on:

- Test Strategy
- Risk
- Features not to test
- Exclusions
- Contingencies
- Tools/Hardware/Resources
- Change Management
- Requirements (frozen and complete specifications, for example ;-)

Test Planning

Never write or agree to anything you can't do.

A good test plan may be controversial. Not everyone is supposed to like your plan, test times, risks, exclusions and features not to be tested.

Avoid “Assumptions” sections.

Test Planning

Meaningful Test Plans can help prevent:

“Why wasn’t this tested?”

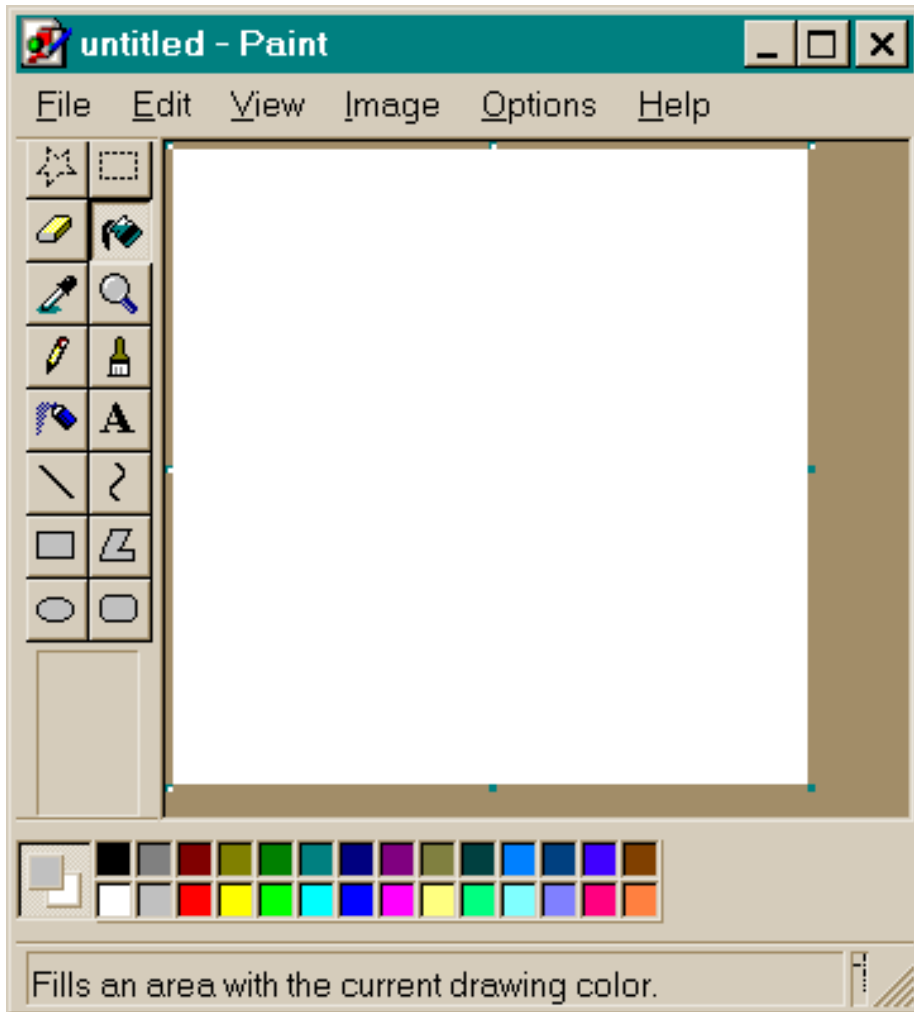
“ Why did you wait so long to test this?”

“ I never knew that!”

Notes

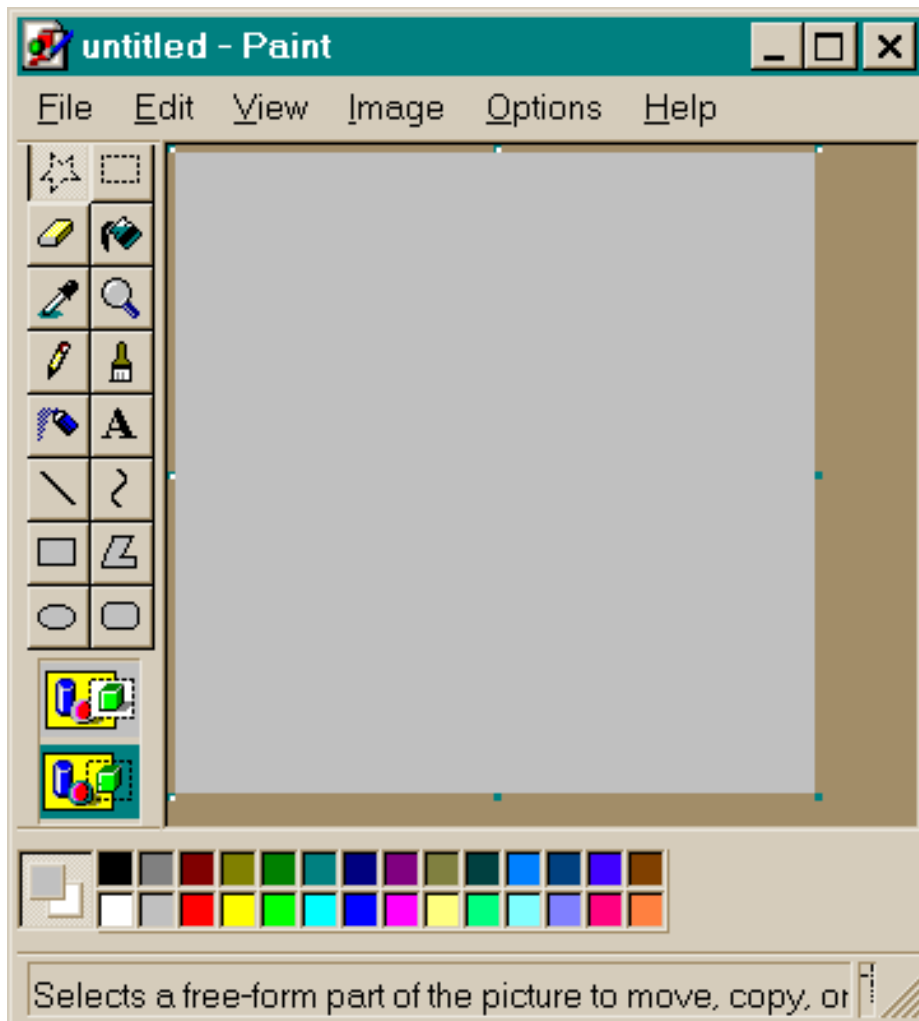
[illegible]

Bug Reporting Exercise 1 (1)



Here's an example of a bug in Win 95 Paint. This is the opening screen, with the paint can selected. Select a color by clicking on the palette, then click on the white box and you get a colored picture.

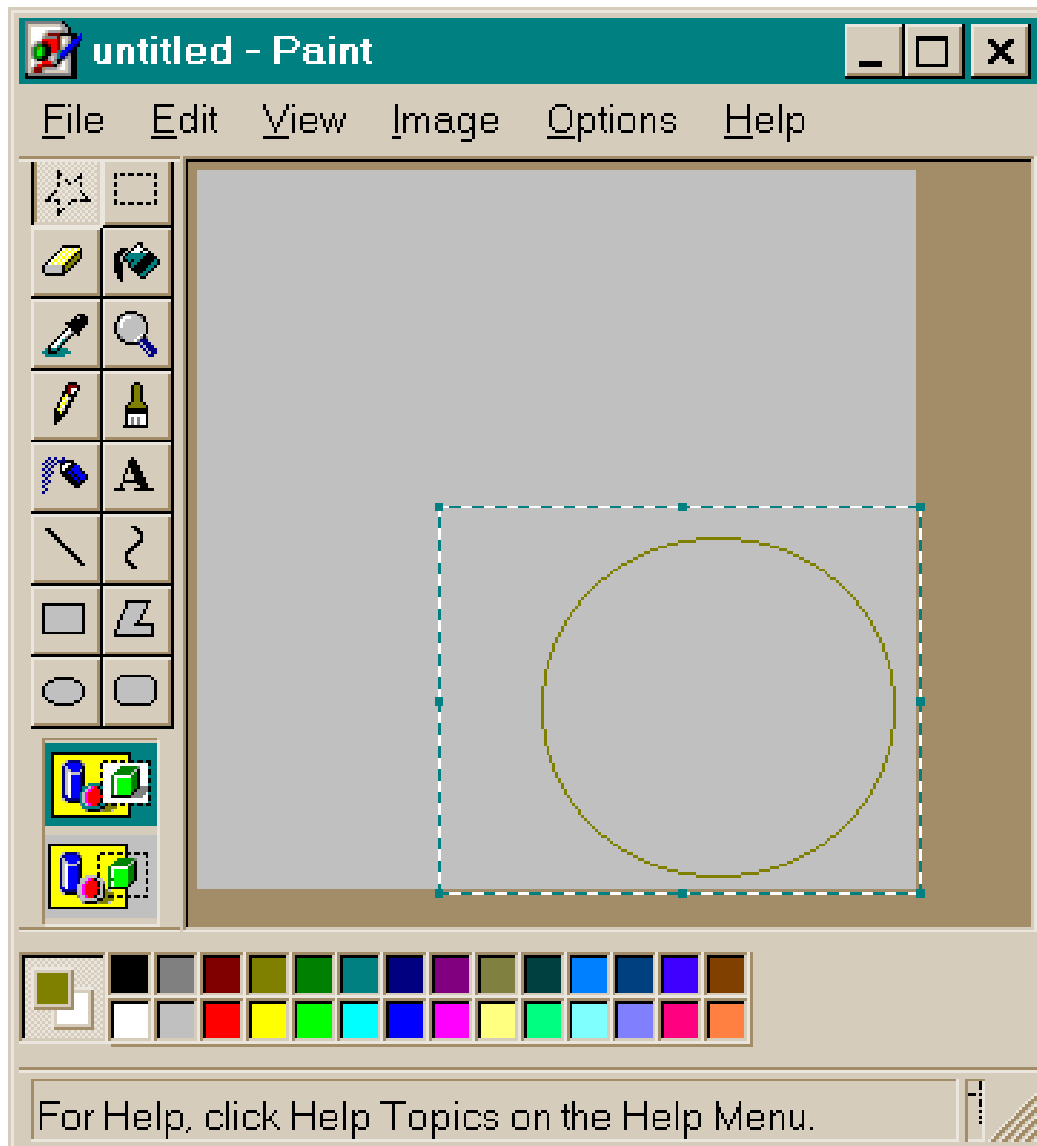
Bug Reporting Exercise 1 (2)



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 move it, cut it, copy it, etc.

Bug Reporting Exercise 1

(3)



First, draw a circle using a different color (so you will see what's selected). Then use the freehand select tool to select the area around it.

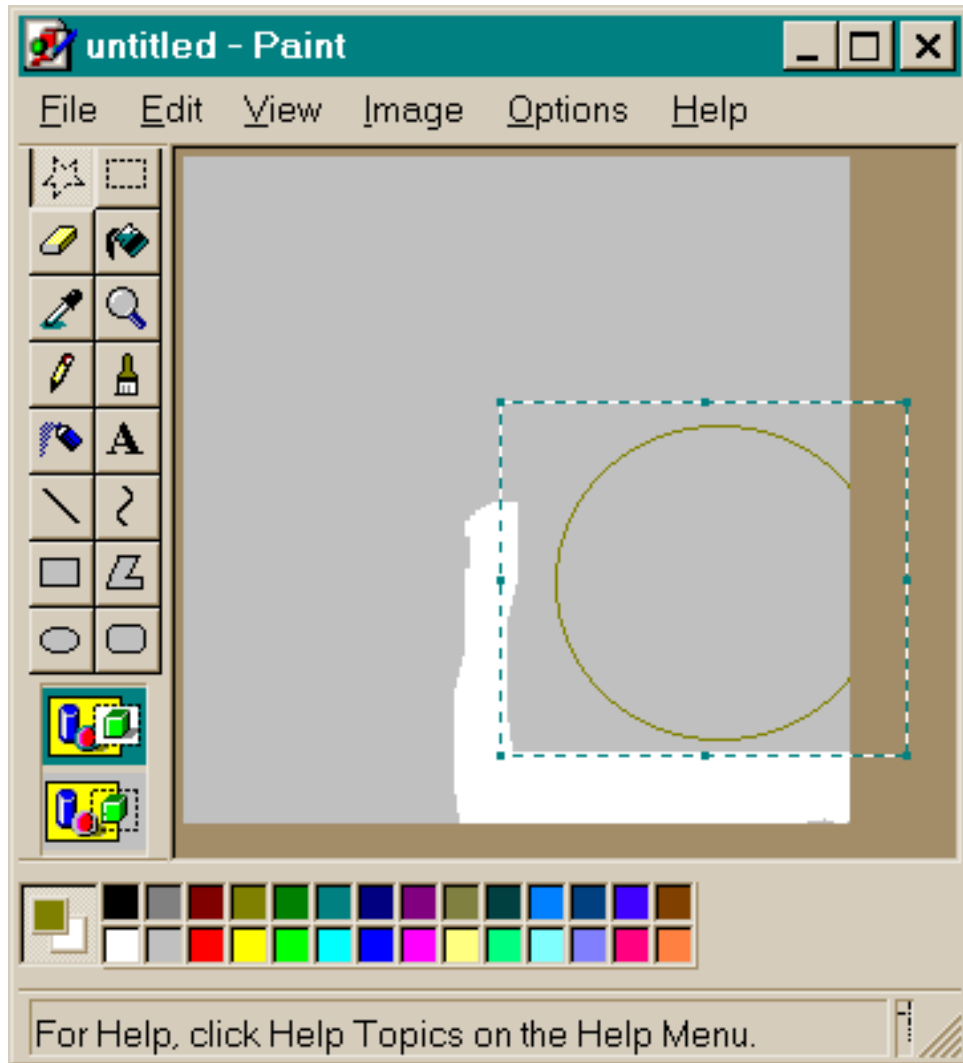
Bug Reporting Exercise 1

(4)



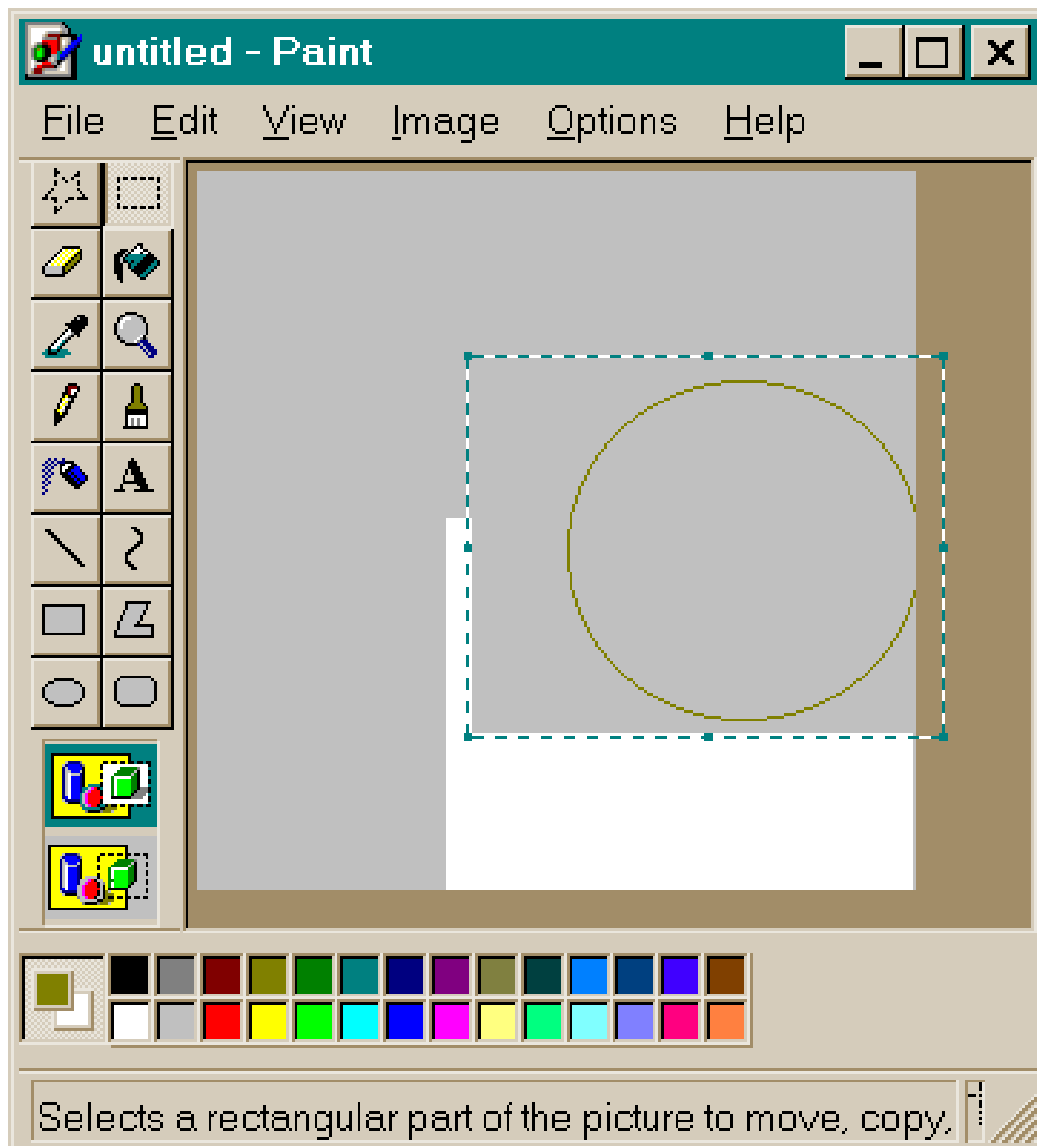
You can cut the selection. (The border is jagged because you used the freehand selection tool to select around the circle. To make a square selection, use the rectangular selection tool.)

Bug Reporting Exercise 1 (5)



Or you can move the selection.

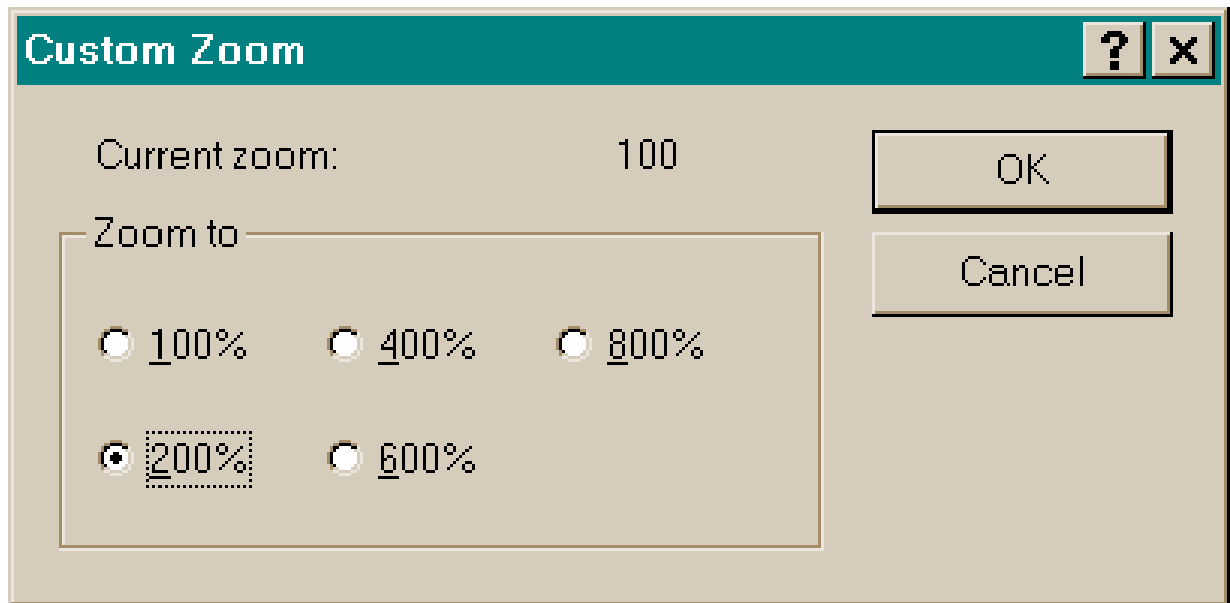
Bug Reporting Exercise 1 (6)



This illustrates a move after using the rectangular selection tool.

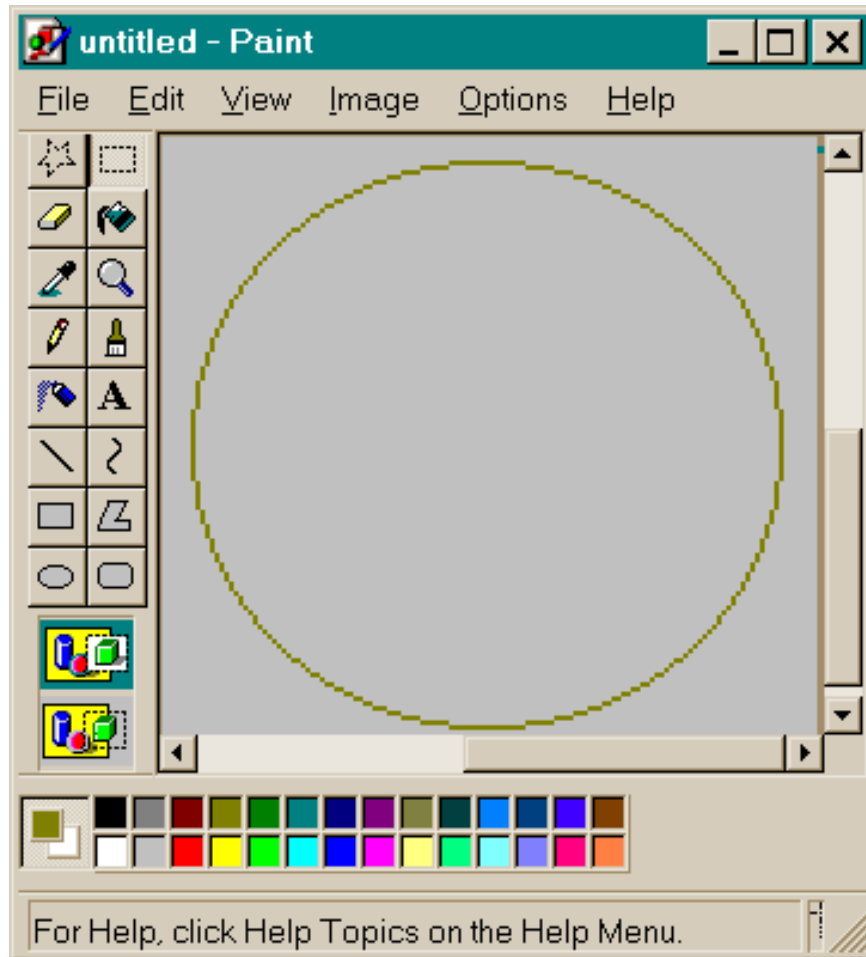
Bug Reporting Exercise 1

(7)



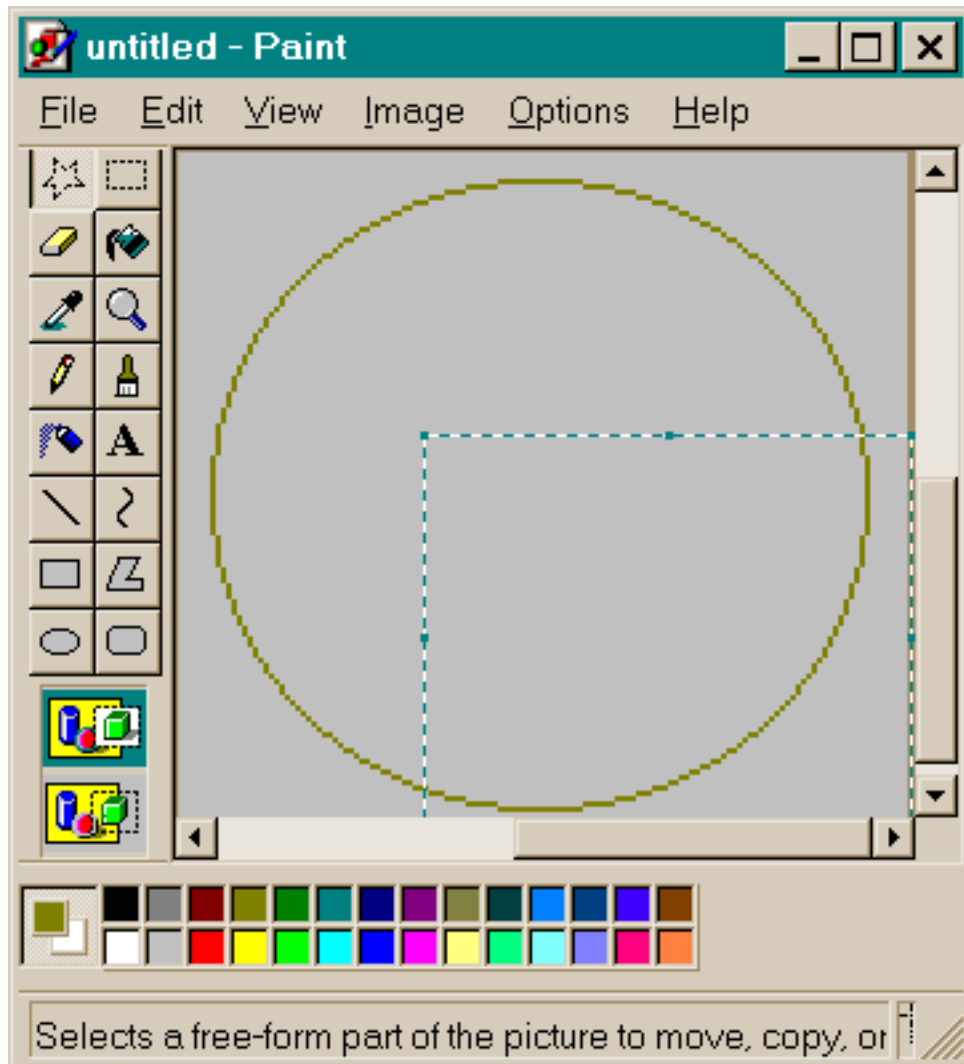
Now play with it in a different scale. Zooming to 200% means that you double the apparent size . . .

Bug Reporting Exercise 1 (8)



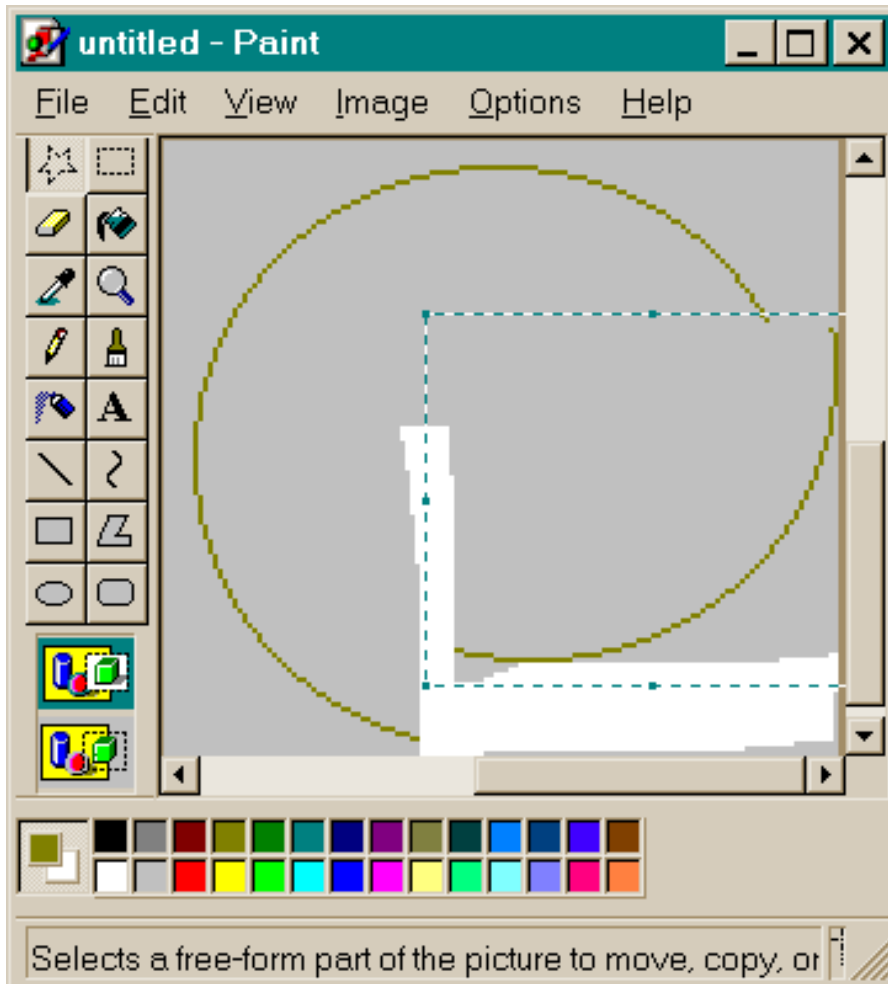
Draw the same circle as before -- it just looks bigger because of the 200% zooming.

Bug Reporting Exercise 1 (9)



Now, select part of the circle. We'll try the move and cut features.

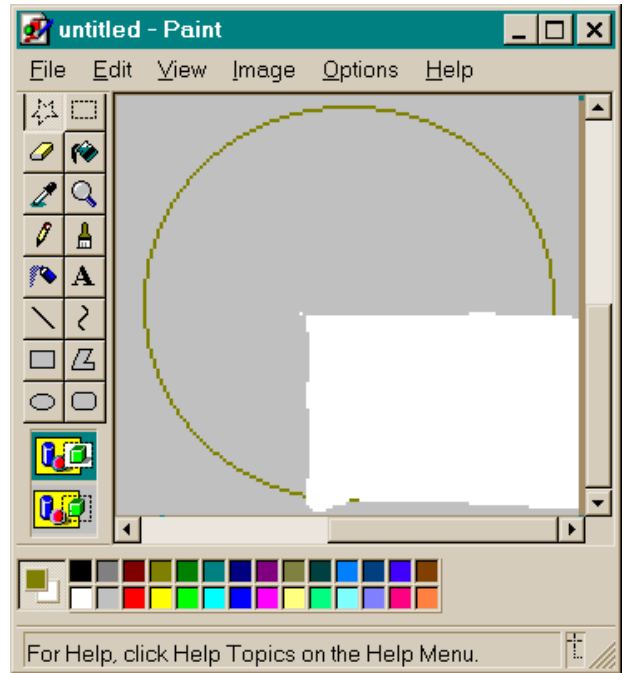
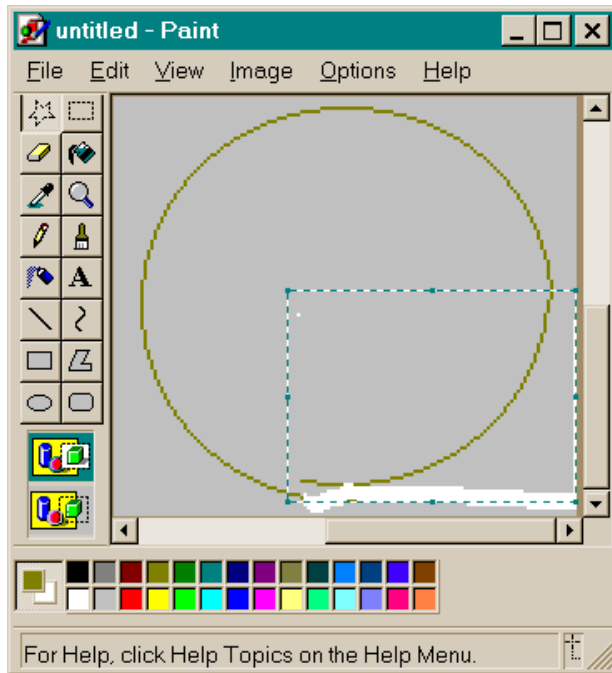
Bug Reporting Exercise 1 (10)



Try to move it. --- That works.
But if you try to cut it instead of trying to move it,
nothing happens. The dashed line around the
selection disappears, but nothing is cut.

Bug Reporting Exercise 1

(11)



But if you move the selected area first, and *then* you try to cut (press Delete or Ctrl-X), it works.

Bug Reporting Exercise 1

(12)

Here is your assignment.

1. Write a bug report that describes this bug. Include only three sections:
 - Problem Summary;
 - Problem Description;
 - Steps to Reproduce.
2. Meet with your group to read each other's reports.
 - How much do you learn from the summary?
 - How clear are the description and steps to reproduce?
 - How complete are the description and steps to reproduce?
 - How accurate are the description and steps to reproduce?

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Notes

[illegible]

Sample Exercise Results

Problem Summary:

Cut fails on a freehand-selected object while in zoom-in mode.

Problem Description:

While in zoom-in mode, if you use the freehand selection tool to select an object and then try to cut it, the cut operation will not work. It works fine in Normal view.

Steps to Reproduce:

1. Start the program.
2. Choose the View|Zoom|Custom menu command and set Custom Zoom to 200%.
3. Draw an ellipse or a rectangle.
4. Use the freehand selection tool to select the drawn object.
5. Now try to cut the selected object.

RESULT--Nothing is deleted.

Note: After step 4, if you try to move the object first, and then try to cut it, it will work fine.

Sample Exercise Results (cont.)

Here is BugNet's description, from The Windows 95 Bug Collection, by Bruce Brown:

“If you cut a region in a Windows 95 Paint image selected with the Free-Form Select tool while you are zoomed in, the wrong region is cut. There is no workaround at present.”

Notes

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Bug Reporting Exercise 1.1

In the previous Microsoft Paint exercise, how would you rate the:

- Severity?
- Frequency?

Sample Exercise Results

Severity = 3-Minor

Frequency = 2-Often

Notes

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Bug Reporting Exercise 2

(1)

Quicken 5 for Windows - TEST101 - [Checking: Bank]

File Edit Activities Lists Reports Plan Add-Ons Online Window Help

Registr Accts Recon Reports Online Calendar QFNet SnpShts Loans Forecast Tax Plan Help

Quicken HomeBase

Checking: Bank

Go To Options Help IconBar

Checking

Delete Find Transfer Options Report Close

Date	Num	Payee	Category	Payment	Clr	Deposit	Balance
5/ 3/96		Opening Balance	[Checking]		R		0 00
5/ 3/96	101	1 test		100 00			-100 00
5/ 3/96	102	2 test		100 00			-200 00
5/ 3/96	103	3 test		100 00			-300 00
5/ 3/96	104	4 test		100 00			-400 00
5/ 3/96	105	5 test		100 00			-500 00
5/ 3/96	106	6 test		100 00			-600 00
5/ 3/96	107	7 test		100 00			-700 00
5/ 3/96	108	8 test		100 00			-800 00
5/ 3/96	109	9 test		100 00			-900 00
5/ 3/96	110	1 test		100 00			-1,000 00
5/ 3/96	111	2 test		100 00			-1,100 00
5/ 3/96	112	3 test		100 00			-1,200 00
5/ 3/96	113	4 test		100 00			-1,300 00
5/ 3/96	114	5 test		100 00			-1,400 00
5/ 3/96	115	6 test		100 00			-1,500 00
5/ 3/96	116	7 test		100 00			-1,600 00
5/ 3/96	117	8 test		100 00			-1,700 00
5/ 3/96	118	9 test		100 00			-1,800 00

Checking

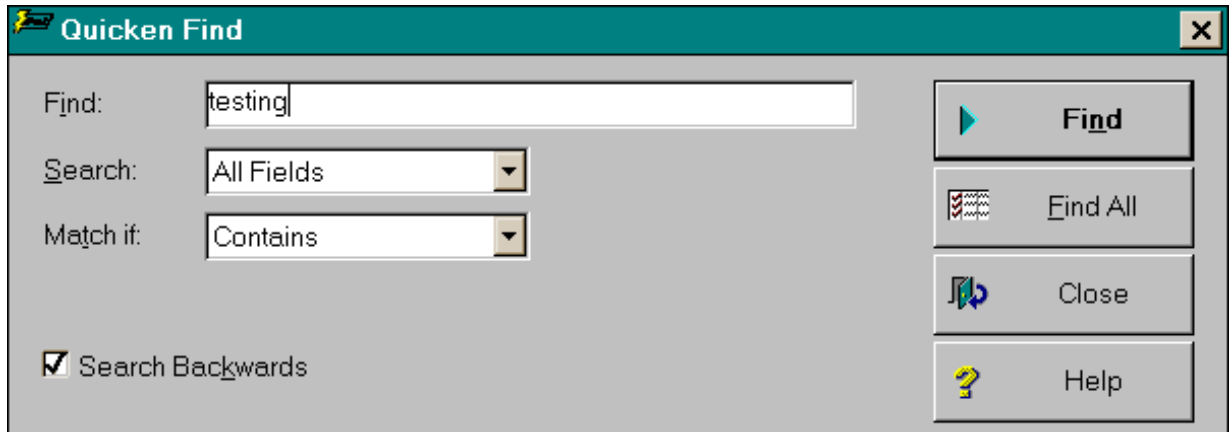
☒ 1-Line Display

Ending Balance: -4,500 00

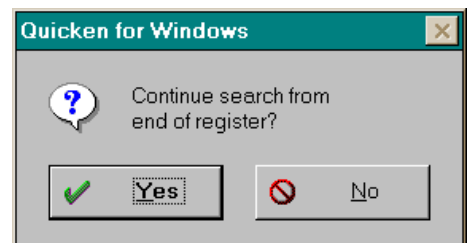
- Create a sample database. These are checks.
- Enter many new checks.

Bug Reporting Exercise 2

(2)

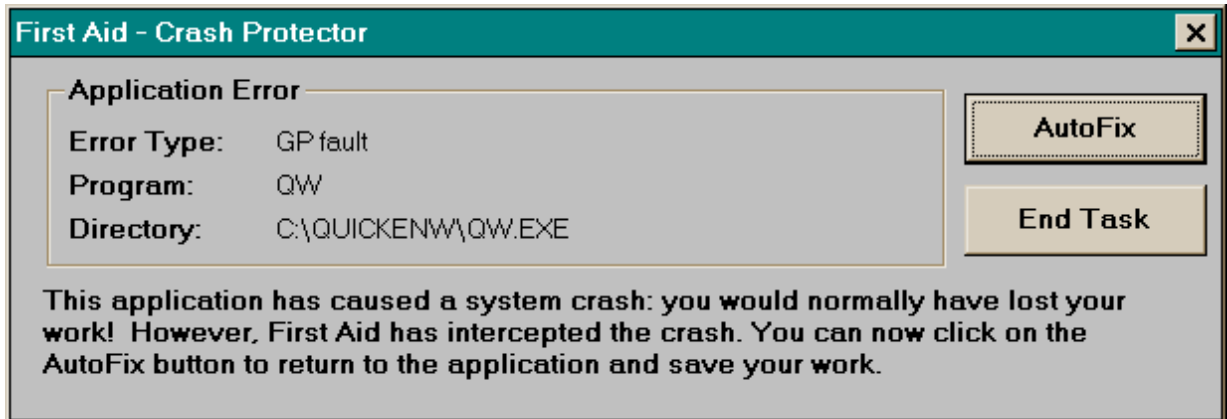


- Now, you search the checks to find one. Here, you search for the word “testing” (expected not to get a hit). The program searches backwards, from the currently selected check to the start of the register.
- It doesn’t find any instances of “testing” so it asks whether it should keep searching from the end of the register backwards. Then ...



Bug Reporting Exercise 2

(3)

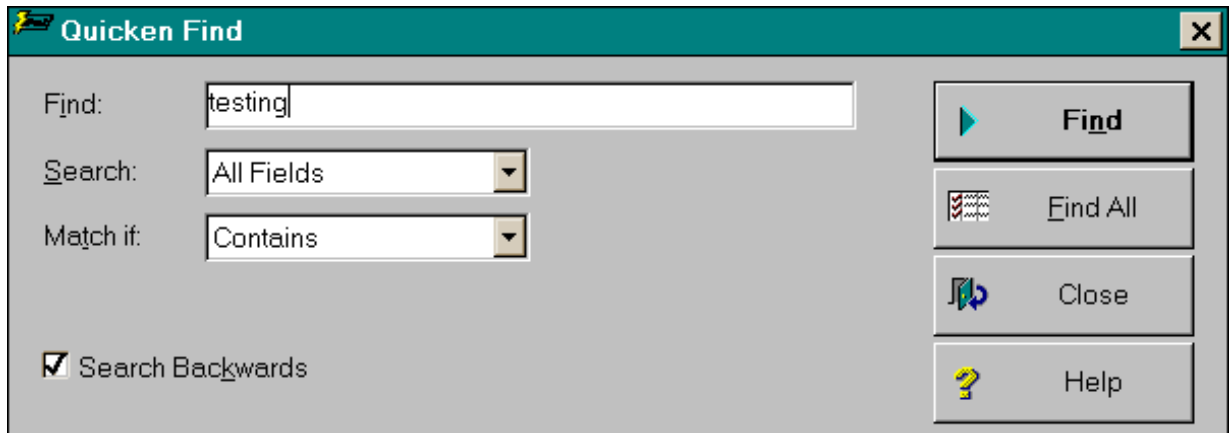


A General Protection Fault (GPF)!

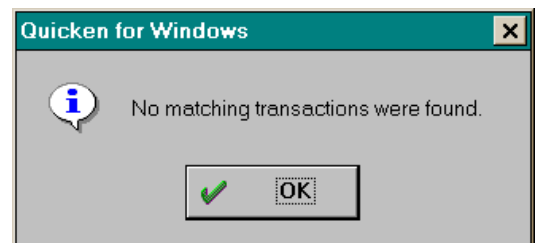
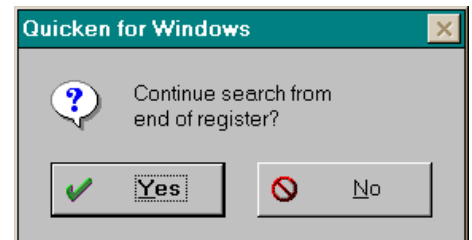
- The “First Aid” application tries to protect the customer from losing data when there is a GPF. It’s always *possible* that the crash was *caused* by an interaction between Quicken and First Aid, so you try the test again after turning off First Aid.
- When you re-ran the test, Quicken crashed again, with a Win 95 system window that identifies a GPF. Therefore the bug was not due to First Aid.

Bug Reporting Exercise 2

(4)



- When analyzing a bug, it's wise to try to recreate it on another computer. So, you re-ran the test again with a second PC. This time, the search didn't crash.
- The crashing computer is a Pentium with 32 megs RAM, a Logitech trackball, the MS keyboard, a 1.6 gig hard drive, no disk compression, a 4 meg high res MPEG video card and a big monitor.
- The other is an 8 meg 486 with an MS Mouse, an old standard keyboard, a 540 meg hard drive (compressed) and basic SVGA video.



Bug Reporting Exercise 2

(5)

Because this is a crash, you decide to get it into the tracking system right away. You'll do more troubleshooting later. So here is your assignment.

1. Write a bug report that describes this bug. Include only three sections:
 - Problem Summary;
 - Problem Description;
 - Steps to Reproduce.
2. What other tests should you run? Why? Write down your list.
3. Meet with your group to read each other's reports.
 - How good is the summary?
 - How clear are the description and steps to reproduce?
 - How complete are the description and steps to reproduce?
 - How accurate are the description and steps to reproduce?

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Sample Exercise Results

Problem Summary:

GPF when the search process prompts the "Continue search from end of register?" message box.

Problem Description:

When you search for a transaction, if the search does not get a hit and then Quicken prompts you with the message box "Continue search from end of register?", a GPF will occur.

Note:

This error seems to be system dependent. A GPF is only produced when I run Quicken using station #1. I run the same test case on station #2 and it works fine. See detailed system configuration for station #1 and #2.

Steps to Reproduce:

1. Start the program.
2. Create a new account.
3. Produce X transactions, \$XXX each payable to "GPF".
4. Choose the Find command and set it up to Find "FPG", Search "All Fields", Match if "Contains", and check "Search Backwards."
5. Click the Find button.

Notice that when Quicken brings up the message box "Continue search from end of register?", you will get a GPF.

Sample Exercise Results (cont.)

Other tests to run

☞ Conditions:

- ☐ Search existing database (instead of creating a new one first), search with hit(s) before Quicken brings up the message box.
- ☐ Test with a few other static and dynamic environments or configurations.
- ☞ Search Methods: Set up other 'Search' and 'Match if' options, uncheck 'Search Backwards', etc.
- ☞ Related Actions: Find All (instead of Find)

Bug Reporting Exercise 2.1

Your assignment:

1. Think back to the previous Microsoft Paint bug reporting exercise
 - Are there other tests that you'd like to run?
 - Why?
 - What do you hope to find?
2. Meet with your group to discuss the results.

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Sample Exercise Results

Other tests to run

- ☞ Condition: Zooming variations
- ☞ Selection Methods: Freehand, Rectangular
- ☞ Related Actions: Cut, Copy, Paste, Clear, Undo, Repeat, Image menu commands (Flip/Rotate, Stretch/Skew, etc.)

Notes

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Bug Reporting Exercise 3

Your assignment:

1. Bug reports are your product. Therefore, you want to *Quality Check* your reports before submitting them.
 - Come up with a Top-Ten list of items for checking your bug reports before submitting them.
2. Meet with your group to discuss the results.

[illegible]

Sample Exercise Results

Top-Ten Quality Checklist

- ❑ 1. The summary is less than 15 words.
- ❑ 2. The report describes no more than one error.
- ❑ 3. Steps to Reproduce are included.
- ❑ 4. Steps to Reproduce are simplified
(minimum number of steps).
- ❑ 5. Steps to Reproduce are verified.
- ❑ 6. Easy-to-understand and non-judgmental
language used.
- ❑ 7. The configuration and operating condition
information is included.
- ❑ 8. Verified steps with a second configuration.
- ❑ 9. Verified steps with the previous release.
- ❑ 10. A sample file is attached only when necessary.

Notes

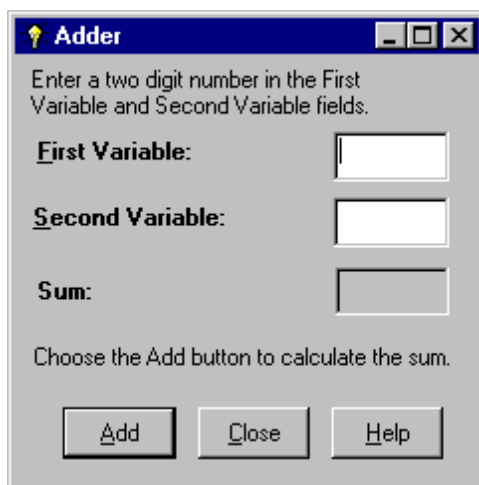
This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Analyzing an Error Exercise 1

(1)

Steps:

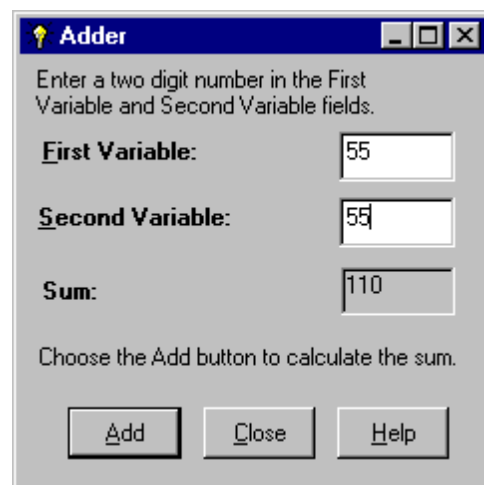
1. Launch the Adder application



2. Enter 55 as the First Variable

3. Enter 55 as the Second Variable

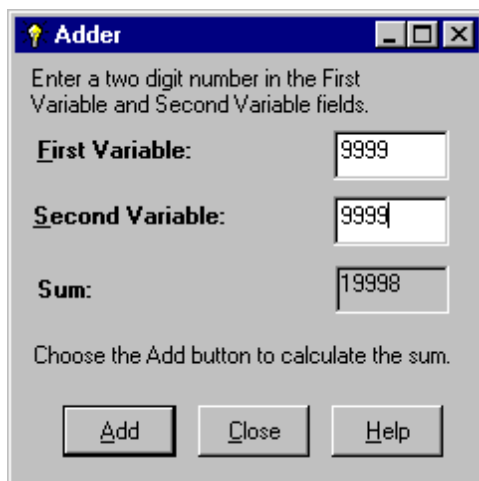
4. Choose Add



5. Enter 9999 as the First Variable

6. Enter 9999 as the Second Variable

7. Choose Add



8. Choose the Help button



Analyzing an Error Exercise 1 (2)

Steps:

9. Click OK to close the About box
10. Choose the Maximize button
11. Enter 99999 as the First Variable
12. Enter 99999 as the Second Variable
13. Choose Add

The screenshot shows the 'Adder' application window. The title bar is blue with a yellow lightning bolt icon and the text 'Adder'. The main area is gray and contains the following text and controls:

Enter a two digit number in the First Variable and Second Variable fields.

First Variable:

Second Variable:

Sum:

Choose the Add button to calculate the sum.

At the bottom, there are three buttons: 'Add', 'Close', and 'Help'. The 'Help' button is highlighted with a dashed border.



Analyzing an Error Exercise 1

(3)

Your assignment is to analyze the steps and answer the following questions:

1. How many (obvious) bugs do you see throughout the steps? Describe them (assuming that you've done your analysis) by writing the summary line for each.
2. If there is more than one bug, are they related? Why? What would you do to validate your theory?
3. If you have ONLY one shot at narrowing down the number of steps-to-reproduce to a minimum set, which set do you think is the most promising for each potential bug? Why? List your set(s).

Exercise Results

[illegible]

Notes

[illegible]

Sample Exercise Results

1. At a minimum, there are 2 obvious bugs:

- ❑ The main dialog box is resizable--UI error.
- ❑ "Run-time error '6' Overflow" when adding 99999 to 99999.

A few other bugs:

- ❑ Help button does not bring up Help.
- ❑ Maximize button is still active while the dialog box has already been maximized.
- ❑ No error detection/handling for >2-digit numbers.
- ❑ No Clear button (and perhaps, reset the cursor to the First Variable input field).

2. Validating your theory. Split the report if necessary:

- ❑ They're probably not related because one seems to be an UI error and the other seems to be a variable overflow error. However, to know for sure we should test our theory.
- ❑ Independently run the set of steps that exposes each error.

3. Simplifying steps-to-reproduce

- ❑ For the UI error, try steps 1 and 10
- ❑ For the "Run-time error '6' Overflow" error, try steps 1, 11, 12 and 13.

Analyzing an Error Exercise 1.1

Your assignment:

1. Assuming the information in the Exercise Results is correct, what else would you do to strengthen your reports (show that they are more serious and/or more general)?
2. What do you think is special or unique about the value 99999? What other tests would you run to check valid and/or invalid values?

Hints:

- The storage size of an integer is 2 bytes or the range is -32,768 to 32,767.
- Think in terms of boundary conditions.

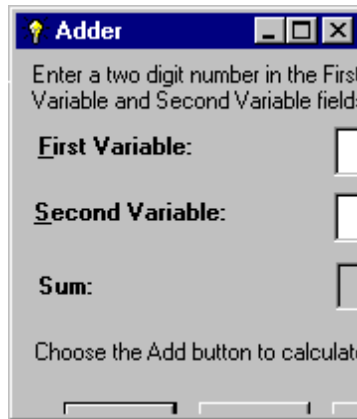
Notes

[illegible]

Sample Exercise Results

1. Analyzing an error. Strengthen the report:

- ❑ UI error: Think in terms of the opposite extreme. Try to resize window to a smaller size.



- ❑ "Run-time error '6' Overflow": Think in terms of boundary conditions. Try 89,999; 79,999; 69,999; etc. Also try -99,999; -89,999; -79,999; etc. Ask yourself what's the smallest positive and/or negative number that causes the overflow? Try to determine the boundaries.

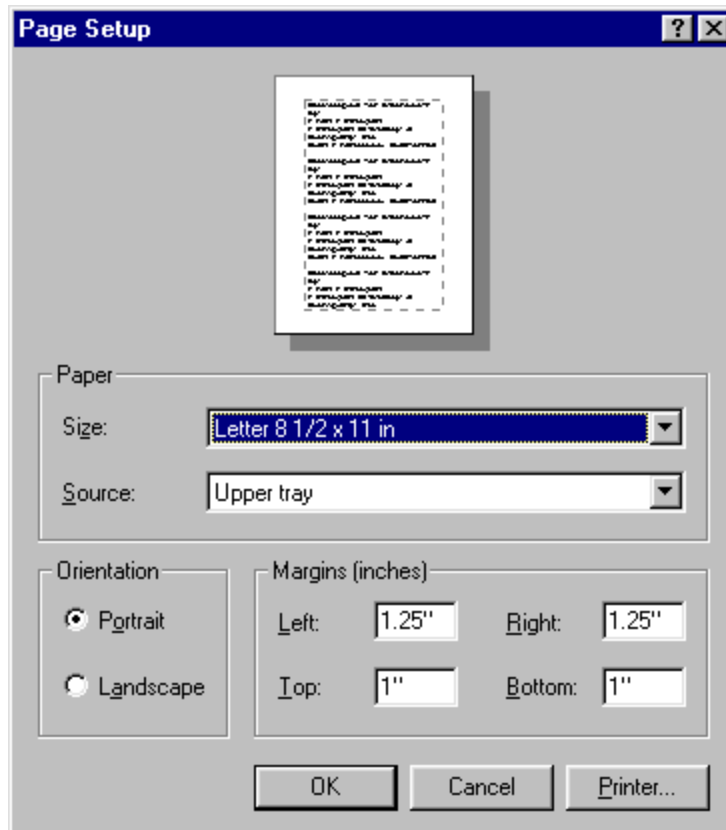
2. Boundary condition tests

- ❑ Consider specification's vs. exploratory boundaries. Consider Error Handling tests.
- ❑ Try 32,767; $32,767 + 1$; -32,768; $-32,768 + (-1)$.
- ❑ Try any set of 2 variables that yields the following sums: 32,767; $32,767 + 1$; -32,768; $-32,768 + (-1)$.

Notes

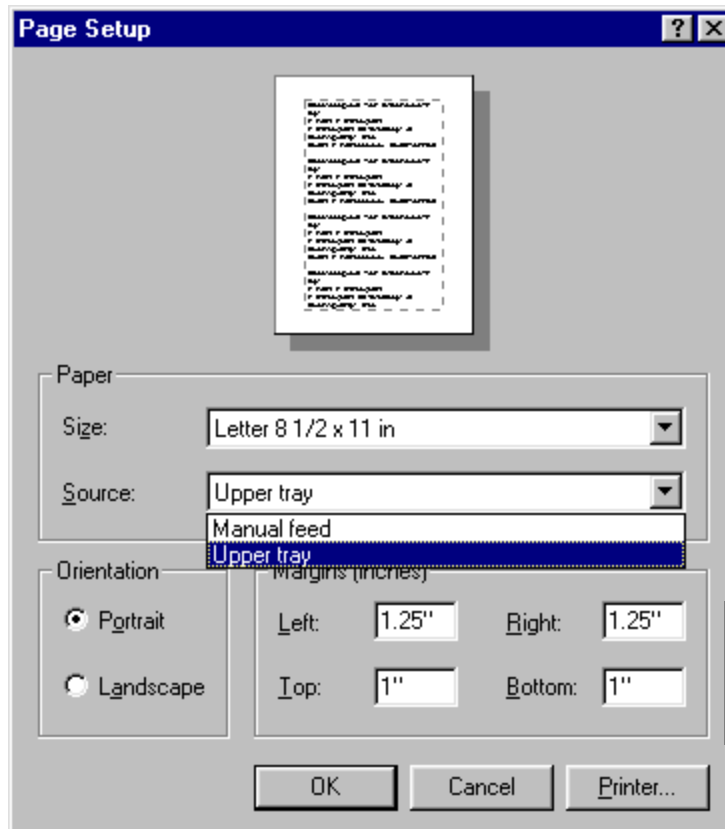
[illegible]

Table/Matrix Exercise (1)



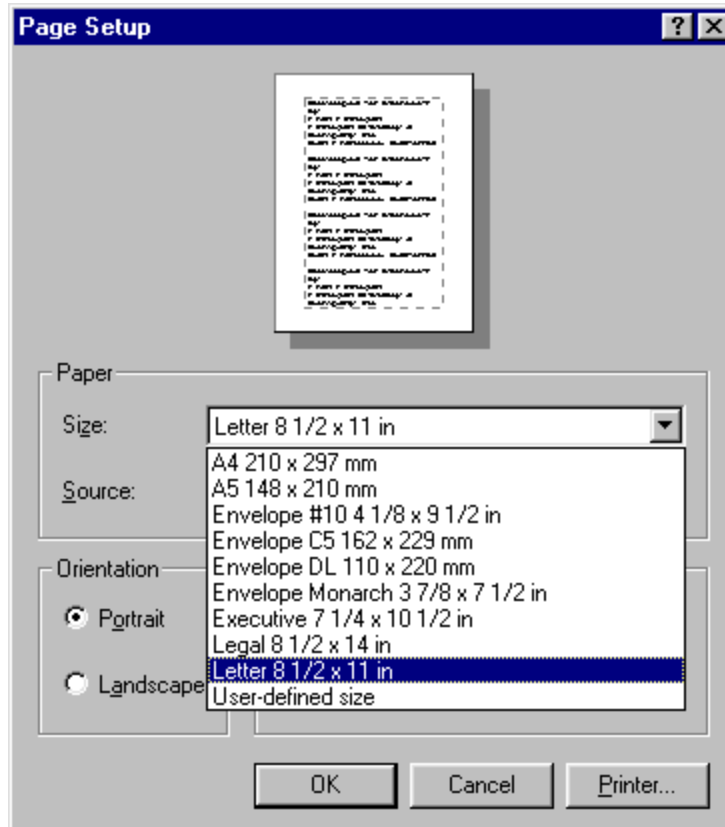
- Here is a Page Setup dialog box. It takes seven inputs:
 - (Paper) Size
 - (Paper) Source
 - Orientation
 - (Margins) Left
 - (Margins) Right
 - (Margins) Top
 - (Margins) Bottom

Table/Matrix Exercise (2)



- For the purpose of simplifying this exercise, let's ignore all Margins variables and consider only three inputs:
 - (Paper) Size: Letter, A4, A5 and Legal
 - (Paper) Source: Manual feed and Upper tray
 - Orientation: Portrait and Landscape

Table/Matrix Exercise (3)



Note:

Let's further simplify this exercise by considering only four values for the (Paper) Size: Letter, A4, A5 and Legal.

Table/Matrix Exercise

(4)

Your assignment:

- 1 Analyze the three input variables and their values to come up with all possible input combinations. Create a table and use it to list all input combinations.
- 2 For each Page Setup input combination, you'll do a print test by sending a sample output to the following printers: HP LaserJet 4, HP LaserJet 5, Cannon PS, Panasonic PS and Epson PS. Create a matrix to plan for this set of tasks.

Table/Matrix 1

Additional Instructions:

[illegible]

Table/Matrix 2

Additional Instructions:

[illegible]

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Notes

[illegible]

Sample Exercise Results

(1)

Size:

- 0: Letter
- 1: A4
- 2: A5
- 3: Legal

Source:

- 0: Upper tray
- 1: Manual

Orientation:

- 0: Portrait
- 1: Landscape

	PROPERTY															
CONTROL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Size	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
Source	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
Orientation	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

	PROPERTY															
CONTROL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Size	Letter	Letter	Letter	Letter	A4	A4	A4	A4	A5	A5	A5	A5	Legal	Legal	Legal	Legal
Source	Upper	Upper	Manual	Manual	Upper	Upper	Manual	Manual	Upper	Upper	Manual	Manual	Upper	Upper	Manual	Manual
Orientation	Portrait	Landscape	Portrait	Landscape	Portrait	Landscape	Portrait	Landscape	Portrait	Landscape	Portrait	Landscape	Portrait	Landscape	Portrait	Landscape

- **There are 16 combinations altogether.**

Sample Exercise Results

(2)

Size:

- 0: Letter
- 1: A4
- 2: A5
- 3: Legal

Source:

- 0: Upper tray
- 1: Manual

Orientation:

- 0: Portrait
- 1: Landscape

PROPERTY																
CONTROL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Size	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
Source	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
Orientation	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
PRINTER																
HP LaserJet 4																
HP LaserJet 5																
Cannon PS																
Panasonic PS																
Epson PS																

PROPERTY																
CONTROL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Size	Letter	Letter	Letter	Letter	A4	A4	A4	A4	A5	A5	A5	A5	Legal	Legal	Legal	Legal
Source	Upper	Upper	Manual	Manual	Upper	Upper	Manual	Manual	Upper	Upper	Manual	Manual	Upper	Upper	Manual	Manual
Orientation	Portrait	Landscape	Portrait	Landscape	Portrait	Landscape	Portrait	Landscape	Portrait	Landscape	Portrait	Landscape	Portrait	Landscape	Portrait	Landscape
PRINTER																
HP LaserJet 4																
HP LaserJet 5																
Cannon PS																
Panasonic PS																
Epson PS																

- Page Setup output test matrix

Notes

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Brainstorm on Equivalence Classes and Boundaries

The classic boundary analysis involves a numeric data entry field. Entries below a minimum or above a maximum are invalid. There are many other types of boundaries. For example, a graph could be too wide or a delay could be too long.

Exercise: For the next 15 minutes, work in your group to create the following:

1. List several different types of equivalence classes.
2. For each equivalence class that you listed, try to list its boundaries.
3. For each class listed, list at least one value that falls outside the class boundary (e.g. 100 is outside of the class -99 to 99, IBM ProPrinter is outside of the class of HP LaserJet compatible printers.)
4. For each equivalence class that you listed, try to list a corresponding “invalid” class.
5. Optional--Try to describe a bug that you’ve seen occur at each type of boundary.

Brainstorm Results

[illegible]

Notes

[illegible]

Equivalence Class and Boundary Brainstorm

There are many types of variables, including input variables, output variables, internal variables, hardware and system software configurations, and equipment states. Any of these can be subject to equivalence class analysis. Here are some common results from the class brainstorm:

- ranges of numbers
- character codes
- how many times something is done
 - (e.g. shareware limits on the number of uses of the software)
 - (e.g. how many times you can do it before you run out of memory)
- how many records in a database, how many names in a mailing list, how many variables in a spreadsheet, how many bookmarks, how many abbreviations
- size of the sum of variables, or the size of some other computed value (think binary and think digits)
- size of a number that you enter (number of digits) or size of a character string
- size of a concatenated string
- size of a path specification
- size of a file name
- size (in characters) of a document
- size of a file (note special values such as exactly 64K, exactly 512 bytes, etc.)
- size of a document on a page, in terms of the memory requirements for the page. This might just be in terms of resolution x page size, but it may be more complex if we have compression algorithms
- size of the document on the page (compared to page margins) (across different page margins, page sizes)
- equivalent output events (such as printing documents)
- amount of available memory (> 128 meg, > 640K, etc.)
- visual resolution, size of screen, number of colors
- operating system version
- variations within a group of “compatible” printers, sound cards, modems, etc.
- equivalent event times (when something happens)
- timing: how long between event A and event B (and in which order)
- length of time after a timeout (from JUST before to way after) -- what events are important?
- speed of data entry (time between keystrokes, menus, etc.)
- speed of input -- handling of concurrent events
- number of devices connected / active
- system resources consumed / available (also, handles, stack space, etc.)
- date (year 2000-related boundaries) and time (23:59; end of week, end of month)
- transitions between algorithms (optimizations) (different ways to compute a function)
- input or output intensity (voltage)
- speed / extent of voltage transition (e.g. from very soft to very loud sound)

Notes

[illegible]

Standard Test Cases

Here are a few examples of other areas that involve repetitive testing. Create lists of standard tests that you'd run for each.

- Launching an application
- Navigating around the window, using the keyboard and the mouse
- Printing an object
- Reading / inserting different types of objects
- Sorting
- Date handling
- Cut, paste, drag and drop, edit, stretch, and similar operations on graphical objects
- Cut, paste, drag and drop, edit, stretch, and similar operations on text objects
- Apply / manipulate styles (all types of text objects)
- Installing/Uninstalling

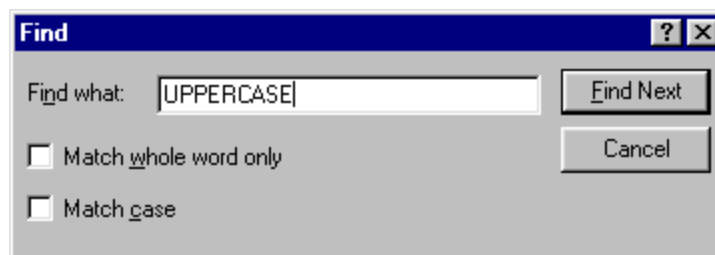
[illegible]

Notes

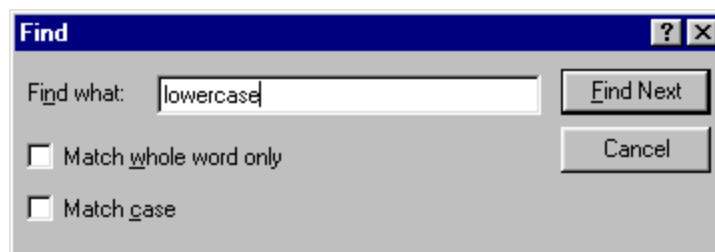
[illegible]

Combinatorial Design Exercise (1)

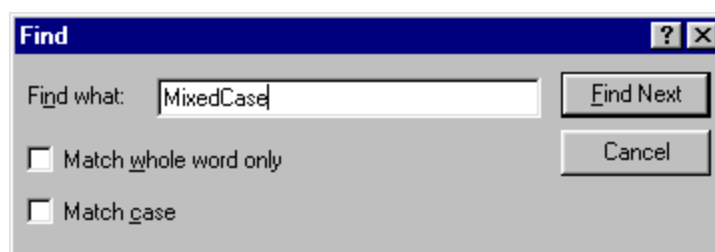
- Here is a simple Find dialog box. It takes three inputs:
 - Find what: a text string (“UPPERCASE”, “lowercase” and “MixedCase”)
 - Match whole word only: ON or OFF
 - Match case: ON or OFF
- Simplify this by considering only three values for the text string, “UPPERCASE” and “lowercase” and “MixedCase”. Let’s combine the inputs.



A screenshot of a standard Windows-style 'Find' dialog box. The title bar is blue with the word 'Find' and standard window controls. The main area has a light gray background. The 'Find what:' label is followed by a text box containing 'UPPERCASE'. To the right are 'Find Next' and 'Cancel' buttons. Below the text box are two checkboxes: 'Match whole word only' and 'Match case', both of which are unchecked.



A screenshot of a 'Find' dialog box, identical in layout to the first one. The text box for 'Find what:' contains the word 'lowercase'. The 'Match whole word only' and 'Match case' checkboxes remain unchecked.



A screenshot of a 'Find' dialog box, identical in layout to the previous two. The text box for 'Find what:' contains the text 'MixedCase'. The 'Match whole word only' and 'Match case' checkboxes remain unchecked.

Combinatorial Design Exercise (2)

CONTROL	PROPERTY											
	1	2	3	4	5	6	7	8	9	10	11	12
Match case	OFF	OFF	ON	ON	OFF	OFF	ON	ON	OFF	OFF	ON	ON
Match whole	OFF	ON	OFF	ON	OFF	ON	OFF	ON	OFF	ON	OFF	ON
Find What	"UPPERCASE"	"UPPERCASE"	"UPPERCASE"	"UPPERCASE"	"lowercase"	"lowercase"	"lowercase"	"lowercase"	"MixedCase"	"MixedCase"	"MixedCase"	"MixedCase"

- There are 12 combinations altogether.

Your assignment:

- Create combination tests that cover all possible pairs of inputs, but don't try to cover all possible triplets. List one such set.

[illegible]

Notes

[illegible]

Sample Exercise Results

(1)

CONTROL	PROPERTY											
	1	2	3	4	5	6	7	8	9	10	11	12
Match case	OFF	OFF	ON	ON	OFF	OFF	ON	ON	OFF	OFF	ON	ON
Match whole	OFF	ON	OFF	ON	OFF	ON	OFF	ON	OFF	ON	OFF	ON
Find What	"UPPERCASE"	"UPPERCASE"	"UPPERCASE"	"UPPERCASE"	"lowercase"	"lowercase"	"lowercase"	"lowercase"	"MixedCase"	"MixedCase"	"MixedCase"	"MixedCase"

- There are 12 combinations altogether.

CONTROL	PROPERTY											
	1			4		6	7			10	11	
Match case	OFF			ON		OFF	ON			OFF	ON	
Match whole	OFF			ON		ON	OFF			ON	OFF	
Find What	"UPPERCASE"			"UPPERCASE"		"lowercase"	"lowercase"			"MixedCase"	"MixedCase"	

- There are 6 pair-wise combinations altogether.

Sample Exercise Results

(2)

	PROPERTY										
CONTROL	1			4		6	7			10	11
Match case	OFF			ON		OFF	ON			OFF	ON
Match whole	OFF			ON		ON	OFF			ON	OFF
Find What	"UPPERCASE"			"UPPERCASE"		"lowercase"	"lowercase"			"MixedCase"	"MixedCase"

1

4

6

Sample Exercise Results

(3)

CONTROL	PROPERTY									
	1			4		6	7		10	11
Match case	OFF			ON		OFF	ON		OFF	ON
Match whole	OFF			ON		ON	OFF		ON	OFF
Find What	"UPPERCASE"			"UPPERCASE"		"lowercase"	"lowercase"		"MixedCase"	"MixedCase"

7

Find [?] [X]

Find what:

☐ Match whole word only

☒ Match case

[Find Next] [Cancel]

10

Find [?] [X]

Find what:

☒ Match whole word only

☐ Match case

[Find Next] [Cancel]

11

Find [?] [X]

Find what:

☐ Match whole word only

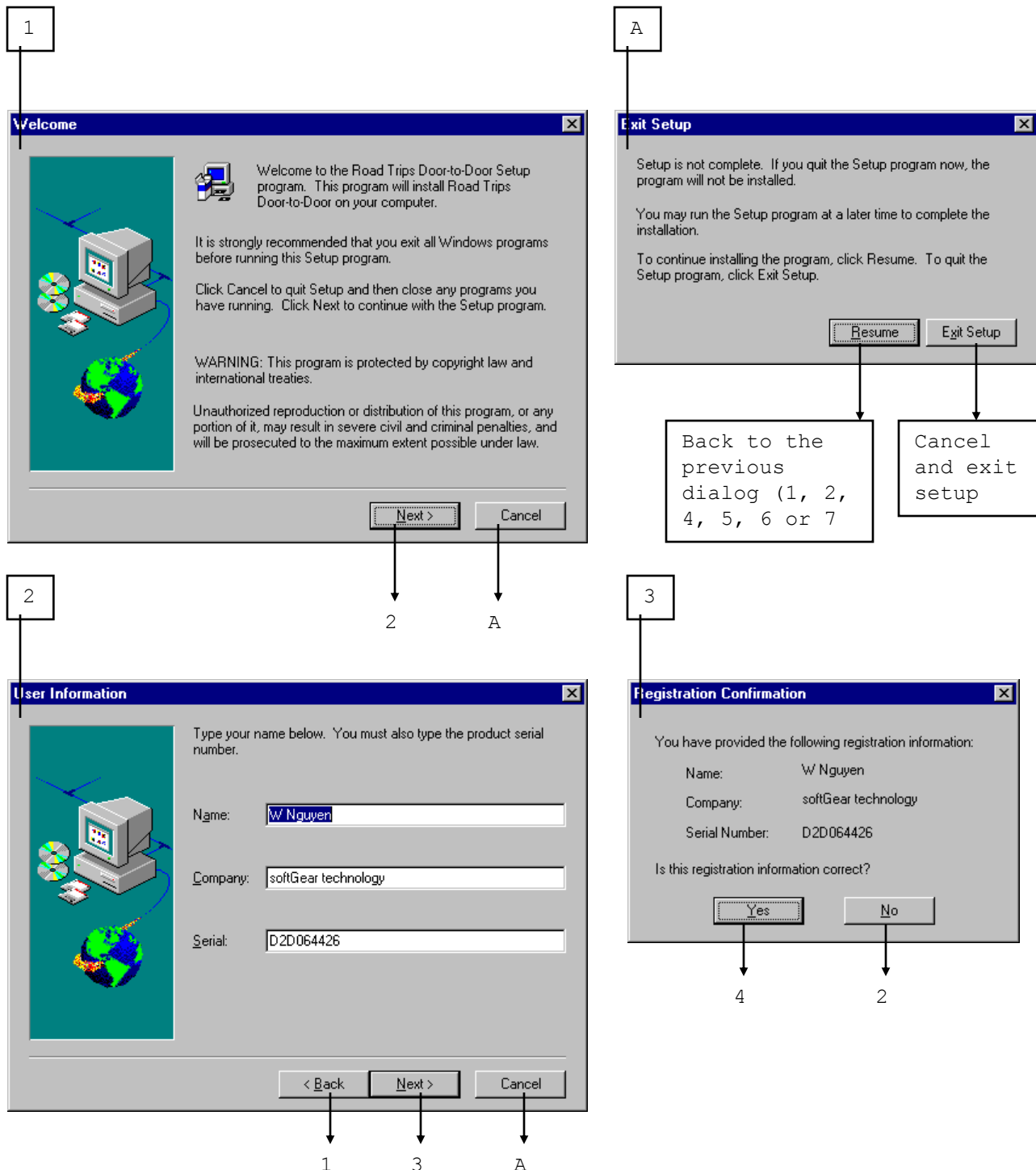
☒ Match case

[Find Next] [Cancel]

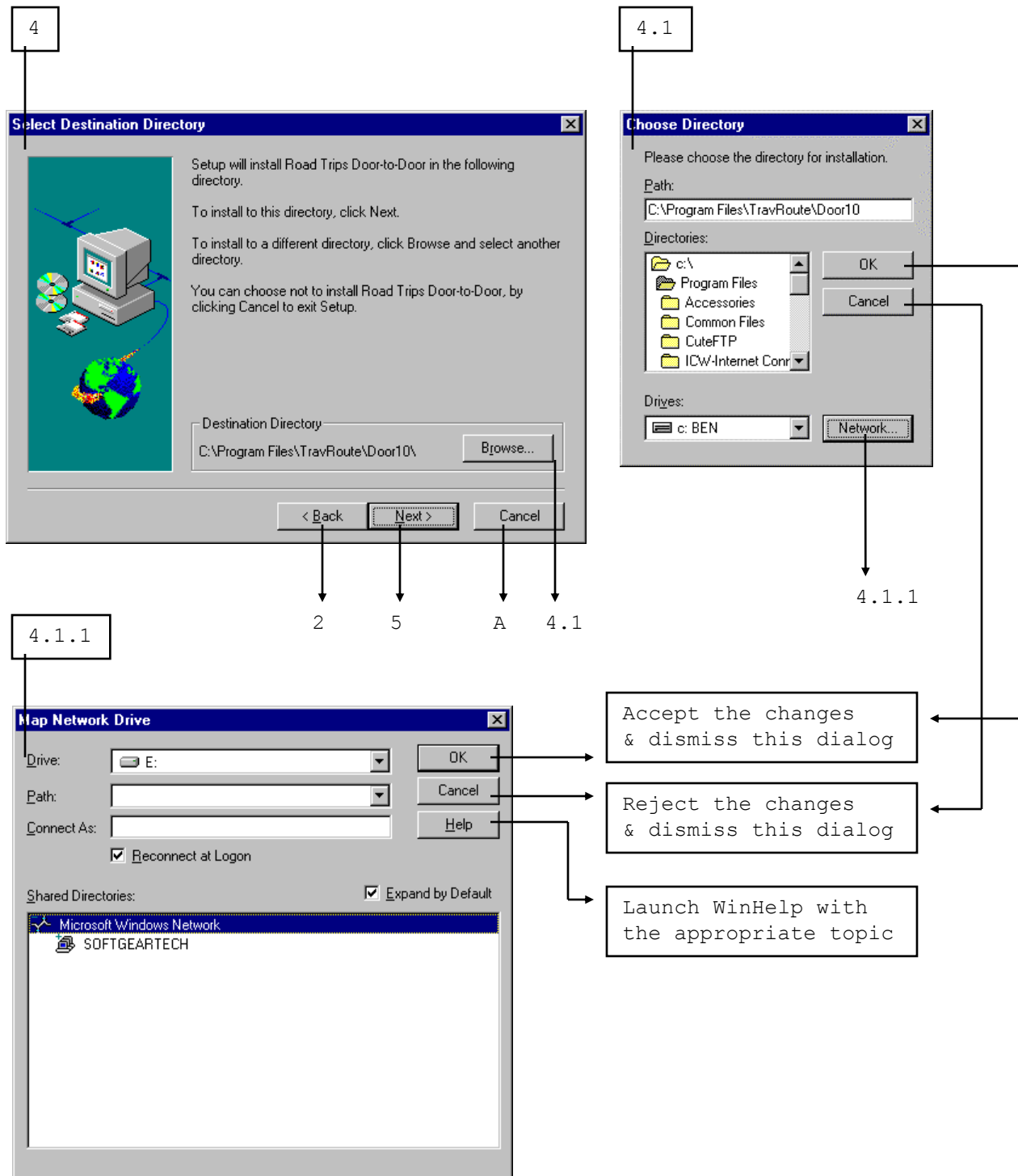
Notes

[illegible]

Installer Exercise (1)



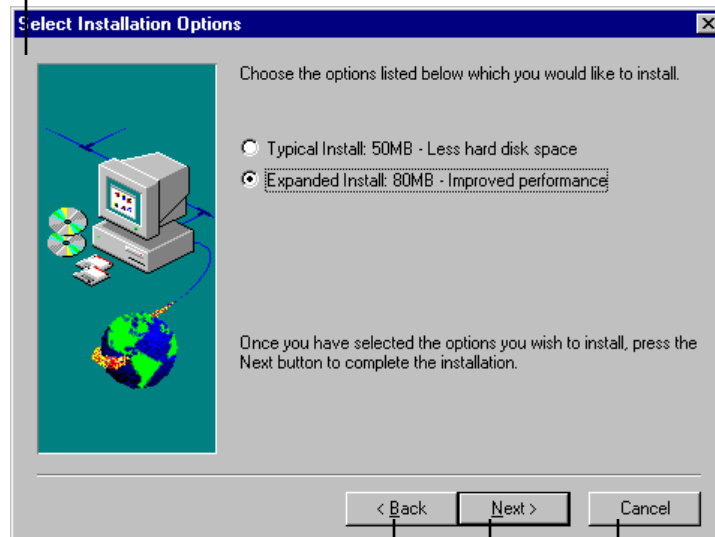
Installer Exercise (2)



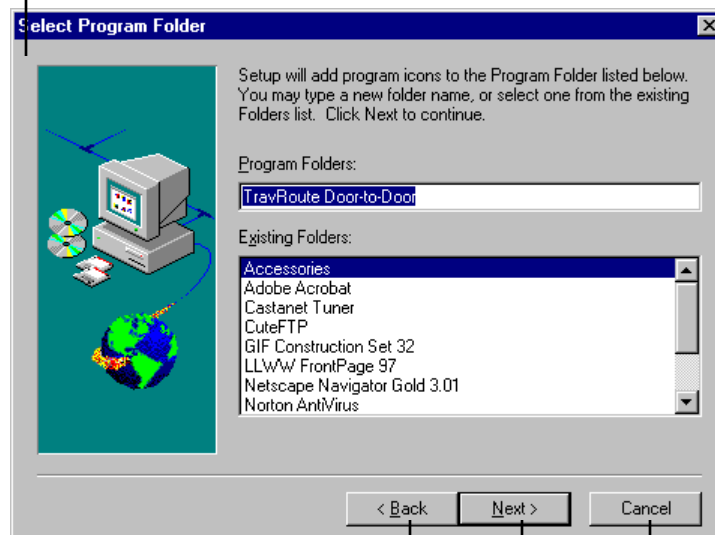
Installer Exercise

(3)

5



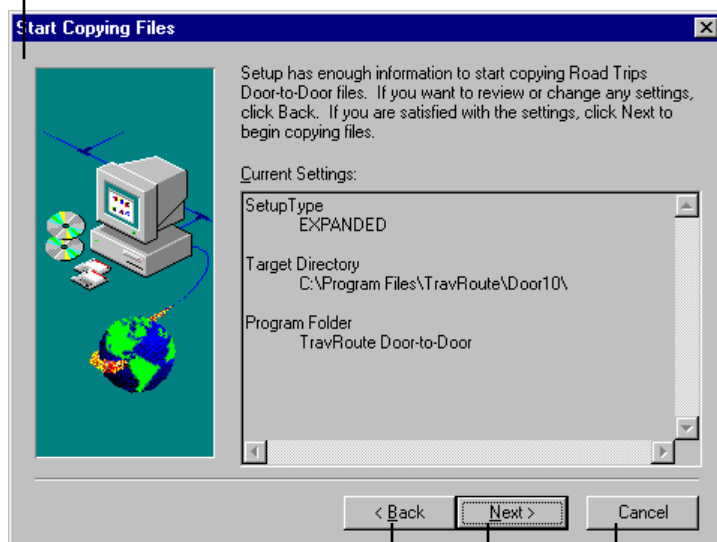
6



Installer Exercise

(4)

7

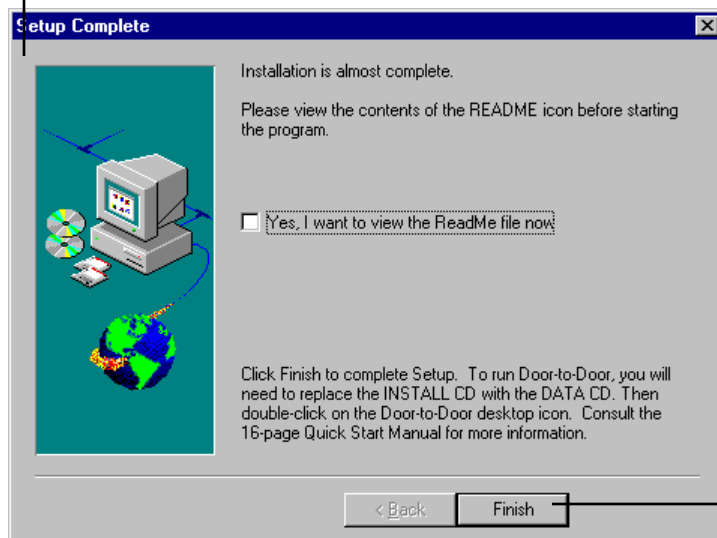


8

6

8

A



Setup complete

Installer Exercise

Here is your assignment.

- 1 Work with your group to generate and list as many test cases and conditions for your installation testing as possible. Prepare to discuss your findings in class.
- 2 Create a one-page draft of your Installation Test Plan and prepare to share your information in class.
 - Ask questions if you don't think you have adequate information.
 - Think in terms of how you would derive a test schedule (i.e., tasks, coverage, resource, stability cycle, etc.)
- 3 Your Project Manager asks you to cut all testing efforts in half across the board, what would you do? (Optional)

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Notes

[illegible]

Test for GUI Behavior -- Tab order and accelerator keys

Tab order

- Tab
- Shift-Tab

Accelerator keys

- Is it functional?
- Is the character used conventional?
 - Is a standard character used for a standard command?
 - For example, F for File, O for Open
 - Is first character of the command used?
- Is there a conflict?
- If there is a conflict,
 - Does the most commonly used command get the first character assigned?
 - If a command does not have the first character assigned, does the design consider the following alternative guidelines:
 - Use a symbolic character? For Example, Ctrl-X for Cut.
 - Use a phonetic character? For Example, X for Exit.
 - One-hand interface if possible?

Test for GUI Behavior -- Issues with The X button

Conventionally, choose the X button means

- CLOSE the current Windows (or dialog box) or
- CLOSE the current Windows (or dialog box) and CANCEL the current operation

Current functionality of the push buttons

- Next = Go to the next dialog and CLOSE this dialog
- Back = Go to the previous dialog and CLOSE this dialog
- Cancel = Cancel the current operation and CLOSE this dialog
- Resume = Resume the current application and CLOSE this dialog
- Exit Setup = Exit the Setup program and CLOSE this dialog
- Yes = Yes to the question and CLOSE this dialog
- No = No to the question and CLOSE this dialog
- Finish = Finish the installation and CLOSE this dialog

Problems

- What is the functionality of the X button in [A]
 - CANCEL the previous Cancel (Resume) or CANCEL the Installation altogether (Exit Setup) and then CLOSE?
- What is the functionality of the X button in [8]
 - CANCEL the installation or Finish the Installation and then CLOSE?

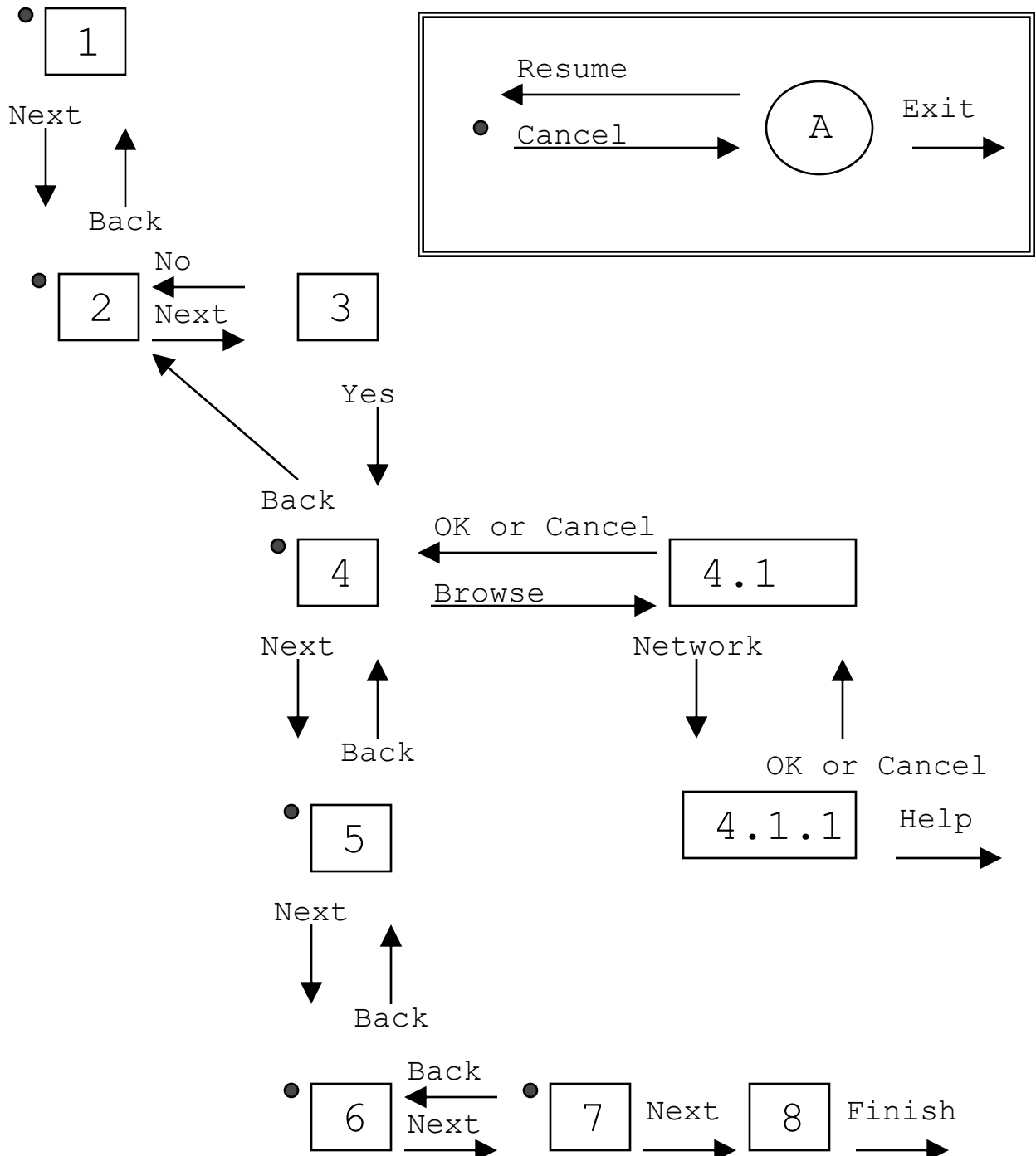
Test for GUI Behavior -- Other considerations

- Check the common function keys such as ESC, ENTER, F1, Shift-F1, WINDOWS, etc.
- Which push button is a DEFAULT button? Is it appropriate?
- Does update information to the dialog boxes get saved and refreshed on the UI?
- Consider executing test cases in the Keyboard Matrix, Mouse Matrix and Input Validation Matrix.

Functionality Testing

- Look for user-level logic errors. For example, run the installation by following the onscreen instructions or user-guide's instruction to check for software and documentation mismatch.
- It is not unusual for the installer to obtain path information from the wrong place, thus installing shared files in the wrong place or updating its registry keys with wrong information.
- Default directory for NT 3.51 and NT4.0 are not the same.
- Test with Full and Typical options.
- Consider minimum and maximum hardware configurations.
- Test with CLEAN configuration.
- Test with various installation paths (see next page).
- See the Microsoft Win95 NT Logo Guidelines document and VeriTest Top Ten Reasons Applications Fail for more ideas. The document can be downloaded from Microsoft at Microsoft.com and VeriTest.Com respectively.

Path Testing



Other Testing to Consider

- Registry issues
- Error detection and handling
- Configuration and compatibility issues

Notes

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

About *Logi*Gear Corporation

*Logi*Gear Corporation provides testing expertise and resources to software development organizations. Our partners benefit from our seasoned testing staff and facilities, practical training programs, and test support products. We help development teams deliver high quality software, improve time-to-market, and optimize development productivity.

Founded in 1994 as *soft*Gear technology, *Logi*Gear has built a reputation on partnering with software development organizations to help make the most of outsourcing and staff training solutions. We assist our clients in delivering the best possible quality products while juggling limited resources and schedule constraints.

About Michael Hackett

MICHAEL HACKETT, Director, Training and Publications, has over a decade of experience in software engineering and the testing of shrink-wrap and Internet based applications. He has developed for Windows, Macintosh and UNIX operating systems. Michael has helped well-known companies including Palm Computing, Electronics for Imaging, Adobe Systems, CNET, The Learning Company, Power Up Software, Oracle, PC World and The Well produce, test and release applications ranging from business productivity to educational multimedia titles in English as well as a multitude of other languages.

Michael is a founding partner of LogiGear Corporation. Prior to joining LogiGear, he served as Director of Quality Assurance at The Well, an online service that is renowned for its electronic conferencing system. Michael has developed professional training courses dealing in engineering, business communication and computer training. He has also written many instructional manuals used by professional trainers. Michael holds a Bachelor of Science in Engineering from Carnegie-Mellon University. michaelh@logigear.com