Software Quality

Topics

- What is quality?
- Defect-removal methods
  - Testing
  - Design and Code reviews
  - Inspections
- Defect prevention

What is Quality?

- Basic definition: meeting the users' needs
  - *needs*, not wants
  - true functional needs are often unknowable
- There is a hierarchy of needs.
  - Do the required tasks.
  - Meet performance requirements.
  - Be usable and convenient.
  - Be economical and timely.
  - Be dependable and reliable.

The PSP Quality Focus -1

- To be useful, software must
  - install quickly and easily
  - run consistently
  - properly handle normal and abnormal cases
  - not do destructive or unexpected things
  - be essentially bug-free
- Defects are not important to users, as long as they do not
  - affect operations
  - cause inconvenience
  - cost time or money
  - cause loss of confidence in the program's results

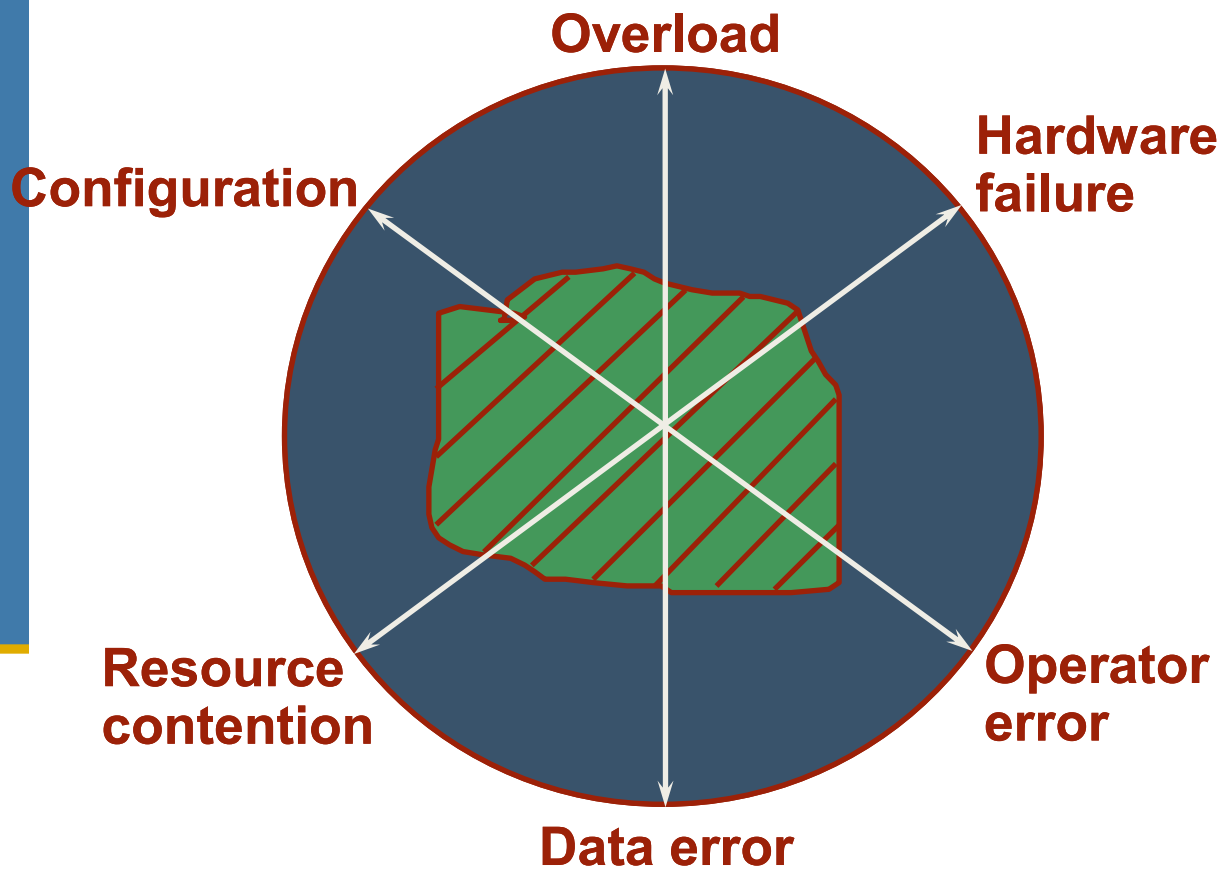
The PSP Quality Focus -2

- The defect content of software products must be managed before more important quality issues can be addressed.
- Low defect content is an essential prerequisite to a quality software process.
- Since low defect content can best be achieved where the defects are injected, engineers should
  - remove their own defects
  - determine the causes of their defects
  - learn to prevent those defects

The Economics of Quality

- Software is the only modern technology that relies on testing to manage quality.
- With common quality practices, software groups typically
  - spend 50+% of the schedule in test
  - devote more than half their resources to fixing defects
  - cannot predict when they will finish
  - deliver poor-quality and over-cost products
- To manage cost and schedule, you must manage quality.
- To get a quality product out of test, you must put a quality product into test.

Testing Alone is Ineffective



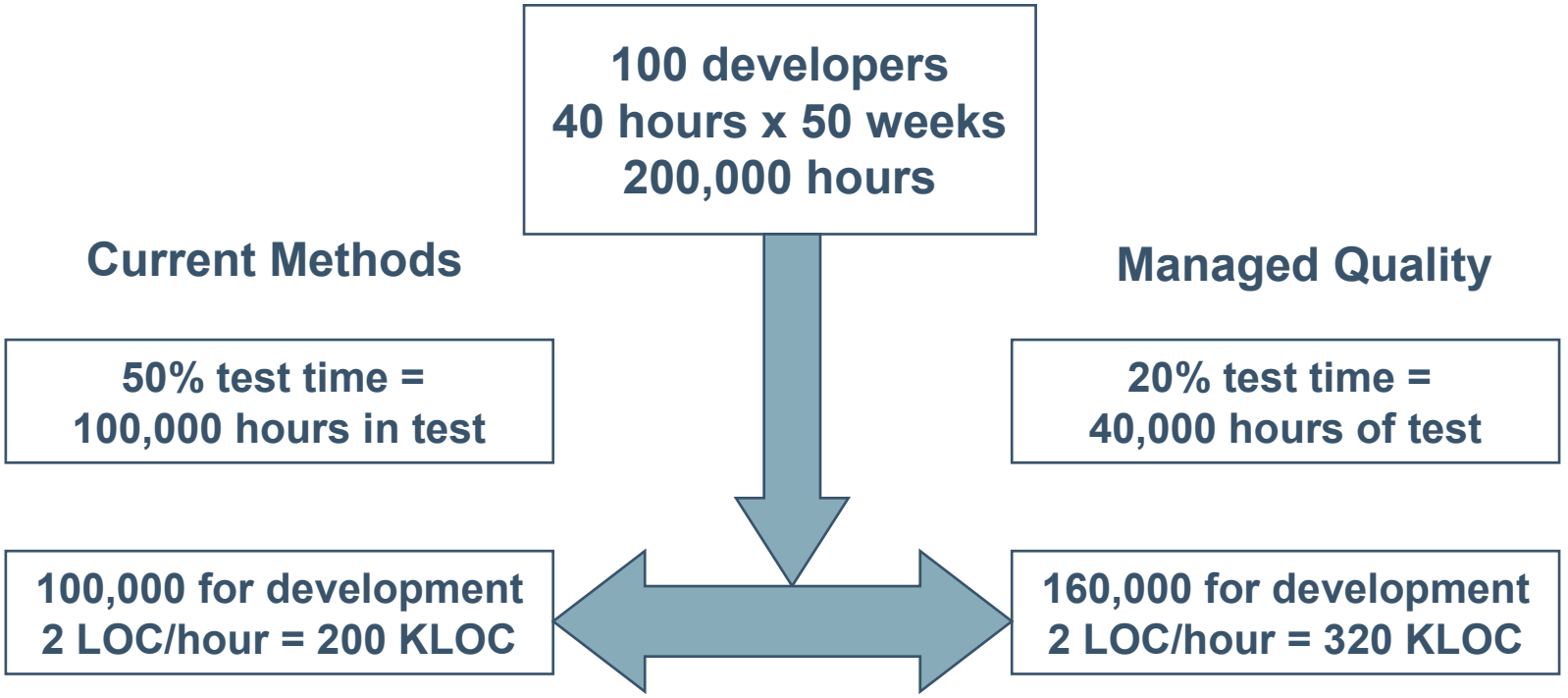
Safe and secure region = tested (shaded)
Unsafe and insecure region = untested (unshaded)

Removing Defects in Test

- When performing a task thousands of times, economics would suggest that you use the most efficient methods.
- A 50,000 LOC system with traditional development methods would
  - have 25+ defects/KLOC at test entry - 1250 defects
  - take 12,500+ programmer hours to test
  - be late and over budget

** typical rate of 10+ hours per defect

Quality and Productivity



Managed quality = 60% increased team productivity

Defect-removal Methods -1

- The principal ways to find and fix defects are by
  - compiling
  - unit testing
  - integration and system testing
  - team inspections
  - personal reviews

- Since you will likely have to remove lots of defects, you should use the most efficient methods.

Defect-removal Methods -2

- In a personal review
  - you privately review your product
  - your objective is to find and fix defects before test
- Reviews are most effective when they are structured and measured.
- Use reviews for requirements, designs, code, and everything else that you develop.
- Also continue to use inspections, compiling, and testing.

Defect-removal Rates -1

- Even at the personal level, it is more efficient to find defects in reviews than in testing.
  - Unit test finds only about 2 to 4 defects per hour.
  - Unit test finds about 50% of the defects.
  - Code reviews find about 6 to 10 defects per hour.
  - Practiced reviewers can find 70% or more of the defects before compiling or testing.

Why Reviews are Efficient

- In testing, you must
  - detect unusual behavior
  - figure out what the test program was doing
  - find where the problem is in the program
  - figure out which defect could cause such behavior
- This can take a lot of time.
- With reviews and inspections, you
  - follow your own logic
  - know where you are when you find a defect
  - know what the program should do, but did not
  - know why this is a defect
  - are in a better position to devise a correct fix

Review Principles

- PSP reviews follow a defined process with guidelines, checklists, and standards.
- The PSP review goal is to find every defect before the first compile or test.
- To address this goal, you should
  - review before compiling or testing
  - use coding standards
  - use design completeness criteria
  - measure and improve your review process
  - use a customized personal checklist

The Code Review Checklist

- Your reviews will be most effective when your personal checklist is customized to your own defect experience.
  - Use your own data to select the checklist items.
  - Gather and analyze data on the reviews.
  - Adjust the checklist with experience.

- Do the reviews on a printed listing, not on the screen.
- The checklist defines the review steps and the suggested order for performing them.
- Review for one checklist item at a time.
- Check off each item as you complete it.

Design Review Principles

- In addition to reviewing code, your should also review your designs.

This requires that you

- produce designs that can be reviewed
- follow an explicit review strategy
- review the design in stages
- verify that the logic correctly implements the requirements

Reviewable Designs

- A reviewable design has a
  - defined context
  - precise representation
  - consistent and clear structure

- This suggests that
  - the design's purpose and function be explicitly stated
  - you have criteria for design completeness
  - the design is structured in logical elements

The Design Review Strategy

Produce designs that can be reviewed in stages.

The suggested review stages are as follows.

1. Review against the requirements to ensure that each required function is addressed by the design.
2. Verify the overall program structure and flow.
3. Check the logical constructs for correctness.
4. Check for robustness, safety, and security.
5. Check the function, method, and procedure calls to ensure proper use.
6. Check special variables, parameters, types, and files for proper use.

Reviews and Inspections

- The principal focus of inspections should be to find problems that you have missed.
- When programs have many simple defects, inspectors are distracted and often miss more important problems.
- Reviewing the code first
  - provides a quality product for the inspection
  - shows respect for the inspectors' time
  - produces higher-quality inspections
  - produces higher-quality products

Defect Prevention

- Defect prevention is important because
  - it is always expensive to find defects
  - if the defects can be prevented, you can avoid the costs of finding and fixing them
  - defect prevention analysis costs are incurred once, but the savings apply to every project
- Defect prevention should follow an orderly strategy and a defined process.
- For the PSP, defect prevention actions include gathering defect data, improving design methods, and prototyping.

Defect Prevention Strategy -1

- Set priorities for the defect types that are most
  - frequently found
  - troublesome
  - easily prevented
  - annoying
- The defect-prevention process has the following steps.
  - Follow an established schedule.
  - Select one or two defect types for initial action.
  - Measure the effectiveness of defect prevention.
  - Make adjustments and continue.

Defect Prevention Strategy -2

- When setting initial priorities, consider the defect types found most frequently in integration and system test.
- Use PSP data to pick one or two defect types for initial action.
- Don’t just try harder; establish explicit prevention actions.
- Incorporate these actions into your process scripts, checklists, and forms.

Summary

- Improve product quality and accelerate development work by
  - doing reviews and inspections to remove defects before test
  - using testing to check product quality, not to remove volumes of defects
- Design and code reviews
  - improve the quality of your programs
  - save development time
- To do effective reviews, you must
  - establish review goals
  - follow a disciplined review process
  - measure and improve your review practices