

# Black Box Software Testing

## *Spring 2005*

by

Cem Kaner, J.D., Ph.D.

Professor of Software Engineering

Florida Institute of Technology

and

James Bach

Principal, Satisfice Inc.

### **Copyright (c) Cem Kaner & James Bach, 2000-2005**

This work is licensed under the Creative Commons Attribution-ShareAlike License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/2.0/> or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

These notes are partially based on research that was supported by NSF Grant EIA-0113539 ITR/SY+PE: "Improving the Education of Software Testers." Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

# About Cem Kaner

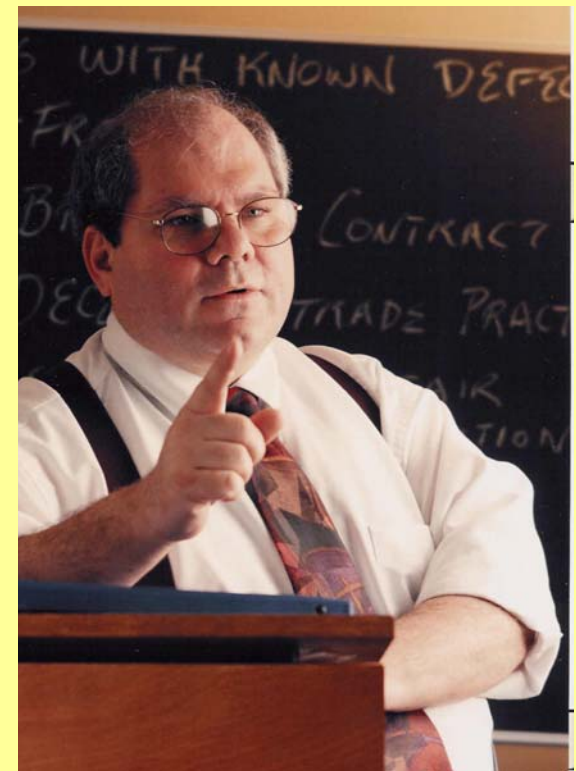
[www.kaner.com](http://www.kaner.com)

My current job titles are Professor of Software Engineering at the Florida Institute of Technology, and Research Fellow at Satisfice, Inc. I'm also an attorney, whose work focuses on same theme as the rest of my career: satisfaction and safety of software customers and workers. I've worked as a programmer, tester, writer, teacher, user interface designer, software salesperson, organization development consultant, as a manager of user documentation, software testing, and software development, and as an attorney focusing on the law of software quality. These have provided many insights into relationships between computers, software, developers, and customers.

I'm the senior author of three books:

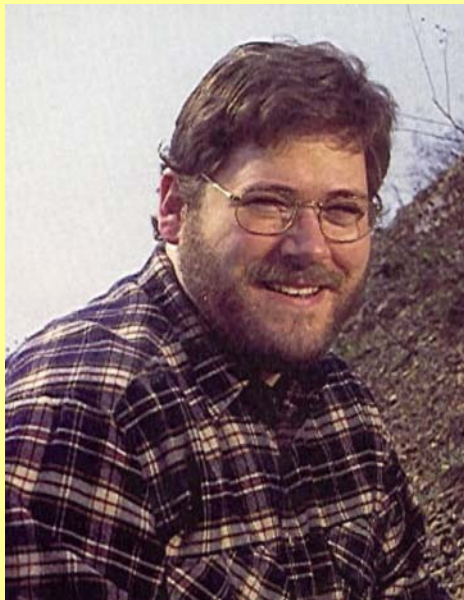
- *Lessons Learned in Software Testing* (with James & Bret Pettichord)
- *Bad Software* (with David Pels)
- *Testing Computer Software* (with Jack Falk & Hung Quoc Nguyen).

I studied Experimental Psychology for my Ph.D., with a dissertation on Psychophysics (essentially perceptual measurement). This field nurtured my interest in *human factors* (and thus the usability of computer systems) and in *measurement theory* (and thus, the development of valid software metrics.)



# About James Bach

[www.satisfice.com](http://www.satisfice.com)



I started in this business as a programmer. I like programming. But I find the problems of software quality analysis and improvement more interesting than those of software production. For me, there's something very compelling about the question "How do I know my work is good?" Indeed, how do I know anything is good? What does good mean? That's why I got into SQA, in 1987.

Today, I work with project teams and individual engineers to help them plan SQA, change control, and testing processes that allow them to understand and control the risks of product failure. I also assist in product risk analysis, test design, and in the design and implementation of computer-supported testing. Most of my experience is with market-driven Silicon Valley software companies like Apple Computer and Borland, so the techniques I've gathered and developed are designed for use under conditions of compressed schedules, high rates of change, component-based technology, and poor specification.

# Credits & Legal Notices

- These notes were originally developed in co-authorship with Hung Quoc Nguyen. James Bach has contributed substantial material. I also thank Jack Falk, Elizabeth Hendrickson, Doug Hoffman, Bob Johnson, Brian Lawrence, Melora Svoboda, and the participants in the Los Altos Workshops on Software Testing and the Software Test Managers' Roundtables. Additional acknowledgements appear on specific slides.
- These notes include some legal information, but you are not my legal client. I do not provide legal advice in the notes or in the course. Even if you ask me a question about a specific situation, you must understand that you cannot give me enough information in a classroom setting for me to respond with a competent legal opinion. I may use your question as a teaching tool, and answer it in a way that I believe would “normally” be true but my answer may be inappropriate for your particular situation. I cannot accept any responsibility for any actions that you might take in response to my comments in this course. If you need legal advice, please consult your own attorney.
- The practices recommended and discussed in this course are useful for an introduction to testing, but more experienced testers will adopt additional practices. I am writing this course with the mass-market software development industry in mind. Mission-critical and life-critical software development efforts involve specific and rigorous procedures that are not described in this course.

# Center for Software Testing Education & Research

[www.testingeducation.org](http://www.testingeducation.org)

*Our mission is to create effective, grounded, timely materials to support the teaching and self-study of software testing, software reliability, and quality-related software metrics.*

Florida Tech's Center for Software Testing Education & Research formed in November 2003, as a collaboration among

- Cem Kaner, Ph.D., J.D. (Professor) (Director)
- Walter P. Bond, Ph.D. (Associate Professor)
- Scott Tilley, Ph.D. (Associate Professor)
- Michael Andrews, Ph.D. (Assistant Professor)
- James Whittaker, Ph.D. (Professor)

We also collaborate closely with:

- James Bach (Satisfice), Bret Pettichord (Thoughtworks), Douglas Hoffman (Software Quality Methods)
- Steve Condly, Ph.D. (U Central Florida: Education), Pat Schroeder, Ph.D. (Milwaukee College of Engineering: Computer Science)

# About this course

Slides are at

<http://www.testingeducation.org/k04>

## How we teach it at Florida Tech

- **Watch the lecture before class**
- **You *might* also be required to do required readings before a given class.**
- **Quiz at the start of class.**
- **In class, we have discussions, student presentations, and labs.**
- **Assessments based on:**
  - **Quiz scores**
  - **Presentations**
  - **Assignments / Labs**
  - **Bonus assignments**
  - **Midterm and final**
- **Also available at the website are demonstrations / examples for many of the test techniques we cover in lecture.**

# What is testing?

*A technical investigation  
done to expose  
quality-related information  
about the product  
under test*

# Defining Testing

- **A technical**
  - We use technical means, including experimentation, logic, mathematics, models, tools (testing-support programs), and tools (measuring instruments, event generators, etc.)
- **investigation**
  - An organized and thorough search for information.
  - This is an active process of inquiry. We ask hard questions (aka run hard test cases) and look carefully at the results.
- **done to expose quality-related information**
  - see the next slide
- **about the product under test.**

# Information Objectives

Different objectives require different testing strategies and will yield different tests, different test documentation and different test results.

- Find important bugs, to get them fixed
- Check interoperability with other products
- Help managers make ship / no-ship decisions
- Block premature product releases
- Minimize technical support costs
- Assess conformance to specification
- Conform to regulations
- Minimize safety-related lawsuit risk
- Find safe scenarios for use of the product
- Assess quality

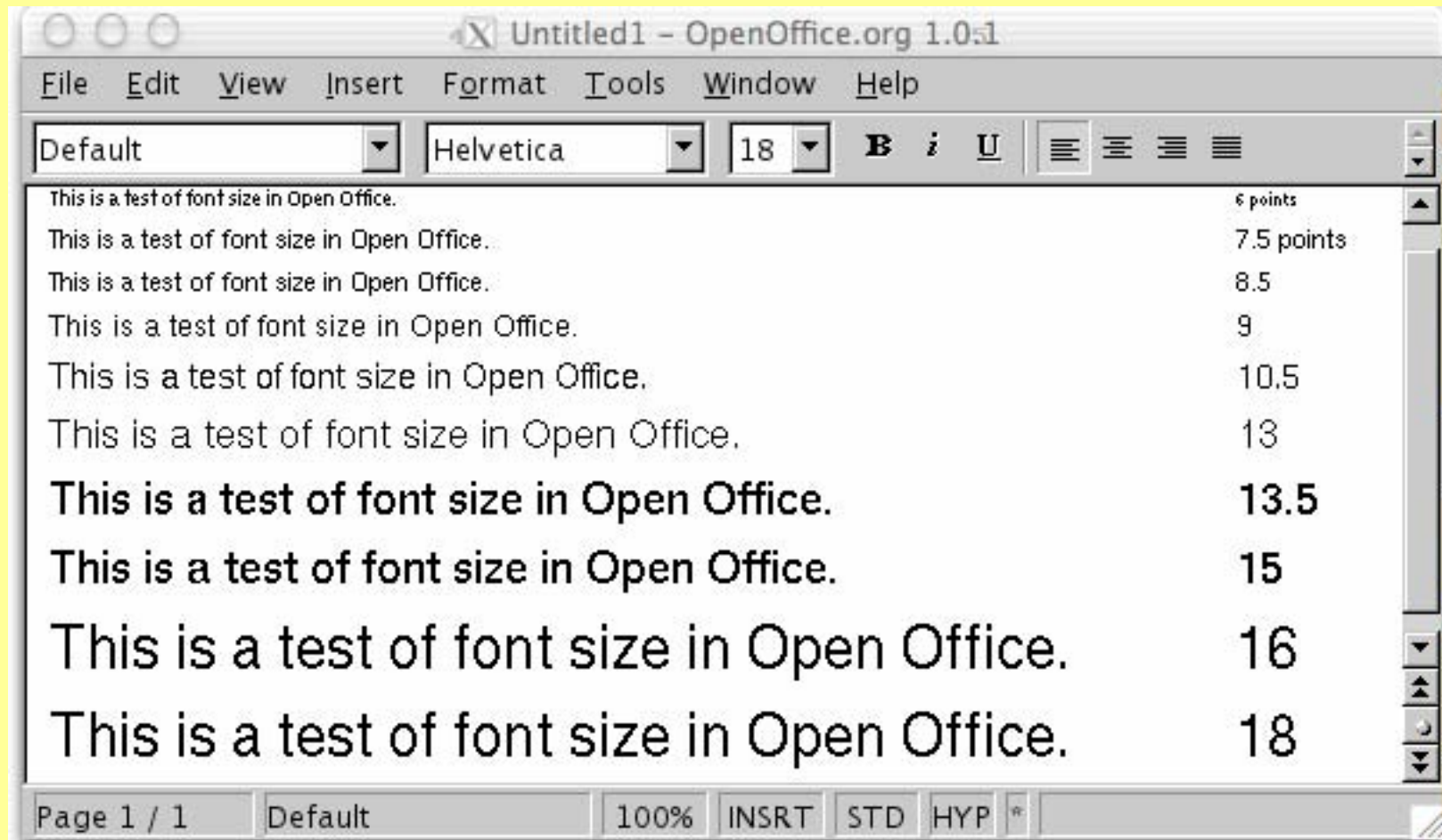
# Black Box Software Testing

## Part 1

### Oracles

*Testing is a cognitive activity,  
Not a mechanical activity.*

# Does font size work in Open Office? *What's your oracle?*



# Oracles

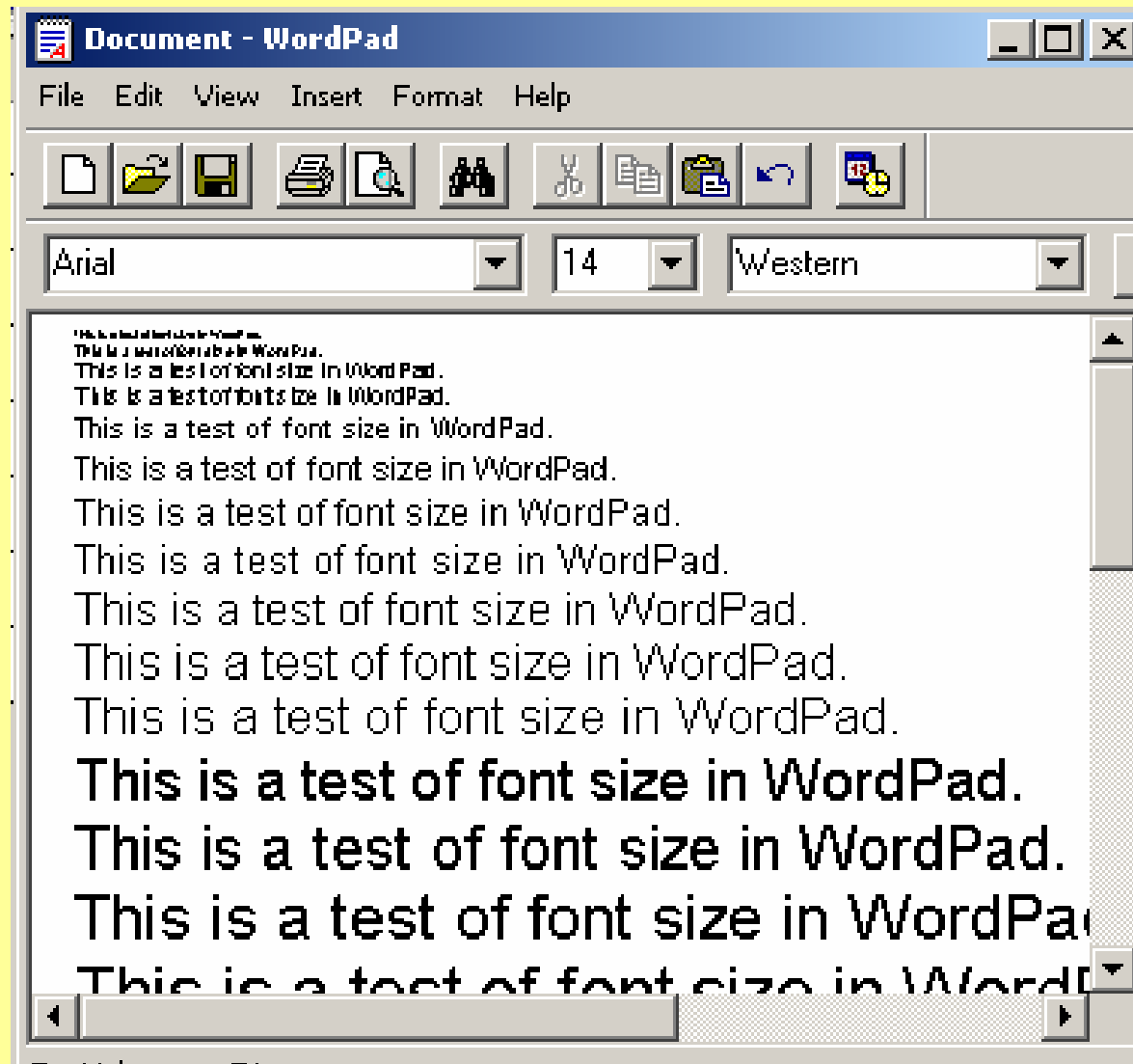
An oracle is the principle or mechanism  
by which you recognize a problem.

**“..it works”**

**really means...**

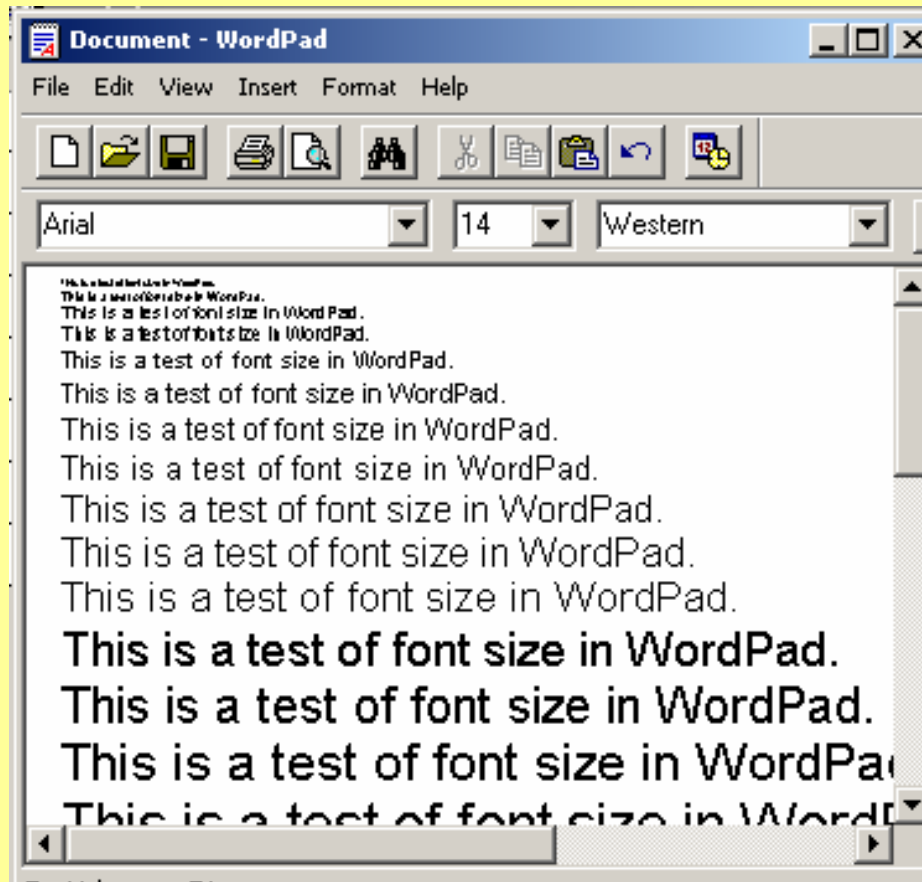
*“...it appeared to meet some requirement  
to some degree.”*

# OK, so what about WordPad?

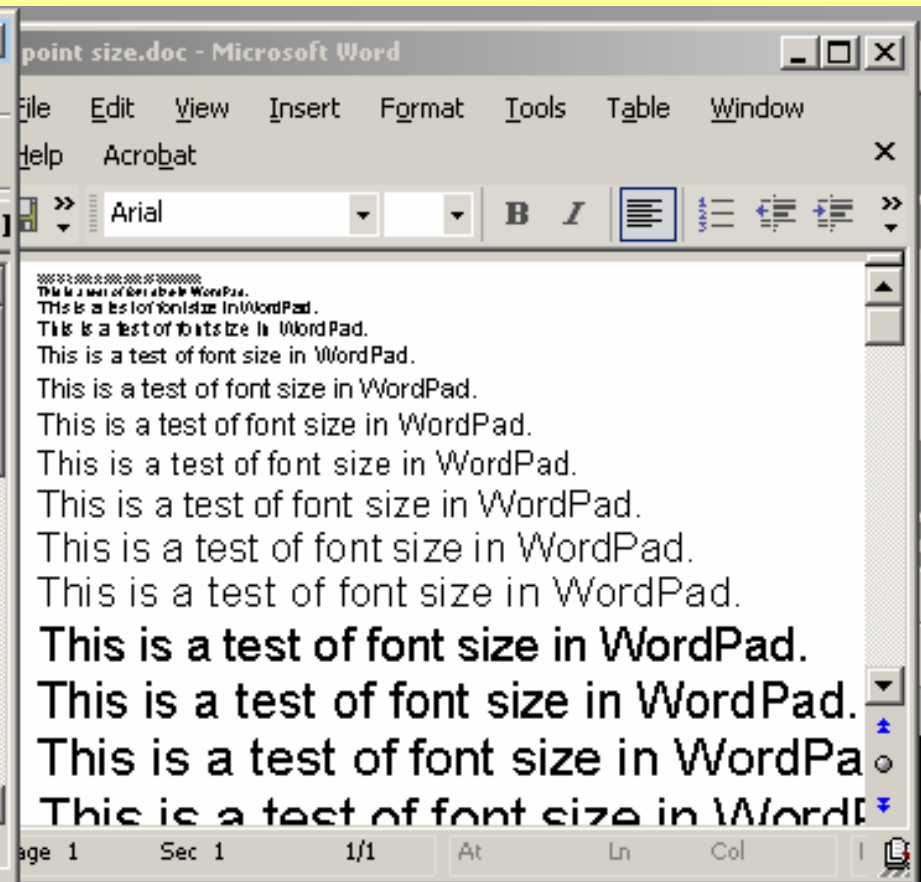


# What if we compared WordPad to Word?

## WordPad



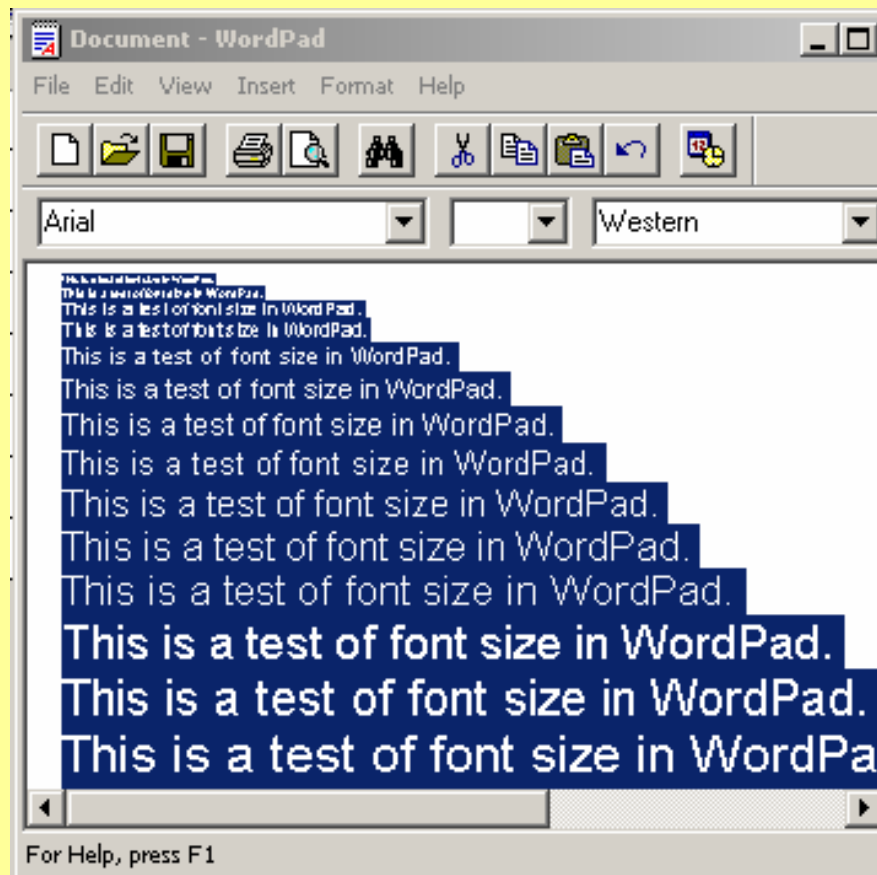
## Word



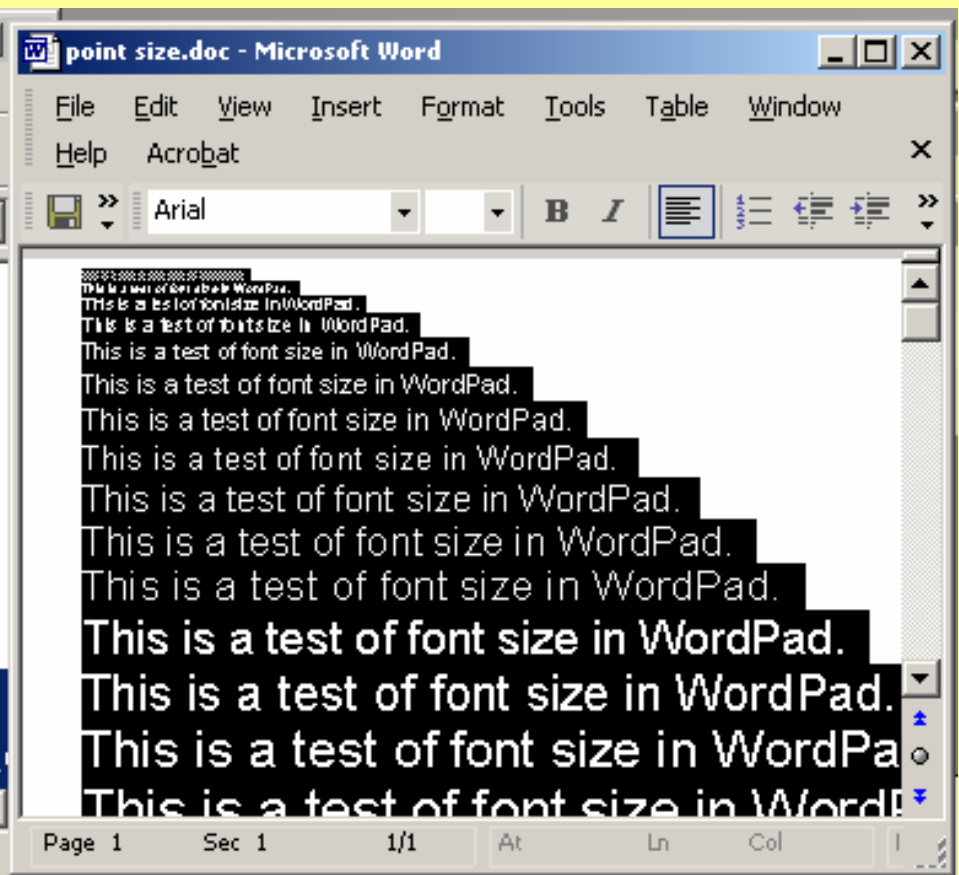
# Let's make the comparison easier

The highlighting makes the relative sizes stand out

## WordPad



## Word

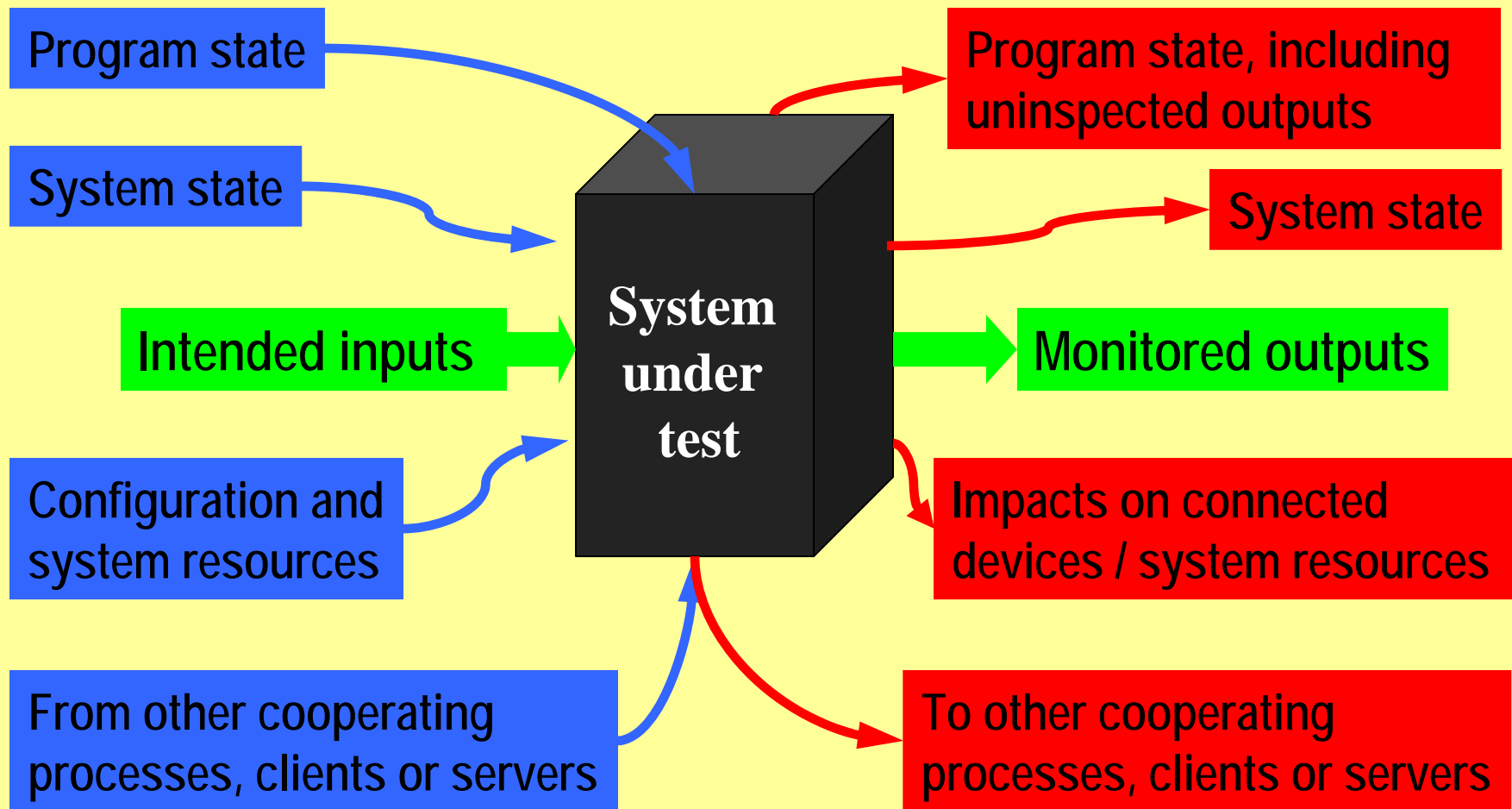


# So, what are we testing for?

- What to cover?
  - Every font size (to a tenth of a point).
  - Every character in every font.
  - Every method of changing font size.
  - Every user interface element associated with font size.
  - Interactions between font size and other contents of a document.
  - Interactions between font size and every other feature.
  - Interactions between font sizes and graphics cards & modes.
  - Print vs. screen display.
- What's your oracle?
  - What do you know about typography?
    - Definition of “point” varies. There are as many as six different definitions (<http://www.oberonplace.com/dtp/fonts/point.htm>)
    - Absolute size of characters can be measured, but not easily (<http://www.oberonplace.com/dtp/fonts/fontsize.htm>)
  - How closely must size match to the chosen standard?
  - Heuristic approaches: relative size of characters; comparison to MS Word.
  - Expectations of different kinds of users for different uses.

# A program can fail in many ways

*Based on notes from Doug Hoffman*



# The program can fail in many ways

- The 1100 embedded diagnostics
  - Even if we coded checks for each of these, the side effects (data, resources, and timing) would provide us a new context for Heisenberg uncertainty.
- The phenomenon of inattentional blindness
  - Humans (often) don't see what they don't pay attention to.
  - Programs (always) don't see what they haven't been told to pay attention to. (Programs are more precise and less flexible.)
- This is often the cause of irreproducible failures. We paid attention to the wrong conditions.
  - But we can't pay attention to all the conditions

Our tests cannot practically address  
All of the possibilities

# Risk as a simplifying factor

- For **Wordpad**, we don't care if font size meets precise standards of typography!
- In general it can vastly simplify testing if we focus on whether the product has a problem that matters, rather than whether the product merely satisfies all relevant standards.
- Effective testing requires that we understand standards as they relate to how our clients value the product.

Instead of thinking pass vs. fail,  
Consider thinking problem vs. no problem.

# Risk as a simplifying factor

- What if we applied the same evaluation approach
  - that we applied to WordPad
  - to Open Office or MS Word or Adobe PageMaker?

The same evaluation criteria lead to different conclusions in different contexts

- In risk-based testing, we choose the tests that we think are the most likely to expose a serious problem, and skip the tests that we think are unlikely to expose a problem, or likely to expose problems that no one would care about.

# Testing is about ideas. Heuristics give you ideas.

- A heuristic is a fallible idea or method that may you help simplify and solve a problem.
- Heuristics can hurt you when used as if they were authoritative rules.
- Heuristics may suggest wise behavior, but only in context. They do not contain wisdom.
- Your relationship to a heuristic is the key to applying it wisely.

“Heuristic reasoning is not regarded as final and strict but as provisional and plausible only, whose purpose is to discover the solution to the present problem.”

- George Polya, *How to Solve It*

# Heuristics

Billy V. Koen,  
Definition of the Engineering Method,  
ASEE, 1985

- A heuristic is anything that provides a plausible aid or direction in the solution of a problem but is in the final analysis unjustified, incapable of justification, and fallible. It is used to guide, to discover, and to reveal.
- Heuristics do not guarantee a solution.
- Two heuristics may contradict or give different answers to the same question and still be useful.
- Heuristics permit the solving of unsolvable problems or reduce the search time to a satisfactory solution.
- The heuristic depends on the immediate context instead of absolute truth as a standard of validity.

Koen offers an interesting definition of engineering

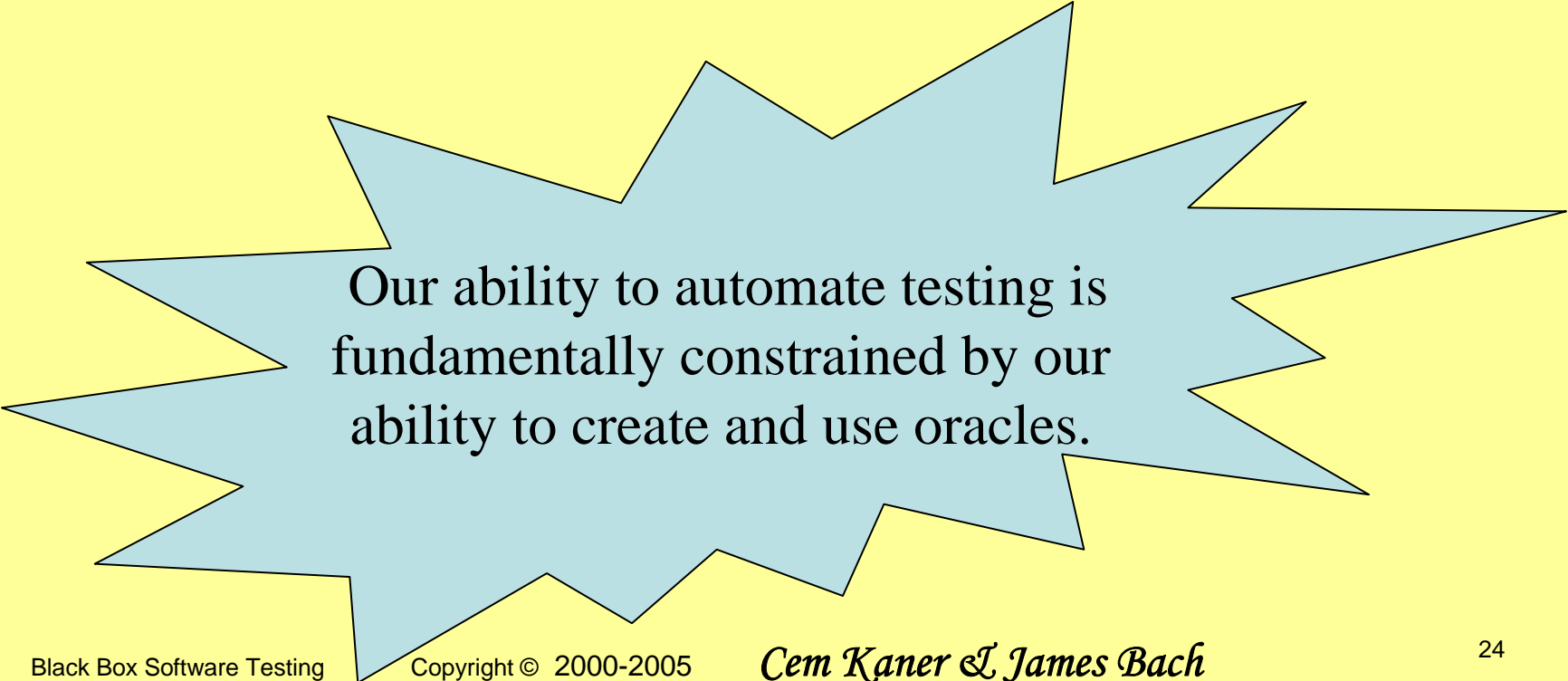
- *The engineering method is the use of heuristics to cause the best change in a poorly understood situation within the available resources (p. 70).*

# Some useful oracle heuristics

- **Consistent within Product:** Function behavior is consistent with behavior of comparable functions or functional patterns within the product.
- **Consistent with Comparable Products:** Function behavior is consistent with that of similar functions in comparable products.
- **Consistent with History:** Present behavior is consistent with past behavior.
- **Consistent with our Image:** Behavior is consistent with an image that the organization wants to project.
- **Consistent with Claims:** Behavior consistent with documentation or ads.
- **Consistent with Specifications or Regulations:** Behavior is consistent with claims that must be met.
- **Consistent with User's Expectations:** Behavior is consistent with what we think users want.
- **Consistent with Purpose:** Behavior is consistent with apparent purpose.

# The oracle problem and test automation

- We often hear that most testing should be automated.
- Automated testing depends on our ability to programmatically detect when the software under test fails a test.



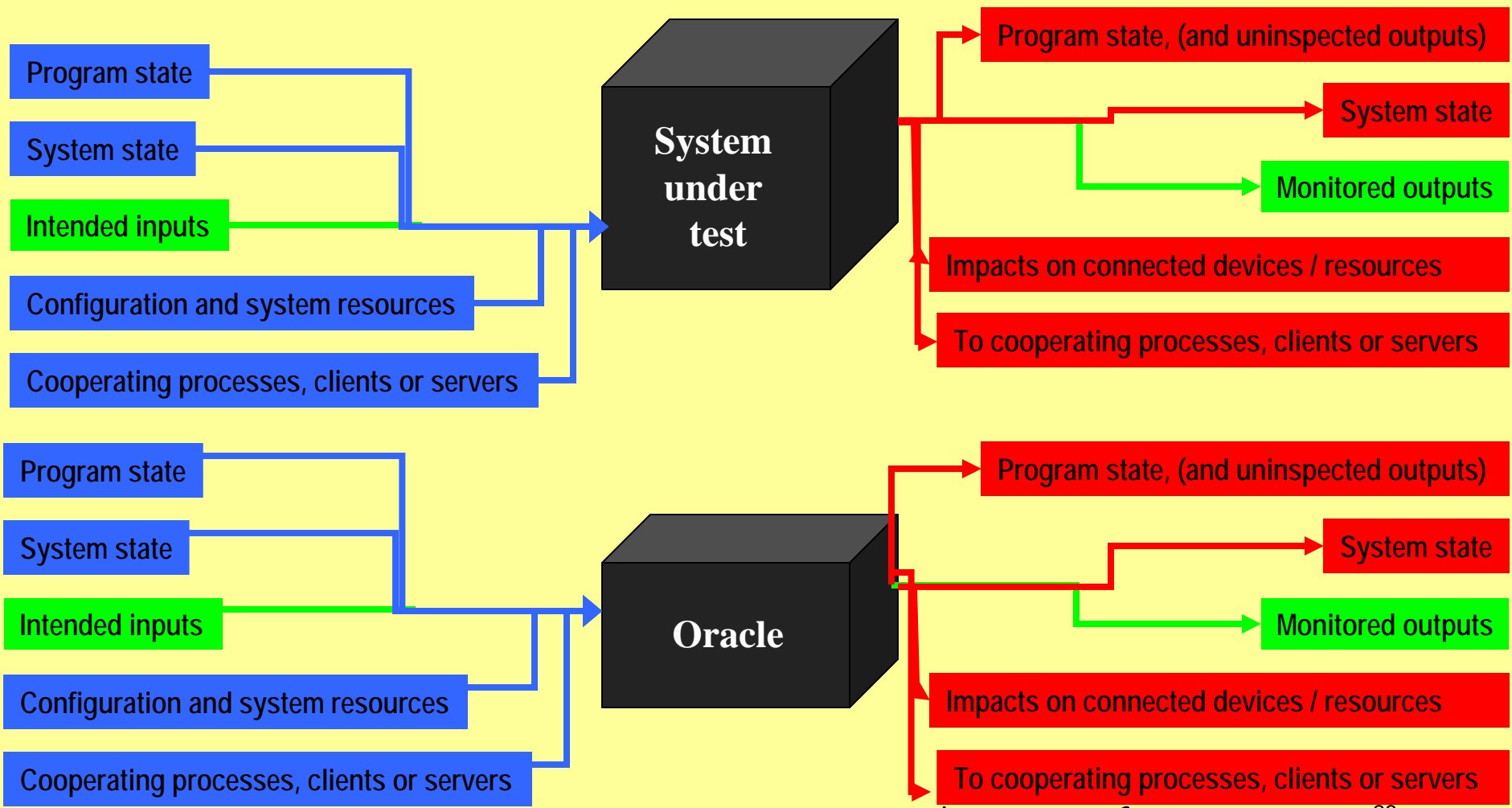
Our ability to automate testing is fundamentally constrained by our ability to create and use oracles.

# The oracle problem and automation

- One common way to define an oracle is as a source of expected results.
  - Under this view, you compare your results to those obtained from the oracle. If they match, the program passes. If they don't match, the program fails.
  - The comparison between MS WordPad and MS Word illustrates this approach.
  - *It is important to recognize that this evaluation is heuristic:*
    - **We can have false alarms:** A mismatch between WordPad and Word might not matter.
    - **We can miss bugs:** A match between WordPad and Word might result from the same error in both programs.
- Automated testing that depends on a comparison oracle is subject to both of these problems, false alarms and missed bugs. Whether you can automate or not, you still have to exercise judgment in picking the risks to test against and interpreting the results you get.

# What do you compare, when you use an oracle?

Based on notes from Doug Hoffman



# Closing thoughts

- Testing is a complex and challenging activity for a variety of reasons. Here are two that we considered in this lecture:
  - **The Strategy Problem.** There are many different objectives for testing. To be effective, testers have to design their strategy (how they'll test, what they'll look for, what test artifacts they'll create) to meet their stakeholders' objectives.
  - **The Oracle Problem.** When you run a test, you need some way to decide whether the program passed the test or failed it, and all of the methods we have available are imperfect.
- Next class, we'll look at the impossibility of complete testing. This is what gives the strategy problem its urgency. There isn't enough time to run all the tests you "should" run, so you have to carefully pick the tests you will run.