KEEPING THE ENGINEERING in Software Engineering

Dick Hamlet

Center for Software Quality Research
Portland State University

Talk Outline

1. What should software engineering be?
   - Not management
2. Programming languages – C and Java
3. Testing tools and processes
   - Testing to {detect failure | measure quality}
   - Software reliability engineering
   - An *engineering* test process
   - Process tools
4. Working too hard
5. Raising hell

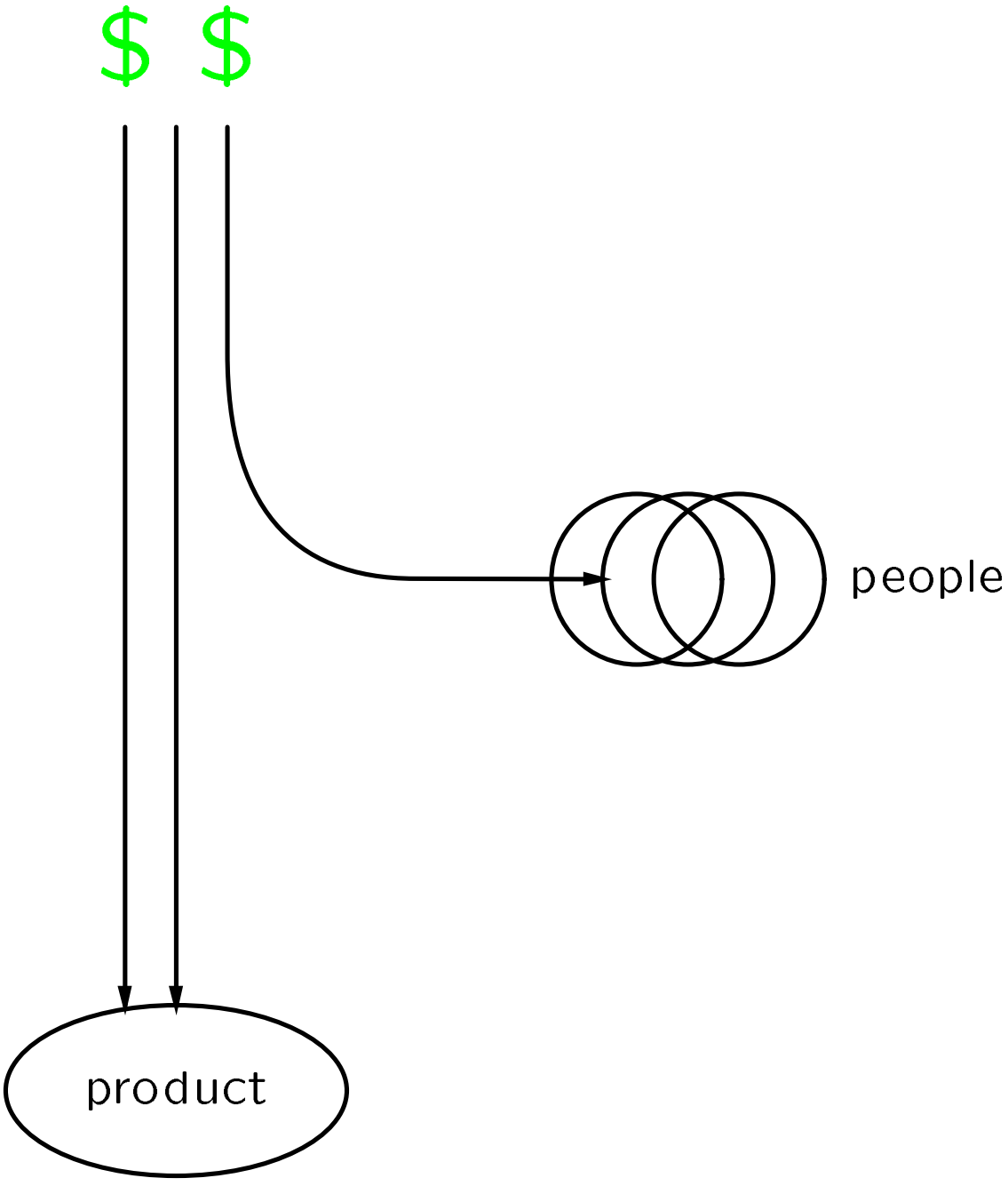
The Management Myth:

Knowledge and skill are unnecessary
if people are properly organized

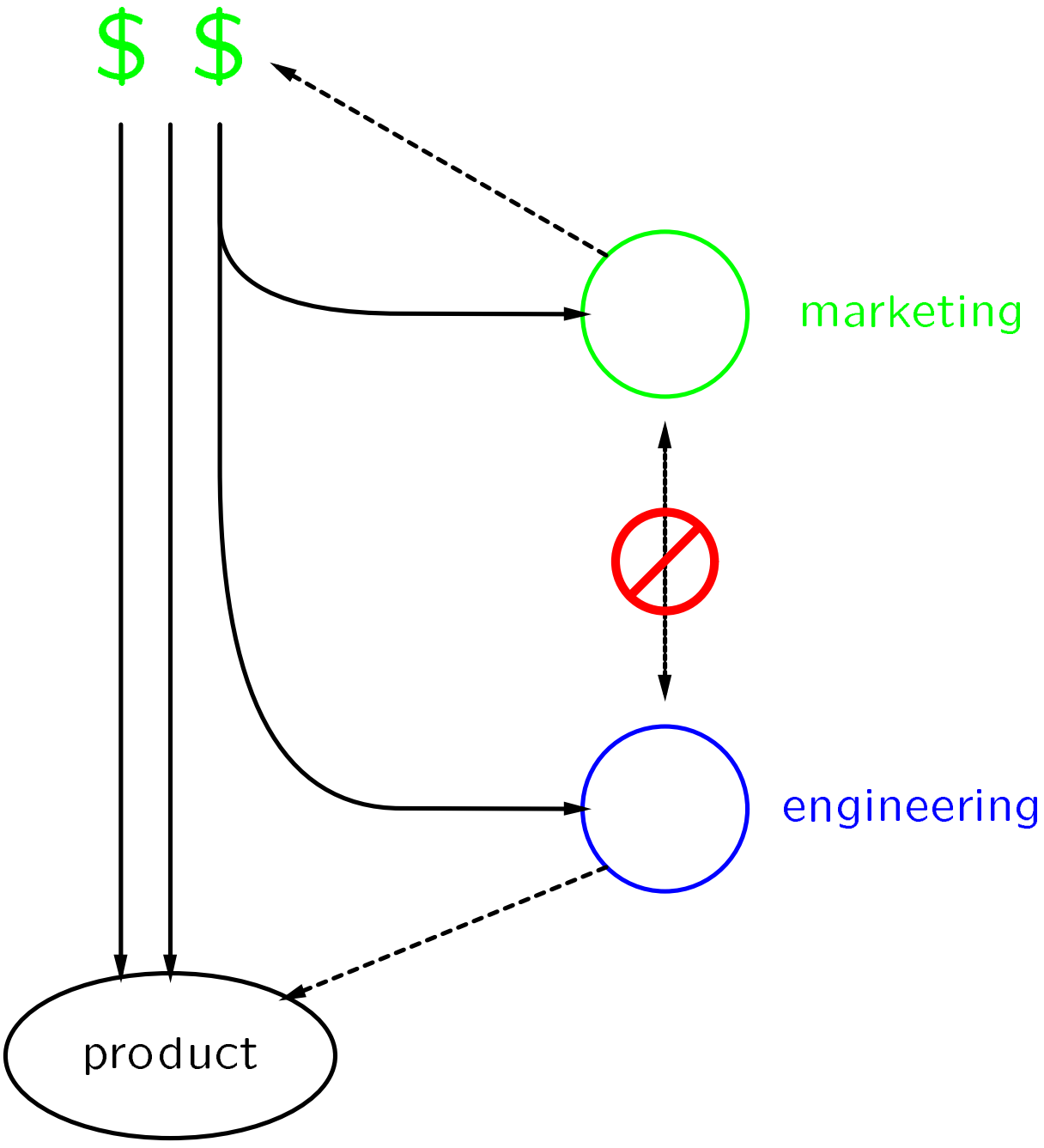
Software-engineer Facts of Life

- 'Manage yourself' – your actual manager can't.
- State-of-the-art tools aren't available.
- 'Best practices' not practiced (not known?).
- You work too long and hard at what isn't fun.
- It's past time to complain.

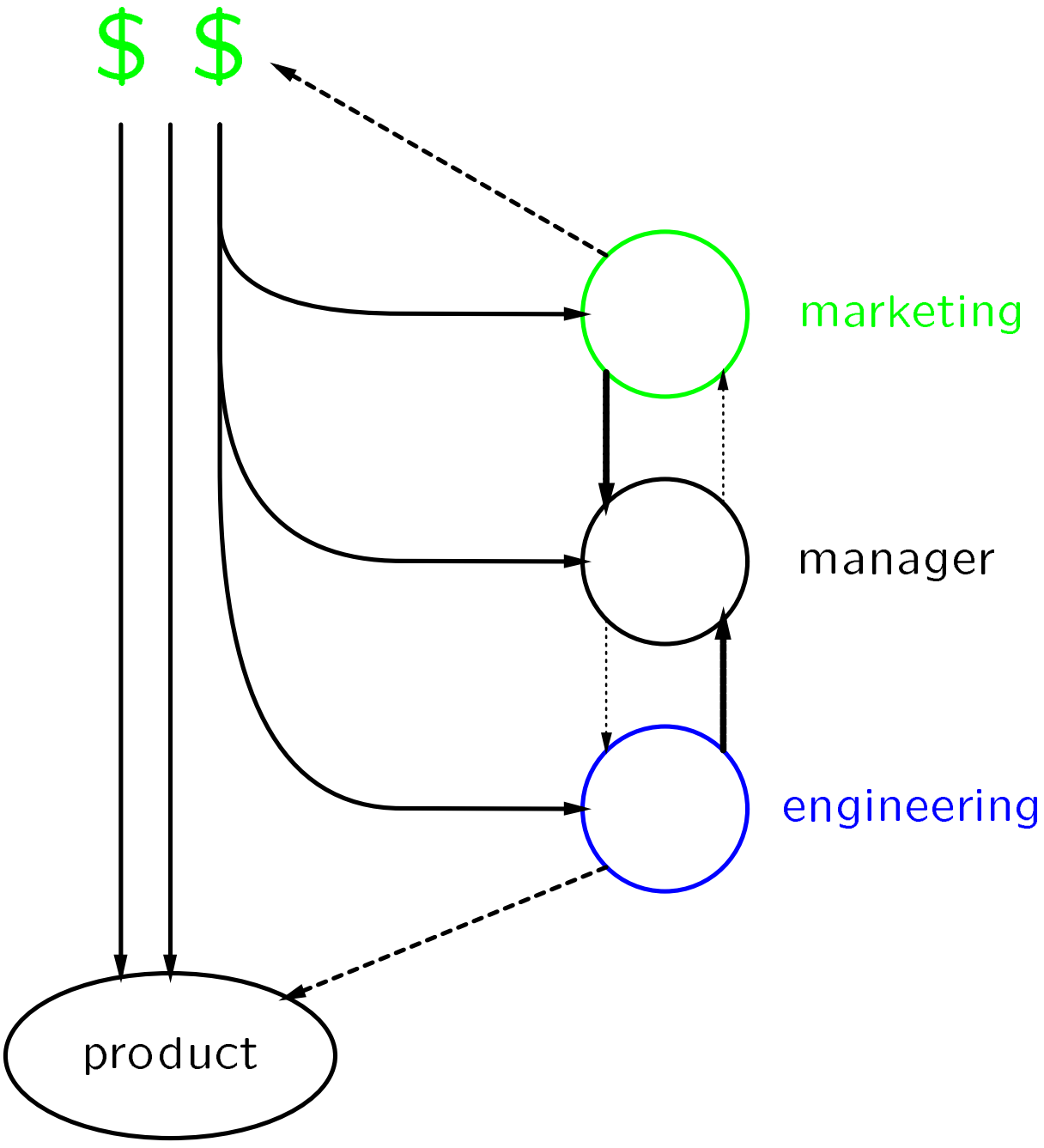
Simplified View of Economics

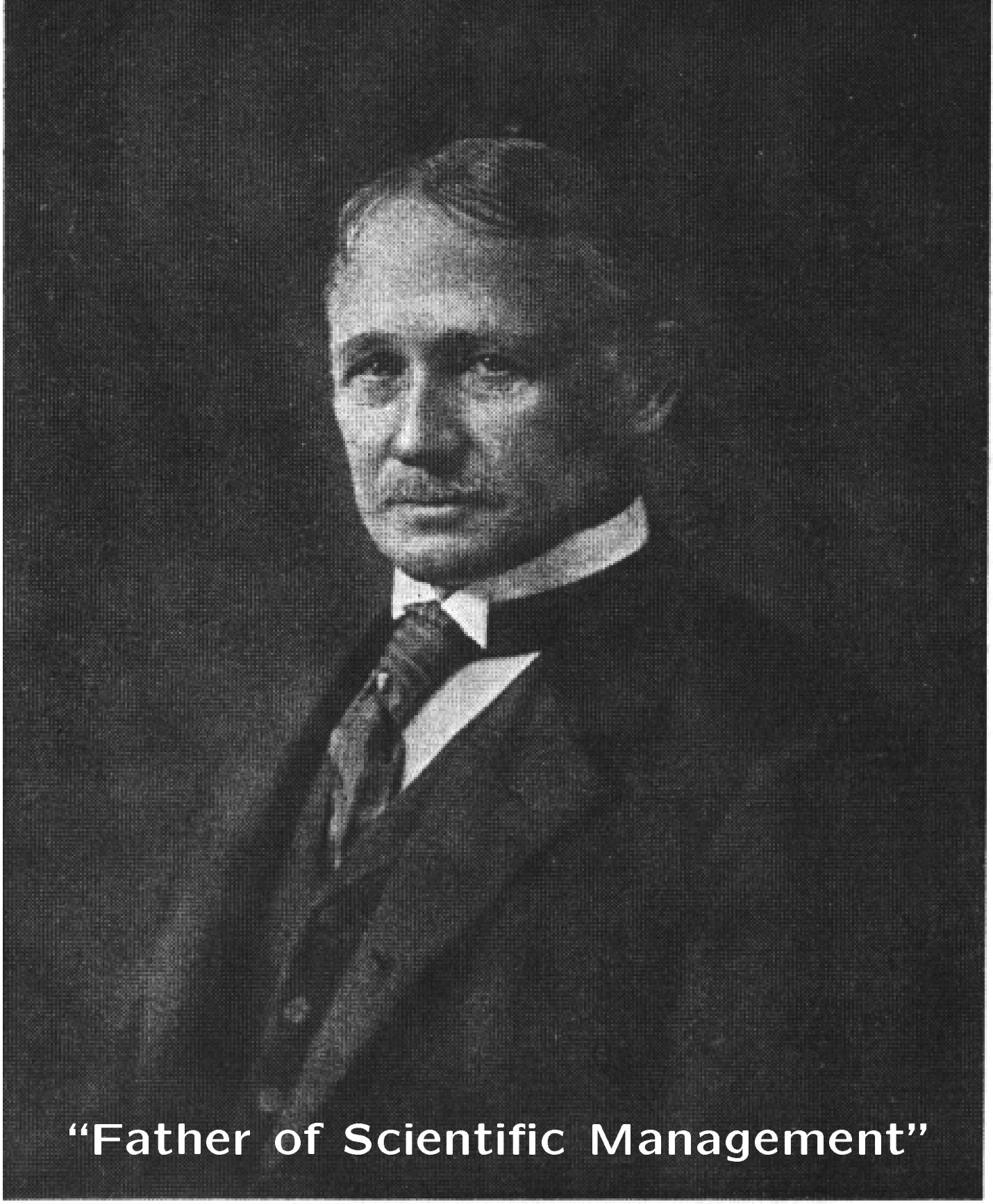


Engineering and Marketing



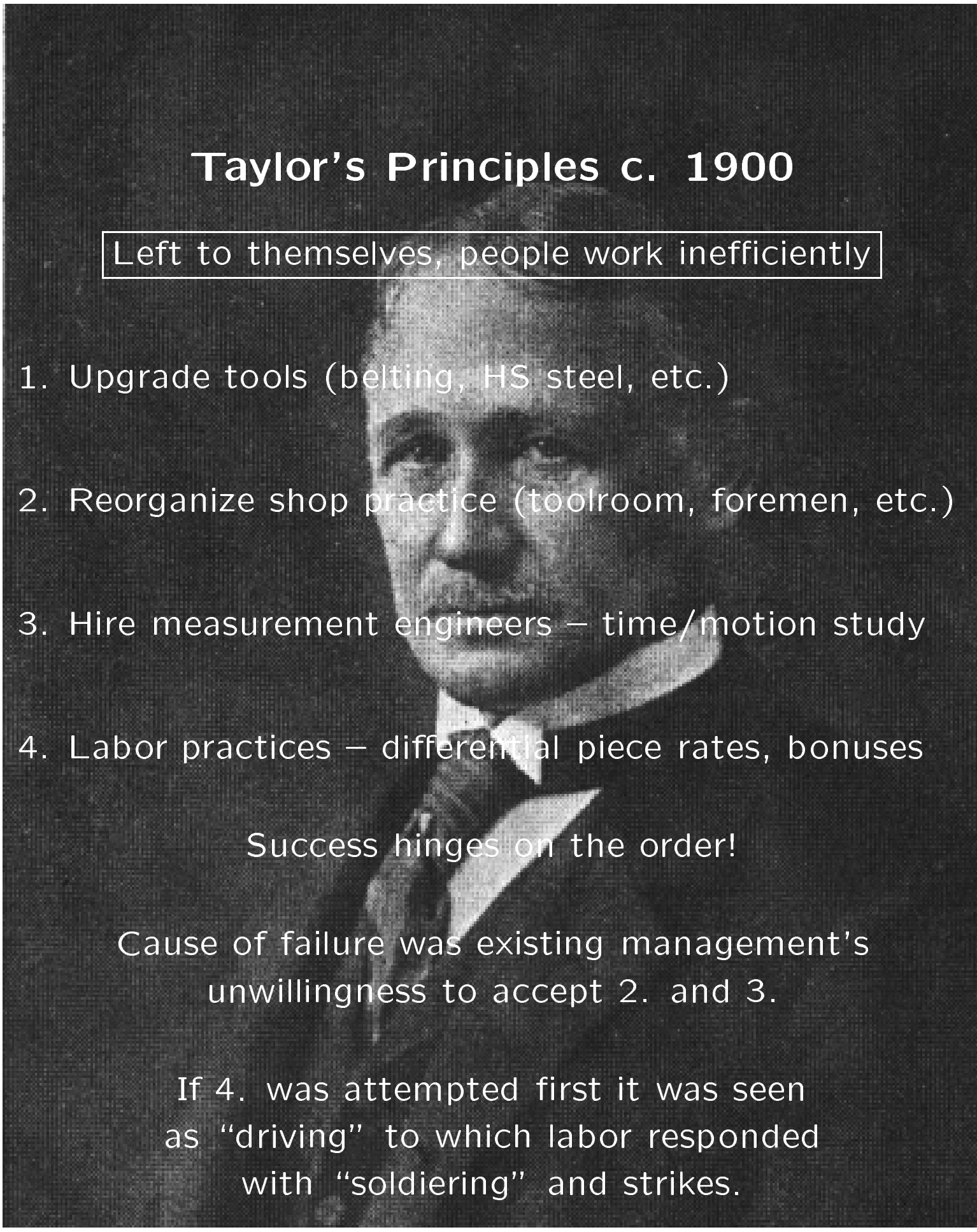
Managers as Filters





"Father of Scientific Management"

Fred. W. Taylor



Taylor's Principles c. 1900

Left to themselves, people work inefficiently

1. Upgrade tools (belting, HS steel, etc.)

2. Reorganize shop practice (toolroom, foremen, etc.)

3. Hire measurement engineers – time/motion study

4. Labor practices – differential piece rates, bonuses

Success hinges on the order!

Cause of failure was existing management's unwillingness to accept 2. and 3.

If 4. was attempted first it was seen as "driving" to which labor responded with "soldiering" and strikes.

The Personal Software Process (PSP)

Read Watts Humphrey's book ...

... before someone orders you to do what it says!

Left to themselves, programmers work inefficiently.
They should manage themselves to do better.

- PSP starts at the wrong end of Taylor's list – labor is asked to "drive" itself

- tools are hardly mentioned

The process vs. product issue split the ASME at the turn of the last century.

Talk Outline

1. What should software engineering be?
   - Not management
2. Programming languages – C and Java
3. Testing tools and processes
   - Testing to {detect failure | measure quality}
   - Software reliability engineering
   - An *engineering* test process
   - Process tools
4. Working too hard
5. Raising hell

Programming Languages: "Security"

C (C++) is dreadful, not a professional tool
– (C the paper).

- "Left to the implementation"
- datatypes like **int**, operators like <<
- uncontrolled pointers, no memory checking

Java is a lot better. How did it happen that the current fad language is a pretty good language?

"We lucked out" – Guy Steele, ICSE '97

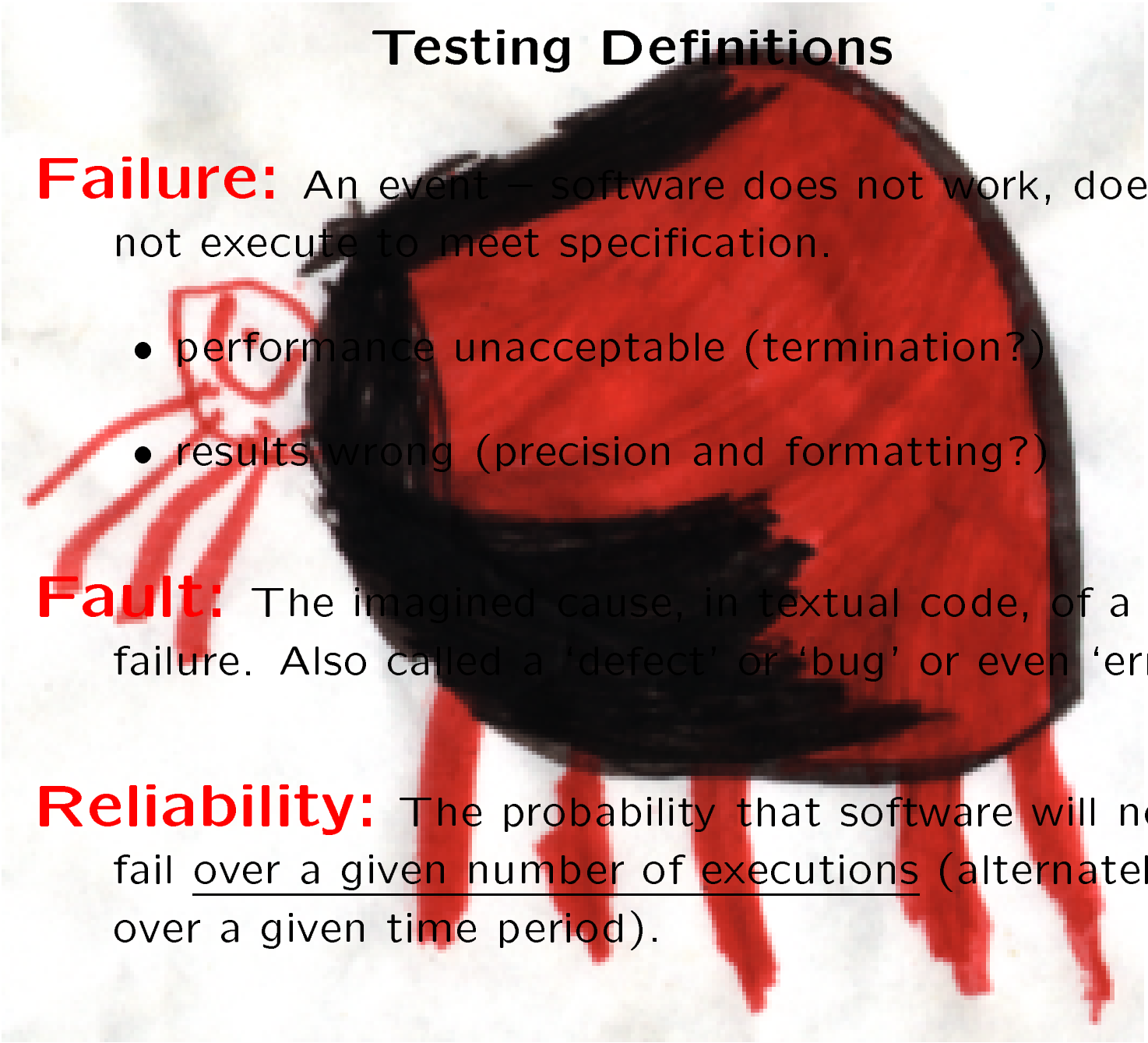
Talk Outline

1. What should software engineering be?
   - Not management
2. Programming languages – C and Java
3. Testing tools and processes
   - Testing to {detect failure | measure quality}
   - Software reliability engineering
   - An *engineering* test process
   - Process tools
4. Working too hard
5. Raising hell

Testing to Find Failures

"Bug" by Matthew McWilliams, age 4

Testing Definitions



Failure: An event – software does not work, does not execute to meet specification.

- performance unacceptable (termination?)
- results wrong (precision and formatting?)

Fault: The imagined cause, in textual code, of a failure. Also called a 'defect' or 'bug' or even 'error.'

Reliability: The probability that software will not fail over a given number of executions (alternately, over a given time period).

Inspection vs. Testing

Code inspections are better than unit testing at finding simple bugs

Inspections can also be used on specifications and designs

Inspections aren't much fun, more sociology than engineering...

Technical problem: Find a way to beat inspection

Candidate technologies:
- Mutation
- Dataflow
- Formal specification

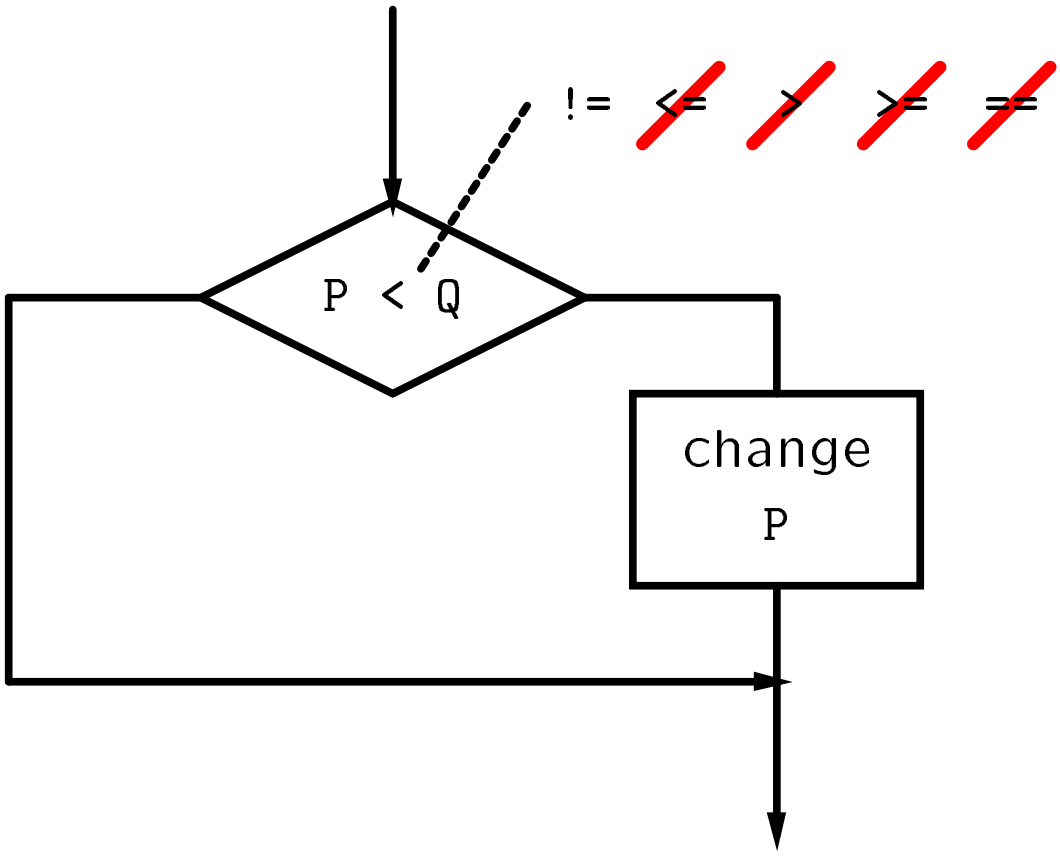
Mutation Technology

Invented 25 years ago, and still unused!

Idea: Introduce a minor change (mutation) to code. Check the behavior under test.

Guaranteed to catch (among others): off-by-one mistakes; wrong relational operator.

Test Data
(P,Q)
(1.7,4.1)
(1.0,1.0)

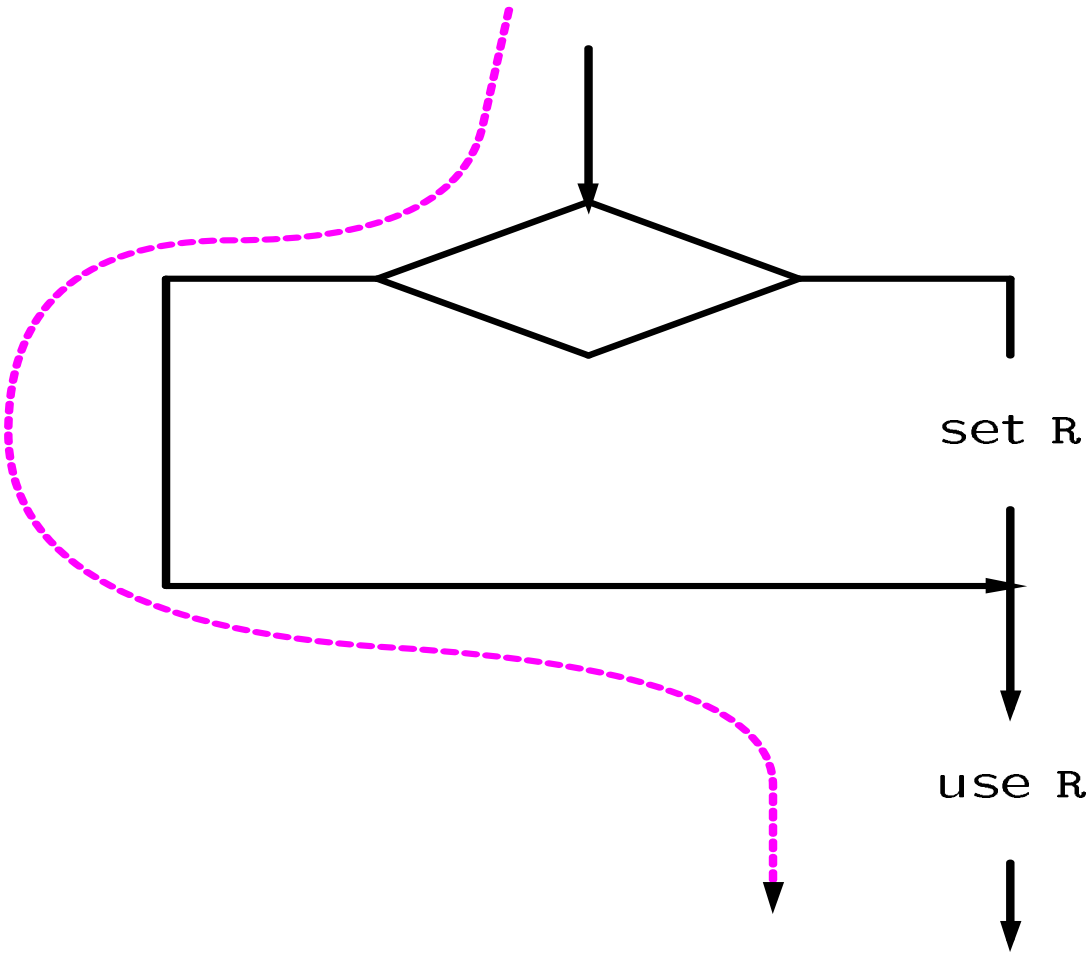


Dataflow Technology

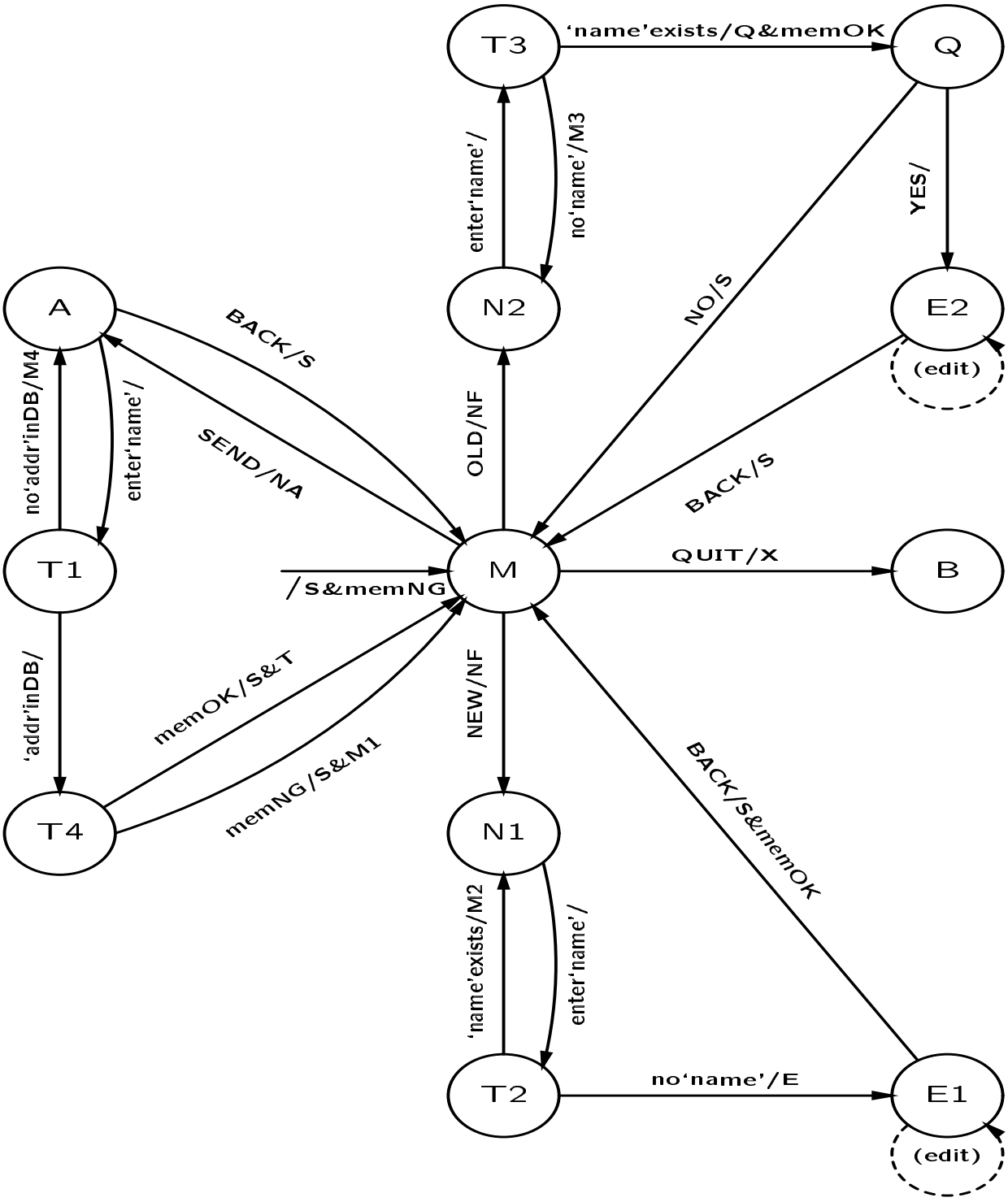
Invented 35(?) years ago; used for memory leaks.

Idea: Trace dependencies for worst-case paths.

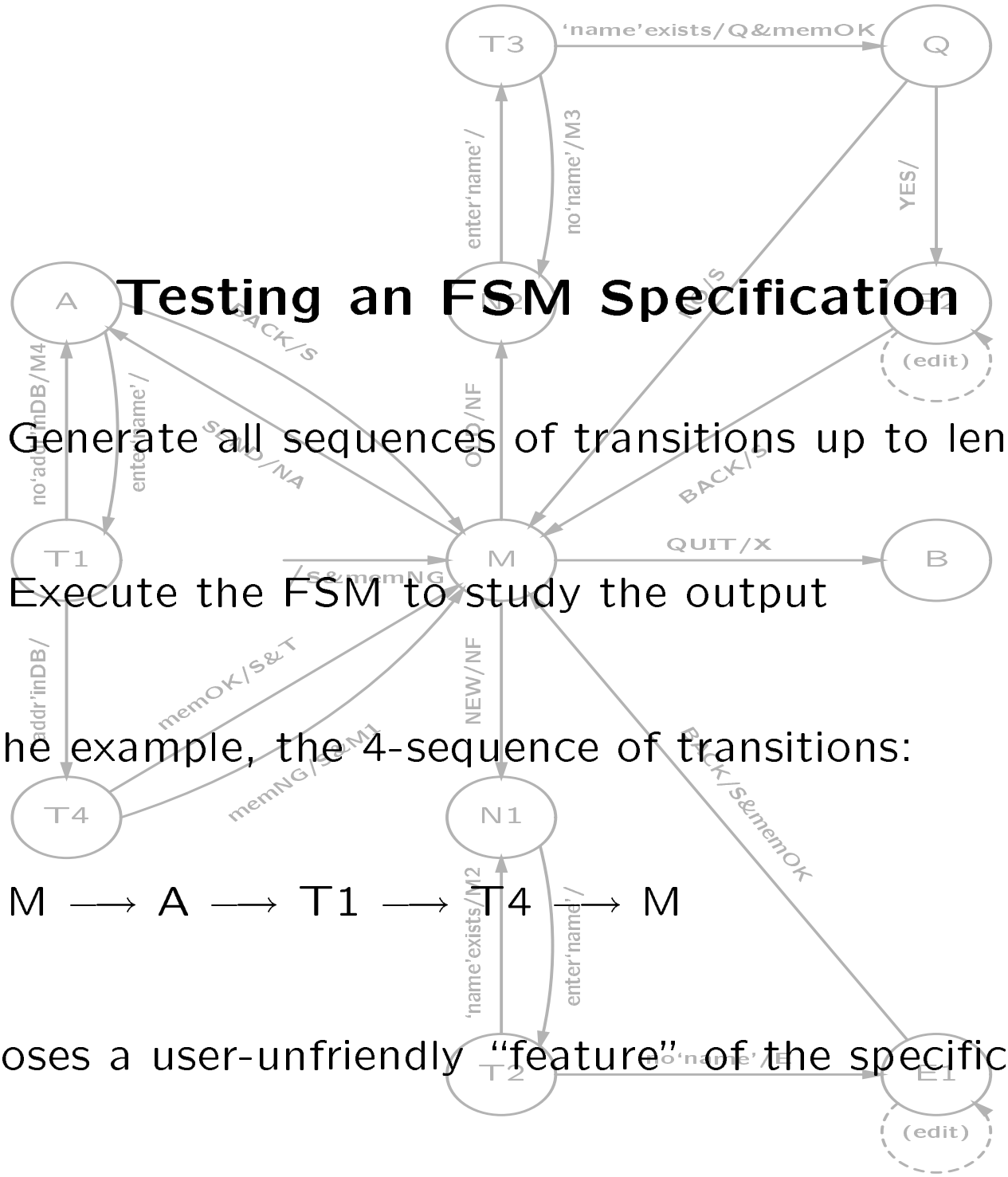
Guaranteed to catch: lack of initialization, resetting variables without use, etc.



FSM Formal Specification



Testing an FSM Specification



- Generate all sequences of transitions up to length N
- Execute the FSM to study the output

In the example, the 4-sequence of transitions:

M $\longrightarrow$ A $\longrightarrow$ T1 $\longrightarrow$ T4 $\longrightarrow$ M

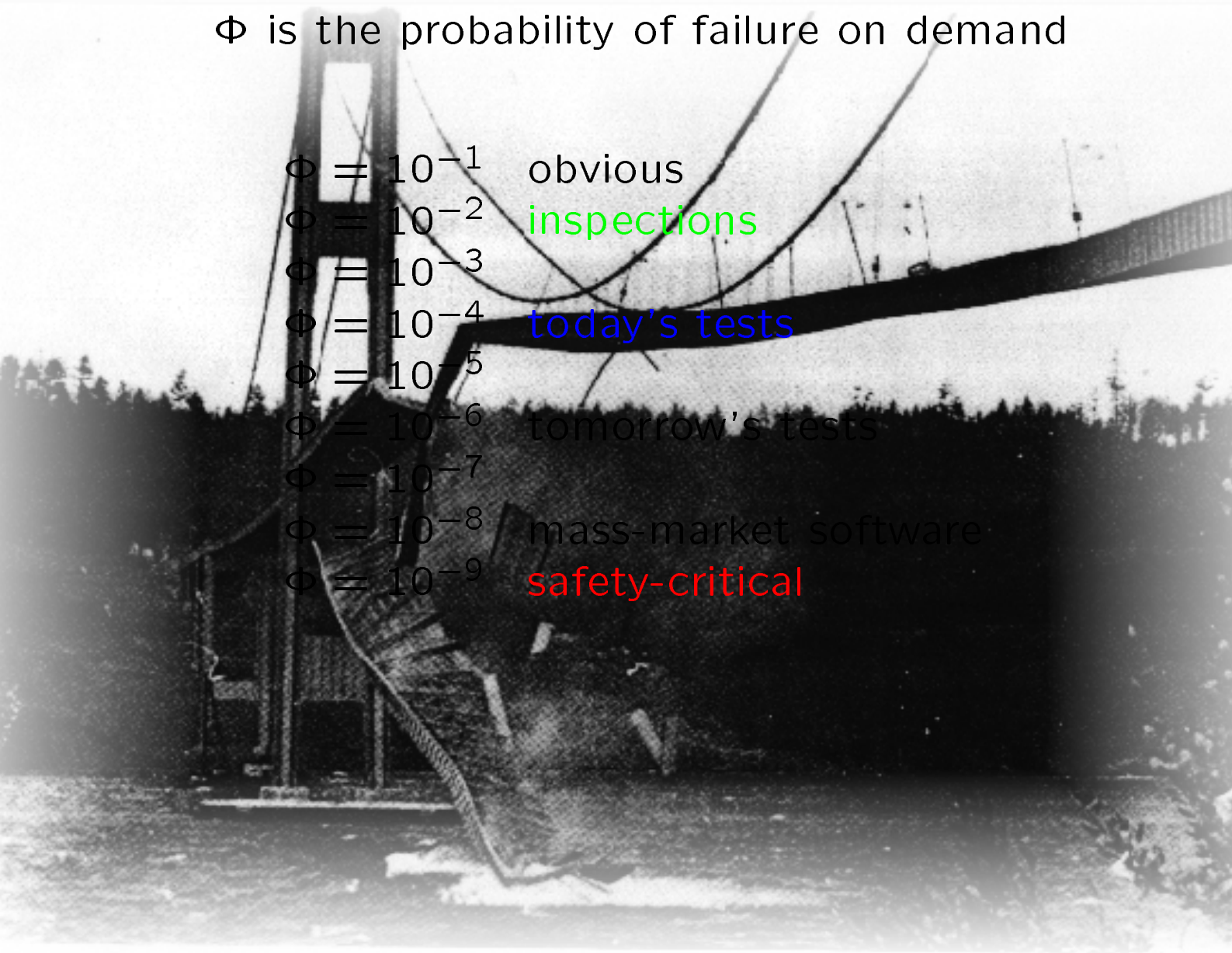
exposes a user-unfriendly "feature" of the specification.

Beyond Simple Failures

"Galloping Gertie" 1940 (UPI photo)

Measuring Software Quality

Φ is the probability of failure on demand



$\Phi = 10^{-1}$	obvious
$\Phi = 10^{-2}$	inspections
$\Phi = 10^{-3}$	
$\Phi = 10^{-4}$	today's tests
$\Phi = 10^{-5}$	
$\Phi = 10^{-6}$	tomorrow's tests
$\Phi = 10^{-7}$	
$\Phi = 10^{-8}$	mass-market software
$\Phi = 10^{-9}$	safety-critical

Software Reliability Engineering

The failure probability Φ is the largest value of θ such that:

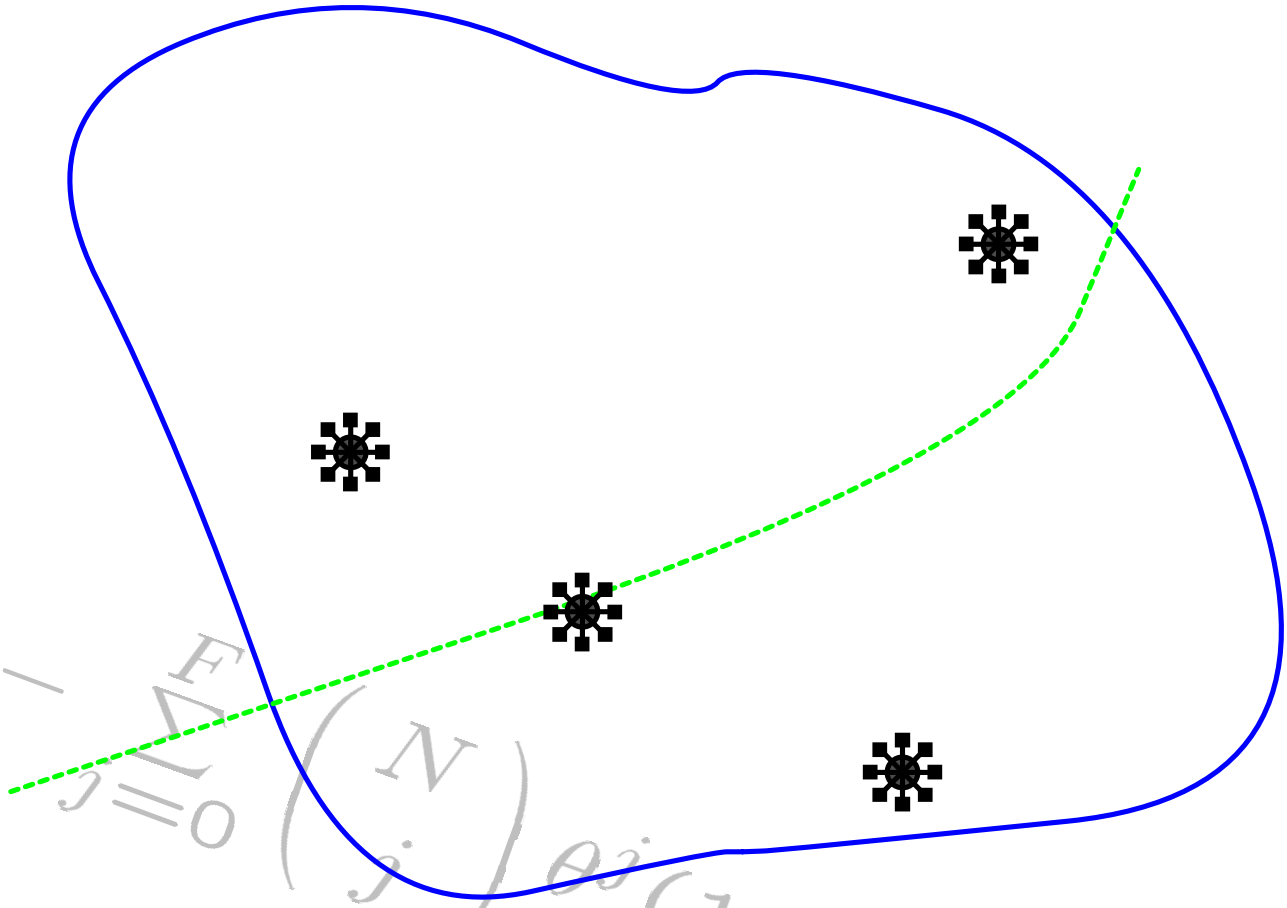
$$1 - \sum_{j=0}^{F} \binom{N}{j} \theta^j (1-\theta)^{N-j} \geq \alpha$$

where: N is the number of tests
F is the number of failures seen
α is the upper confidence bound

Tests must be drawn according to an *operational profile*.

Example: 90% confidence in $\Phi < 10^{-5}$ takes 230,000 tests without a failure.

The Software Minefield



$$1 - \sum_{j=0}^{F} \binom{N}{j} \theta^j (1-\theta)^{N-j} \geq \alpha$$

An Engineering Test Process

- Define operational profile, reliability requirement
- Specification formalism & analysis w/tools
- Design formalism, ditto
- Foolproof structural test for simple failures
- Random testing to reliability requirement
- Feedback to process from field performance

Without the technical basis for each step, the last is vacuuous.

"Process" Tools

Management support These tools help managers to control the process and the people using it. Example: recording, scheduling, organizing inspections.

Engineering support These tools help in doing the technical work itself. Example: automating (and thus eliminating) inspection steps.

The crucial questions:

Who benefits when this tool is used?
Is it fun to use?

Test question: What color is a CASE tool?

Talk Outline

1. What should software engineering be?
   - Not management
2. Programming languages – C and Java
3. Testing tools and processes
   - Testing to {detect failure | measure quality}
   - Software reliability engineering
   - An *engineering* test process
   - Process tools
4. Working too hard
5. Raising hell

Working too Hard

Why do software engineers put in such long hours?

What is the difference between a "professional" and a "wage slave"?

It is good to be a professional (civil) engineer (PE). Why are there no professional software engineers?

What are the differences among:

A professional association,
A company union,
A real labor union?

Try these: Are doctors who work for an HMO professionals? Is the AMA a union? What is the difference between the AMA and the ACM?

Raising Hell – Reliability

Ask Marketing for a user profile on a new product, and for the reliability that will be advertised.

Hand Marketing the technical articles on how to obtain a profile, and the definition of reliability. Request support tools and training for software reliability engineering (because Marketing requires them).

Explain that (1) your competitors have no idea what profiles they will face and no idea of their products' reliability, and (2) you can't possibly test software without a profile and a reliability goal.

Raising Hell – Unit Testing

Get QA to request that you submit a report on the mutation- and dataflow adequacy of unit testing.

Refuse to have any code inspected without these reports. (Because QA requires them.)

When an inspection is held, object to each item on the checklist as unnecessary because the test reports show that it has been mechanically checked.

Bring copies of the technical papers on mutation and dataflow, and wave them in the face of anyone who objects.

Raising hell – Training

Don't go to talks like this one! Instead try:

PNSQC Pacific Northwest Software Quality Conference (October)
`http://www.teleport.com/~pnsqc/`

ISSRE International Symposium on Software Reliability Engineering (November)
`http://admin.one2one.com/issre97/`

ISSTA International Symposium on Software Testing and Analysis (March)
`http://www.cs.pitt.edu/issta98`

Raising Hell – Last Resort

Vote with your feet. Go where engineering is valued.

When you leave, make sure that your co-workers and managers know why.

Explain in writing what is wrong, and why you gave up trying to fix it.

At job interviews, ask about turnover, and why people quit. No answer is an answer. Don't get your old job back.