BIO
PRESENTATION

F7

November 8, 2002
11:15 AM

SMARTER TESTING WITH THE 80:20 RULE

Erik Petersen
Software Testing Consultant

International Conference On
Software Testing Analysis & Review
November 4-8, 2002
Anaheim, CA USA

Erik Petersen

Erik Petersen has moved through many SDLC roles since the mid 1980s, focusing on software testing and QA since the early 90s, working for small software houses, customized software providers (including CSC) and large companies. He has consulted on, tested and managed test projects for a diverse range of Australian and international companies (including IBM and Philip Morris).

Erik spent four and a half years at the ANZ Bank (one of Australia's largest), mainly consulting on test tools and process, as well as running testing for ANZ Internet Banking, ANZ-E*Trade and other eCommerce and intranet projects.

Erik has been an active participant in several software testing discussion groups since 1996, having email contact with many European and North American testers. Erik was a reviewer of Brian Marick's "Classic Testing Mistakes" presented at STAR 1997, and a reviewer of "Lessons Learned in Software Testing" by Kaner, Bach and Pettichord. He presented at AsiaSTAR 2001 and AsiaSTAR 2002.

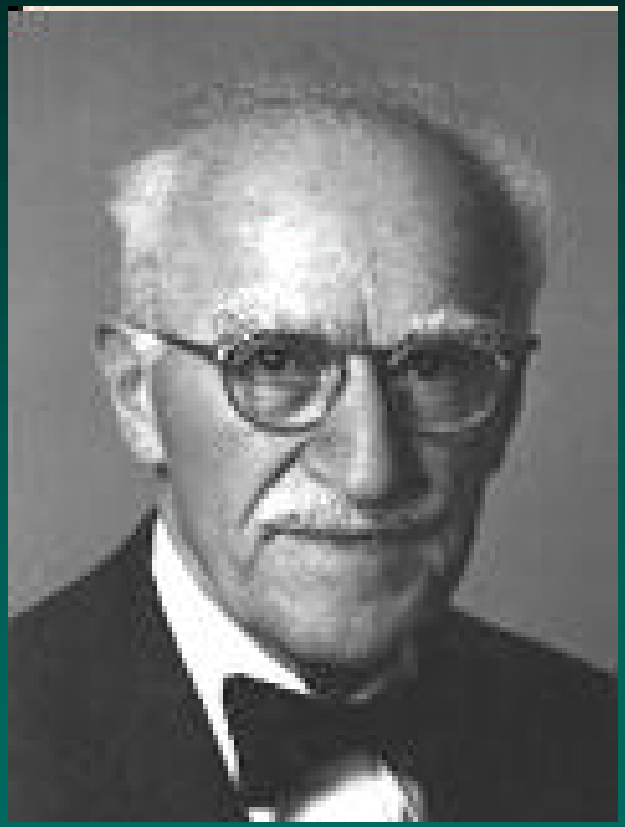
Contact him via **ecp@computer.org**

Smarter testing with the 80:20 rule

Erik Petersen,
Melbourne, Australia
ecp@computer.org

Visit the Software Testing Spot
at www30.brinkster.com/wvole

Who is this?



- Who is this man?
- Clue 1: He is an engineer and a lawyer
- Clue 2: He discovered the 80:20 rule

On the 80:20 rule

- " … [The 80/20 Principle] can multiply the profitability of corporations and the effectiveness of any organization. It even holds the key to raising the quality and quantity of public services while cutting their cost."

  From "***The 80/20 Principle : The Secret to Success by Achieving More With Less*** ", by Richard Koch, 1998

The Pareto Principle

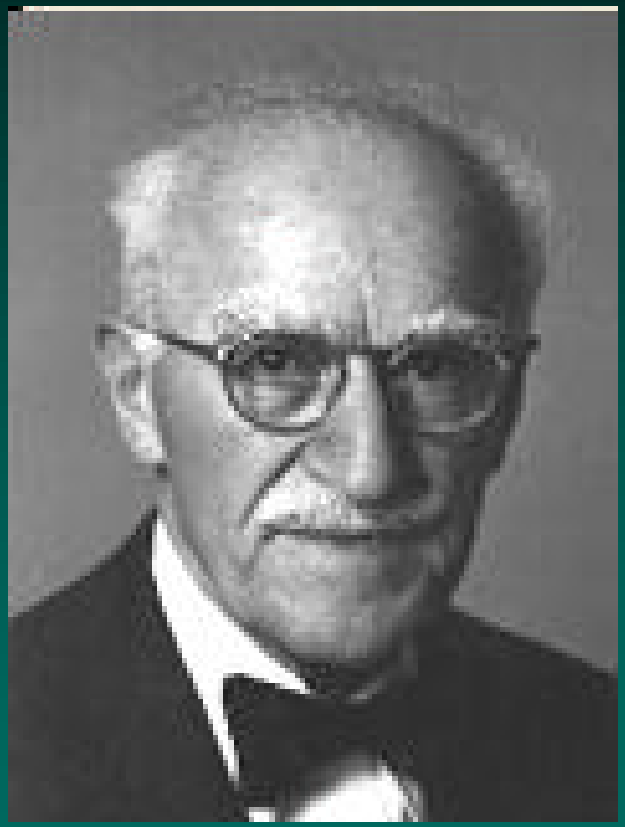
- What is the 80:20 rule? (a.k.a the Pareto Principle)
- **Answer**: For any group of things, 80% of the attributes of the group are due to only 20% of the members of the group.

Note that Pareto percentage varies but typically it is 80:20

Pareto Principle example

- In 1963, IBM discovered that roughly 80% of a computer's time was spent executing only 20% of the instructions, which was a counter-intuitive breakthrough at the time. IBM rewrote the operating system to make the 20% faster and easier to use, improving computer performance.
- Web traffic is also following the Pareto Principle, with 10% of web sites having 90% of traffic

Who is this, take 2?

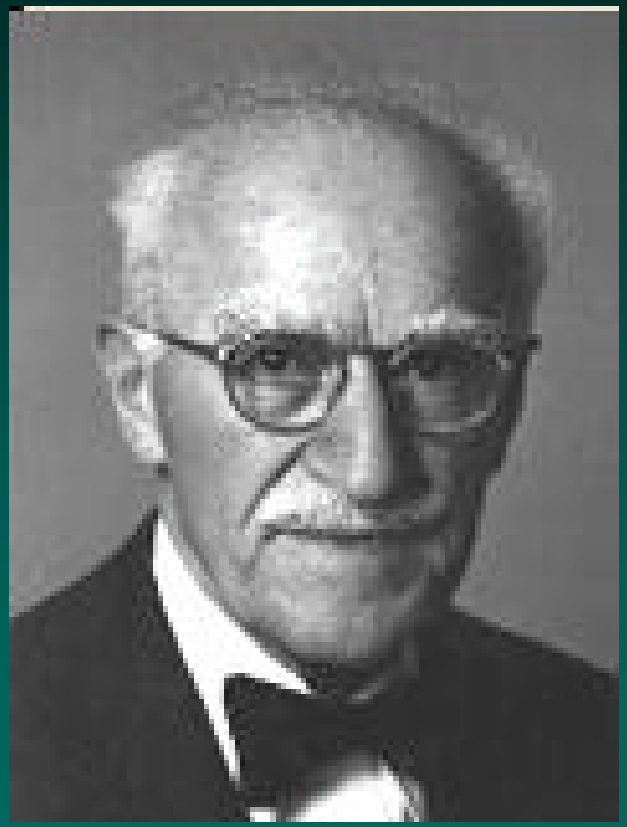


- So who is this man who discovered the Pareto Principle?

Vilfredo Pareto

- Economist and sociologist
- Discovered in 1880s that wealth of Italian population followed an 80:20 rule (80% wealth for 20% people)

Joseph Juran



- Engineer and lawyer
- Discovered 80:20 rule in 1940s after studies of quality in American industry, “the vital few and the trivial many”. Modestly named it after Pareto.
- Father of Total Quality Management (TQM)

Pareto sweet spots

- Sweet spots occur in a cricket or baseball bat, golf club or tennis racket, where a sportsperson can hit the ball and hardly notice the contact. It is the most productive way to hit the ball, with the least effort.
- The Pareto Principle offers a way to identify sweet spots - productive ways of achieving our goals quicker or maximizing use of focus points while minimizing effort.
- How do we identify the sweet spots and test smarter? Use Pareto analysis!

What is Pareto analysis?

- Named for Pareto by Juran
- Identifying the **main classifications/ causes** of the **properties/results** to identify sweet spots.
- There may be many different criteria for Pareto analysis.
- Uses a Pareto Diagram as visual representation to simplify analysis

What is a Pareto diagram?

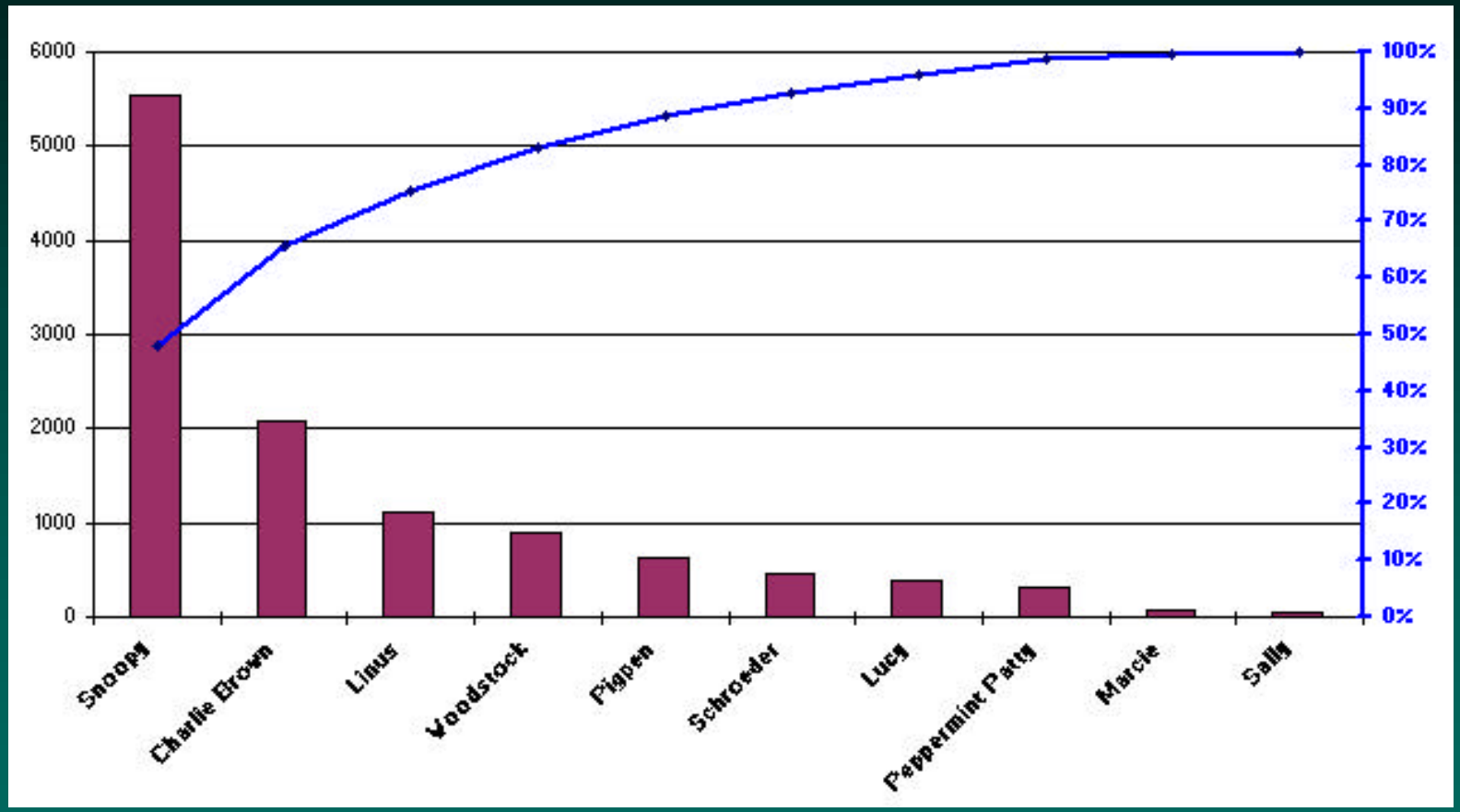
- Named for Pareto by Juran
- A Pareto diagram is a graph of **results** grouped logically by **cause** and **sorted** by most to least
- Pareto diagrams can be made with a spreadsheet or graphing tool.

The Peanuts Popularity Survey

- An internet poll in early 2000 invited people to vote for their favorite character from Snoopy. The survey had these results:
  - Snoopy **5529** - Charlie Brown **2076**
  - Linus **1108** - Woodstock **900**
  - Pig Pen **629** - Schroeder **463**
  - Lucy **392** - Peppermint Patty **323**
  - Marcie **82** - Sally **59**
  - Assorted others 199 (we'll ignore these)

Documented by Scott Jones of KPI Inc. and used with permission

The Peanuts Pareto



This descending curve across all outcomes is typical.

On living by the 80:20 rule

- "You'll miss the glorious world of timing, luck and surprise, all of which lurk in the shadows, not the light, of life. It's fine to understand life in terms of the Pareto Principle. Just don't live it that way."
  from Dr Contrarian's Guide to the Universe
- This "glorious world" sounds like a typical IT project! If the Pareto Principle can help us avoid it, it might be a good thing to know.

On Experience

- "Experience is the name a person gives to their mistakes"

  Oscar Wilde

- What can we learn from the experience of our mistakes?

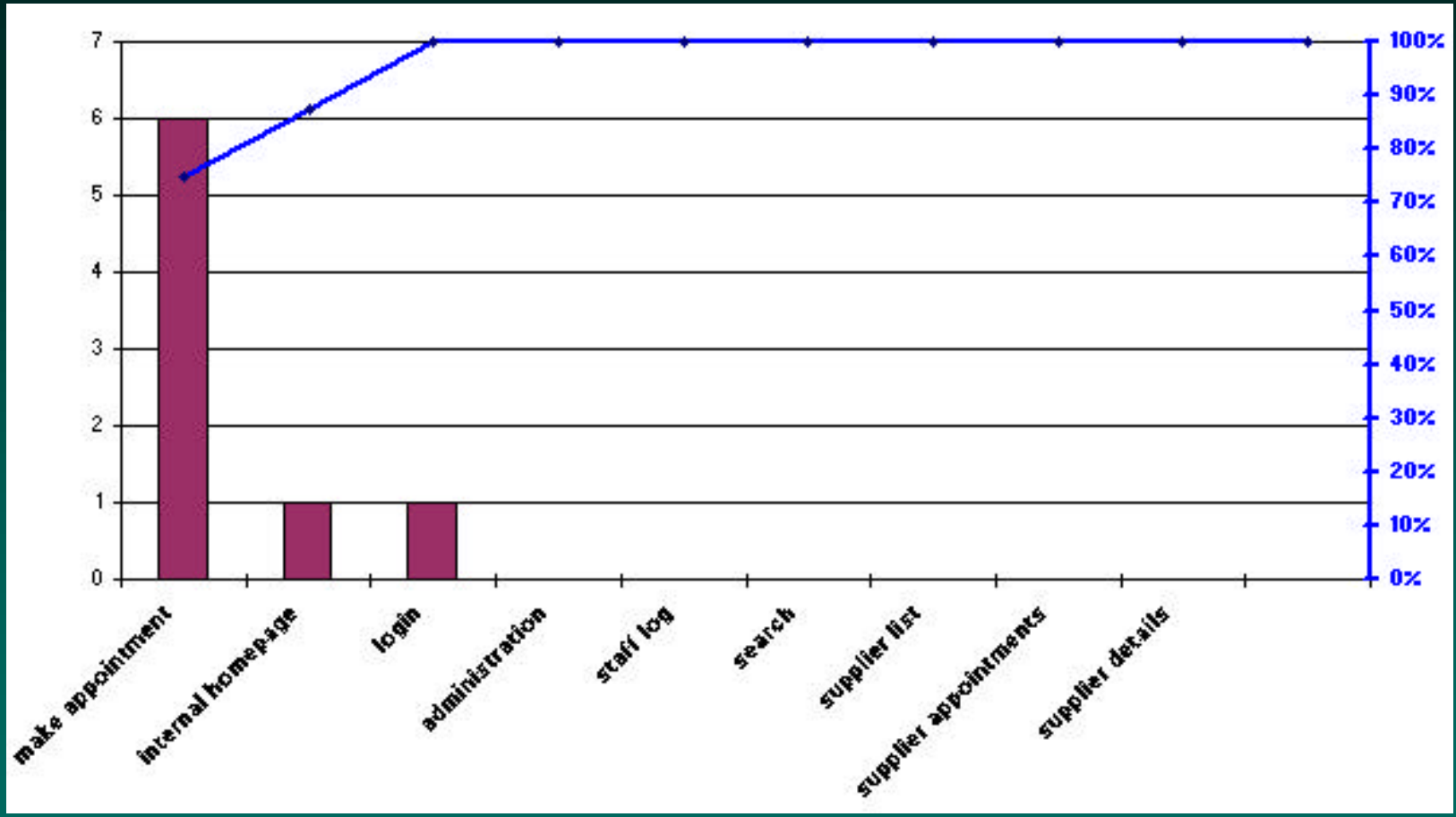
Pareto analysis of defects

- A Pareto diagram can be created for number of defects found against where/how they were found.
- Use a spreadsheet or some test management tools (though they may not sort the data).

Example : Project X

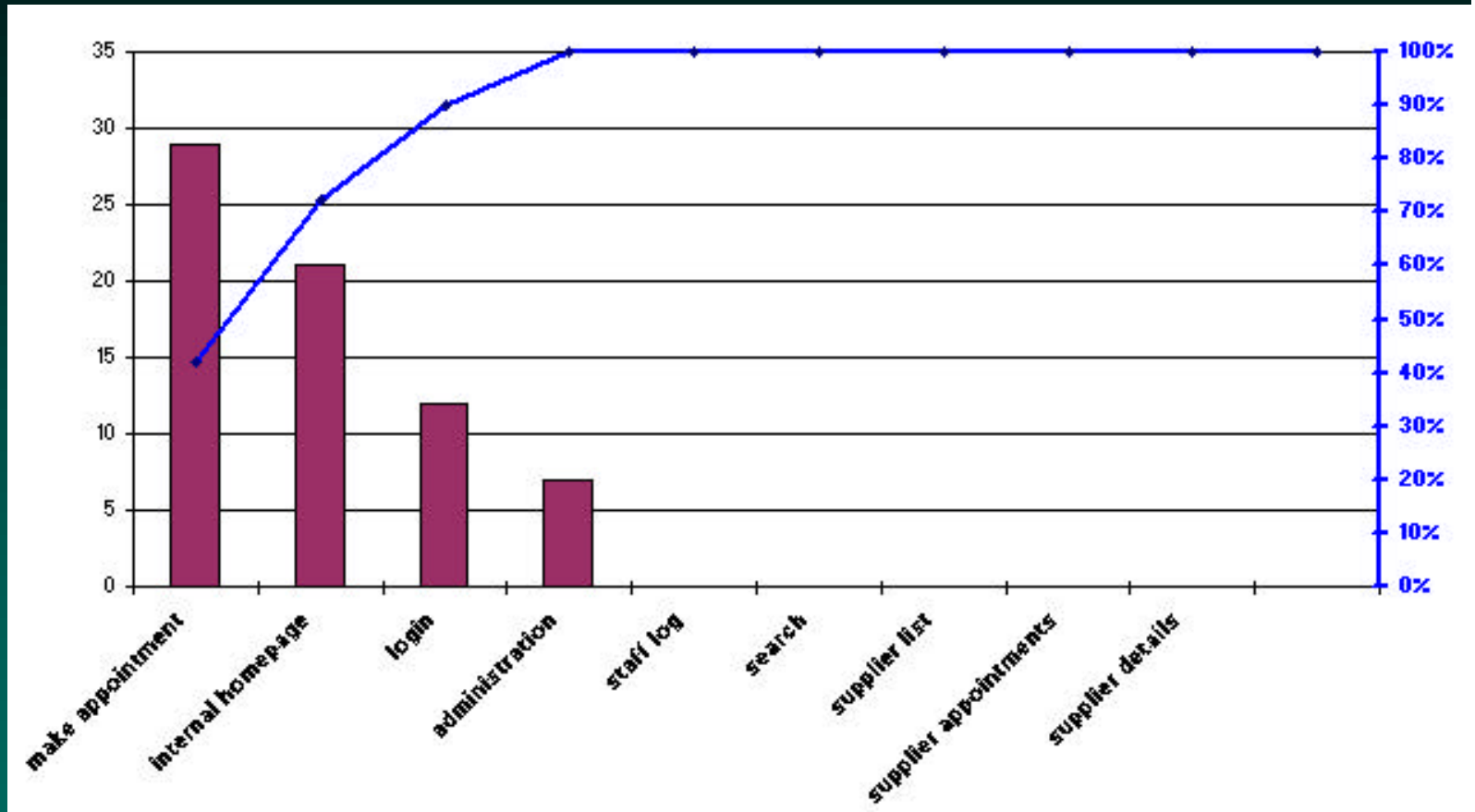
- Example of a small web system
- 9 major functional areas, renamed to protect the innocent. Login, internal homepage, administration, staff log, search, make appointment, supplier list, supplier appointments, supplier details.
- 69 defects found, classified into 5 severities
- Major defects were 1 sev one and 7 sev two
- What do the Pareto diagrams look like?

Project X (Pareto for sev 1 & 2 bugs)



The major bugs are concentrated in a few functions.

Project X (Pareto for all bugs)



All bugs are concentrated in about half of the functions!

This is not a normal Pareto graph. Is this typical? YES!

Defect density

- Snap shot from article "*Software Defect Reduction Top 10 List*", Basili & Boehm, IEEE Computer, Jan 2001. From a testing point of view, we are interested in 2 rules

- **Number 4**: About 80% of the defects come from 20% of the modules (**standard Pareto**)
  and about half the modules are defect free (**unique to software!**)
  (*range is 60-90%, with 80% median*)
  (*load testing: about 40% of modules have 60% defects*)

- **Number 5**: About 90% of the downtime comes from at most 10% of the defects

Defect density implications

- Once we find a defect, there is a strong chance we are in a sweet spot and will find more in the same place
- If we are running tests in an area and not finding defects, there is a strong chance it may be defect free (Note: Chance not Guarantee)

Identifying defect sweet spots

- During test execution, do Pareto analysis (with spreadsheet or test management/ defect tracking software) to identify sweet spots where defects are being found.
- Concentrate on sweet spots for scripted and exploratory testing

Finding deeper defects 1

- Recall the other relevant item from the Defect Reduction Top 10:
  Number 5: About 90% of the downtime comes from at most 10% of the defects
- We can identify our defect sweet spots during testing and concentrate on them for further scripted or exploratory testing. This will identify functional errors in behaviour, but what if there are further errors in the code that are not visible at the functional level?
- What can we do about this?

Finding deeper defects 2

- We need to push back on developers when further structural (unit or integration) testing is needed.
- This is much easier to say than do
- One approach is to look for bug clusters. Robert Sabourin and Mr Kim Davis, motivated by the 80:20 rule, wanted to try and see if they could save time and effort in web projects by concentrating on looking for bug clusters wherever possible or practical after initial bugs were found.

(continued...)

Finding deeper defects 3

- They performed fast probabilistic root cause analysis on bug clusters, prioritizing them then investigating them with tester-developer pairs. This resulted in 13 defect corrections for every 10 reported
- Ongoing investigation, now being used on multiple projects

Smarter Testing: Tip 1

- When you find bugs, look for others nearby

Reactive versus proactive testing

- We cannot do Pareto analysis until we have real defect information. Unfortunately, most of the defects are in 20% of the software
- Once we start finding defects, we can focus on where they are occurring but we may have to run many tests first. This is very reactive. Can we be more proactive and try to anticipate where the defects will be and start testing there first?

On tomorrow's problems

- "Today's risks are tomorrow's problems"
  Software Engineering Institute

- If we identify potential risk areas and build/test them first, we increase chances of finding defects quicker and maximizing fix and retest time

Risk identification

- Where we don't want problems
  - high use/important functions
- Where we may have problems
  - Functionally risky
    - use intuitive assessment, historical model or
    - use calculation risk = impact times likelihood
  - Structurally risky
    - assess dev risks (new or changed complex fn, late spec'd fn, fn with many interfaces, etc)
    - update with code review and unit test results
    - Feed structural risks into functional ones

Software usage

- Typically, 20% of a system used 80% of the time
- Identify the sweet spots (20%)
- Improve usability of the sweet spots if possible
- Use sweet spot focus to drive the functional, acceptance and load testing
- Number of tests in each area should reflect area's usage or importance

Smarter Testing: Tip 2

- Spend time testing where your users spend their time

Risk reduction

- Where we don't want problems
  - build prototypes with core functions first, e.g Extreme Programming (XP), EVO, other agile methods
- Where we may have problems
  - Functionally risky
    - schedule riskiest work first for dev and testing
    - revise risks constantly with new information, e,g from code reviews

(continued.....)

Risk reduction, continued

- Where we may have problems
  - Structurally risky
    - Use test team in reviews, early test planning, etc
    - do Pareto analysis of design/dev issues
    - improve (eg use Personal Software Process or similar) negative sweet spots of individual developers in estimating
      - estimate then write small programs to build up estimating skills
    - and unit testing
      - track errors in small programs then create custom checklists to trap main personal coding errors.
    - Pair program where possible, e.g XP

Risk based testing

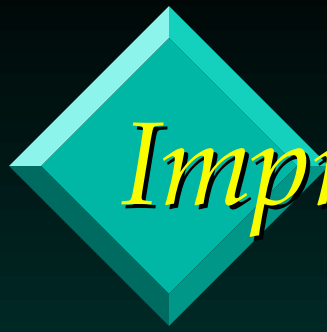
- This can be used in system, acceptance, usability, security, load testing, etc
- Reality replaces risk, so always revise after code reviews, unit tests or system tests
- A spreadsheet or test management software (e.g. TestDirector with customized fields, etc) can be used for this, both to verify the number of tests in each area reflects anticipated risk and to prioritise the order of testing
- If software is not safety critical, we can reach acceptable quality rapidly (if risk assumptions correct)
- Capture/replay tools used on high risk parts of a system will cover a large part of regression functionality

Smarter Testing: Tip 3

- Anticipate where the bugs will be and look there first

The largest room

- "The largest room in the world is the room for improvement" - Anonymous
- The Pareto Principle belongs in the middle of the largest room



Improving software development

- If we make these improvements using the 80:20 rule, can we leverage similar relationships to improve overall development and testing?
- Yes. Pareto Principle is a major analysis tool used in process improvement, particularly in the CMM (Capability Maturity Model).

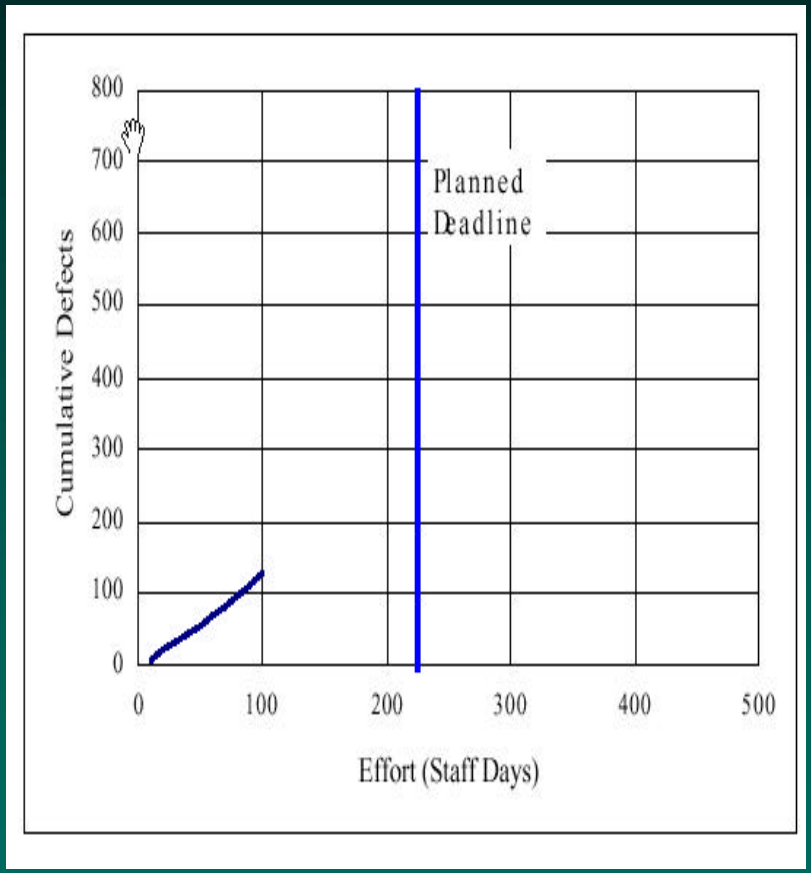
Defect based improvement

- We've found that defects (like diamonds) aren't found everywhere. Can we use this information to improve how we test? Yes.
- Pareto analysis of defect causes can be used to identify the most common type of defects created across the team (not just by individuals a la PSP) improving process and speeding delivery.
- ODC is a good technique for test process improvement (TPI).

Orthogonal Defect Classification

- ODC invented at IBM in 1990 by Ram Chillarege
- Used by IBM, Motorola, Lucent, Nortel, Cisco and Phillips among others
- Defects classified by independent, invariant attributes, & independent of dev model
- Reduces root cause analysis cost by factor of 10

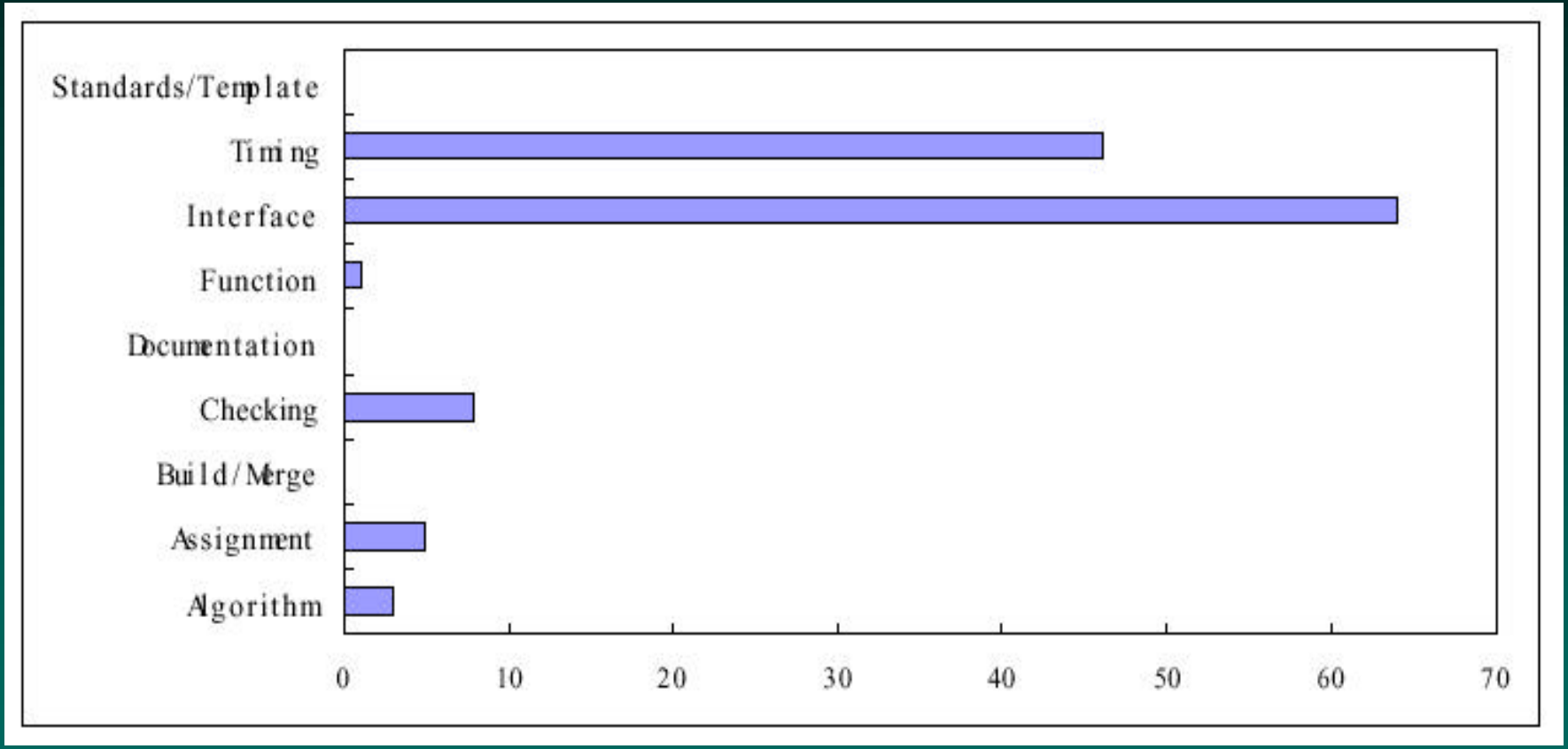
ODC TPI: Early Defect count on Project Y



- Example: New features added to 2nd product release. New feature testing and regression testing was not finding defects fast enough to meet deadline.
- Pareto analysis of ODC was done on 1st release defects

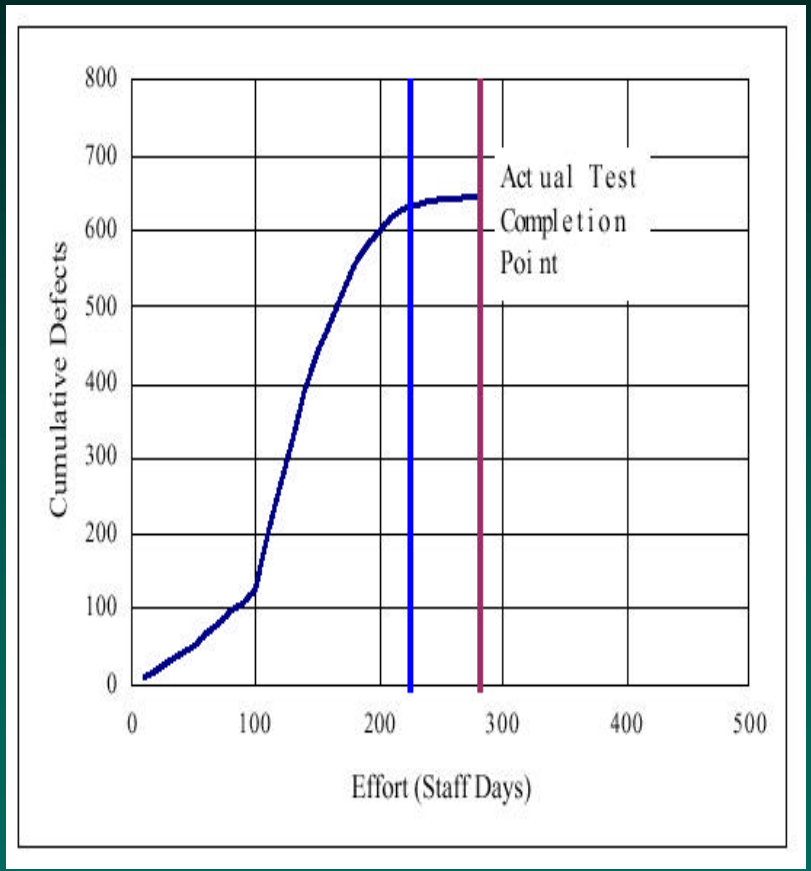
Project Ydetails used with permission from Albert Liu of Motorola Global Software Group, China.

ODC TPI:
ODC defect analysis on Project Y



The sweet spots are in Timing and Interface defects.

ODC TPI: Final Defect count on Project Y



- Test focus switched to sweet spot areas (timing and interfaces) and test effectiveness improved
- Testing completed with minimal delay.

Smarter Testing: Tip 4

- Leverage the 80:20 rule to improve the test and development process at all levels

Smarter Testing tips

- When you find bugs, look for others nearby
- Spend time testing where your users spend their time
- Anticipate where the bugs will be and look there first
- Leverage the 80:20 rule to improve the test and development process at all levels
- The 

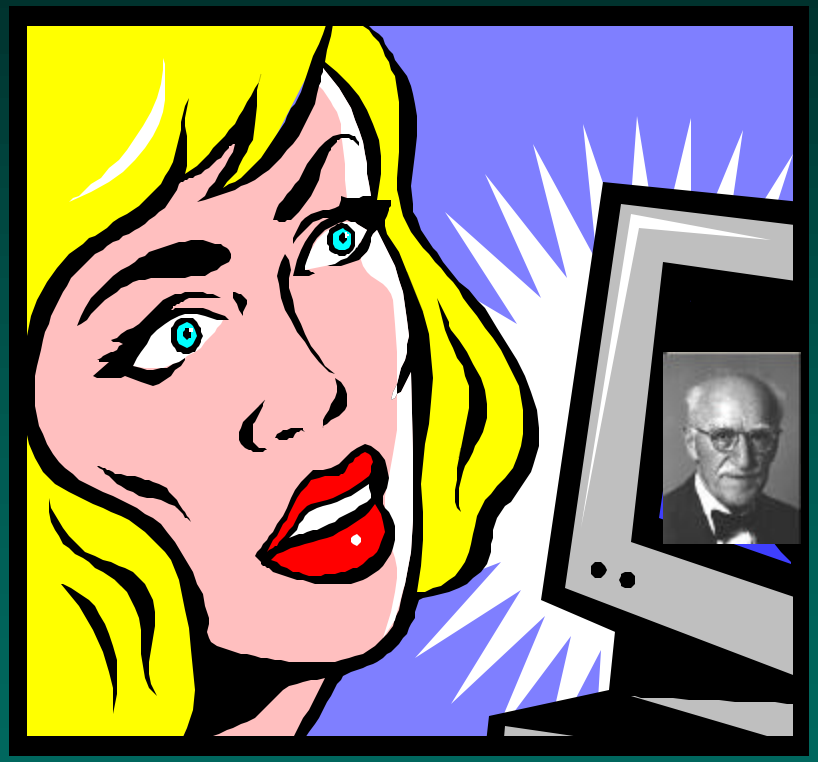
rule is smarter testing rocket fuel!!!

Last Words

- "Give me fruitful error any time, full of seeds, bursting with its own corrections."
  [comment on Kepler]
  - Vilfredo Pareto, 1848-1923
- "Give me fruitful sweet spots any time, full of seeds, bursting with Pareto benefits."
  - Erik Petersen, 2002

Project Z: over to you.....

- Questions now?
  Ask away.
- Questions after?
  Grab me, or email
  ecp@computer.org



REFERENCES

80:20 quote and IBM Pareto example, from “The 80:20 principle: The secret of achieving more with less”, intro at www.portalalfa.com/time/knjige/book1.htm

IBM example also in “Pareto Programming” at//softwaredev.earthweb.com

Web Pareto example, from article “90% of web traffic goes to 10% of web sites”, at www.onlinepublishingnews.com

Pareto picture from http://www.bently.com/articles/999pareto.asp

Juran Picture from http://qualite.univ-lyon1.fr/historique/juran.html

Peanuts and Pareto (used with permission) at http://www.keyperfin.com/Articles/Peanuts%20&%20Pareto.htm

Dr Contrarian’s Guide to the Universe, at http://www.allskewedup.com/why%20do%20we%20live%20this%20way.html

Rob Sabourin material at www.amibug.com

For risks material, see web and stickyminds.com, e.g A risk based testing strategy, I.Ottevanger

See web for XP, PSP and ODC material. For usability, see the Software Testing Spot.

Pareto “bursting” quote, at http://www.ndirect.co.uk/~greenprac/jamie/quotations.html