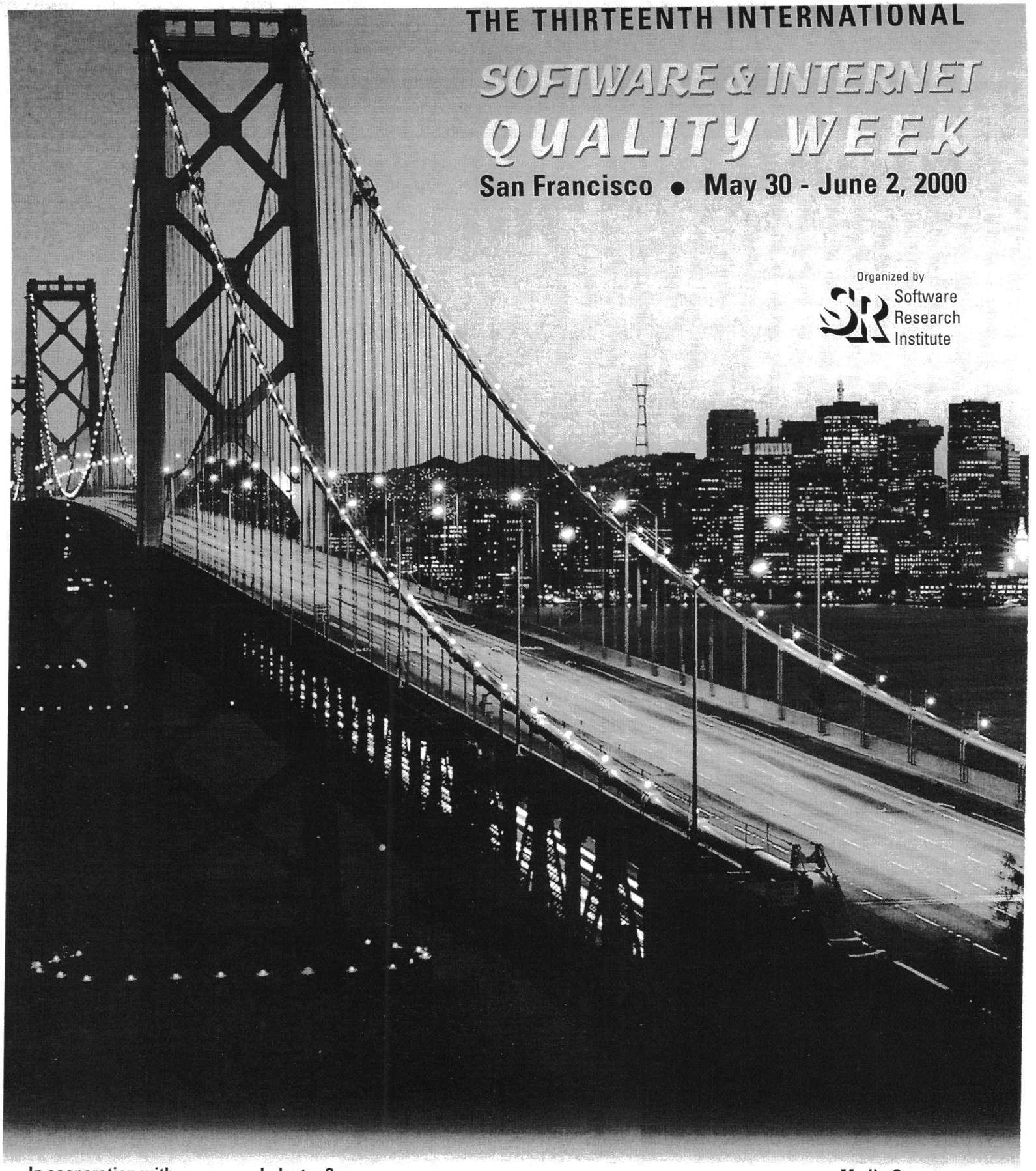


In cooperation with

acm

American Society for Quality
ASQ
Software Division

Industry Sponsors

COMPUWARE.

Rational
the e-development company™

Valid
Your e-Business Partner

SOFTWARE RESEARCH

Microsoft
Where do you want to go today?

VANTEON

NASXXCAL
TESTING LABORATORIES

ZD LABS

Media Sponsors

SOFTWARE
Development

CUTTER
CONSORTIUM

Software
BUSINESS

Friday, June 2, 2000
QW 2000 Speaker Evaluation

Workshops

❑ W1 ❑ W2 ❑ W3 ❑ W4

Speaker:_____

Overall rating of presentation

(5 = excellent, 1 = poor):

5	4	3	2	1
❑	❑	❑	❑	❑

Comments:_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Please return this form to the registration desk by 10:30 AM, Friday June 2, at the very latest! Thank you!

WORKSHOP NOTES

Thirteenth International Software & Internet Quality Week 2000

30 May 2000 – 2 June 2000

Hyatt Regency Embarcadero
San Francisco, California USA

Software Research Institute
901 Minnesota Street
San Francisco, CA 94107 USA

Phone: (415) 550-3020 – FAX: (415) 550-3030 – E-mail: qw@soft.com

Session W1

Oracle Strategies for Automated Testing

(Workshop)

Mr. Douglas Hoffmann
Software Quality Methods LLC

Oracle Strategies for Automated Testing

Quality Week 2000

Douglas Hoffman
Software Quality Methods, LLC.
24646 Heather Heights Place
Saratoga, California 95070-9710
Phone 408-741-4830
Fax 408-867-4550
doug.hoffman@acm.org

Describe Your Typical Automated Test

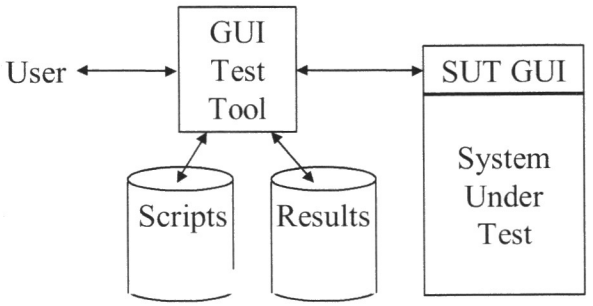
- What is being tested
- How is the test set up
- Where are the inputs coming from
- What is being checked
- Where are the expected results
- How do you know pass or fail

Your Typical Automated Test

Regression Strategy

- Write the test exercise
- Run it
- Capture the results
- "Verify" the results
- Run it again as a regression test
- Compare current with captured results

A Regression Test Model



Regression Advantages

- Straightforward
- Same approach for all tests
- Fast implementation
- Variations easy
- Repeatable tests

Regression Disadvantages

- "20 Questions"
- Master validation
- Analysis of failures
- Limited scope

There's A Universe Beyond Regression Automation

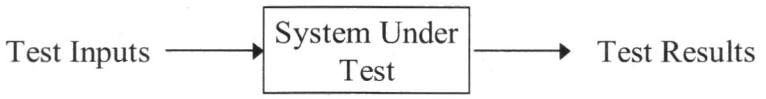
Automated Software Tests

- No intervention needed after launching tests
- Automatically sets-up and/or records relevant test environment
- Runs test exercise
- Captures relevant results
- Evaluates actual against expected results
- Reports analysis of pass/fail

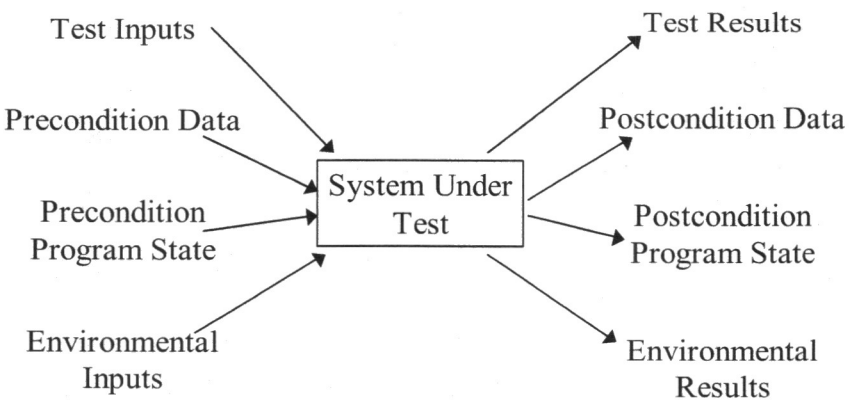
Levels of Automation

- Fully automated software testing
- Semi-automated software testing
- Manual software testing

Simple Testing Model
(Black Box)



Expanded Testing Model
(Black Box)



Implications of the Expanded Model

- We don't control all inputs
- We don't verify everything
- Multiple domains are involved
- Test exercise may be the easy part
- We can't verify everything
- We don't know all the factors

Size Of The Testing Problem

- Input one value in a 10 character field
- 26 UC, 26 LC, 10 Numbers
- Gives 62^{10} combinations
- How long at 1,000,000 per second?

What is your domain size?

We can only run a

vanishingly small portion

of the possible tests

Choosing The Subset

- High value tests
- Exploratory testing
- Automated tests
- More powerful exercises

More Powerful Exercises

- Increasing the number of combinations
- Self-verifying tests and diagnostics
- More frequency, intensity, duration
- Increasing the variety in exercises

Pseudo Random Numbers

- Used for selection or construction of inputs
  - With and without weighting factors
  - Selection with and without replacement
- Statistically "random" sequence
- Randomly generated "seed" value
- Requires oracles to be useful

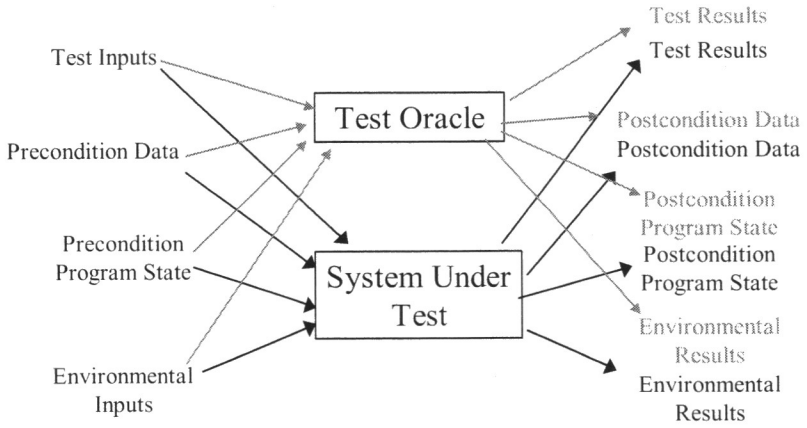
Random Selection Among Alternatives

- Pseudo random numbers
- Partial domain coverage
- Small number of combinations
- Use an oracle for verification

Where Do We Fit In The Oracle?

- Identify what to verify
- How do we know the "right answer"
- How close to "right" do we need
- Decide when to generate the expected results
- Decide how and where to verify results
- Get the oracle

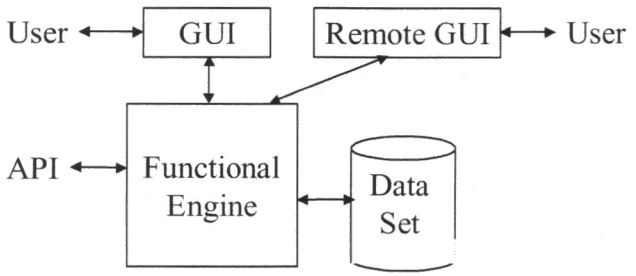
Testing With An Oracle



Automation Architecture

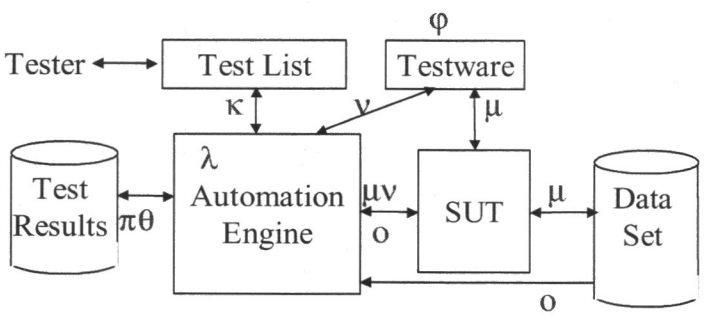
- Model for SUT and environment
- Break down software testing problem
- Decide on location(s) of automation
- Decide on level(s) of automation
- Describe automation architecture

A Model For SUT



System Under Test

Automated Software Testing Process Model



Process Model Example

1. Testware creation, version control, and configuration management
2. Selecting the subset of test cases to run
3. Set-up and/or record environmental variables
4. Run the test exercises
5. Monitor test activities
6. Capture relevant results
7. Compare actual with expected results
8. Report analysis of pass/fail

Oracle Characteristics

- Completeness of information from an oracle
- Accuracy of information from an oracle
- Usability of results
- Temporal relationships
- Supportability

Completeness

- Sufficiency
- Input Coverage
- Result Coverage
- SUT environments
- Types of errors possible

Accuracy

- How similar
  - Arithmetic accuracy
  - Statistically similar
- How independent
- Types of possible errors

Usability

- Form of information
- Location of information
- Availability of comparators
- Support in SUT environments
- Cost

Temporal Relationships

- How fast to generate results
- How fast to compare
- When is it run
- When are results compared

Supportability

- COTS or custom
- Correspondence through SUT changes
  - Test exercises
  - Tools
- Ancillary support activities required

Oracle Approaches

	True Oracle	Heuristic Strategy	Consistency	Self Referential	No Oracle
Definition	-Independent generation of all expected results	-Verifies some values, as well as consistency of remaining values	-Verifies current run results with a previous run (Regression Test)	-Embeds answer within data in the messages	-Doesn't check correctness of results, (only that some results were produced)
Advantages	-No encountered errors go undetected	-Faster and easier than True Oracle -Much less expensive to create and use	-Fastest method using an oracle -Verification is straightforward -Can generate and verify large amounts of data	-Allows extensive post-test analysis -Verification is based on message contents -Can generate and verify large amounts of complex data	-Can run any amount of data (limited only by the time the SUT takes)
Disadvantages	-Expensive to implement -Complex and often time-consuming when run	-Can miss systematic errors (as in *sine* wave example)	-Original run may include undetected errors	-Must define answers and generate messages to contain them	-Only spectacular failures are noticed.

True Oracle

- Independent implementation
- Complete coverage over domains
  - Input ranges
  - Result ranges
- "Correct" results
- Usually expensive

Heuristic Strategy

- Rules of thumb
  - Estimates
  - Approximations
  - Trends
- Levels of abstraction
  - General characteristics
  - Statistical properties

Consistency Strategy

- A / B compare
- Check for changes
- Regression checks
  - Validated
  - Unvalidated
- Alternate versions or platforms
- Foreign implementations

Self-Referential Strategy

- Embed results in the data
- Cyclic algorithms
- Shared keys with algorithms

'No Oracle' Strategy

- Easy to implement
- Tests run fast
- Only spectacular errors are noticed
- False sense of accomplishment

Choosing Which Strategy

- Decide how the oracle fits in
- Identify the oracle characteristics
- Prioritize testing risks
- Choose a combination of approaches

Heuristic Oracle Examples

- Live data base
  - selected records using specific criteria
  - selected characteristics for known records
  - standard characteristics for new records
- Data base engine
  - correlated field values (time, order number)
- Data communications
  - CRC
- Sine function

Self-Referential Oracle Examples

- Data base
  - embedded linkages
- Data communications
  - value patterns (start, increment, number of values)
- Noel Nyman's "Self Verifying Data"*

* "Self Verifying Data - Validating Test Results Without An Oracle," STAR East 1999

Mutating Automated Tests

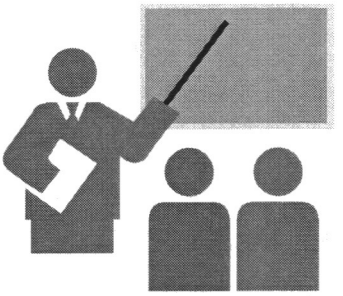
- Closely tied to instrumentation and oracles
- Using pseudo random numbers
- Positive and negative cases possible
- Diagnostic drill down on error

Mutating Tests Examples

- Data base contents (Embedded)
- Processor instruction sets (Consistency)
- Compiler language syntax (True)
- Stacking of data objects (None)

Summary

- Decide what you are testing
- Decide how to exercise it
- Decide what can tell pass from fail
- Analyze technical factors, costs, and risks
- Select oracle strategy

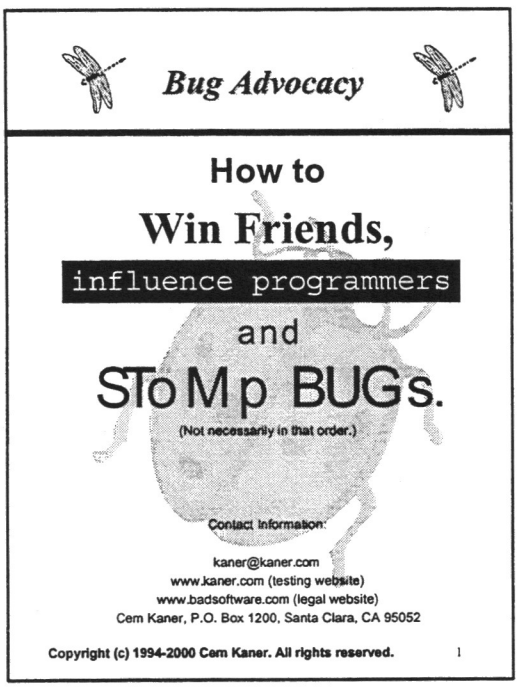


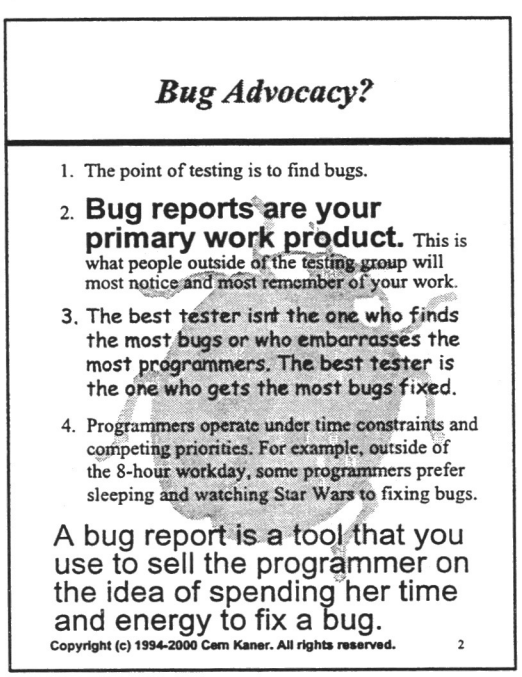
Session W2

Bug Advocacy

(Workshop)

Mr. Cem Kaner, Ph.D.
Attorney At Law





Selling Bugs

Time is in short supply. If you want to convince the programmer to spend his time fixing your bug, you may have to sell him on it.

(Your bug? How can it be your bug? The programmer made it, not you, right? It's the programmer's bug. Well, yes, but you found it so now it's yours *too.)*

Sales revolves around two fundamental objectives:

- **Motivate the buyer (*Make him WANT to fix the bug.)***
- **Overcome objections (*Get past his excuses and reasons for not fixing the bug.)***

 3

Motivating the Bug Fixer

Some things that will often make programmers want to fix the bug:

- It looks really bad.
- It will affect lots of people.
- Getting to it is trivially easy.
- It has embarrassed the company.
- One of its cousins embarrassed the company.
- Management (that is, someone with influence) has said that they really want it fixed.
- You've said that *you* want the bug fixed, and the programmer likes you, trusts your judgment, is susceptible to flattery from you, owes you a favor or accepted bribes from you.

 4

Overcoming Objections

Things that will make programmers resist spending their time on the bug:

- The programmer cant replicate the defect.
- Strange and complex set of steps required to induce the failure.
- Not enough information to know what steps are required, and it will take a lot of work to figure them out.
- The programmer doesnt understand the report.
- Unrealistic (e.g. "corner case")
- It will take a lot of work to fix the defect.
- A fix will introduce too much risk into the code.
- No perceived customer impact
- Unimportant (no one will care if this is wrong: minor error or unused feature.)
- Management doesnt care about bugs like this.
- The programmer doesnt like / trust you (or the customer who is complaining about the bug).

 5

Bug Advocacy

Motivating Bug Fixes

By Better Researching

The Failure Conditions

 6

Motivating The Bug Fix: Looking At The Failure

Some vocabulary

- An ***error*** (or ***fault***) is a design flaw or a deviation from a desired or intended state.
- An error won't yield a failure without the ***conditions*** that trigger it. Example, if the program yields 2+2=5 on the 10th time you use it, you won't see the error before or after the 10th use.
- The ***failure*** is the program's actual incorrect or missing behavior under the error-triggering conditions.
- ***Defect*** *is frequently used to refer to the failure or to the underlying error.*

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

 7

Motivating The Bug Fix: Looking At The Failure

VOCABULARY EXAMPLE

Here's a defective program

```
INPUT A
INPUT B
PRINT A/B
```

What is the fault?
What is the critical condition?
What will we see as the failure?

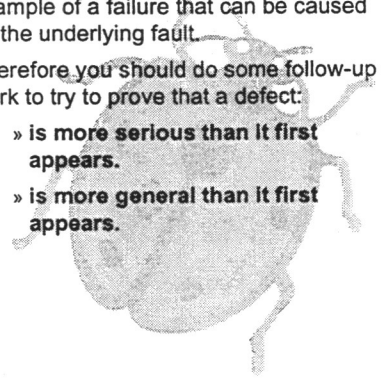
 8

Motivating the Bug Fix

When you run a test and find a failure, you may or may not have found the best example of a failure that can be caused by the underlying fault.

Therefore you should do some follow-up work to try to prove that a defect:

- **is more serious than it first appears.**
- **is more general than it first appears.**



 9

Motivating the Bug Fix: Make it More Serious

LOOK FOR FOLLOW-UP ERRORS

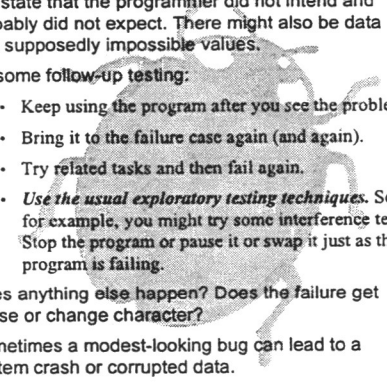
When you find a coding error, you have the program in a state that the programmer did not intend and probably did not expect. There might also be data with supposedly impossible values.

Do some follow-up testing:

- Keep using the program after you see the problem.
- Bring it to the failure case again (and again).
- Try related tasks and then fail again.
- ***Use the usual exploratory testing techniques.*** So, for example, you might try some interference tests. Stop the program or pause it or swap it just as the program is failing.

Does anything else happen? Does the failure get worse or change character?

Sometimes a modest-looking bug can lead to a system crash or corrupted data.



 10

Motivating the Bug Fix: Make it More Serious

LOOK FOR NASTIER CIRCUMSTANCES

This is just another form of follow-up testing, but I find it useful to think of it as a separate question.

Often when I do "follow-up" testing, I focus on the failure, and on the steps I'm taking to create the failure.

Here, I'm up to something slightly different. The failure steps are taken as given. What I am changing are program options and other things happening on the computer as background activity. For example, reproduce the bug:

- While the program is doing a background save. Does that cause data loss corruption along with this failure?
- With a different database
- With other options set
- With more (or fewer) device interrupts coming in

 11

Motivating the Bug Fix: Make it More Serious

LOOK FOR NASTIER CONFIGURATIONS

This is another form of follow-up testing, focusing on the hardware and environment rather than on steps and program settings.

Sometimes a bug will show itself as more serious if you run the program with less memory, a higher resolution printer, etc.

- *If there is anything involving timing, use a really slow computer, a really slow link, a really slow modem or printer. And use very fast ones.*
- *If there is a video problem, try higher resolutions on the video card. Try displaying MUCH more complex images (and much simpler ones).*

Note that we are not checking standard configurations here . We aren't asking how broad the range of circumstances is that produces the bug. Instead ***we're asking whether there is a particular configuration that will show the bug more spectacularly.***

 12

Motivating the Bug Fix: Make it More Serious

IS THIS BUG NEW TO THIS VERSION?

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.)
- The bug will be taken more seriously if it is new.
- You can argue that it should be treated as new if you can find a new variation or a new symptom that didn't exist in the previous release. What you are showing is that the new version's code interacts with this error in new ways. That's a new problem.
- This type of follow-up testing is especially important during a maintenance release that is just getting rid of a few bugs. Bugs won't be fixed unless they were (a) scheduled to be fixed because they are critical or (b) new side effects of the new bugfixing code.

 13

Motivating the Bug Fix: Show it is More General

Question: How many programmers does it take to change a light bulb?

Answer: *What's the problem? The bulb at my desk works fine!*

LOOK FOR CONFIGURATION DEPENDENCE

In the ideal case (standard in many companies), you test on 2 machines

- Do your main testing on Machine 1. Maybe this is your powerhouse: latest processor, newest updates to the operating system, fancy printer, video card/USB devices, huge hard disk, lots of RAM, cable modem, etc.
- When you find a defect, use Machine 1 as your bug reporting machine and replicate on Machine 2. Machine 2 is totally different. Different processor, different keyboard and keyboard driver, different video, barely enough RAM, slow, small hard drive, dial-up connection with a link that makes turtles look fast.
- Some people do their main testing on the turtle and use the power machine for replication.
- Write the steps, one by one, on the bug form at Machine 1. As you write them, try them on Machine 2. If you get the same failure, you've checked your bug report while you wrote it. (A valuable thing to do.)
- If you don't get the same failure, you have a configuration dependent bug. Time to do troubleshooting. But at least you know that you have to.

AS A MATTER OF GENERAL GOOD PRACTICE, IT PAYS TO REPLICATE EVERY BUG ON A SECOND MACHINE.

 14

Motivating the Bug Fix: Show it is More General

TRY VARIANTS THAT SHOULDNT MATTER

The point of this exercise is to show that you get the same failure whether a given variable is set one way or another.

In follow-up testing, we varied "irrelevant" variables with an eye to seeing differences in the failure symptoms. We picked variables that looked promising for this.

In generalization testing, we're still looking to see whether the failure symptoms change, but we're picking variables that we don't expect to cause a change. The point is to take a variable that has been set one way throughout the testing of this bug, show that you get the same problem with a different setting, and then you can ignore this variable, not discuss it in the bug report, treat it as irrelevant.

 15

Motivating the Bug Fix: Show it is More General

UNCORNERYOUR CORNER CASES

We test at extreme values because these are the most likely places to show a defect. ***But once we find the defect, we don't have to stick with the extreme value test.***

- Try mainstream values. These are easy settings that should pose no problem to the program. Do you replicate the bug? If yes, write it up, referring primarily to these mainstream settings. This will be a very credible bug report.
- If the mainstream values don't yield failure, but the extremes do, then do some troubleshooting around the extremes. Is the bug tied to a single setting (a true corner case)? Or is there a small range of cases? What?

 16

Corner Conditions
A Thought Experiment

(Ive never tried anything like this on Quicken. This example is purely hypothetical.)

- Imagine that you were testing Quicken. I understand that it can handle an arbitrarily large number of checks, limited only by the size of the hard disk. So, being a tester, you go buy the biggest whopping hard disk on the market (about 30 gig at the moment, I think) and then write a little program to generate a Quicken-like data file with a billion or so cheques.
- Now, open Quicken, load that file, and add 1 more check. It works — sort of. The clock spins and spins while Q sorts the new check into place. At the end of four days, you regain control.
- So you write a bug report— "4 days to add a check is a bit too long."
- The programmers respond— "Not a bug. Only commercial banks manage a billion checks. If you're Citibank, use a different program."

Suppose that you decided to follow this up because you thought the programmers are dismissing a real problem without analysis. What would you do?

 17

Overcoming Objections:
Unrealistic (e.g., Corner Conditions)

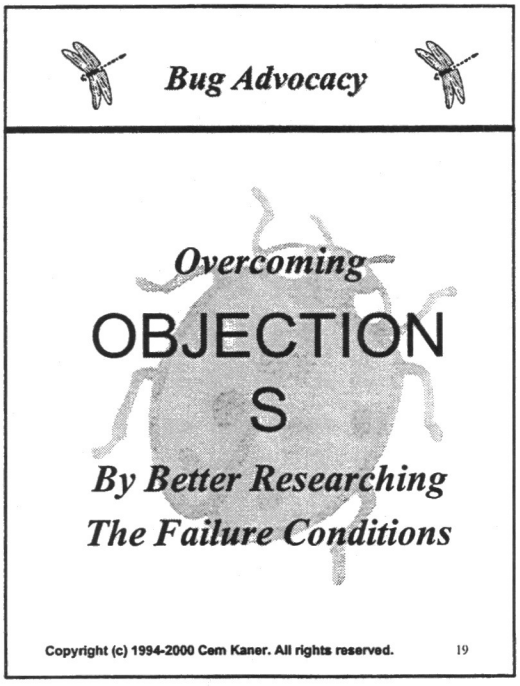
Some reports are inevitably dismissed as unrealistic (having no importance in real use).

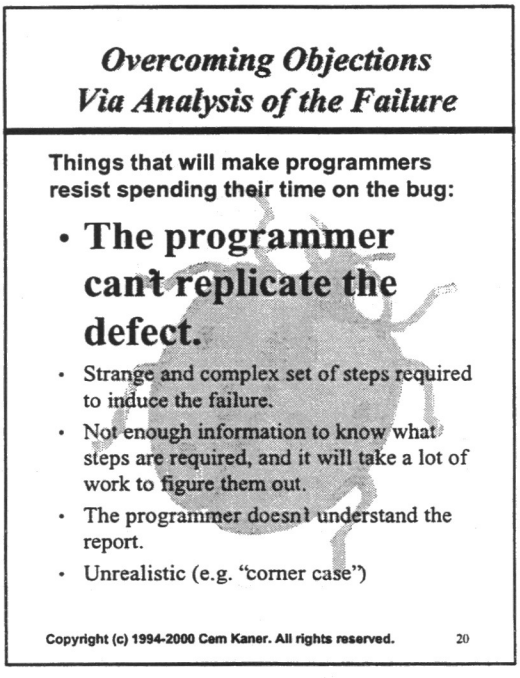
- If you're dealing with an extreme value, do follow-up testing with less extreme values.
- If you're protesting a bug that has been left unfixed for several versions, realized that it has earned tenure in some people's minds. Perhaps, though, customer complaints about this bug have simply never filtered through to developers.
- If your report of some other type of defect or design issue is dismissed as having "no customer impact," ask yourself:

  Hey, how do they know the customer impact?

- Then check with
  - Technical marketing
  - Technical support
  - Human factors
  - Documentation and training
  - Network administrators
  - In-house power users
  - Maybe sales or corporate communications.

 18





Non-Reproducible Errors

- Always report non-reproducible errors, but describe your steps and observations precisely. Programmers will often figure them out.
- When we try to replicate a bug, we hypothesize about the cause or about the nature of conditions that might be relevant. We then try to recreate the conditions that make the error visible.
- Many conditions are potentially implicated in any bug:
  - » the ones you expect to affect program behavior,
    - *includes some that dont and some that actually cant affect behavior.*
    - *Some of the ones that dont have effects should -- thats the bug.*
  - » the hidden input variables (that you dont expect to have an effect but sometimes they do)
  - » the ones that youre not even aware of because you cant manipulate them.

 21

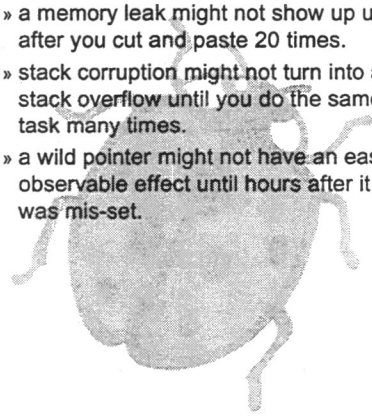
Non-Reproducible Errors

- The fact that a bug is not reproducible is data. The program is telling you that you have a hole in your logic. You are not entertaining certain relevant conditions. Why not?
- See Watts Humphrey, *Personal Software Process*, for recommendations to programmers of a system for discovering and then eliminating characteristic errors from their code. As he suggested to me, we can think of consistently missed replication conditions as characteristic errors made by testers, and apply similar techniques.
  - » **To improve over time, keep track of the bugs youre missing and what conditions you are not attending to (or find too hard to manipulate).**
- The following pages give a list of some conditions commonly ignored or missed by testers. Your personal list will be different in some ways, but maybe this is a good start.

 22

Non-Reproducible Errors: Examples of Conditions Often Missed

- Some problems have delayed effects:
  - » a memory leak might not show up until after you cut and paste 20 times.
  - » stack corruption might not turn into a stack overflow until you do the same task many times.
  - » a wild pointer might not have an easily observable effect until hours after it was mis-set.



 23

Non-Reproducible Errors: Examples of Conditions Often Missed

- The bug depends on the value of a *hidden input variable*. (Bob Stahl teaches this well.) In any test, there are the variables that we think are relevant and then there is everything else. If the data that you think are relevant don't help you reproduce the bug, look at all of the other variables that were set, and their values, by the time the test was run.
- Some conditions are hidden and others are invisible. You cannot manipulate them and so it is harder to recognize that they're present.
- Some conditions are *catalysts*. They make failures more likely to be seen. Example: low memory for a leak; slow machine for a race. But sometimes catalysts are more subtle, such as use of one feature that has a subtle interaction with another.
- Some bugs are predicated on corrupted data. They don't appear unless there are impossible configuration settings in the config files or impossible values in the database. What could you have done earlier today to corrupt this data?
- The bug might appear only at a specific time of day or day of the month or year. (We're all waiting to see how many bugs will show up on Jan. 1, 2000 and on Feb. 29 (is there a Feb. 29?) 2000.)

 24

Non-Reproducible Errors: Examples of Conditions Often Missed

- Programs have various degrees of *data coupling*. When two modules use the same variable, oddness can happen in the second module after the variable is changed by the first. (Books on structured design, such as Yourdon/Constantine often analyze different types of coupling in programs and discuss strengths and vulnerabilities that these can create.) In some programs, interrupts share data with main routines in ways that cause bugs that will only show up after a specific interrupt.
- Special cases appear in the code because of time or space optimizations or because the underlying algorithm for a function depends on the specific values fed to the function (talk to your programmer).
- The bug depends on you doing related tasks in a specific order.
- The bug is caused by a race condition or other time-dependent event, such as:
  - » An interrupt was received at an unexpected time.
  - » The program received a message from another device or system at an inappropriate time (e.g. after a time-out.)
  - » Data was received or changed at an unexpected time.
- The bug is caused by an error in error-handling. You have to generate a previous error message or bug to set up the program for this one.

 25

Non-Reproducible Errors: Examples of Conditions Often Missed

- *Time-outs* trigger a special class of multiprocessing error handling failures. These used to be mainly of interest to real-time applications, but they come up in client/server work and are very pesky.

  Process A sends a message to Process B and expects a response. B fails to respond. What should A do? What if B responds later?
- Another inter-process error handling failure -- Process A sends a message to B and expects a response. B sends a response to a different message, or a new message of its own. What does A do?
- You're being careful in your attempt to reproduce the bug, and you're typing too slowly to recreate it.
- The program might be showing an initial state bug, such as:
  - » The bug appears only the first time after you install the program (so it happens once on every machine.)
  - » The bug appears once after you load the program but won't appear again until you exit and reload the program.

    (See Testing Computer Software's Appendix's discussion of initial state bugs.)

 26

Non-Reproducible Errors: Examples of Conditions Often Missed

- The program may depend on one version of a DLL. A different program loads a different version of the same DLL into memory. Depending on which program is run first, the bug appears or doesn't.
- The problem depends on a file that you think you've thrown away, but it's actually still in the Trash (where the system can still find it).
- A program was incompletely deleted, or one of the current program's files was accidentally deleted when that other program was deleted. (Now that you've reloaded the program, the problem is gone.)
- The program was installed by being copied from a network drive, and the drive settings were inappropriate or some files were missing. (This is an invalid installation, but it happens on many customer sites.)
- The bug depends on co-resident software, such as a virus checker or some other process, running in the background. Some programs run in the background to intercept foreground programs' failures. These may sometimes trigger failures (make errors appear more quickly).

 27

Non-Reproducible Errors: Examples of Conditions Often Missed

- The bug depends on a crash or exit of an associated process.
- The program might appear only under a peak load, and be hard to reproduce because you can't bring the heavily loaded machine under debug control (perhaps it's a customer's system).
- On a multi-tasking or multi-user system, look for spikes in background activity.
- The bug occurred because a device that it was attempting to write to or read from was busy or unavailable.
- It might be caused by keyboard bounce or by other hardware noise.
- Code written for a cooperative multitasking system can be thoroughly confused, sometimes, when running on a preemptive multitasking system. (In the *cooperative* case, the foreground task surrenders control when it is ready. In the *preemptive* case, the operating system allocates time slices to processes. Control switches automatically when the foreground task has used up its time. The application is suspended until its next time slice. This switch occurs at an arbitrary point in the application's code, and that can cause failures.

 28

Non-Reproducible Errors: Examples of Conditions Often Missed

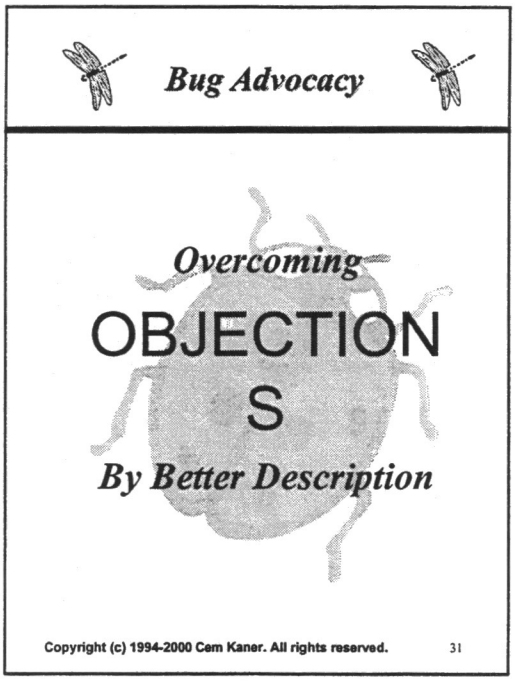
- The bug is specific to your machine's hardware and system software configuration. (This common problem is hard to track down later, after you've changed something on your machine. It's good reporting practice to replicate the bug on a second configuration.)
- The bug is specific to your machine's hardware and system software configuration. (This common problem is hard to track down later, after you've changed something on your machine. That's why good reporting practice involves replicating the bug on a second configuration.)
- Someone tinkered with your machine.
- You forgot some of the details of the test you ran, including the critical one(s) or you ran an automated test that lets you see that a crash occurred but doesn't tell you what happened.

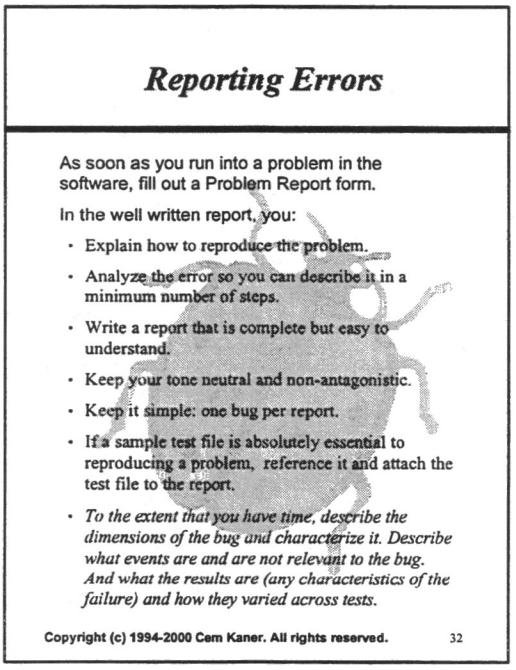
 29

Making Errors Reproducible

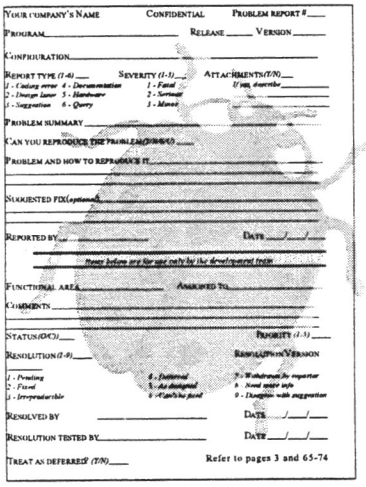
- Write down everything you remember about what you did the first time.
- Note which things you are sure of and which are good guesses.
- Note what else you did before starting on the series of steps that led to this bug.
- Review similar problem reports you've come across before.
- Generate predictions from your guesses about what makes this bug hard to reproduce, and try them out.
- There are tools such as videotape, capture programs, debugger, debug-logger, or memory meter that can help you identify things that you did before running into the bug.
- Find ways to affect timing of your program or of your devices, Slow down, speed up.
- ***Talk to the programmer*** and/or read the code.

 30





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 any, describe_____

2 - Design issue 5 - Hardware 2 - Serious

3 - Suggestion 6 - Query 3 - Minor

PROBLEM SUMMARY_____

CAN YOU REPRODUCE THE PROBLEM (Y/N/S/O) ___

PROBLEM AND HOW TO REPRODUCE IT_____

SUGGESTED FIX (optional)_____

REPORTED BY_____ DATE ___/___/___

Items below are for use only by the development team

FUNCTIONAL AREA_____ ASSIGNED TO_____

COMMENTS_____

STATUS (O/C) ___ PRIORITY (1-3) ___

RESOLUTION (1-9) ___ RESOLUTION VERSION

1 - Pending 6 - Deferred 7 - Withdrawn by reporter

2 - Fixed 5 - As designed 8 - Need more info

3 - Irreproducible 4 - Can't be fixed 9 - Duplicate with suggestion

RESOLVED BY _____ DATE ___/___/___

RESOLUTION TESTED BY_____ DATE ___/___/___

TREAT AS DEFERRED? (Y/N) ___ Refer to pages 3 and 65-74

 33

Important Parts of the Report Problem Summary

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

- The project manager will use it in 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 gives the reader enough information to help her decide whether to ask for more information. It should include:

- A brief description that is specific enough that the reader can visualize the failure.
- A brief indication of the limits or dependencies of the bug (how narrow or broad are the circumstances involved in this bug)?
- Some other indication of the severity (not a rating but helping the reader envision the consequences of the bug.)

 34

Important Parts of the Report
Can You Reproduce The Problem?

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

- Never say it's reproducible unless you have recreated the bug. (Always try to recreate the bug before writing the report.)
- If you've tried and tried but you can't recreate the bug, say "No". Then explain what steps you tried in your attempt to recreate it.
- If the bug appears sporadically and you don't yet know why, say "Sometimes" and explain.
- You may not be able to try to replicate some bugs. Example: customer-reported bugs where the setup is too hard to recreate.

The following policy is not uncommon:

- **If the tester says that a bug is reproducible and the programmer says it is not, then the tester has to recreate it in the presence of the programmer.**

 35

Important Parts of the Report
Description; How to Reproduce It.

- First, describe the problem. What's 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. NUMBER THE STEPS. Take it one step at a time.
- Describe the erroneous behavior and, if necessary, explain what should have happened. (Why is this a bug? Be clear.)
- List the environmental variables (config, etc.) that are not covered elsewhere in the bug tracking form.
- If you expect the reader to have any trouble reproducing the bug (special circumstances are required), be clear about them.

 36

Simplify, Simplify: Eliminate Unnecessary Steps

You might decide that it would be useful to the programmer to know that the bug doesn't appear if you first move the selected area, then try to delete. If so, report this step AFTER you describe the main bug. Say: NOTE:" and then add this information.

Sometimes it's 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.

You have a list of all the steps that you took to show the error. You're now trying to shorten the list. Look carefully for any hint of an error as you take each step -- A few things to look for:

- 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

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. If this doesn't work, try taking out individual steps or small groups of steps.

Think back to the Microsoft Paint example. Here are my steps to reproduce it:

1 Start the program.
2 Use the paint can to color the background. (Or load any image.)
3 Zoom to 200%.
4 Use the freehand tool to select part of the image.
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.

There is no need to talk about moving the selected area.

 37

Simplify, Simplify: Split the Report in Two

If you see two failures, write two reports.

Combining failures on one report creates problems:

- The summary description is typically vague. You say words like "fails" or "doesn't work" instead of describing the failure more vividly. This weakens the impact of the summary.
- The detailed report is typically lengthened. It's common to see bug reports that read like something written by an inept lawyer. Do this unless that happens in which case don't do this unless the first thing and then thetestcase of the second part and sometimes you see this but if not then that.
- Even if the detailed report is rationally organized, it is longer (there are two failures and two sets of conditions, even if they are related) and therefore more intimidating.
- You'll often see one bug get fixed but not the other.
- When you report related problems on separate reports, it is a courtesy to cross-reference them.

 38

Editing Bug Reports

Some groups have a second tester (usually a senior tester) review reported defects before they go to the programmer. The second tester:

- checks that critical information is present and intelligible
- checks whether she can reproduce the bug
- asks whether the report might be simplified, generalized or strengthened.

If there are problems, she takes the bug back to the original reporter.

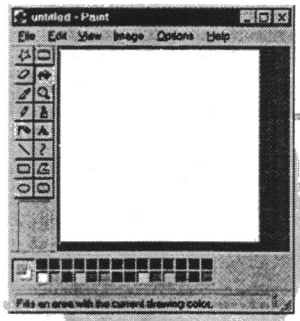
- If the reporter was outside the test group, she simply checks basic facts with him.
- If the reporter was a tester, she points out problems with an objective of furthering the tester's training.

This tester might review:

- all defects
- all defects in her area
- all of her buddy's defects.

 39

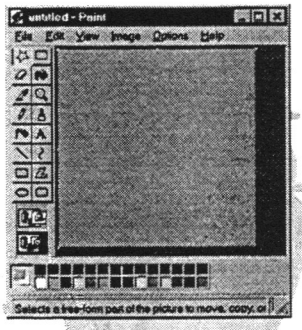
Bug Reporting Exercise (1)



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

 40

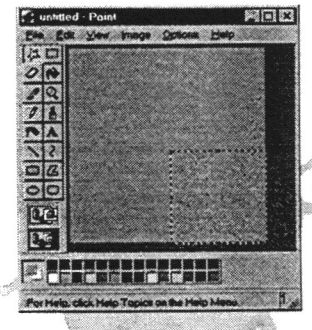
Bug Reporting Exercise (2)



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 cut it, copy it, move it, etc.

 41

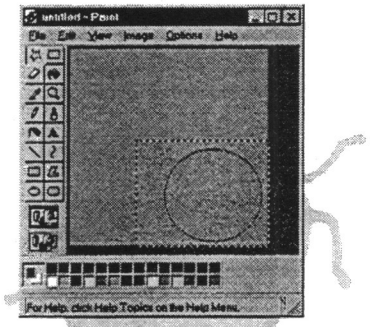
Bug Reporting Exercise (3)



This shows an area selected with the freehand selection tool. The bottom right corner is selected. The actual area might not be perfectly rectangular, but the freehand tool shows a rectangle that is just big enough to enclose the selected area.

 42

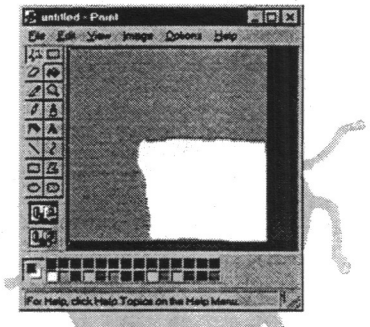
Bug Reporting Exercise (4)



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

 43

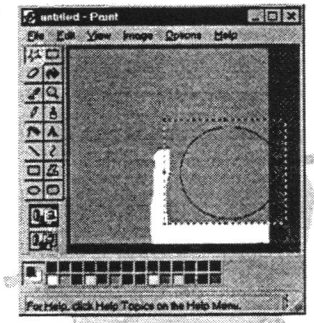
Bug Reporting Exercise (5)



You can cut the selection. (Note the jagged border. When you use the freehand selection tool, you select an area by moving the mouse. To make a square selection, use the rectangular select tool.)

 44

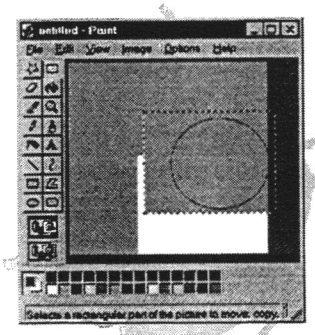
Bug Reporting Exercise (6)



And you can move the selection.

 45

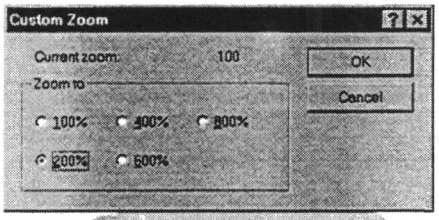
Bug Reporting Exercise (7)



This shows a move after using the rectangular selection tool.

 46

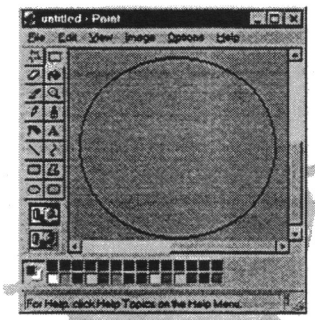
Bug Reporting Exercise (8)



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

 47

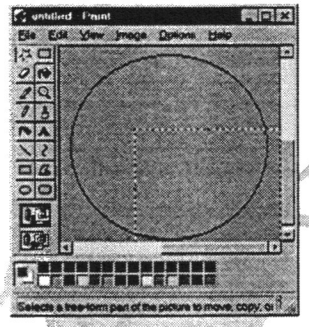
Bug Reporting Exercise (9)



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

 48

Bug Reporting Exercise (10)

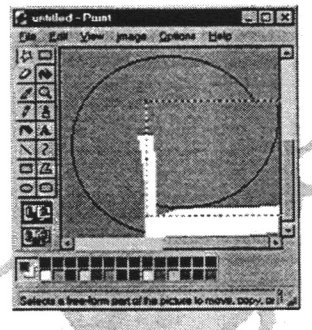


So, select part of the circle. Well try the move and cut features.

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

 49

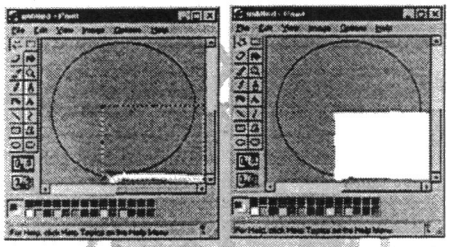
Bug Reporting Exercise (11)



Select the area again, and try to move it. Moving works.

 50

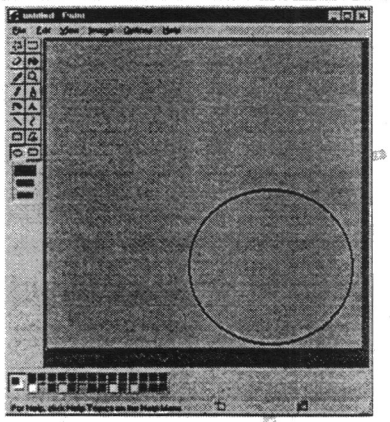
Bug Reporting Exercise (12)



Now try this. Move the selected area first, and *then* try to cut (press Delete or Ctrl-X). It works.

 51

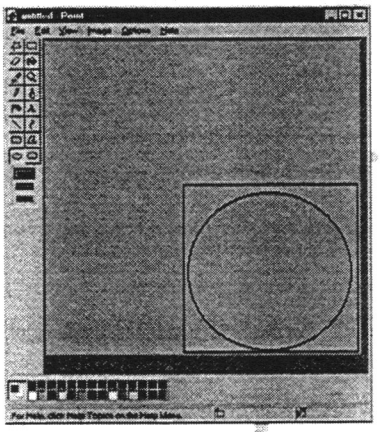
Bug Reporting Exercise (13)



Set up the circle like slide 4, zoom to double the size (slide 8) and then grow the window so that you can see the whole picture.

 52

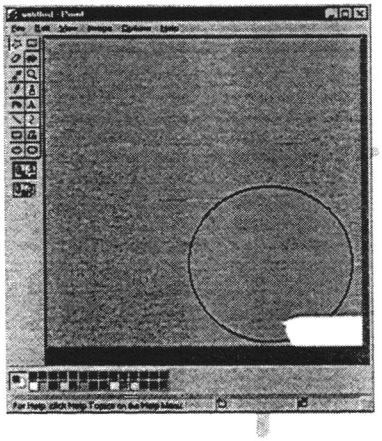
Bug Reporting Exercise (14)



Select around the circle, and hit Delete

 53

Bug Reporting Exercise (15)



It cut the wrong part of the picture!

 54

Bug Reporting Exercise (16)

Here is your assignment.

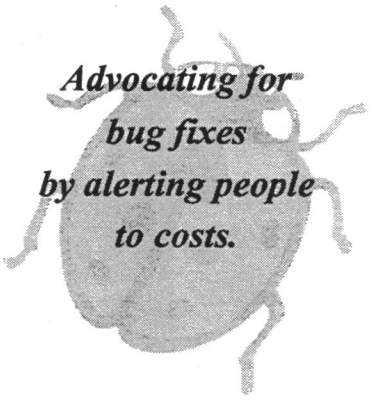
1 Write a bug report that describes this bug. Include only two sections:
   - » Problem Summary
   - » Problem Description

2 Are there any more tests that you'd like to run? If so, what are they and what do you hope to find?

3 Meet with your group to read each other's reports.
   - » How much do you learn from the summary?
   - » How clear is the description?
   - » How complete is the description?
   - » How accurate is the description?

 55

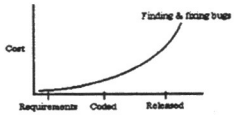
Bug Advocacy



Advocating for bug fixes by alerting people to costs.

 56

Cost of Finding and Fixing Software Errors



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

- if we find bugs in requirements, we can fix them without having to recode anything;
- 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, think of the marketing assistant who wastes days trying to create a demo, but can't because of bugs.

 57

Influencing Others Based on Costs

Its probably impossible to fix every bug. Sometimes the development team will choose to not fix a bug based on their assessment of its risks for them, without thinking of the costs to other stakeholders in the company.

- Probable tech support cost
- Risk to the customer.
- Risk to the customer's data or equipment.
- Visibility in an area of interest to reviewers.
- Extent to which the bug detracts from the use of the program.
- How often will a customer see it?
- How many customers will see it?
- Does it block any testing tasks?
- Degree to which it will block OEM deals or other sales.

To argue against a deferral, ask yourself which stakeholder(s) will pay the cost of keeping this bug. Flag the bug to them.

 58

Cem Kaner, Ph.D., J.D. P.O. Box
1200
Attorney at Law Santa Clara, CA 95052
kaner@kaner.com 408-244-7000

Quality Cost Analysis: Benefits and Risks

Published in *Software QA*, , 3, #1, 1996, p. 23.

"Because the main language of [corporate management] was money, there emerged the concept of studying quality-related costs as a means of communication between the quality staff departments and the company managers."

Joseph Juran, one of the world's leading quality theorists, has been advocating the analysis of quality-related costs since 1951, when he published the first edition of his *Quality Control Handbook*. Feigenbaum made it one of the core ideas underlying the Total Quality Management movement. It is a tremendously powerful tool for product quality, including software quality.

www.kaner.com/qualcost.htm

 59

Bug Advocacy

What About The Objection That It Is Not Really A Bug?

Really, it's a feature.

Or, at least, it's not a problem for my release so I don't have to fix it.

It won't matter until we ship it to Germany. Let them fix it.

 60

Software Errors: What is Quality?

Here are some of the traditional definitions:

- 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 (ASQ)
- 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)

Note the absence of "conforms to specifications."

 61

Software Errors: What Should We Report?

I like Gerald Weinberg's definition:

Quality is value to some person

But consider the implication:

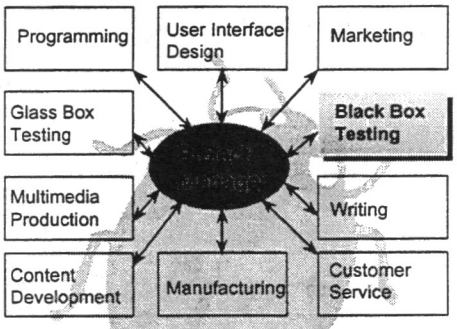
- **It's appropriate to report any deviation from high quality as a software error.**
- **Therefore many issues will be reported that will be errors to some and non-errors to others.**

Glen Myers' definition:

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

 62

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 slide gives some examples.

 63

Quality is Multidimensional: Different People, Different Visions

Localization Manager: *A good product is easy to translate and to modify to make it suitable for another country and culture. Few expereienced localization managers would consider acceptable a product that must be recompiled or relinked to be localized.*

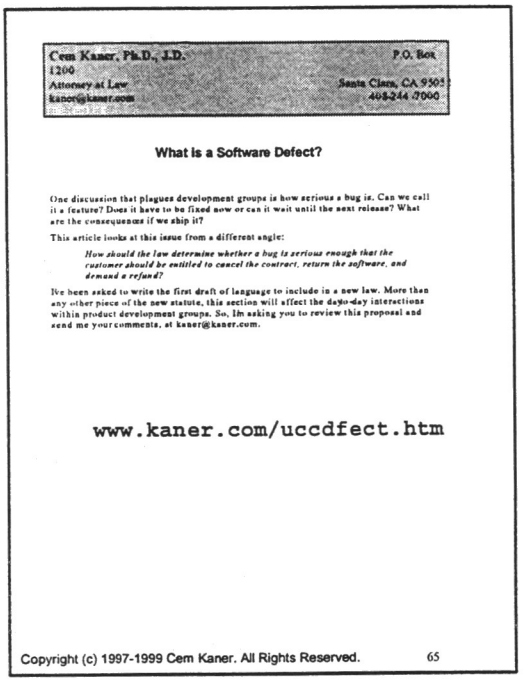
Tech Writers: *A high quality program is easily explainable. Aspects of the design that are confusing, unnecessarily inconsistent, or hard to describe are marks of bad quality.*

Marketing: *Customer satisfiers are the things that drive people to buy the product and to tell their friends about it. A Marketing Manager who is trying to add new features to the product generally believes that he is trying to improve the product.*

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

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

 64



Cem Kaner, Ph.D., J.D.
1200
Attorney at Law
kaner@kaner.com
P.O. Box
Santa Clara, CA 9505
408-244-7000

What is a Software Defect?

One discussion that plagues development groups is how serious a bug is. Can we call it a feature? Does it have to be fixed now or can it wait until the next release? What are the consequences if we ship it?

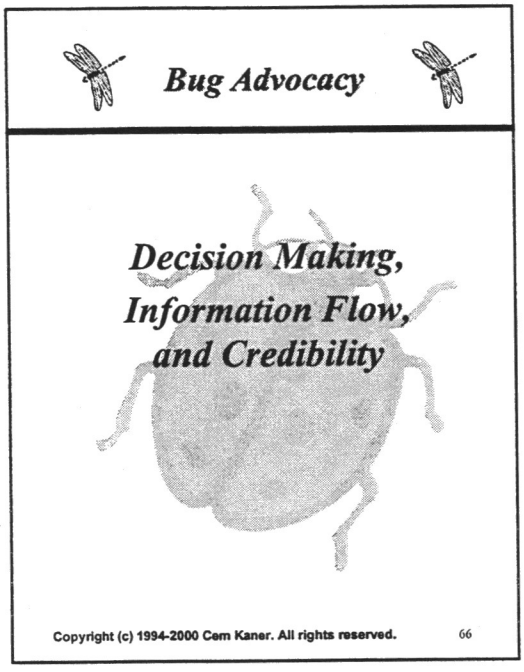
This article looks at this issue from a different angle:

> *How should the law determine whether a bug is serious enough that the customer should be entitled to cancel the contract, return the software, and demand a refund?*

Ive been asked to write the first draft of language to include in a new law. More then any other piece of the new statute, this section will affect the day-to-day interactions within product development groups. So, Im asking you to review this proposal and send me your comments, at kaner@kaner.com.

www.kaner.com/uccdfect.htm

 65

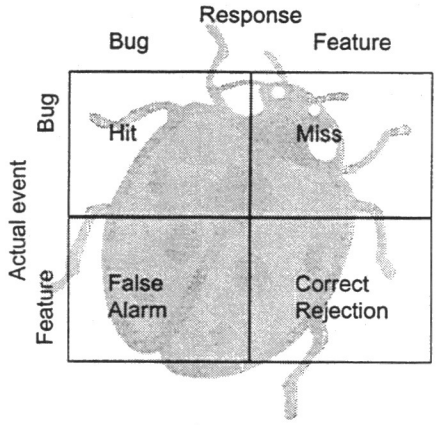


Bug Advocacy

Decision Making, Information Flow, and Credibility

 66

The Signal Detection & Recognition Problem



Refer to Testing Computer Software, pages 24, 116-118

 67

Lessons From Signal Detection: We Make Decisions Under Uncertainty

When you try to decide whether an item belongs to one category or the other (bug or feature), your decision will be influenced by your expectations and your motivation.

- Can you cut down on the number of false alarms without increasing the number of misses?
- Can you increase the number of hits without increasing the number of false alarms?
- Pushing people to make fewer of one type of reporting error will inevitably result in an increase in another type of reporting error.
- Training, specs, etc. help, but the basic problem remains.

 68

Lessons From Signal Detection: Decisions Are Subject To Bias

We make decisions under uncertainty.

Decisions are subject to bias, and much of this is unconscious.

The prime biasing variables are:

- ***perceived probability:***
  If you think that an event is unlikely, you will be ***substantially*** less likely (beyond the actual probability) to report it.
- ***perceived consequence of a decision:***
  What happens if you make a False Alarm? Is this worse than a Miss or less serious?
- ***perceived importance of the task:***
  The degree to which you care / don't care can affect your willingness to adopt a decision rule that you might otherwise be more skeptical about

 69

Lessons From Signal Detection: Decisions Are Subject To Bias

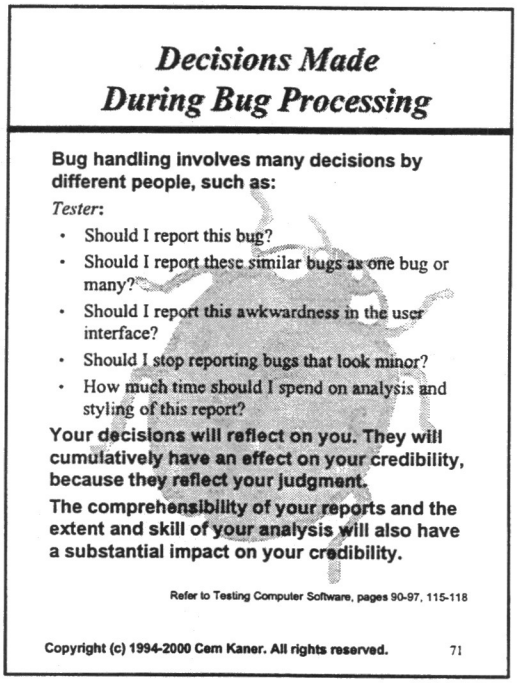
Decisions are made by a series of people.

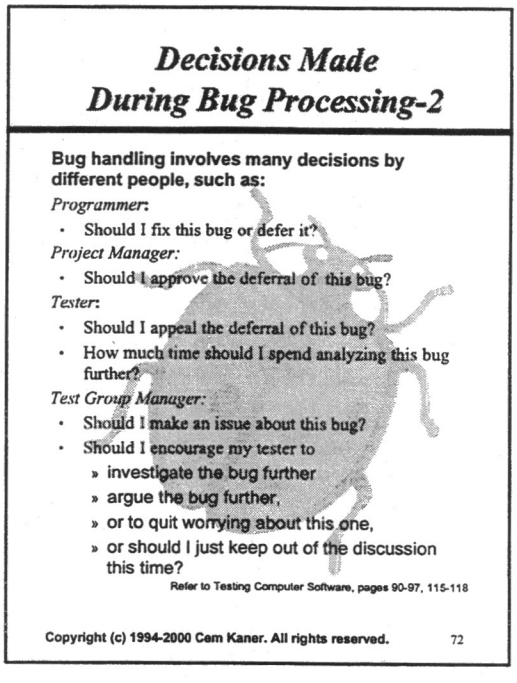
- **Bug reporting policies must consider the effects on the overall decision-making system, not just on the tester and first-level bug reader.**

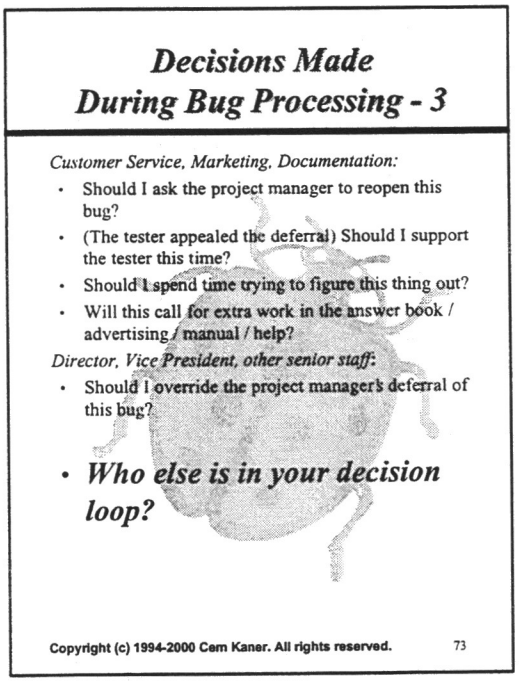
Trace these factors through the decisions and decision-makers (next slides). For example, what happens to your reputation if you

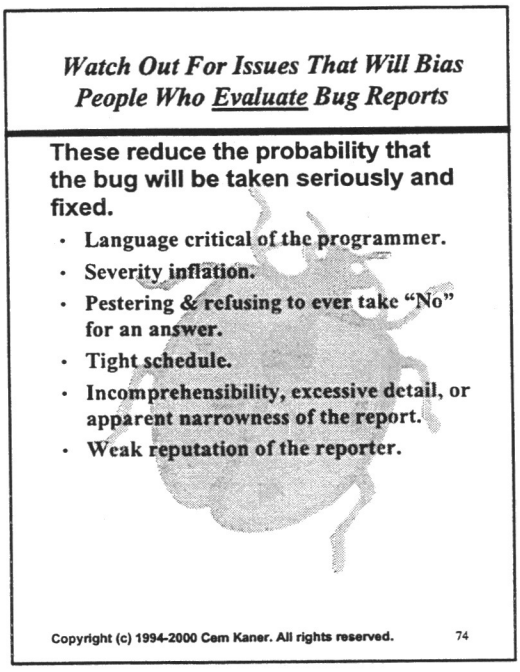
- Report every bug, no matter how minor, in order to make sure that no bug is ever missed?
- Report only the serious problems (the 'good bugs')?
- Fully analyze each bug?
- Only lightly analyze bugs?
- Insist that every bug get fixed?

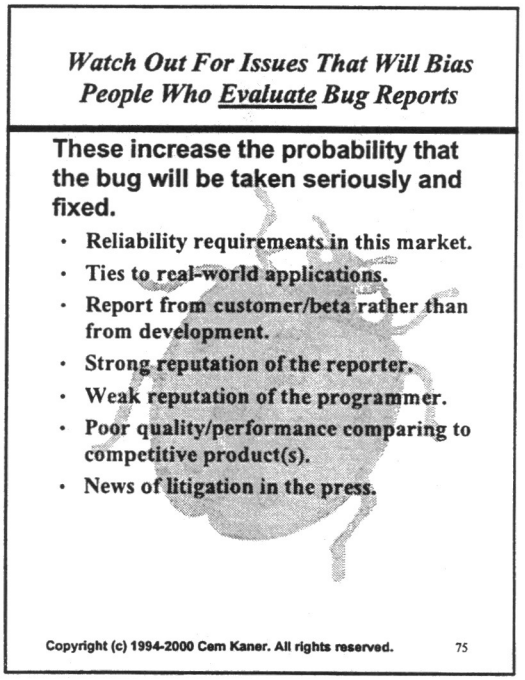
 70

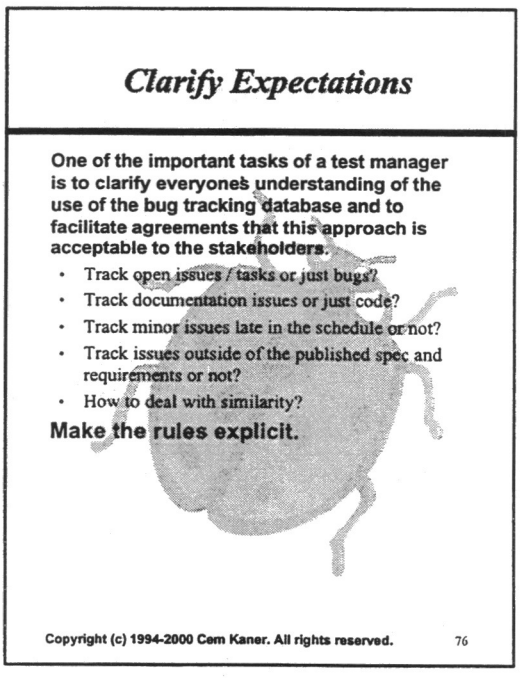












Biasing People Who Report Bugs

These help increase the probability that people will report bugs.

- Give results feedback to non-testers who report bugs.
- Encourage testers to report all anomalies.
- Adopt a formal system for challenging bug deferrals.
- Weigh schedule urgency consequences against an appraisal of quality costs. (Early in the schedule, people will report more bugs; later people will be more hesitant to report minor problems).
- Late in the schedule, set up a separate database for design issues (which will be evaluated for the start of the next release).

These will reduce the probability that bugs will be reported, either by discouraging reporters, by convincing them that their work is pointless or will be filtered out, or by creating incentives for other people to pressure people not to report bugs.

- Never use bug statistics for employee bonus or discipline.
- Never use bug statistics to embarrass people.
- Never filter reports that you disagree with.
- Never change an in-house bug reporter's language, or at least not without free permission. Add your comments as additional notes, not as replacement text.
- Monitor language in the reports that is critical of the programmer or the tester.
- Beware of accepting lowball estimates of bug probabilities.

 77

Notes on Exercise

I do some analysis before writing. Here's a structure for making your notes:

OBSERVED FAILURES
Didn't cut the object
Cut the wrong part of the object

CONDITIONS
zoom 200%
freehand tool
grow the window (#2)

OTHER CONDITIONS (maybe irrelevant)
Grey background
Bottom right corner

NOTES
Doesn't happen if you zoom before cutting

MY SUMMARIES
(1) Cut doesn't cut (intermittent with zoom and freehand tool)
(2) Cuts the wrong area (intermittent with zoom, freehand, grow window)

MY PROBLEM DESCRIPTION (SINGLE)
PROBLEM: Fails to cut the selected area.
STEPS TO REPRODUCE:
1. Start the program
2. Use the paint can to color the background
3. Draw a circle
4. Zoom 200%
5. Use the freehand tool to select part of the circle
6. Attempt to cut the selected area. Result: selection marquee goes away but nothing is deleted.
NOTES: The bug doesn't appear if you zoom the object before trying to cut it.

IDEAS FOR FURTHER TESTING:
(a) Other zooms, backgrounds, shapes. Do we need the circle at all?
(b) follow up to see if memory is corrupted. Try other images, higher resolution video, larger images, many repeated cuts. I'm curious about memory corruption because of the cut that cut the wrong area. Try pasting.

SOME NOTES ON THE EXERCISE
This example contains three intentional problems.

- First, there's the issue of one bug vs. two.
- Second is the language usage -- "cut" versus "delete." For people who don't understand paint programs, these seem like equivalent terms, but of course they're not. Using both creates confusion. Watch your language. Use ONE term consistently in reports. Avoid synonyms (even correct ones).
- Third is the problem of verbal versus written description. Were you confused about when I entered and exited the program? How far back did you think you had to go to get the program to a known initial state? I told you what I did, and these notes aren't outrageously ambiguous, but many people get confused anyway. Remember this confusion when YOU write a report. Verbal explanations don't add much of lasting value to a written bug report. If it's worth saying, get it into the report.

 78

Session W3

Achieving WebSite Quality

(Workshop)

Dr. Edward Miller
Software Research, Inc.

Achieving WebSite Quality

Edward Miller

Software Research, Inc.
eValid, Inc.

What is WebSite Quality?

- Visual
- Balance
- Content
- Accessibility
- Behavior
- Customer Satisfaction

WebSite Classifications

- Complexity
- Size
- Additional Factors

WebSite Complexity Classifications

WebSite Complexity Classifications

Class 1: Static	"Corporate Brochure"
Class 2: Passive	+ News Updates
Class 3: Collective	+ Collect Data
Class 4: Interactive	+ Search/Respond
Class 5: Responsive	+ e-Commerce

Software Research, Inc

WebSite Size Classifications

- Small: < 50 URLs
- Medium: 50 - 500 URLs
- Large: 500 - 1000 URLs
- Very Large: 1000 - 10,000 URLs
- Enormous: > 10,000 URLs

Software Research, Inc

WebSite Additional Factors

- Criticality
- Security
- Availability
- Other

Software Research, Inc

Levels of Concern (Examples)

- None: Class 1, Large+
- Moderate: Class 2+, Medium+
- High: Class 3+, Medium+
- Very High: Class 4+, Large+
- Life Critical: Class 5, Medium+

Browser Dependencies

- Older browsers
- JavaScript
- VB Script
- Java Applets
- XML

WebSite Failure Modes

- Availability
- Time/Speed
- Content
- Other

Software Research, Inc

WebSite Quality Approaches

- Static Analysis
  - Content
  - Connectivity
- Dynamic Analysis
  - Functional Behavior
  - Performance
- Capacity Analysis
  - "Safe" Load
  - Maximun Load
  - Saturation Load

Software Research, Inc

Typical Static Analysis/Testing

- HTML Checking
- HTML Tidy
- Link Checking
- Content Checking
- Other

Dynamic Analysis/Testing Validation

- Text
- Images
- Dialogs
- Sequences
- Other

Dynamic Performance Timing

- Single and Multiple Download Timings
- Overall User-level Response Times
- Perceived User-level Response Times, Thresholds
- Web Effects

Difficult to Test Pages

- Java Applets
- Dynamically Generated Pages
  - Adaptive Playback Solution
  - Dynamically updated pages
- Pages with Frames
- Pages with ID Tags

Hurdles for Dynamic Testing

- Repeatability
- Databases that "Remember"
- Fancy Page Effects
- Multi-Media Displays
- Browser Differences
  - Rendering
  - Adaptive Servers

Unsolved Problems in Dynamic Testing

- Streaming Audio/Video
- Applet control of Browser Area
- Some types of modal dialogs
- Page Transition Effects
- Voice Response

Load and Capacity Checking/Testing

- Load Imposition
- Load Measurement
- User Scenarios
- Realistic vs. Non-Realistic Experiments
- Client-Based vs. Server-Based Experiments
- Web Variability

Site Monitoring

- Confirmation of "Aliveness"
- Confirmation of Content
- Confirmation of Operation
- Timing of Operation
- Variable Rate of Reporting

Dynamic Properties of the Web

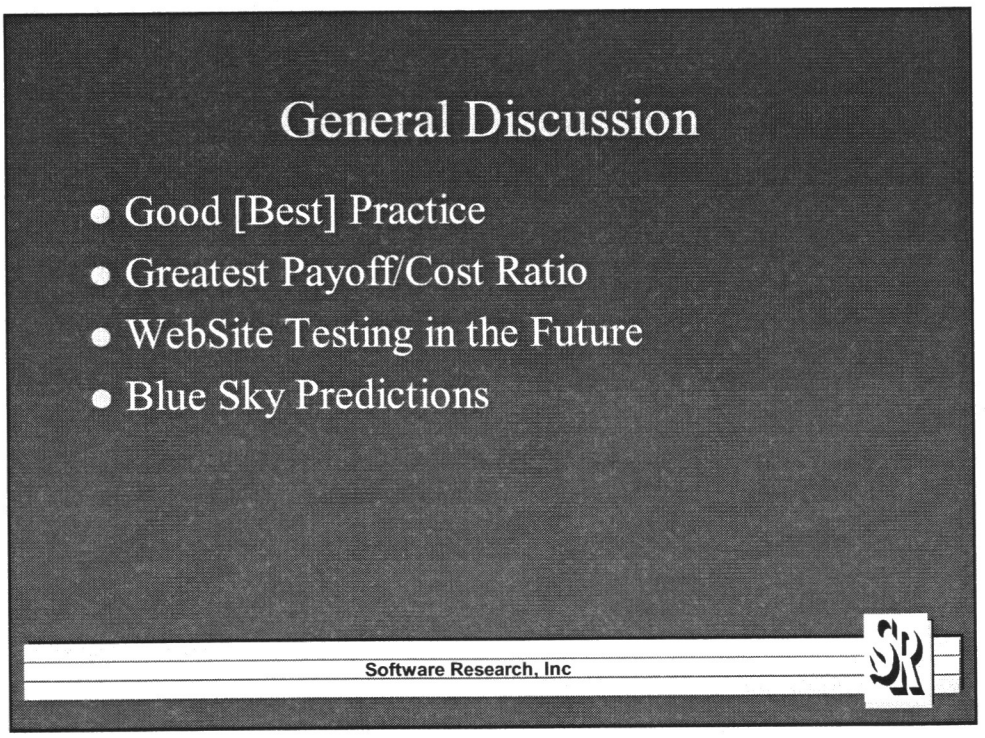
- Adaptive network speed
- Adaptive Routor
- Path geometrics
- "Throbbing" Phenomena

Typical Monitoring Services

- 24x7
- Geographic Coverage
- What is monitored
- Response method:
  - Email
  - Page
  - Direct

General Discussion

- Good [Best] Practice
- Greatest Payoff/Cost Ratio
- WebSite Testing in the Future
- Blue Sky Predictions

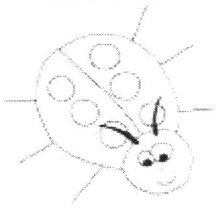


Session W4

The Effective SQA Manager: Getting Things Done

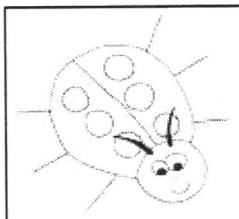
(Workshop)

Mr. Robert Sabourin
Purkinje Inc.



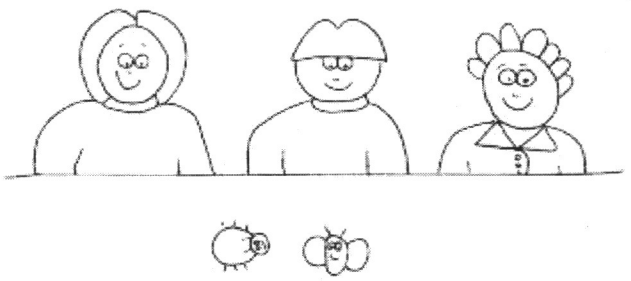
Becoming an Effective SQA Manager

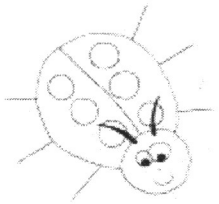
Robert Sabourin
President
AmiBug.Com, Inc.
Montreal, Canada
robsab@sympatico.ca



Becoming an Effective SQA Manager

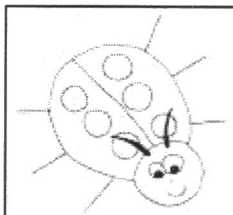
- It's all about people! (and the occasional bug too)



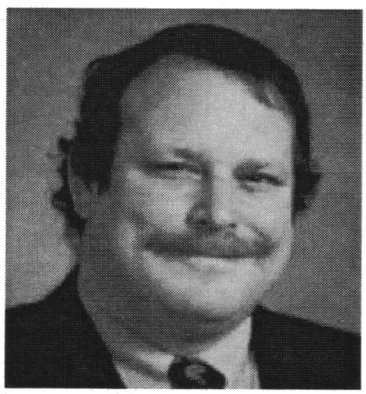


Becoming an Effective SQA Manager

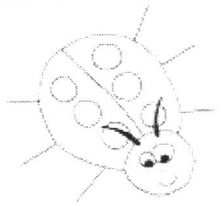
- Overview
  - Introductions
  - Fundamental Question in Software Engineering!
  - Parable of the Effective SQA Manager
  - Some Examples



Becoming an Effective SQA Manager



- Robert Sabourin , *Software Evangelist*
- President
- AmiBug.Com Inc.
- Montreal, Quebec, Canada
- robsab@sympatico.ca



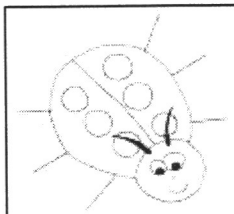
I am a Bug

Robert & Catherine Sabourin

ISBN: 0-9685774-0-7

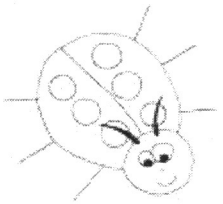
www.amazon.com

In the style of a children's book. Explains elements of software development process in a fun easy to read format.



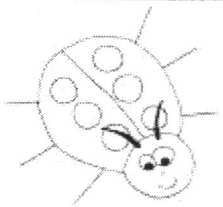
AmiBug.Com, Inc.

- Software Development & SQA Consulting
- Services
  - Training, Coaching and Professional Development
  - Light Effective Process
  - Team Building and Organization
  - We help people to get things done!



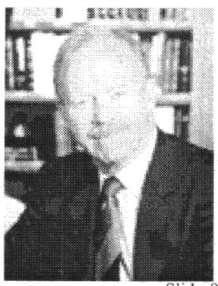
Fundamental Question

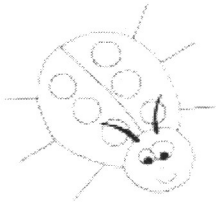
- How do you know when you are finished?



Crosby on Quality

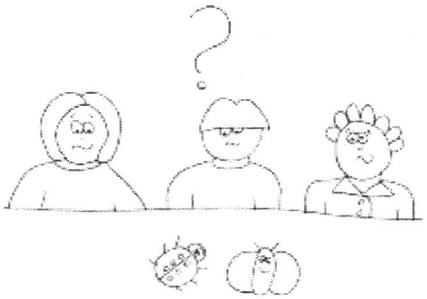
- "Quality is defined as conformance to requirements"
- "Quality is not a measure of GOODNESS"
  - Phil B. Crosby, *Quality is Free*

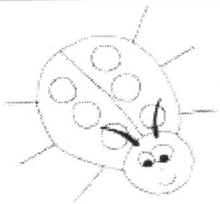




Definition of a Bug

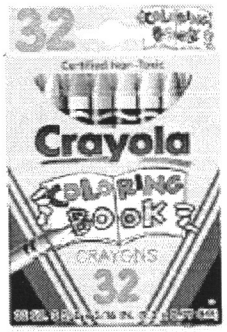
- To make our job more fun, whenever we have a *problem with software*, we call it a "bug".

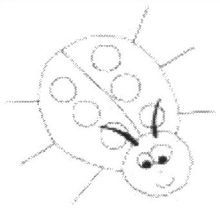




Crayons

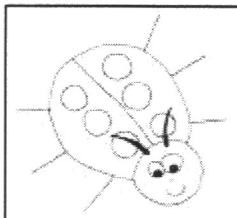
- Fun to draw pictures





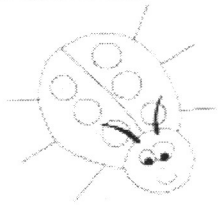
Index Cards

- Small
- Pocket Size
- Effective



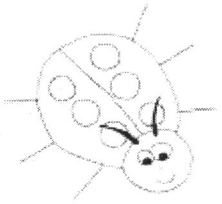
Arc

- Draw an arc on the card



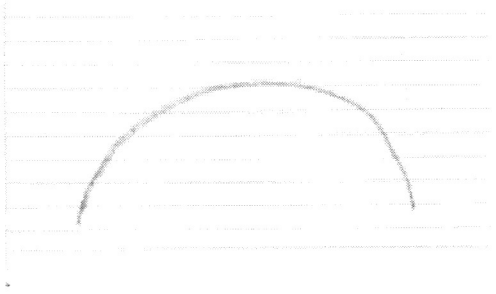
Smile

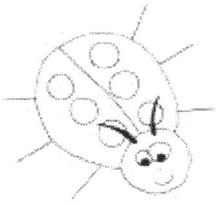
- An Arc is a smile



Frown

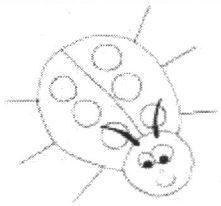
- An Arc can be a frown





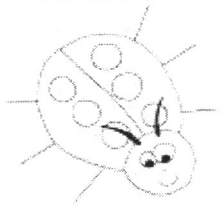
Bell

- we are going to measure our performance as a team
- not Pavlov
  - ring the bell and Dog will salivate



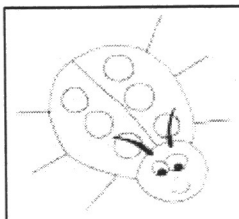
Goals

- teach some fundamental truths about becoming an effective SQA manager
  - cover material on time
  - ensure people are with me and are learning



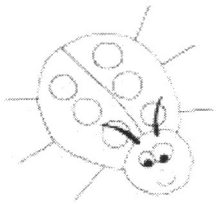
Goals

- have fun
  - the goal is to end the session with the same number/or more people attending than I started with
  - as we go ensure people have a good time
  - ensure people are happy



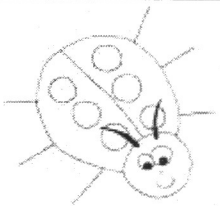
Measures

- appoint a coordinator to drive operational logistics
  - bell ringer on 30 minute mark
  - progress through session



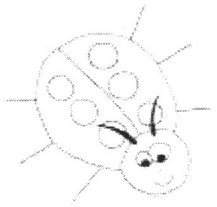
Measures

- appoint a coordinator to drive operational logistics
  - measure satisfaction as we go
    - count smiles
    - count frowns
    - count number of attendees
  - write down the score so everyone can see where we stand at all times!



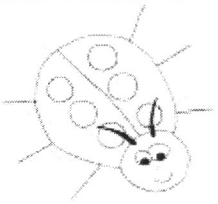
Added Incentives (stock options!)

- OK folks so it is Friday afternoon in San Francisco and you are listening to a flake explaining to you how to become an effective SQA manager! You deserve a bit of a bonus for sticking it out!

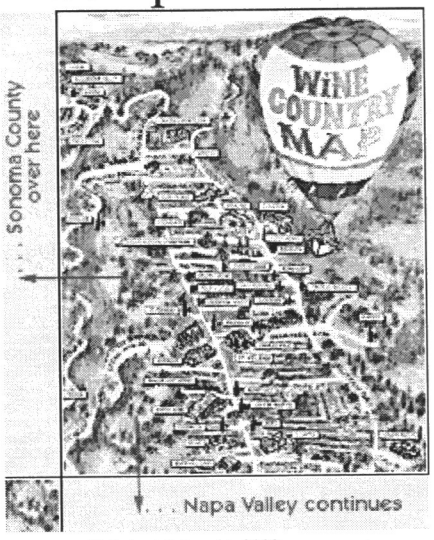


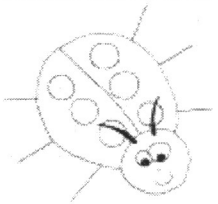
Added Incentives (stock options!)

- So we have a contest!
  - Submit your business card or write your name on a piece of paper
  - put it in the hat
  - at every 30 minute mark we will draw two names and award copies of my book
    - "I am a Bug"



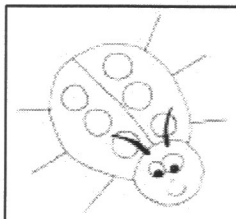
Temptation





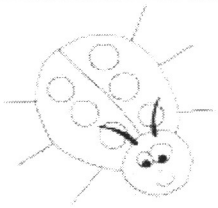
A note about parables

- teaching
- learning
- retaining
- applying knowledge
- share experiences



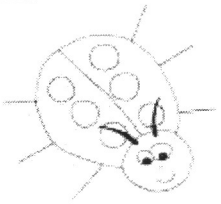
Fred and NoGo.com

- Story about Fred
- Fred will have a simple adventure
- Meet many interesting people
- Observe many things



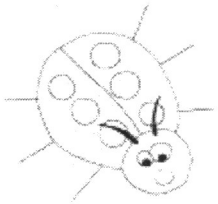
Fred

- a little bit of many people
- composite of many I have worked with
- young energetic



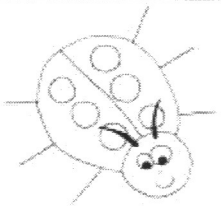
Fred

- his history
  - worked as a guru in software testing
  - expert and wizard
  - hands on



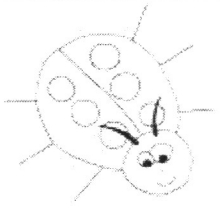
Fred

- his background
  - worked for a well organized company
  - isolated from the big picture
  - saw his work and interacted well with some developers
  - took charge of assignments responsibly



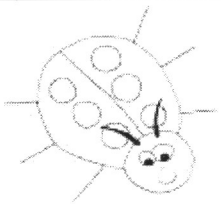
Fred

- his work
  - he did good
  - helped find important bugs
  - recognized as a key player
  - well liked by his peers and his manager



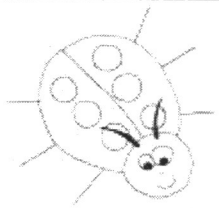
Fred

- But his company for some reason just didn't slice it
  - ran out of funds
  - could not sustain the pressure
  - even with great testers like Fred



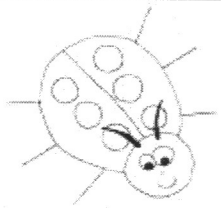
Fred's New Job

- head hunters found Fred a new job in no time flat
  - Fred was hired as an SQA director at NoGo.Com
  - Fred was brought in to make things happen and *get things done*



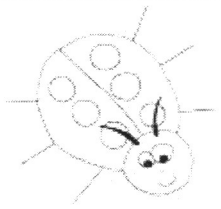
NoGo.Com Evaluation Criteria

- Does he breath?
- Does he get along with developers?
- Can he find serious, damaging and dangerous bugs?
- Available Now?



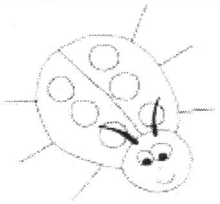
Fred

- Young and a bit innocent!
- Takes the challenge
- Starts right away



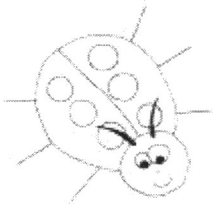
A day at Work

- Meet the team
- Brews, pizza, party
- Feel really welcome!



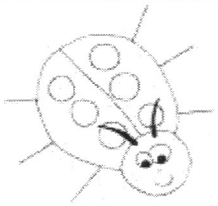
Developer bashing

- Fred was a bit surprised
- They just throw *bleep* over the wall
- "Unit Testing" HA
- They are just a bunch of arrogant *bleeps*



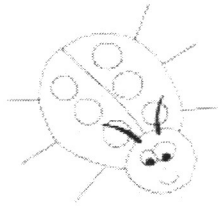
Product Management bashing

- pressure
- blame
- no requirements
- to many demo releases
- are we in the trade show business?



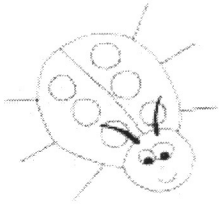
The Bug Review Meetings

- jovial atmosphere (at the start)
- people coming to the meeting
- arriving at arbitrary times
- some folks stick head



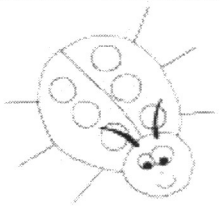
The fun begins

- a hundred or so bugs to review
- chaotic
- loudest person in the room becomes moderator



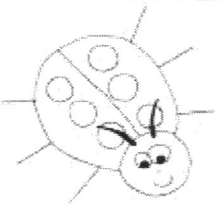
The fun continues

- endless debate between pairs of people
- everyone else waits and basically starts falling asleep
- "this is not a real bug"
- "this does not happen on my computer"



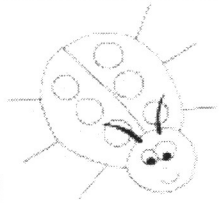
The fun continues (continues)

- discussions about how to fix the bug
- defer until we try this or try that
- unclear description of problems
- inconsistent - some descriptions very detailed - others vague



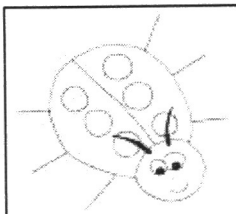
The test lab

- grab build off of the network
- nice name for build
  - version from Bills machine



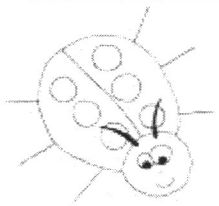
The test lab

- contrived install
  - walk on you hands, backwards and roll over
  - patch registry and manually copy many versions of a dll



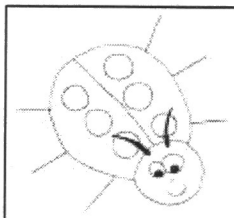
The test lab

- does not run
  - wrong person helping for wrong reason
  - good access to developers!



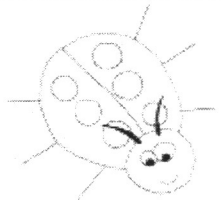
The test lab

- noisy
- lots of machines
- cool network game
- surfing between tests



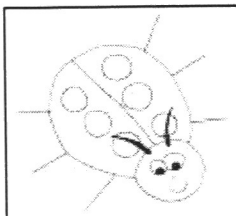
The Project Status Meeting

- regular periodic meeting
- just heard about it at the last minute
- what is project status



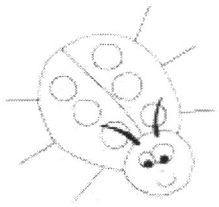
The Project Status Meeting

- development is almost finished so we decided to add a couple of new features that the folks figured
- super the sales guys will love that
- and what about our testing folks



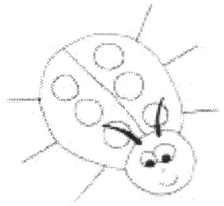
The Project Status Meeting

- so why the heck is the project so late
- "still bottlenecked in SQA"



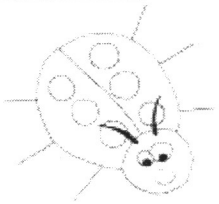
The Project Status Meeting

- Well Fred, we are counting on you to correct the situation
- Let me emphasize how critical it is that your department stop blocking the shipment



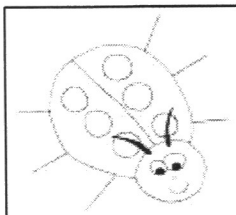
The Project Status Meeting

- In no uncertain terms
  - this is your responsibility
  - this is your job
  - we are counting on you
  - we expect you to get things done



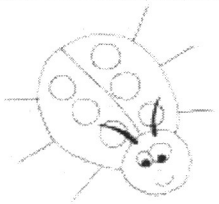
What to do?

- Fred left the meeting a bit concerned
- Heavy weight on his shoulders
- First reflex is to blame everyone
- The damn thing is that there was no status discussed at the status meeting
- It seemed more like a *witch hunt*
- Or was it more like a *lynching*



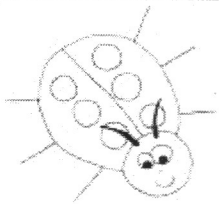
Learning about the E-SQA Manager?

- Fred's wife showed him the article in the career section of the newspaper
- "This article talks about a new kind of SQA Manager, they call him the E-SQA Manager!"



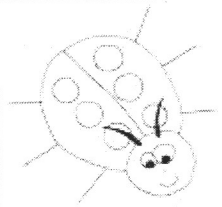
Learning about the E-SQA Manager?

- It sounds like his company is really making lot's of money, his team is happy and the damn thing is that their products really work
- What a revolutionary concept!



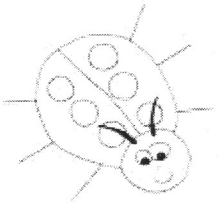
Curious about the E-SQA Manager?

- Fred spent some time reading the article
- Fred is curious about what this E-SQA guy does
- Fred has very little to loose



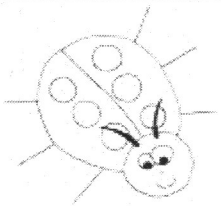
Curious about the E-SQA Manager?

- Zaps an email to the E-SQA manager asking if they can get together sometime to discuss how to get things done.



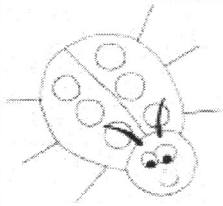
The email

- After Fred sent the email he stayed on his PC
- Fred is at his PC surfing a bit
- Looking at his favorite SQA site
- Suddenly his PC beeps "***You've got mail***"



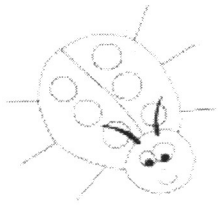
The response

- WOW
- The E-SQA Manager has replied to the email already
- And it was not an automatic reply either



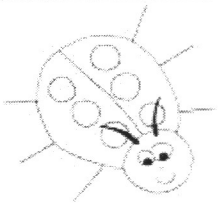
The email text

```
Fred,
I always enjoy meeting people who are new to
  SQA Management!
Welcome, welcome and welcome!
Why not pass by and see me sometime
I am always available at about 11:30 AM
Just confirm your favorite time with my team
  coordinator
See you soon
    • Signed The E-SQA Manager
```



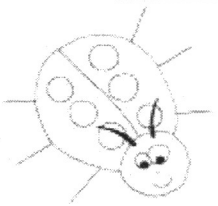
Sure Enough

- a bit dazzled by the speedy response
  - timestamps are within minutes of each other!
- Fred calls the E-SQA team coordinator
  - indeed it was expected and the meeting is set up for the very next day



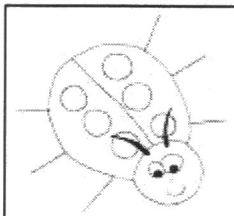
What Fred has in mind

- How can I loose
  - E-SQA manager can probably point me to a better company
  - Maybe even hire me himself
  - or maybe there is a remote chance of teaching me how to make things better at NoGo.Com



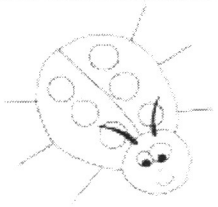
The E-SQA Manager

- Warm and welcoming
- Available
- Door is open
- People around office look busy
- Seems to have time



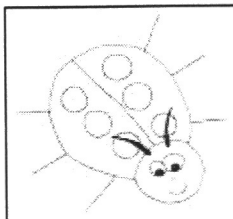
Welcome to Q - II

- E-SQA Manager said something that caught Fred off guard
- He said "Welcome to Quadrant II"
- And he shook my hand …
- what is going on?



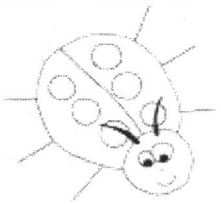
Q - II

- E-SQA Manager explained Q - II
- Steve Covey "7 Habits of Highly Effective People"
- A new paradigm of time management



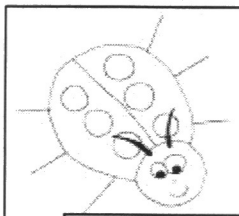
Quadrants

- What we do with our time?
- How do we use our time



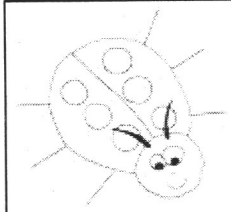
Quadrants

- Urgency
  - things that require and demand our attention now
- Importance
  - things that have significance, meaning and value



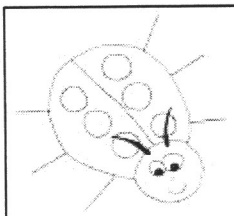
Time Management Matrix

Urgent Important	Urgent Not Important
Not Urgent Important	Not Urgent Not Important



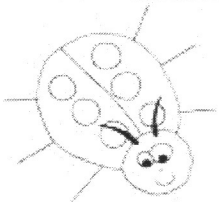
Four Quadrants

- QI
  - Urgent / Important
  - the pressing issue of the day that if it is not dealt with all other things become irrelevant!



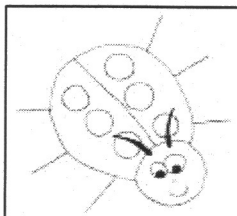
Four Quadrants

- QII
  - Not Urgent / Important
  - long term issues which have significance and which improve things
  - not pressing



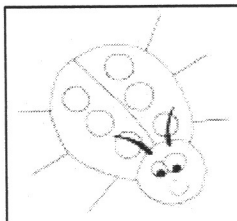
Four Quadrants

- QIII
  - Urgent / Not Important
  - those unimportant activities which take your immediate attentions
  - time stealers
  - some phone calls or unimportant interruptions



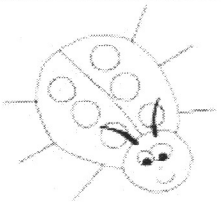
Four Quadrants

- QIV
  - Not Urgent / Not Important
  - Some wasteful mindless activities
  - watching a mindless TV show
  - reading a romance novel
  - some unreasonably popular web sites



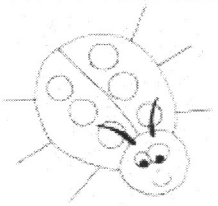
What about Vacations?

- should be in QII
- not QIV



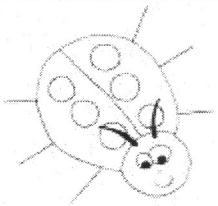
What about learning and teaching?

- should be in QII



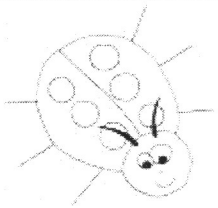
QII

- this is where you have to be in order to make a significant impact of your environment
- to make things better and have a lasting influence
- to be effective
- to really get things done!



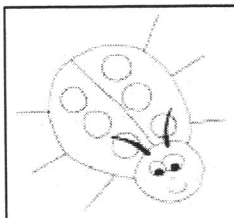
QII

- by seeing the E-SQA manager to learn about how to become a more effective SQA manager Fred is essentially in QII
- this is not an urgent activity but obviously important



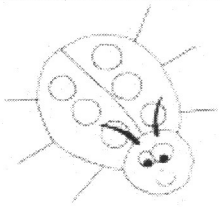
QII

- you can force yourself into QII by doing activities such as
  - retreats
  - writing a diary
  - taking time to get advice from other
  - sharpen the saw



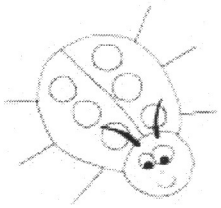
Servant Model

- The mindset of the effective manager is that of a servant leader
- You should work for your team
- Foster strength
- Asking them always "How can I help you"



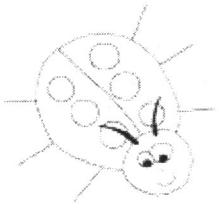
Inverted pyramid

- The effective SQA manager facilitates
  - Makes it possible for the team to succeed
  - Makes it possible for individuals to succeed



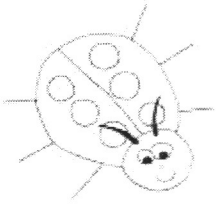
Fred's concern

- this all sounds a bit artsy
  - we are after all responsible for testing
  - we are very technical workers
  - are you saying that I should test software for my team?



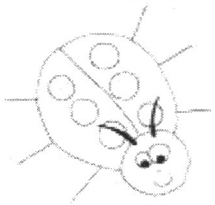
Fred's concern

- the E-SQA manager replies
  - you should not do the work for them
  - you should make it possible for them to do their job the best they can
    - "be the best that you can be"
  - the effective SQA manager keeps the road clear of obstacles
  - the effective SQA manager shields his team from the BS



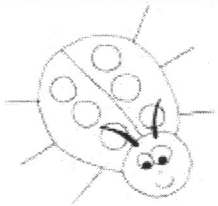
Fred's concern

- Fred says:
  - "So basically you recommend that I should isolate my team in order to protect them!"



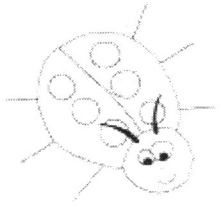
Fred's concern

- E-SQA Manager clarifies
  - "No, they must not be isolated from everything, they have to know the business context of everything going on. Just shield them from ignorant politics motivated by power struggles!"



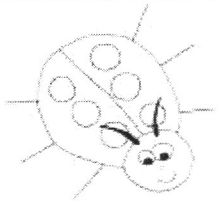
The 4 Ps

- E-SQA Manager explains that to get things done you and all of your team must understand the 4 Ps
  - Purpose
  - People
  - Practical Process



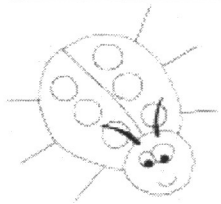
Purpose

- The E-SQA Manager asks the question:
  - "What is the purpose of SQA in your organization?"
  - "What do you think the purpose of SQA should be in your business?"



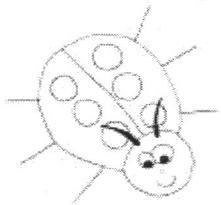
Purpose

- Fred is caught off guard by this question!
- "Our purpose is to find bugs before our customers do" was his quick reply
  - this was very clear to Fred
  - it was not just words -- he really believed!



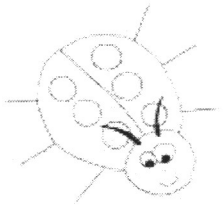
Purpose

- The E-SQA Manager explains
- "At our organization we have a special focus on helping increase the value of our organization, we look at things from a business prospective always keeping in mind the key stakeholders, *our customers, employees and shareholders*"



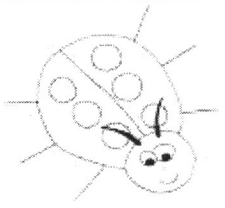
Purpose

- "The role of SQA in our organization is to provide objective input to facilitate business decisions (wise smart and good decisions)"
- "SQA keeps internal stakeholders aware of all the issues that relate to shipping a product"
- Some friends of mine in Washington State have a similar purpose for the testing role!



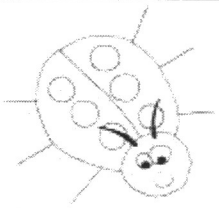
Microsoft® SDD Team Model

- Testing Role defined simply and clearly in the sense of SQA not corporate QA
  - ensure all issues are KNOWN to team
  - develop testing strategy and plans
  - *FACILITATE BUSINESS DECISIONS*
  - *PROVIDE OBJECTIVE INFORMATION*



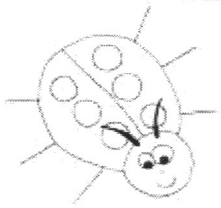
Purpose

- To be an effective SQA manager you must be an "*on purpose*" SQA manager
  - *On Time*
  - *On Quality*
  - *On Budget*
  - are meaningless unless you are *On Purpose*



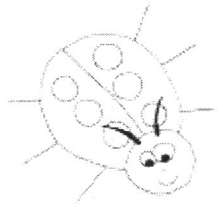
Purpose of SQA Team

- The E-SQA Manager asked Fred
  - "how does SQA fit into the development process at NoGo.Com"



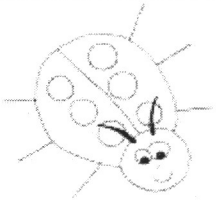
Purpose of SQA Team

- Fred after some reflection replied
  - "it seems that we are a testing job shop"
  - "we get all sorts of stuff from developers and try and test it"
  - "we give them feedback about how the testing went"



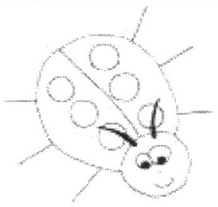
Purpose of SQA Team

- The E-SQA Manager Summarized
  - So you, as almost every SQA manager in the software industry, are providing a service to a development team or organization
  - and you know you cannot test quality into a product!



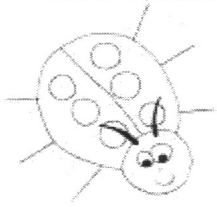
Purpose of SQA Team

- The E-SQA Manager on Service Model
  - We have a service model for SQA generalized to Software Engineering
    - Metrics collection tracking as a service
    - Analysis as a service
    - Configuration management and construction as a service
    - Integration and System testing services
    - Formal inspection services



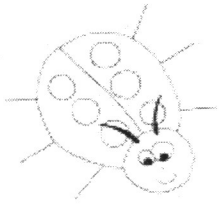
Purpose of SQA Team

- On the E-SQA Service Model
  - we like to ensure that the customers of our service are Raving Fans!
  - A Raving Fan customer is a customer who is not just satisfied, but is so excited that they are like a walking, talking sales promotions department!
  - If you really want a booming business you need raving fans!



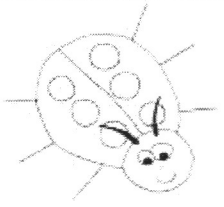
Getting to Raving Fan Service

- Fred agreed
  - presently NoGo.Com's SQA Team does not have Raving Fan's



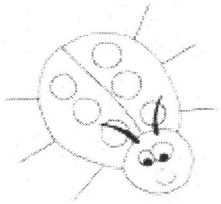
Getting to Raving Fan Service

- Fred suggested
  - the SQA Team was obviously a service to development projects
  - most SQA Team members did not treat the development people like a customer!



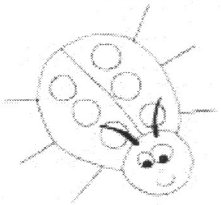
Getting to Raving Fan Service

- Instead of trying to cultivate and develop Raving Fan Customers the current state was based on confrontation



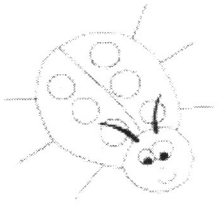
Getting to Raving Fan Service

- Some steps to get there
  - Read "Raving Fan's" by Ken Blanchard and Sheldon Bowles



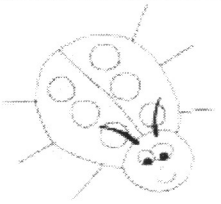
Getting to Raving Fan Service

- Three Secrets to Raving Fan SQA
  - DECIDE WHAT YOU WANT
  - DISCOVER WHAT THE CUSTOMER WANTS
  - DELIVER PLUS ONE PERCENT



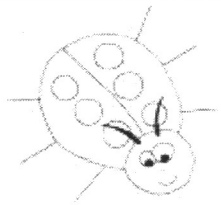
Getting to Raving Fan Service

- DECIDE WHAT YOU WANT
  - create a clear vision of what you want the SQA department to be like at some time in the ideal future
  - the vision should be of the internal customer using the services offered by the team
  - have a good idea of what is excellence!



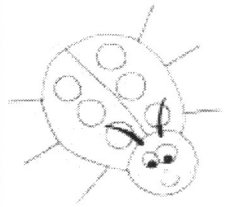
Getting to Raving Fan Service

- DISCOVER WHAT THE CUSTOMER WANTS
  - identify who in the organization are your customers
    - not just the leads and managers but all those touched by your service



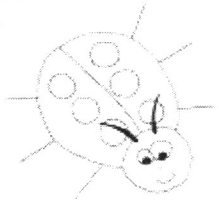
Getting to Raving Fan Service

- DISCOVER WHAT THE CUSTOMER WANTS
  - on an individual basis find out what they expect from your team, what type of products, services, information, data etc. (be polite and try to get specific not vague input)



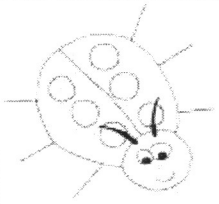
Getting to Raving Fan Service

- DISCOVER WHAT THE CUSTOMER WANTS
  - Developers what clear bug descriptions which help them find and correct the associated defects
  - Management wants the status of the product in terms of what works, what does not work, how close is the product to something which can he shipped!
  - Help Desk support people want good descriptions of a work around for all of the bugs we decide to leave in the product so that they can help end users



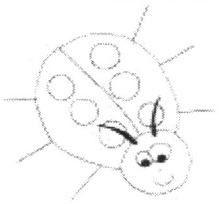
Getting to Raving Fan Service

- DISCOVER WHAT THE CUSTOMER WANTS
  - as required adapt your vision to mesh with that of the internal customers
    - if there are wide gaps and gaping holes consider redirecting the customer to some other department or organization (do not try to be all things for all people)



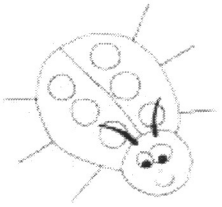
Getting to Raving Fan Service

- DISCOVER WHAT THE CUSTOMER WANTS
  - sales may expect SQA to provide platform recommendations
    - in this case you should redirect the customer to Product Management and make sure that it is clear that you do not provide that service



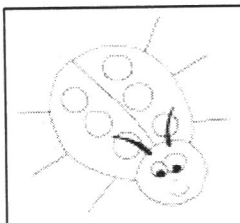
Getting to Raving Fan Service

- DISCOVER WHAT THE CUSTOMER WANTS
  - end user support may expect SQA to provide a work around for bugs left in the product
    - if this is a service you intend to offer make sure it is part of your teams work on any project and take care to document in a clear straight forward manner any work around



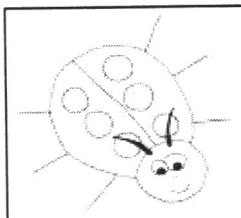
Getting to Raving Fan Service

- DISCOVER WHAT THE CUSTOMER WANTS
  - end user support may expect SQA to provide a work around for bugs left in the product
    - do we provide this in a language our end users should understand or in a language our customer service representatives understand!



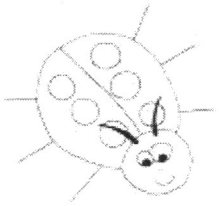
Getting to Raving Fan Service

- DELIVER THE VISION PLUS ONE PERCENT
  - With a vision in hand establish a strategy which will allow you to deliver



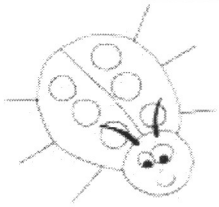
Getting to Raving Fan Service

- DELIVER THE VISION PLUS ONE PERCENT
  - Baby steps are the order of the day!
    - We do not jump from the current state to the ideal vision in one step



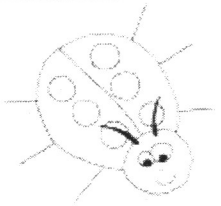
Getting to Raving Fan Service

- DELIVER THE VISION PLUS ONE PERCENT
  - On each project implement some process change which brings you closer to the idea
    - consistency, consistency, constancy
    - Consistency creates credibility!



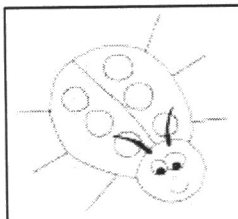
Getting to Raving Fan Service

- DELIVER THE VISION PLUS ONE PERCENT
  - Inconsistency can destroy a lot of built up good will and productivity
    - » bug reports descriptions varying depending on who wrote the report can make the whole team look incompetent even if it is only due to the fact that one junior tester was taking a great initiative to help out in an area he was not familiar with



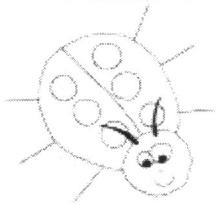
Getting to Raving Fan Service

- DELIVER THE VISION PLUS ONE PERCENT
  - The way you treat one project should be the same way you treat all projects!
    - » Do not try to add all sorts of new process steps or deliverables until you can consistently deliver what is presently required



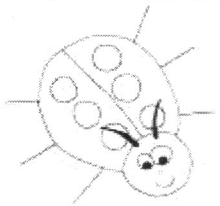
Getting to Raving Fan Service

- DELIVER THE VISION PLUS ONE PERCENT
  - » example BABY STEPS
  - » Bug Graph - update it once a week, consistently, accurate, punctual, available
  - » Bug Graph - update daily ONLY after weekly is working perfectly
  - » Bug Graph - on demand in real time ONLY after daily is working perfectly



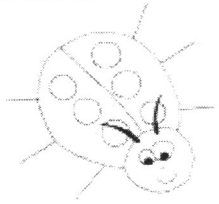
Getting to Raving Fan Service

- DELIVER THE VISION PLUS ONE PERCENT
  - Meet first and then exceed customers expectation
    - if the customer expects a great test plan outline with coverage of all features, ensure this is consistently met before adding requirement tracing or usage scenarios



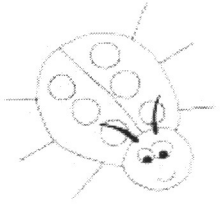
Getting to Raving Fan Service

- DELIVER THE VISION PLUS ONE PERCENT
  - Figure out how to measure whether you are generating Raving Fan Internal Customers
    - do they come back for more help, advise, guidance
    - do they use the deliverables
    - are they excited?
    - are they having fun?



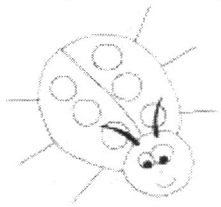
Getting to Raving Fan Service

- Of course getting to Raving Fan Service is not a one man job
- The leader has to have the vision but the vision must be consistent with the purpose
- And then you have to get the people involved!



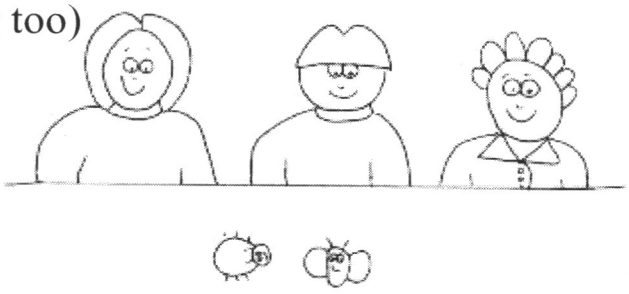
People

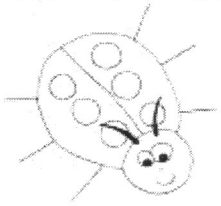
- The second P is "People"
- It is all about people!



People

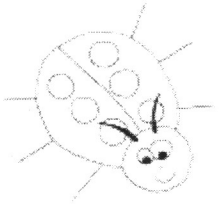
- It's all about people! (and the occasional bug too)





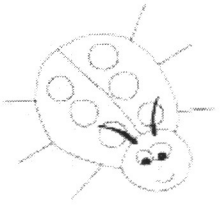
People

- The E-SQA Manager asked Fred:
  - How would you describe the attitude of the people in your team?



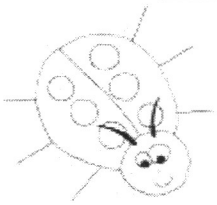
People

- Fred reflected:
  - A bit confrontational, definitely acting like typical techies
  - Good team spirit
    - you should see them when they get together over a brew to complain about the guys in development



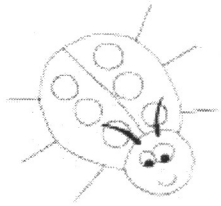
People

- The E-SQA Manager commented:
  - I cannot help but notice that you did not mention anything about your people being excited about their job!
  - Are they keen and enthusiastic
  - Do you see the bounce in their step
  - Do they keep bringing in CVs of all their friends from other companies?



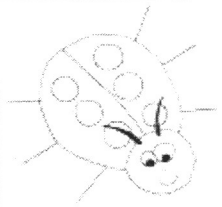
People

- Fred's look gave a clear message:
  - no or at least not all of them and certainly I never get referrals from within the team
  - In fact -- I am convinced that some of them do not want to be in SQA at all -- some of them probably got shoved into SQA because they were lousy developers -- or so I suspect



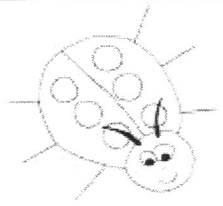
People

- E-SQA Manager:
  - It certainly makes a big difference if people are in SQA because they want to be in SQA rather than otherwise
  - with a smaller team of people who liked to work in SQA you can be more productive that with a larger team including staff who did not want to work in SQA



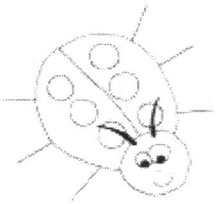
People

- If you want to get "Raving Fan Customers" for SQA you will need to have "Gung Ho" staff to deliver the service!
- You should read "Gung Ho!" also by Ken Blanchard and Sheldon Bowles which talks about how to increase Productivity, Profits and Prosperity by having a "Gung Ho!" team!
- People who are excited about going to work and being productive!



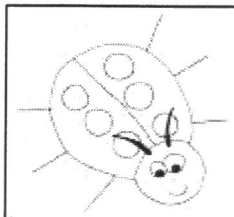
People

- The basics about "Gung Ho" staff
  - Worthwhile work
    - important
    - leading to shared goals
    - value driven



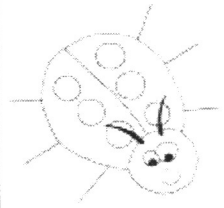
People

- The basics about "Gung Ho" staff
  - In control of achieving the goals
    - well marked territory
    - listen to and respect
    - able but challenged



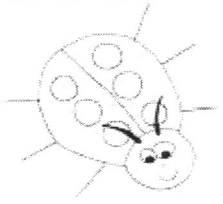
People

- The basics about "Gung Ho" staff
  - Cheering others on
    - feedback timely and true
    - keep score and cheer progress
    - enthusiasm



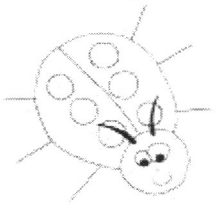
People

- The basics about "Gung Ho" staff
  - As you go reassess and redirect



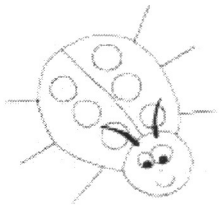
People

- E-SQA Manager
  - When I was starting out in SQA management I believed that "Happy people are productive"
  - I used to take the gang out for a beer or to the ball game
  - We partied and had a great team spirit



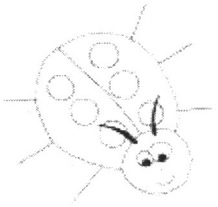
People

- E-SQA Manager
  - But it didn't make things work better at the office
  - in fact in some ways it was worse because people were more focused on the social extra curricular activities than on the job at hand!
  - Something was missing



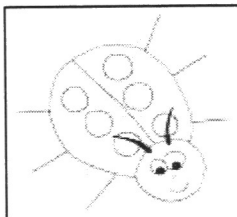
People

- E-SQA Manager
  - Then I learned that the more appropriate model was that "Productive people are happy"
  - If people have a clear important role, are allowed to succeed, and are given solid timely feedback
  - So now I focus my management efforts on my people and ensuring that they are and want to be productive!



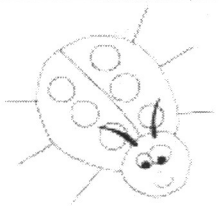
People

- E-SQA Manager
  - We still have parties to celebrate important achievements and project milestones and even sometimes to highlight individual success, but we are celebrating the productivity of people not celebrating to make people productive!



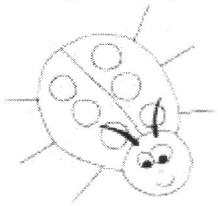
People

- E-SQA Manager
  - Feedback is the "Breakfast of Champions"
  - Remember feedback is about the behavior not the person
    - next time we can do better by trying this instead of that



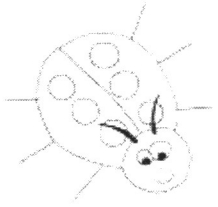
People

- E-SQA Manager offered some tips about managing people
  - Different Strokes for different folks at different times!
  - Adapt leadership style to the situation
  - Choose leadership styles deliberately!



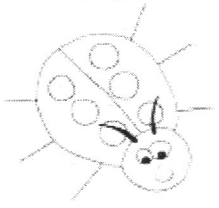
People

- 4 Basic Situational Leadership Styles
  - Directing
  - Coaching
  - Supporting
  - Delegating



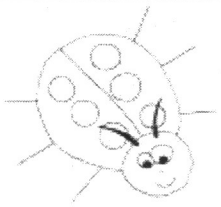
People

- Directing Leadership
  - tell people specifically what to do
  - provide constant feedback, praising and redirection
  - used when someone is new to a task and uncertain as to how to successfully achieve the task
  - used sometimes in an emergency situation



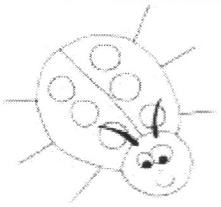
People

- Coaching Leadership
  - provide guidance and advice on how to achieve goals based on input from staff
  - does not need close direction but needs to learn how to achieve success
  - used when someone has a proven track record but is new to this specific task
  - team member is mature enough to ask for assistance



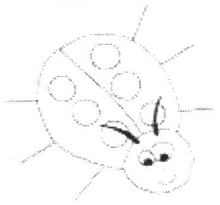
People

- Supporting Leadership
  - staff participate in decision making with leader
  - staff works with leader to establish goals and milestones
  - person can work quite autonomously but needs leaders help
  - team member is mature



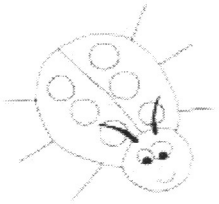
People

- Delegating Leadership
  - staff is given broad goal and parameters and then takes full ownership of task
  - constant feedback is not required and the need for it is driven by team member mostly to confirm that big picture business drivers have not changed
  - team member is capable of successfully achieving assignment
  - team member is autonomous



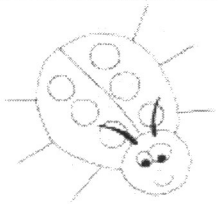
People

- Force them into QII
  - weekly activity reports + SQA News
  - *THIS IS NOT PROJECT STATUS*
  - what the person did, what activities did they participate in, who did they interact with, about what was done not what is planned to do!
  - Discretionary time



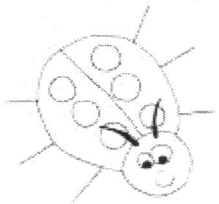
People

- Capture some information frequently and on purpose
  - at least once a week, probably when the manager reads the Weekly Activity Report
    - for each team member capture at least one comment
    - place in a spreadsheet one text field per week
    - answer what did the person do last week that is excellent



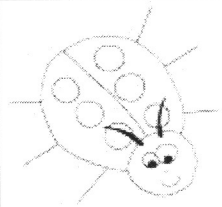
People

- answer what did the person do last week that could be improved or needs redirection
- answer what did I do to help that person succeed in the assigned work
- answer what could I have done better to help that person succeed in the assigned work
- answer did I provide the employee with timely feedback, praising or redirection



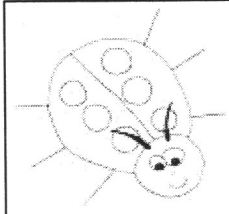
People

- takes only a few minutes per team member per week but is a gold mine
- this spreadsheet, plus weekly activity report, plus project status information is great



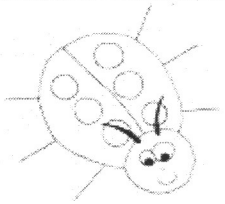
People

- take the time about once every couple of weeks to sit down, have a coffee and review the goals
  - are they still relevant
  - are they still the right goals for this person
  - what progress have we made toward these goals
  - what can the manager do to help
  - what can the employee do to help



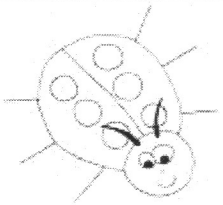
People

- meet with the people and review their goals
  - have one page of point form written goals for each person



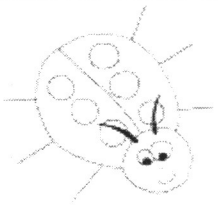
People

- professional/individual development goals
- project/role related goals
- change the goals if the business context changes!
- Check out measures!



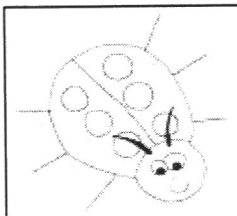
People Employee Evaluations

- Why wait a year?
  - ZAP - SURPRISE ATTACK style
  - Save up goodies for a year
  - Let it all out in one painful shot



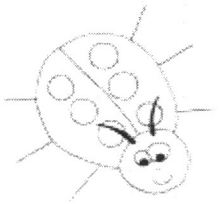
People Employee Evaluations

- Best is immediate
  - When something happens provide feedback
  - Be relevant and aware
    - direct!
  - Don't avoid bad and over emphasize good
    - remember behavior not person!



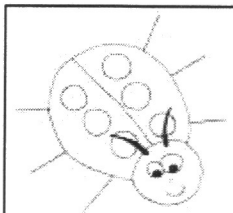
People Employee Evaluations

- Always Collecting Data
  - Normal work deliverables
    - just getting the job done right deserves feedback (don't wait for something extraordinary)
  - Keep additional or special notes on file about the person



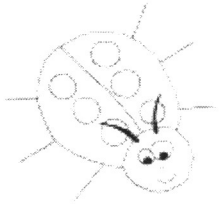
People Employee Evaluations

- Employees may be a bit worried about feedback
  - watch out for big brother is watching syndrome
  - make sure employee is comfortable with frequency of feedback (to much or too little?)
  - all employees should be treated uniformly
    - if any one employee is getting all the feedback a little paranoia may set in!



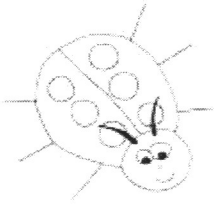
People Employee Evaluations

- Example of feedback regarding bad behavior
  - ZAK a valued developer sent an angry email company wide criticizing sales about their weak understanding of how to clearly specify requirements - a job ZAK feels should be left to product managers.



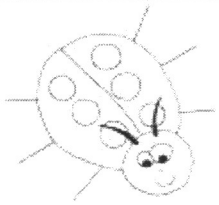
People Employee Evaluations

- Example of feedback regarding bad behavior
  - This email further ticked off SALES. Improvements in relations between the departments are out of the window
  - Feedback is in order -- you are the boss -- what do you do?



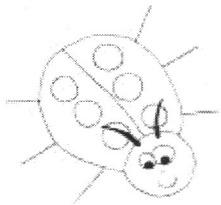
People Employee Evaluations

- Example of feedback regarding bad behavior
  - ZAK, you are one of the finest engineers who has worked for this company! The software you write is always top quality, and you have a great track record.



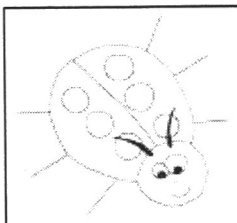
People Employee Evaluations

- Example of feedback regarding bad behavior
  - That's why I was surprised to find that you were the author of an email which in a very unprofessional manner criticizes the sales department regarding their understanding of product requirements. Frankly, the management in sales is quite concerned!



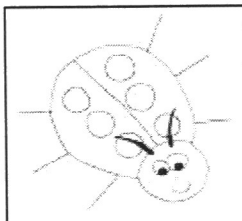
People Employee Evaluations

- Example of feedback regarding bad behavior
  - I would expect you to be able to make the same point in a professional manner by doing …
  - *** how can manager and employee come up with a way to deal with this issue better ***



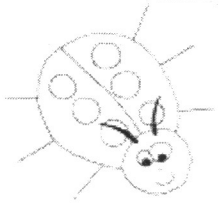
People Employee Evaluations

- Example of feedback regarding bad behavior
  - In order to improve your behavior try the following:
    - wait and re-read emails (especially critical ones) before sending them out
    - here are some examples of tactful ways to make the same point
    - there is a seminar in two months regarding … I'll make sure you can attend…



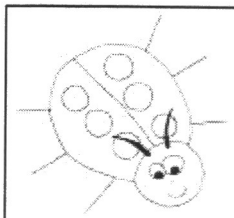
People Employee Evaluations

- Clear Goals an Example
  - in the next year you will ensure that all employee evaluations in your team are done on time or ahead of schedule except in the case of a delay explicitly requested by the employee.
  - in the next year you will ensure that before any programming activity takes place, in the project you are managing, an associated specification activity has been completed.



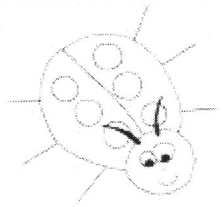
People Employee Evaluations

- 360 Evaluation
  - feedback to managers by team members
  - customers
  - suppliers
  - up and down hierarchy



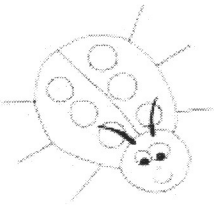
People

- SQA Bill Of Rights
  - Right to know the business context for assigned activities. Staff must be able to answer the question: "What is the business reason for doing this assigned activity?"
  - Right to know what it means to finish assigned activity



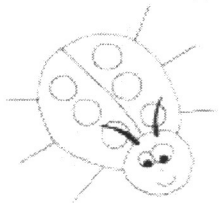
People

- SQA Bill Of Rights
  - Right to know what software being tested is supposed to do and if assumptions are to be made the right to double check with product or development management before testing activity starts
  - Right to get software which the development team honestly believes works



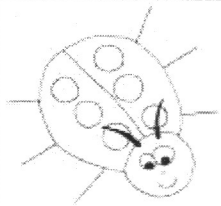
People

- SQA Bill Of Rights
  - Right to have fun at work
  - Right to learn new work methods, techniques and technologies
  - Right to try out innovations which may fail



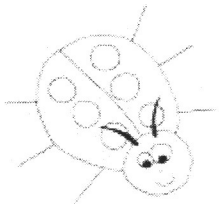
People

- SQA Bill Of Rights
  - Right to speak directly with developer responsible for code being tested
  - Right to report a bug discovered even if it may already be in the bug list (never loose a bug)
  - Right to know how much effort to spread across a testing assignment



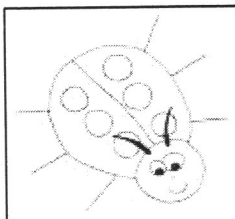
People Earn Respect

- you must earn respect
- from peers
- from customers
- from stakeholders



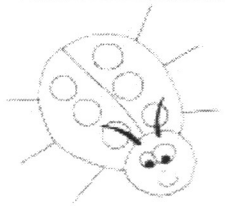
People Human Side

- The human side of the equation is the most unpredictable
  - there is always something you do not know
  - problems at home
  - peer personality conflicts
  - poor self-esteem of team members
  - be sensitive, clear, firm and honest



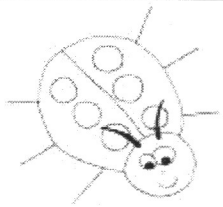
Practical Process

- Baby Step Innovation
  - on each project implement two innovations
    - technical or technology innovation
    - process or management innovation



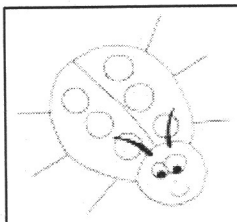
Practical Process

- Baby Step Innovation
  - ensure that all projects operate within a couple of innovations from each other
  - every project is a pilot project for some innovation
  - pull innovation if it does not look promising after being given a fair chance



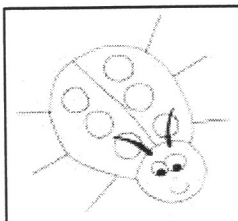
Practical Process

- Specifications
  - try to avoid long written dissertations
  - keep to as many tables, state diagrams and schematics as possible
  - only specify what is specific
    - current state
    - input
    - outcome



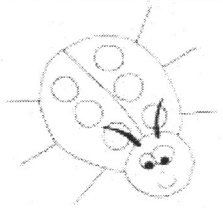
Practical Process

- Specifications
  - keep document and terse and clear as possible
    - faster to write
    - faster to review
    - less likely to be misinterpreted
  - tie to requirements
  - reference in test plans



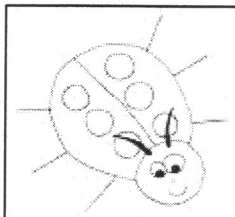
Practical Process

- Have effective meetings
  - as few people as possible
  - efficient use of time
  - separate project meetings from team meetings
  - team meetings invited guests, info from exec



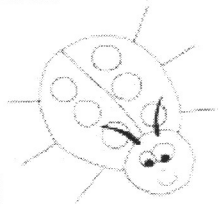
Practical Process

- Be punctual at all times
  - an SQA Manager must set an example
  - be on time in all matters at all times
  - if you expect your people to deliver on time you must deliver on time
  - make sure administrative issues, pay issues and all people issues are dealt on time



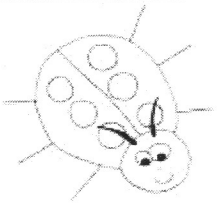
Practical Process

- it is really quite simple
  - all you have to do is always MAKE AND KEEP COMMITMENTS!



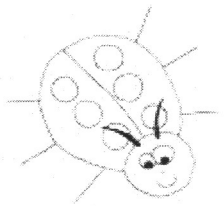
Practical Process

- one page reports
  - project status reports, short term goals, risks
  - department progress reports
  - effort per project past projected versus actual
  - graphs of bug status
  - all on SQA INTRANET



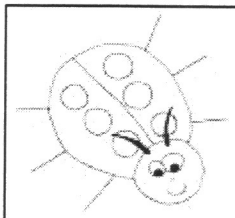
Practical Process

- Process improvement discretionary time stuff
- P Team Present 0-6 months
  - improve present process, use of technology
- F Team Future 6-12 months
  - guiding light for future, new process and technologies



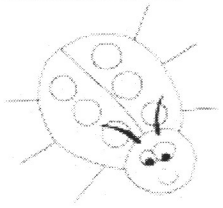
Practical Process

- Off Site SQA Team Retreats
  - focus on what can be changed
  - look at past recent experience
  - all team members come prepared
  - capture results and recommendations



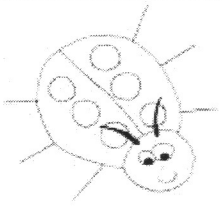
Practical Process

- Project Post Mortem Review
  - key team members bring lists
    - 5 excellent things to be encouraged in future projects
    - 5 things that could have been done better and should be improved in future projects
    - a couple of specific recommendations or personal comments



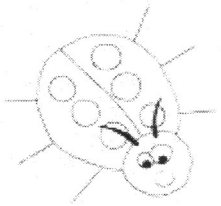
Practical Process

- Project Budgets
  - need to know how many resources to commit to project
  - effort based and spread across project in rational way
  - reviewed and revised frequently with project stake holders



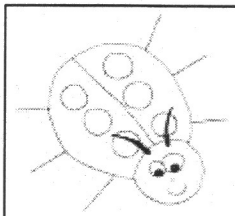
Practical Process

- Staffing
  - have access to contract resources to increase capacity for short bursts during crunch periods
  - good test scripts and plans help
  - have permanent staff coach contract staff for leverage



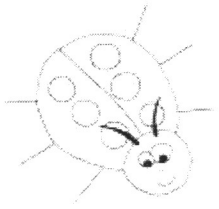
Practical Process

- Platform testing
  - lab management, different OS on different days rotated through project
  - have a test lab manager



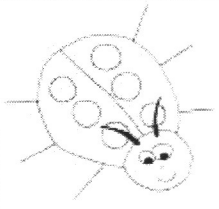
Practical Process

- Peer reviews on bug data being entered
- ensure clear description!
- double check



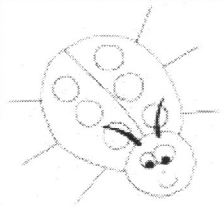
Practical Process

- Use formal inspections
  - test plans against specifications
  - any documents prepared!
  - use Gilb based methods!

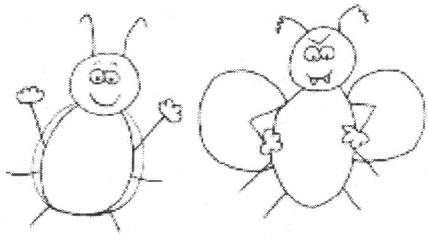


Practical Process

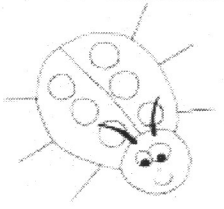
- Test Strategy for each test cycle
  - development lead identifies technical risks
  - product manager identifies commercial risks
  - also identify what not to test in a particular cycle



About Bugs

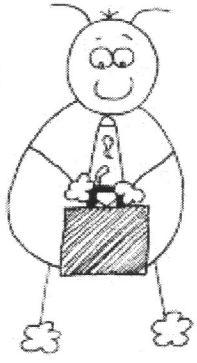


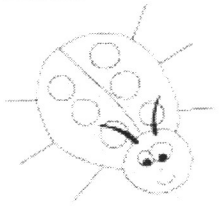
Bugs are not Good or Bad



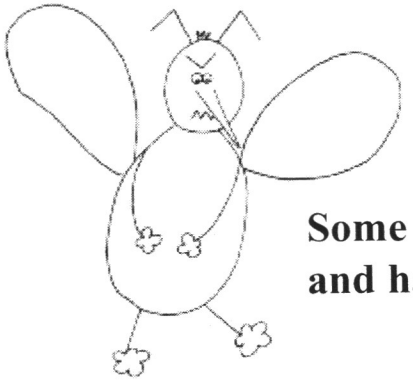
About Bugs

Some bugs are important and have a high priority!

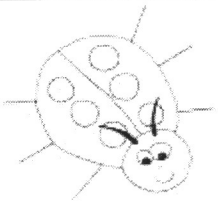




About Bugs

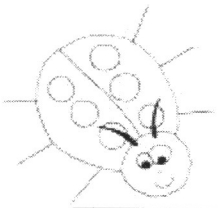


Some bugs are dangerous and have a high severity!



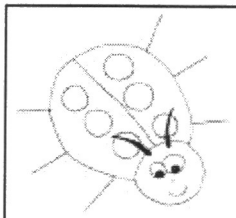
About Bugs

- Setting the priority and severity of a bug is a business decision
- Changing business conditions impact the priority and severity of a bug!
  - Always review previous decisions in light of changing business context
  - ensure staff assigning priority and severity are aware of all relevant business drivers



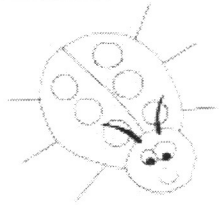
Bug Quadrants

Urgent Severe	Urgent Not Severe
Not Urgent Severe	Not Urgent Not Severe



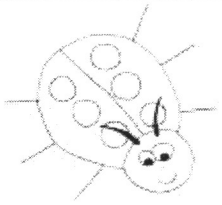
Business Decisions

- SQA:
  - objective input
- Development:
  - technical implementation
- Product Management:
  - customer driven requirements



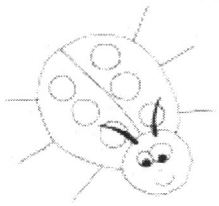
Quadrant Changing

- Same technical bug can be in a different quadrant depending on the business context
- Monitor business drivers!
- Focus find and fix quadrant -1- bugs high priority/high severity



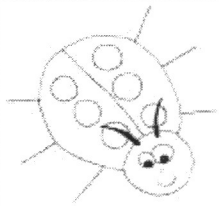
Practical Aspects

- Bug Review Meeting
  - bug review meetings are among the most important activities in a software development project
  - decisions are made as to what the priorities are for all bugs in the system
  - when business conditions have changed a review of all lower priority bugs is needed to reclassify as required



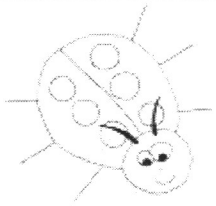
Practical Aspects

- Bug Review Meeting
  - all attendees should have access to the bug list before the meeting and have sufficient lead time to know what priorities they would assign to the bugs
  - assume approximately 2 hours are required per person reviewing about a weeks worth of NEW UNCLASSIFIED BUGS



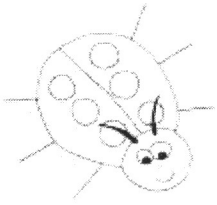
Practical Aspects

- Bug Review Meeting
  - if more than 2 hours review time is needed then increase the frequency of your bug review meetings
  - as few people as possible should be attending the bug review meeting
    - development lead
    - product manager
    - sqa lead



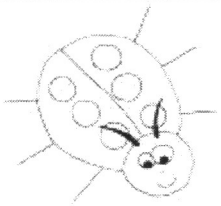
Practical Aspects

- Bug Review Meeting
  - other staff should be available on demand to help clarify technical or business issues related to the bug
  - run bug review meeting objectively
    - avoid finger pointing
    - avoid assigning blame
    - be objective and unemotional



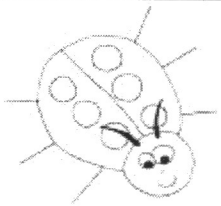
Practical Aspects

- Bug Review Meeting
  - agree on an order to review the bugs
    - for low volume the easiest is sequentially
    - logical order could be by function and then from most dangerous to least dangerous severity
    - order should be rational
  - moderator should be diplomatic
  - training in how to run a meeting



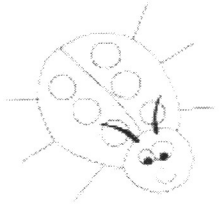
Practical Aspects

- Bug Review Meeting
  - can be run similar to a defect logging meeting of a formal inspection (Gilb style) but with more discussion encouraged to come to a business decision especially on gray subjective areas about what product users would could or should do!



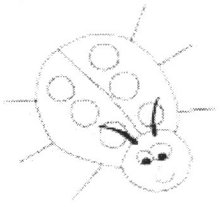
Practical Aspects

- Bug Review Meeting
  - all stakeholders should be notified of decisions made in Bug Review Meetings
    - developers
    - testers
    - technical writers
  - ensure a process exists to notify staff that they have work to do related to fixing a problem.



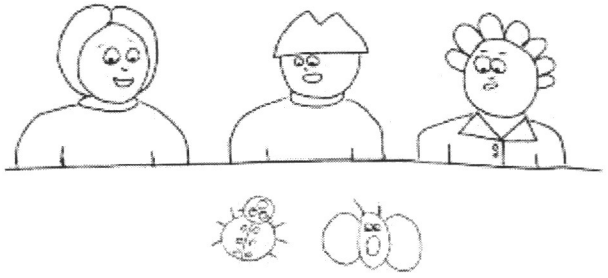
Practical Aspects

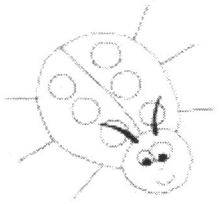
- Bug Review Meeting
  - although email and electronic notification is very popular I recommend communicating in person (or by phone or by some form of electronic conferencing)



Finished?

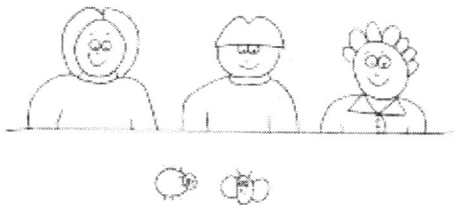
- How do you know you are finished?

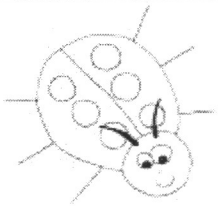




You know you are finished when …

- … the only bugs left are the ones that Product Management and Development agree are acceptable (based on objective SQA input) ...

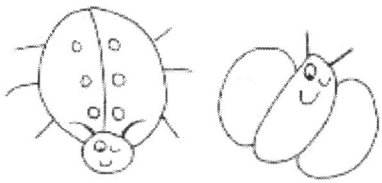


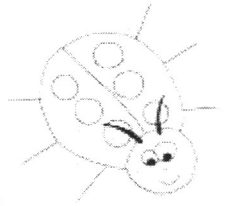


You know you are finished when …

- … the only bugs left are the ones that Product Management and Development agree are acceptable (based on objective SQA input) …

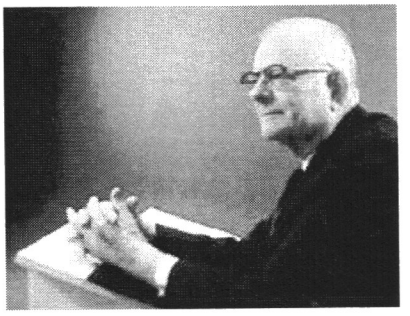
At least for now!

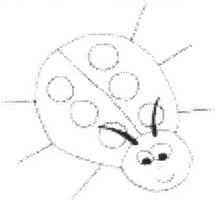




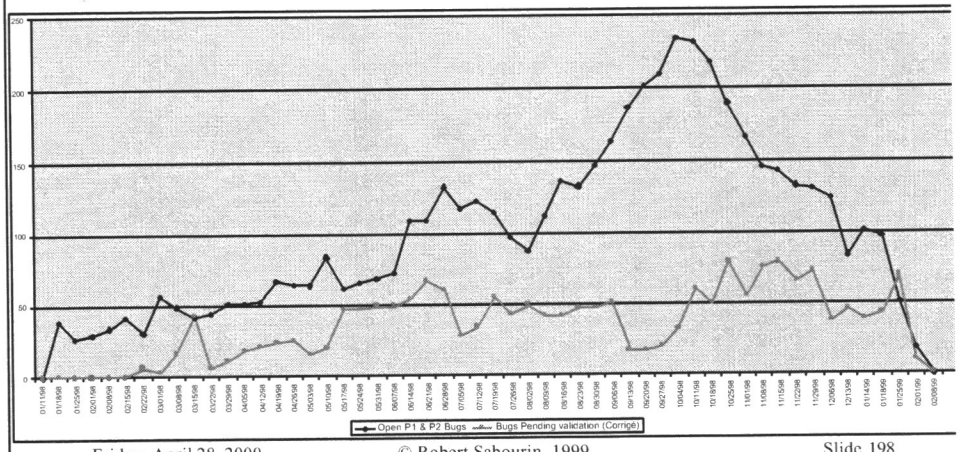
Dr. Edwards Deming

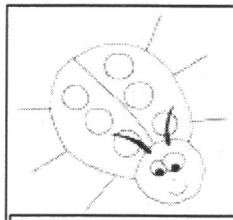
- "Management of quality needs quality management"



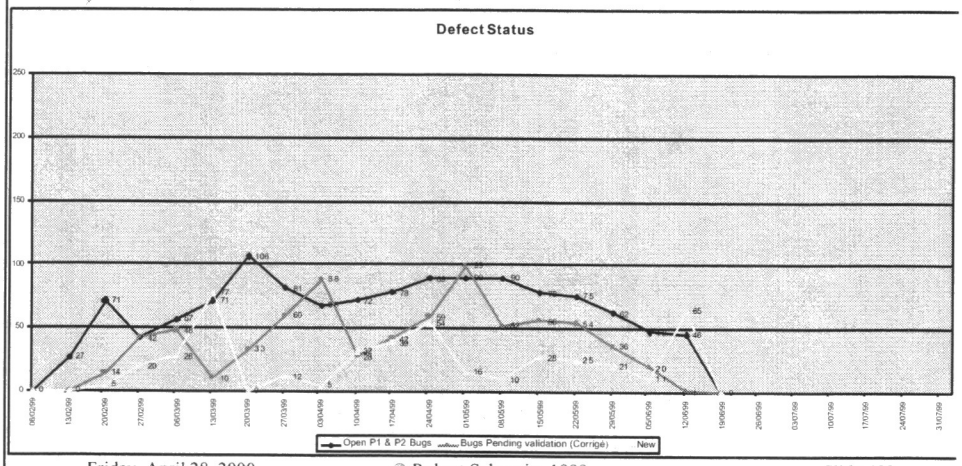


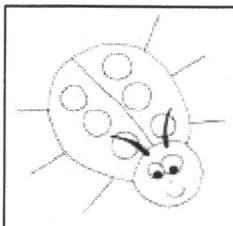
CNC 2.10 Final



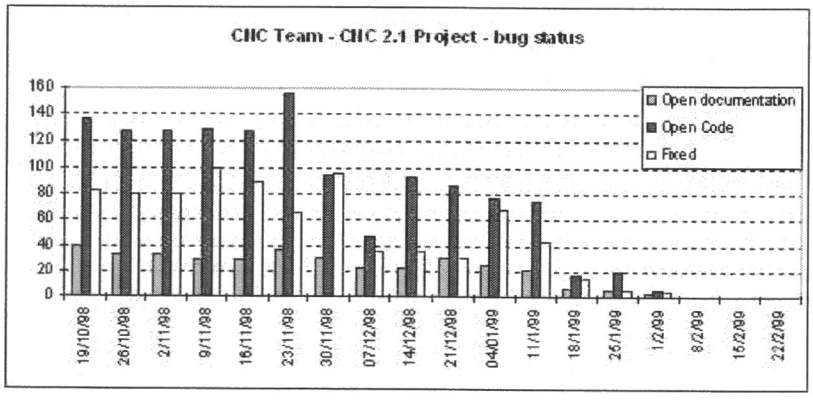


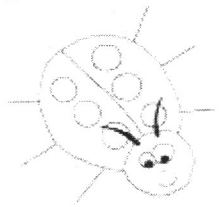
CNC 2.20 Final



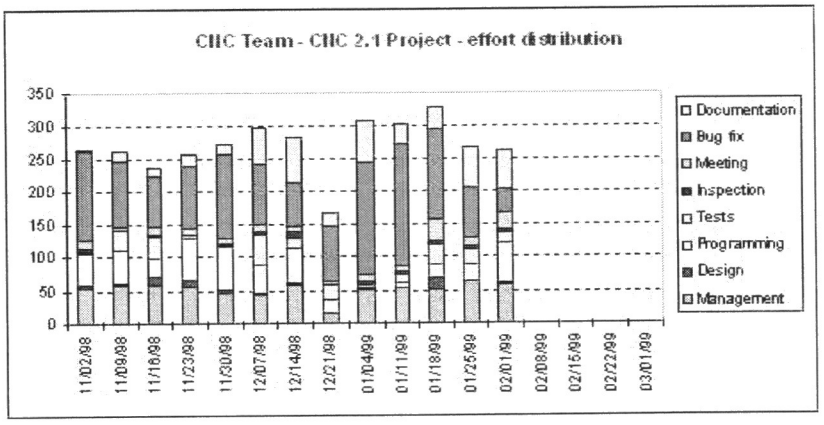


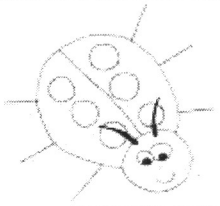
CNC 2.1 Project Data



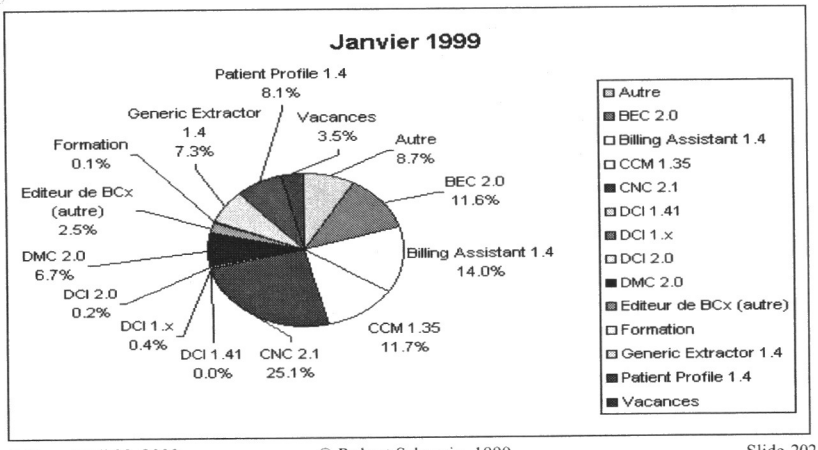


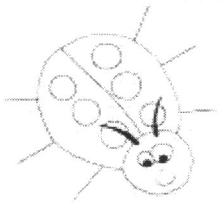
CNC 2.1 Project Data



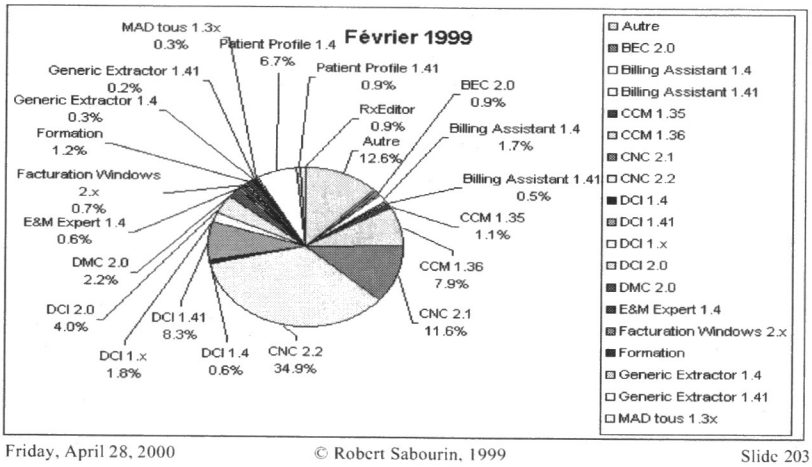


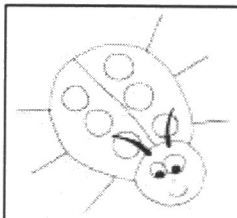
SQA Effort Distributions





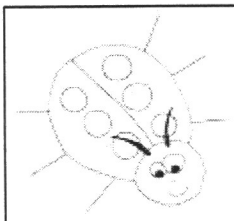
SQA Effort Distributions



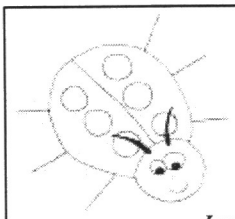


Bibliography

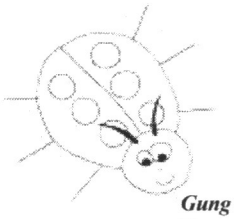
Empowerment Takes More than A Minute
Kenneth H. Blanchard, John P. Carlos, Alain Randolph
Barrett-Koehler, 1996 ISBN: 1881052834
Description: *Empowerment Takes More Than a Minute* explains that empowerment is not "giving power to people". Rather, it is "releasing the knowledge, experience, and motivation they already have."
Drawing on ten years of research and consultation with a wide variety of companies, the authors define the three essential keys for achieving true empowerment. With these keys in place, employees will be able to make good business decisions for themselves rather than replying on superiors, and be able to draw from each other the skills and knowledge necessary to get the job done.



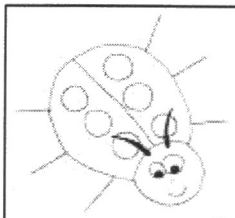
Bibliography



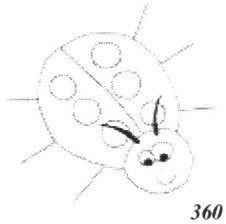
Leadership and the One Minute Manager : Increasing Effectiveness Through Situational Leadership
Kenneth Blanchard, Patricia Zigarmi, Drea Zigarmi
William Morrow & Company, 1985 ISBN: 0688039693
Description: The third book in the extraordinary One Minute Manager series goes straight to the heart of management as it describes the effective, adaptive styles of Situational Leadership. In clear, simple terms it shows why "nothing is so unequal as the equal treatment of unequals," while it teaches you how to become flexible and successful leader.



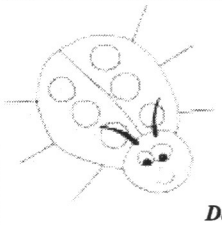
Gung Ho! Turn on the People in Any Organization
Kenneth H. Blanchard, Sheldon Bowles
William Morrow & Company, 1997 ISBN: 068815428X
Synopsis: The authors of *Raving Fans* now offer an invaluable new strategy for creating enthusiastic employees. Here is the story of how two managers saved a failing company and turned in record profits with record productivity. These three core ideas of *Gung Ho!* are surprisingly simple: worthwhile work guided by goals and values; putting workers in control of their production; and cheering one another on. Also includes a clear game plan with a step-by-step outline for instituting these ground-breaking ideas that boost enthusiasm and performance and usher in astonishing results for any organization.



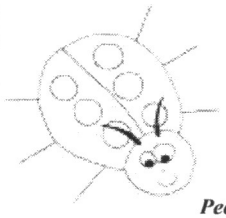
Software Inspection
Tom Gilb, Dorothy Graham
Addison-Wesley Publishing Company, 1993 ISBN: 0201631814
Synopsis: Gilb and Graham show software professionals how to achieve high-quality software through inspection. They show how to do a formal review of documents to find errors, giving effective statistical process improvement. The book includes many examples and case studies based on actual experience at IBM, AT&T, McDonnell Douglas, and other companies.



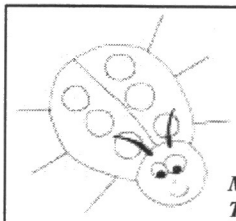
360 degree Feedback : The Powerful New Model for Employee Assessment & Performance Improvement
Mark R. Edwards, Ann J. Ewan
AMACOM, 1996 ISBN: 0814403263
Synopsis: Once, the only opinion that counted in a performance evaluation was the supervisor's. Now, as teamwork and employee empowerment become more prevalent, organizations are turning to "360° feedback" -- a multi-perspective approach perfected by the authors of this pioneering "how-to" guide. Drawing on 20 years of field research, they show how to design and implement the 360° feedback method and also tell what not to do.



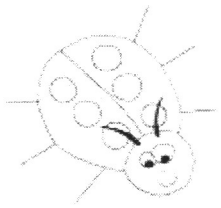
Debugging the Development Process : Practical Strategies for Staying Focused, Hitting Ship Dates, and Building Solid Teams
Steve Maguire
Microsoft Press, 1994 ISBN: 1556156502
Synopsis: This book explains how to get the most out of a software team without sacrificing quality of work or quality of life. Filled with simple, proven strategies that keep projects on track.
In *Debugging the Development Process*, Maguire describes the sometimes controversial but always effective practices that enable his software teams at Microsoft to develop high-quality software - on schedule.



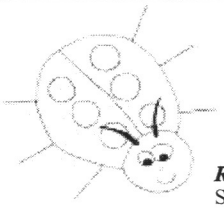
Peopleware : Productive Projects and Teams
Tom DeMarco, Timothy Lister
Dorset House, 1987 ISBN: 0932633056
Synopsis: *Peopleware* asserts that most software development projects fail because of failures within the team running them. This strikingly clear, direct book is written for software development-team leaders and managers, but it's filled with enough common-sense wisdom to appeal to anyone working in technology. Authors Tom DeMarco and Timothy Lister include plenty of illustrative, often amusing anecdotes; their writing is light, conversational, and filled with equal portions of humour and wisdom, and there is a refreshing absence of "new age" terms and multistep programs. The advise is presented straightforwardly and ranges from simple issues of prioritization to complex ways of engendering harmony and productivity in your team. *Peopleware* is a short read that delivers more than many books on the subject twice its size.



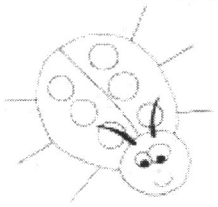
Mission Possible : Becoming a World-Class Organization While There's Still Time
Ken Blanchard, Terry Waghorn, Jim Ballard
McGraw-Hill, 1996 ISBN: 007009403
Synopsis: Blanchard, author of the best-selling *One Minute Manager* (1982), insists that for today's businesses to remain viable, managers must be "working on the present and the future of [their] organizations at the same time." His latest book is built around that central thesis. For managers to put this concept into practice, Blanchard offers three important premises: success in business is based on the creative use of untapped human energy, tapping into this resource requires managers to make partners of the people who work for them, and making people partners requires that they are meaningfully engaged "in either improving the present operation of the organization or creating its future." In elucidating his ideas, Blanchard brings into the discussion numerous specific examples and many words of wisdom from other management gurus. His logic makes sense; leaders need to pay attention.



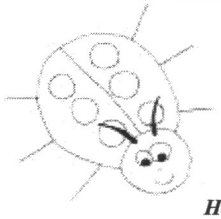
Raving Fans : A Revolutionary Approach to Customer Service
Ken Blanchard, Harvey MacKay, Sheldon Bowles
William Morrow & Company, 1993 ISBN: 0688123163
Review: "Your customers are only satisfied because their expectations are so low and because no one else is doing better. Just having satisfied customers isn't good enough anymore. If you really want a booming business, you have to create Raving Fans." This, in a nutshell, is the advise given to a new Area Manager on his first day -- in an extraordinary business book that will help everyone, in every kind of organization or business, delivery stunning customer service and achieve miraculous bottom-line results.



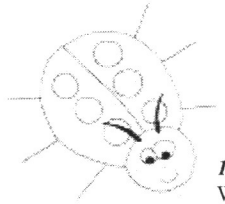
Rapid Development : Taming Wild Software Schedules
Steve McConnell
Microsoft Press, 1996, ISBN: 1556159005
Review: Written for team leaders, managers, and programmers alike, *Rapid Development : Taming Wild Software Schedules* is an excellent book on scheduling software development effectively and quickly. McConnell, an experienced software consultant for Microsoft and other companies, convincingly outlines potential hazards in the development process and possible steps for avoiding them. The chapter on classic mistakes will fill some with an overwhelming sense of déjà vu. The first two-thirds of the book are filled with clear-headed takes on topics ranging from scheduling to productivity tools, interspersed with examples, supporting data, and insightful anecdotes The last section of the book is devoted to "Best Practices," suggestions for more efficient development, each explicitly described and analysed for potential effectiveness. Since its publication, *Rapid Development* has developed a reputation for success and earned a place on the desks of software developers everywhere.



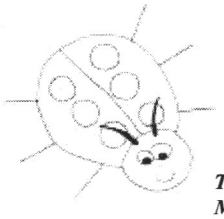
Software Project Survival Guide Software
Steve McConnell
Microsoft Press, 1997, ISBN: 1572316217
Review: *Congratulations, you're in charge. Now what?* Equip yourself with *Software Project Survival Guide*. It's for everyone with a stake in the outcome of a development project - especially those without formal software project management training. That includes top managers, executives, clients, investors, end-user representatives, project managers, and technical leads.
Here you'll find guidance from the acclaimed author of the classics *Code Complete* and *Rapid Development*. Steve McConnell draws on solid research and a career's worth of hard-won experience to map the surest path to your goal - what he calls "one specific approach to software development that works pretty well most of the time for most projects." Nineteen chapters in four sections cover the concepts and strategies you need for mastering the development process, including planning, design, management, quality assurance, testing and archiving.



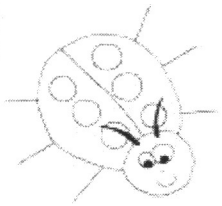
How to Run a Successful Meeting in Half the Time
Milo O. Frank
Simon & Schuster Inc., 1989, ISBN: 0671726013
Description: At last - a way to make meetings brief and productive! Milo O. Frank shows you how to :
Determine whether a meeting is really necessary - and say no if it isn't
Plan for a constructive meeting - setting the right time, place participants and agenda
Get the decision makers on your side *before* the meeting starts
Keep every meeting on target and on time - setting limits, discouraging latecomers, and cutting off digressions



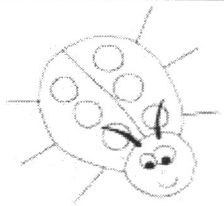
13 Fatal Errors Managers Make And How You Can Avoid Them
W. Steven Brown
Berkley Publishing Group, 1995, ISBN: 0425096440
Synopsis: Are you guilty of:
Being a buddy, not a boss?
Never admitting that you are accountable?
Managing different people in the same way?
Failing to set common business goals?
Trying to control your people instead of influencing their thinking with enthusiam?
These are just a few of the 13 fatal errors managers make. Errors that waste valuable time, money and talent. This book will show you how to recognize problems - and avoid them - before they happen. Author Steven Brown, a nationally recognized professional trainer and consultant, provides the essential guide for *effective* managers and show you how to get the best from your workers, your company - and yourself.



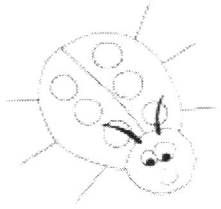
The Dilbert Principle : A Cubicle's-Eye View of Bosses, Meetings, Management Fads & Other Workplace Afflictions
Scott Adams
Harper Business, 1997, ISBN: 0887308589
Reviews:
"Ever since Machiabelli set out the prinicples of princehood, the public has been hoodwinked into believing that with the right advice on swimming with the sharks, managing in one minute, and searching for excellence, accompanied by the application of reason and logic, success would be theirs. In *The Dilbert Principle*, Scott Adams finally puts things in perspective, fearlessly acknowledging an eternal truth : "If you've worked in the business world for more than ten minutes, you know it's an immense exercise in the absurd." Unlike other business books, Adams actually illustrates the truth in action : examples of his hilarious comic strip, *Dilbert*, are packed into these pages, a "vision" statement par excellence.



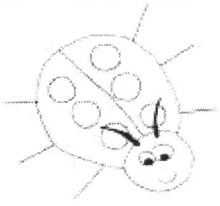
Make It So : Leadership Lessons from Star Trek : The Next Generation
Wess Roberts, Bill Ross
Pocket Books, 1996, ISBN: 0671520989
Synopsis:
The fast-changing business world of today is far different from just a few years ago. Success in today's marketplace requires new leadership techniques, new thinking, and an eye on the future....
In Make It So : Leadership Lessons from Star Trek : The Next Generation,Wess Roberts and coauthor Bill Ross take their inspiration from today's most striking and most popular vision of the future -- *Star Trek* -- an unprecedented television, feature film and publishing phenomenon.



Death March : The Complete Software Developer's Guide to Surviving "Mission Impossible" Projects
Edward Yourdon
Prentice Hall, 1997, ISBN: 0137483104
Synopsis:
The complete software developer's guide to surviving projects that are "doomed to fail".
In the course of a career, practically every software developer and manager will encounter projects with outrageous staffing, scheduling, budgeting, or feature constraints : projects that seem destined to fail. In the wake of re-engineering, such "Death March" projects have become a way of life in many organizations.
In *Death March*, Edward Yourdon guides you through : Surviving projects that are "doomed to fail!"



I am a Bug
Robert Sabourin, illustrated by daughter Catherine
1999, ISBN: 0-9685774-0-7
Synopsis:
"I am a Bug" uses the style of a children's book to explain elements of the software development process in a fun easy to read format."



Thank You

- Questions?