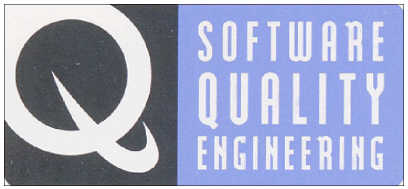
The Banana Principle For Testers:

Knowing When To Stop Testing

Lee Copeland

lee@sqe.com



The Banana Principle

- **A little boy comes home from school and his mother asks, “What did you learn in school today?” The boy responds, “Today we learned how to spell ‘banana’ but we didn’t learn when to stop.”**

– Jerry Weinberg
An Introduction to General Systems Thinking

The Banana Principle For Testers

- As testers we know how to design effective and efficient test cases

- BUT
  - How do we know when to stop?
  - How do we know we have done enough testing?

Knowing When To Stop

- **"Testing ends when we have measured system capabilities and corrected enough of the problems to have confidence that we are ready to run the acceptance test.**

– Bill Hetzel
The Complete Guide to Software Testing

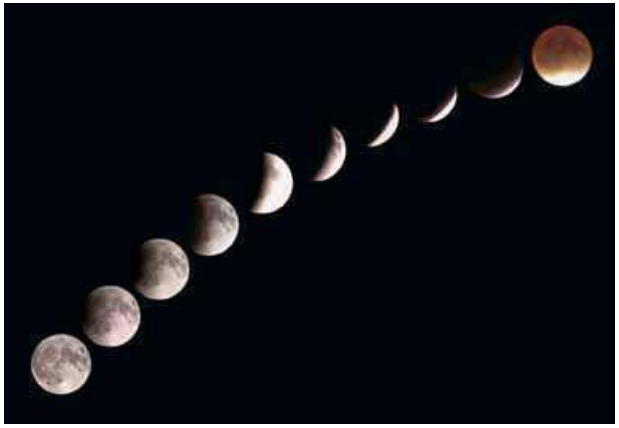
- **Unfortunately, "corrected enough" and "have confidence," while correct, are certainly vague.**

Five Basic “Stopping” Criteria

- You have met previously defined coverage goals
- The defect discovery rate has dropped below a previously defined threshold
- The marginal cost of finding the “next” defect exceeds the expected loss from that defect
- The project team reaches consensus that it is appropriate to release the product
- The boss says, “Ship it!”

Coverage Goals

- **‘Coverage’ is a measure of how much has been tested compared with how much is available to test.**



Code Coverage Goals

- Coverage can be defined at the code level with metrics such as:
  - Statement coverage
  - Branch coverage
  - Path coverage

Integration Coverage Goals

- Coverage can be defined at the integration level with metrics such as:
  - APIs tested
  - API/parameter combinations tested

System Coverage Goals

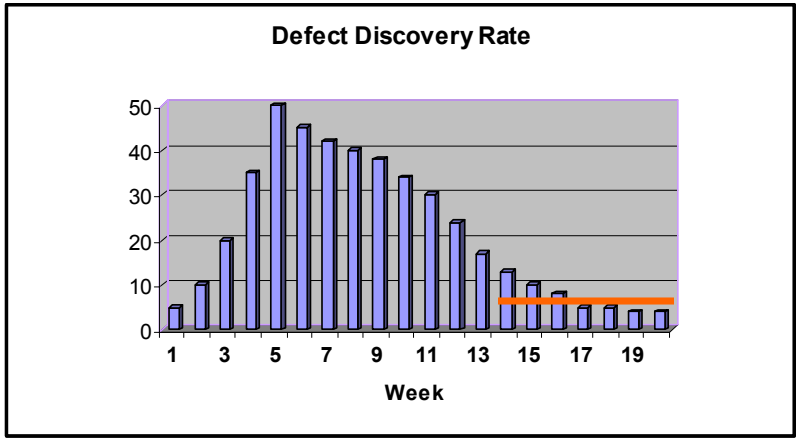
- Coverage can be defined at the system level with metrics such as:
  - Functions tested
  - Use cases tested
  - Use case scenarios (main path and exception paths) tested

Coverage Based Stopping Criteria

- A project's stopping criteria could be defined, for example, as:
  - 100% statement coverage AND
  - 90% use case scenario coverage

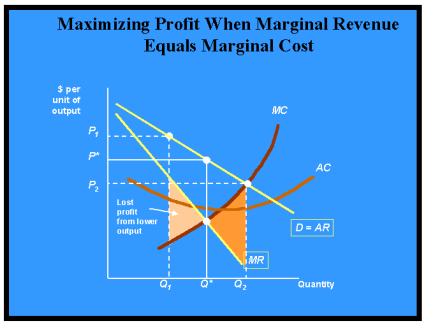
Defect Discovery Rate

- Each week (or some other short period of time) we count the number of defects discovered
- When the discovery rate is less than a certain previously selected threshold, we are finished testing



Marginal Cost

- In manufacturing, ‘marginal cost’ is defined as the cost associated with one additional unit of production
- In manufacturing, the marginal cost typically decreases as the number of units increases



- In software testing, however, just the opposite occurs

Marginal Cost

- Finding the first few defects is relatively simple and inexpensive

- Finding each additional defect becomes more and more time consuming and costly

- At some point, the cost of finding the “next” defect exceeds the loss that defect would cause

Consensus

- **Based on various factors including technical, financial, political, and just "gut feeling," the project team (managers, developers, testers, marketing, sales, quality assurance, etc.) decide that the benefits of delivering the software now outweigh the potential liabilities**

“Ship It!”

- For many of us, this will be the only strategy we will ever personally experience
- What testers must remember is that there may be very reasonable and logical reasons for shipping the product before we think it is ready
- Remember, our role is to make sure management is adequately informed of the risks, not to make this decision for them

Summary

- Let's discuss the advantages and disadvantages of each of these approaches:
  - Coverage goals
  - Defect discovery rate
  - Marginal cost
  - Consensus
  - "Ship it!"

Thanks

- Thanks for joining with me today
- If I can be of assistance, or if you'd just like to chat, please contact me at

lee@sqe.com

- And remember, bananas are fat-free, sodium-free, cholesterol-free, contain eight amino acids, and are an excellent source of vitamins B6 and C, and potassium