Socrates went around giving people a lot of good advice.
They poisoned him.

- California 6th grader
Objectives

1. Automate the testing process
2. A structured review process
3. Structured development with reusable code and reusable tests
4. Appropriate progress and performance metrics
5. On-going quality initiatives
Objectives

6. Identify testable conditions
7. Organize testing
8. Teamwork
9. Communications
10. Enjoy your job
Observation - 1

Nothing is obvious

Specifications must be written
Examples
Graphics
Quantify everything
Observation - 2

Do it twice

Test and retest
Design all tests to be repeatable
Test bed should be maintainable
Test initialization & re-initialization
Observation - 3

Everything has a limit

Identify the limits
Test the limits
Language imposed limitations
Platform imposed limitations
Specification imposed limitations
Observation - 4

Design systems with testing in mind

- Insert diagnostic tools (instrumentation)
- Control totals
- Audit trails
- Balancing routines
- File comparisons
Observation - 5

Practice tact and diplomacy

Don’t be critical all of the time
Offer positive comments
Encourage the right behavior
It is better to find agreement than to win
Observation - 6

The specifications may be wrong

All specifications will change
Include test plans in the specs
Identify a formal change process
Don’t fight change
Get better estimates of change impact
Change Requests: must be in writing and require 3 estimates
First Shopping List

• 1 box cereal
• 2 cans orange juice
• 3 1/2 gallons of milk
• 1 loaf of bread
Shopping List (Spec)

- olives
- laundry detergent
- oatmeal
- juice
- coffee
- dessert for tonight
Shopping List (Spec) Questions

• olives (large, jumbo, can, jar, pimentos)
• laundry detergent (brand, size, aroma)
• oatmeal (instant, flavored, can, box)
• juice (orange, tomato, apple, cranberry)
• coffee (instant, Jamaican, Hawaiian)
• dessert for tonight
Bug Propagation

1 bug in analysis can result in 3-15 bugs in design can result in 2-10 bugs in code

**Worst case:** 1 bug in analysis causes 15 bugs in design and they in turn create 10 bugs in the code. Total = 150 code defects. How many will you find in testing?
What are the major impediments to quality initiatives in this organization?

1.
2.
3.
4.
Impediments, Opportunities, and Managing

• No time

• “We just don’t have time to change the way we test or develop systems. Quality improvement is a great idea, but we don’t have anyone available right now.”
Impediments, Opportunities, and Managing

• Inertia

• “Things aren’t so bad, why should I want to do anything differently. No one has yelled at me in over two weeks.”
Impediments, Opportunities, and Managing

• Need a management buy-in

• “If management doesn’t tell us to improve quality, gives us time, and a budget, nothing is going to happen. Discussing quality with us is simply preaching to the choir.”
Impediments, Opportunities, and Managing

- *We’re not ready and we need training*

- “We have to get the rest of the shop in order, before we can consider quality improvement suggestions. When will we have time for training?”
Who is responsible for testing?

Customer
    They know what they want better than I do.

Business Analysts
    They should do all of the testing. They’re always talking to the customers.

Programmers
    If they do their job properly, no one else has to test.

Quality Assurance
    Its their job.
Quality Questions

1. What is quality.
2. What does it cost?
3. How is quality measured?
4. Where does quality come from?
Quality Improvement Suggestions: 1

Joint Application Design sessions (JAD/JAR)
Define business objectives
Insist on written specifications
Use prototyping tools
Write user manuals (before coding)
Estimate the testing effort
Allocate resources to automate testing
Quality Improvement Suggestions: 2

Instrumentation
Written unit test plans (before coding)
Enforce development methodologies
Enforce coding standards
Earlier involvement of QA personnel
Capture and report metrics
More documentation
Quality Improvement Suggestions: 3

Risk and Contingency analysis
Follow-up process for suggestions
Reusable tests
Maintainable test beds
Walkthroughs
Earlier Reviews
Written unit test plans
Concise Project Management Course

- Milestones
  - 40 hour rule
  - 80 hour rule
- Deliverables
  - Measurable
  - Reviewable
  - Achievable
- Document everything
- Review everything promptly
<table>
<thead>
<tr>
<th></th>
<th>Front End</th>
<th>Coding</th>
<th>Back End</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional</td>
<td>35%</td>
<td>15%</td>
<td>50%</td>
</tr>
<tr>
<td>Quality Process</td>
<td>42%</td>
<td>18%</td>
<td>28%</td>
</tr>
</tbody>
</table>

12% productivity increase
50+% fewer defects
12-15% faster to market
It may cost less to leave the defects out of the system, than to pay to put them in, pay to find them, and then pay to take them out again.
Reliability Metrics

Mean Time Between Failures

- MTBF$_1$ Crash, software inoperable
- MTBF$_2$ Functional failure
- MTBF$_3$ Quality failure
- MTBF$_4$ Client Server failure
Reliability Metrics

Mean Time To Repair

- $MTTR_1$: Actual time to fix
- $MTTR_2$: Total time in queue
Test Scripts (Generic Tests)

Unit Test Scripts
  Component level tests

System and Acceptance Test Scripts
  Functional tests
  Test threads
  End-to-end tests
  Start-to-finish
No time to perform a regression test?

Alternatives

- Parallel testing
- Volume testing with live data
- Volume testing with test data
- Local, regional, expanded regional testing
- Statistical sampling
- Testing changes only
- Combine sample data with known problems
Traceability Matrix

Customer Requirements -> Functional Specification -> System Test Scripts -> Test Cases

- Code Modules
- Data Structures
Black Box Testing

Specification:
C = (A*B)
A: Integer, 1 to 100
B: Integer, -5 to +5

Test Plan

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

Dynamic Testing
Validation
White Box Testing

Specification:

\[ C = (A \times B) \]

A: Integer, 1 to 100
B: Integer, -5 to +5

Code Example 1:

- Move 0 to A, B
- GetInput(A,1,100,B,-5,5)
- Compute \( C = A \times B \)
- Compute Tot = Tot + C
- Print C

Code Example 2:

- Move 0 to A, B
- GetInput(A,0,100,B,-5,5)
- Compute \( C = A \times B + .000001 \)
- Compute Tot = Tot + C
Quotes

- It compiled, it’s got to be good.
- It usually works.
- No reasonable customer would ever do that.
- Trust me, it’s okay.

- It worked yesterday.
- It works on my machine.
- I tested it for you.
- Of course it doesn’t pass that test.
- What could go wrong?
How many paths? How many test cases?

A

B

1

2

3

4

Waist
<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>nnnn</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>nnnny</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>nnnyn</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>nnyyy</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>nynnn</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>nyny</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>nyyny</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>nyyyn</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>ynnnn</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>ynyny</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>yynny</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>yyyyy</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>yynnn</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>yyyny</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>yyyy</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>yyyy</td>
</tr>
</tbody>
</table>
Screen Edits

- Screen defaults
- Function keys
- Escape key
- Minimize screen
- Maximize screen
- Drag screen
- Resize screen
- Initial screen size
- Initial screen location

- O/S Characteristics
- Resolution
- Color
- Fonts
- Menu bars
- Button bars
- Navigation bars
- Slide bars
- Screen title
Button Edits

- Single click
- Double click
- Look & feel
- Space bar
- Tab
- Enter
- Hot key

- Other events that can be triggered
- Escape
- Default setting
- Color
- Relational edits
- Focus box
Character Entry

• Leading spaces
• Trailing spaces
• Embedded spaces (multiples)
• Permitted spaces
• Special characters (numeric, CTRL, ALT, SHIFT, foreign)

• Specific valid or invalid characters
• Font
• Color
• Case sensitivity
• Entry templates
• Minimum field length
• Maximum field length
Recommendation Guidelines

1. All recommendations must be written.
   Verbal doesn’t count.

2. Provide a means to measure the results.
   Results that can not be predicted, or quantified, make the suggestion seem less credible.

3. Offer a brief assessment of the current situation.

   Try to get everything on one page.
Root Cause Analysis

- When was the error made?
- Who made the error?
- What was done incorrectly?
- How could the error have been prevented?
- Why wasn’t the error detected earlier?
- How could the error have been detected earlier?
- How was the error found now?
- Has this happened before?
- Is the problem in any other systems?
Question

Which best describes your job?

1. Defect detection
2. Defect prevention
Assessing Readiness to Automate

1. Are good testing policies and procedures in place today?
2. Is there time to learn how to use the automated testing tool?
3. Will there be ongoing training?
4. Are the development team and test team in agreement about the use of automated test tools and procedures?
5. Are you ready to make a total commitment to the testing tool?
Assessing Readiness to Automate

6. Are your development language and platform supported?

7. How many platforms will you test?

8. Are you testing embedded systems?

9. Is the budget adequate?

10. How much support will your organization require?
1. Design all tests to be repeatable

2. Design the test bed to be maintainable
Question

Which activity will enable you to make the greatest contribution to your organization?

1. Running tests

2. Identifying testable conditions