

Disciplined Software Engineering

Lecture #8

**Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213**

Sponsored by the U.S. Department of Defense

Copyright © 1994 Carnegie Mellon University

Disciplined Software Engineering - Lecture 8 1

Lecture Overview

What is quality?

- **product and process quality**
- **quality economics**

The quality strategy

- **characterizing a process**
- **benchmarking a process**

Yield management

- **defect removal**
- **defect prevention**

Copyright © 1994 Carnegie Mellon University

Disciplined Software Engineering - Lecture 8 2

What is Software 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

Dependable and Reliable

To be used, the software must

- install quickly and easily
- run consistently
- properly handle normal and abnormal cases
- not do destructive or unexpected things
- be essentially bug free

The latent bugs must

- be operationally insignificant
- not be destructive
- rarely be evident

A Quality Process

Produces quality products

Meets its users needs

You are the user of the PSP process

Your customers are your

- management**
- peers and associates**
- product's users**

The PSP Quality Focus - 1

In this course defects are the basic quality measure.

Note that bugs are not important to the user 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 first be managed before other more important quality issues can be addressed.

Current software processes manage defects so poorly that little if any time is available for such important software quality issues as

- installability**
- safety**
- performance**
- recovery**
- usability, etc.**

The PSP Quality Focus - 3

Low defect content is an essential prerequisite to a quality software process.

Low defect content can best be achieved at the PSP level.

This is where the defects are injected and this is where the engineers should

- remove them**
- determine their causes**
- learn to prevent them**

Tests and Inspections - 1

Without inspections and a 50,000 LOC system

- **25+ defects/KLOC at test entry**
- **that is 1250 defects**
- **at the typical 10+ hours per defect, that is 12,500+ programmer hours**
- **that is 6 programmer years**

If properly planned, these tests could be done in 12 to 15 months.

If unplanned, testing could take two years or more.

Tests and Inspections - 2

With inspections and a 50,000 LOC system

- **inspections take about 10 programmer hours per 250 LOC, or about 2,000 hours**
- **this is 1 programmer year**
- **if done well, inspections can remove about 80% of the defects**

This means, 250 defects would be left for test

- **this would take about 2,500 hours**
- **or a saving of 8,000 hours**
- **or a saving of 4 programmer years**

Tests and Inspections - 3

With the PSP

- code quality will be sharply improved
- several thousand hours could probably be saved

Inspection should still be done

- the inspection focus should be on design

The principal advantages are

- improved product quality
- a more predictable schedule
- time to focus on the important quality issues

Some Fix Time Data

Some typical fix time ratios

- IBM rules of thumb: coding - 1.5; testing - 60; usage - 100
- Boehm: design - 1; development test - 15 to 40; acceptance test - 30 to 70; operation - 40 to 1000
- Remus: design - 1, code - 20, test - 82
- Ackerman: test 2 - 10 times inspection time
- Russell: inspection - 1, test - 2 to 4, use - 33
- PSP research: unit test takes 12 times longer than code review to find and fix defects

The Cost of Quality (COQ) - 1

COQ is a way to measure process quality.

COQ has the following components

- **failure costs**
- **appraisal costs**
- **prevention costs**

The Cost of Quality (COQ) - 2

Failure costs

- **repair, rework, and scrap**
- **in PSP, failure costs include all compile and test time**

Appraisal costs

- **costs of inspecting for defects**
- **in PSP, appraisal costs include all design and code review time**

The Cost of Quality (COQ) - 3

Prevention costs

- finding and resolving defect causes
- generally handled before projects start
- should typically be a process and not a project activity

In the PSP, examples of prevention costs are

- formal specification or design work
- prototyping
- process analysis and improvement

The Cost of Quality (COQ) - 4

A useful COQ measure is the ratio of appraisal to failure costs (A/FR). This is

- $100 * (\text{appraisal COQ}) / (\text{failure COQ})$

A/FR experience

- the A/FR measure is not widely used
- if measured, most software organizations would be near zero
- in the PSP, A/FR should exceed 2.0
- high A/FR is associated with low numbers of test defects and high product quality

The Quality Strategy - 1

Identify your PSP quality objectives, i.e.

- removing all defects before the first compile
- achieving high productivity
- producing accurate plans

Establish PSP process quality measures, i.e.

- overall process yield
- COQ appraisal vs. failure costs - A/FR
- LOC reviewed per hour
- Cost performance index - CPI

The Quality Strategy - 2

Examine the projects you have completed

- determine their ratings on these measures
- see what behaviors impacted these results

Based on these data, identify the most effective practices for your work.

Incorporate these practices in your process

- process scripts
- checklists
- forms

The Quality Strategy - 3

Identify measures that will reasonably predict process quality

- **establish these as control variables**
- **set specifications for these variables**

Track your performance against these specifications.

Track your process to determine

- **if and when to change the specifications**
- **actions to take to improve the process further**

Process Benchmarking

A method for tracking process improvement should

- **consider quality and productivity**
- **provide means for comparing process used by different people or organizations**
- **be insensitive to project specifics**

Industrial process benchmarking typically deals with the ability of the process to

- **produce products within specifications**
- **withstand drift and perturbation**

Software Benchmarking

At present, software benchmarking techniques are process dependant.

They are still useful, however as long as we

- establish objective measures
- track them over time
- use them for improving the process for which they are designed

Comparisons should not be made among individuals or organizations using process sensitive benchmarks.

Using Software Benchmarks

Establish a consistent set of measures for evaluating your process performance

- take these measures on every project
- compare individual projects to determine trends or problems

Establish and track short-term improvement goals against these measures.

Establish and track long-term improvement goals against these measures.

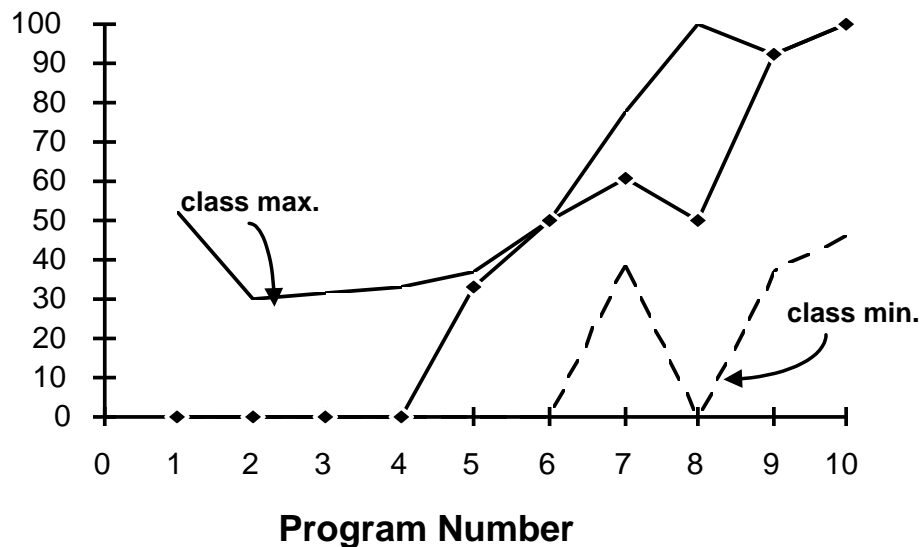
Benchmarking Data

The following data are from various of the students in the PSP course at Carnegie Mellon University in the Spring of 1994.

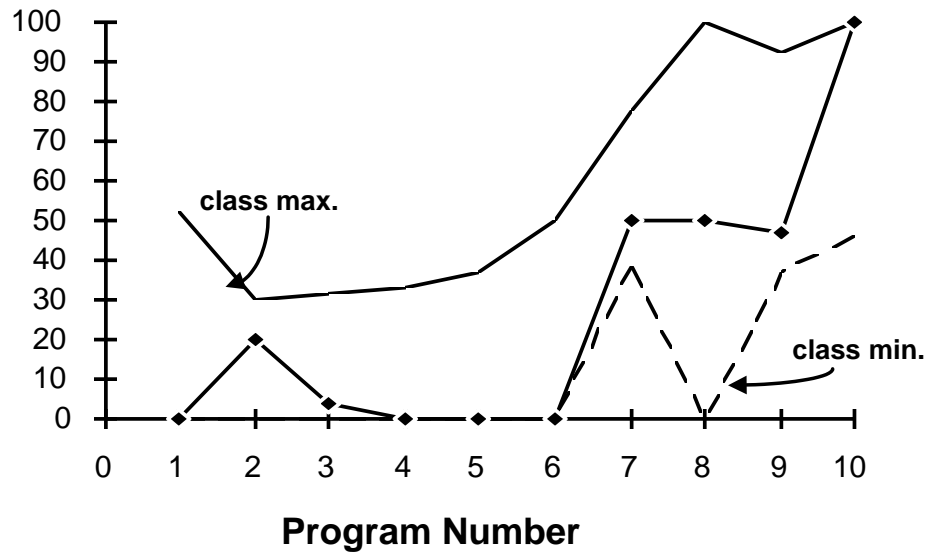
The data are

- yield by project
- yield vs A/FR
- A/FR vs test defects
- productivity by project
- yield vs productivity
- A/FR vs productivity

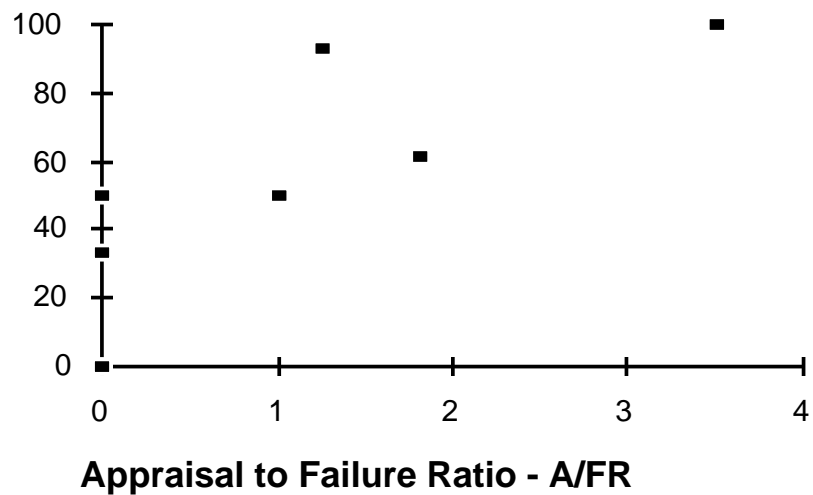
Yield - Student 3



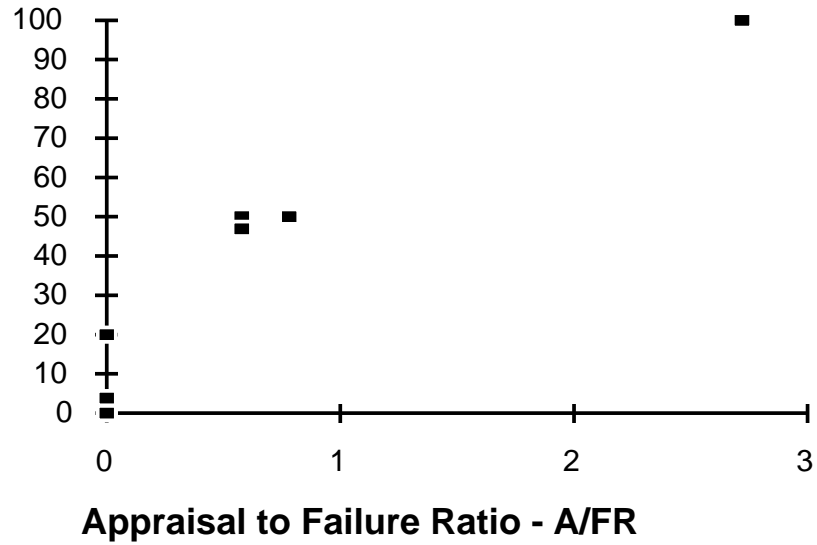
Yield - Student 20



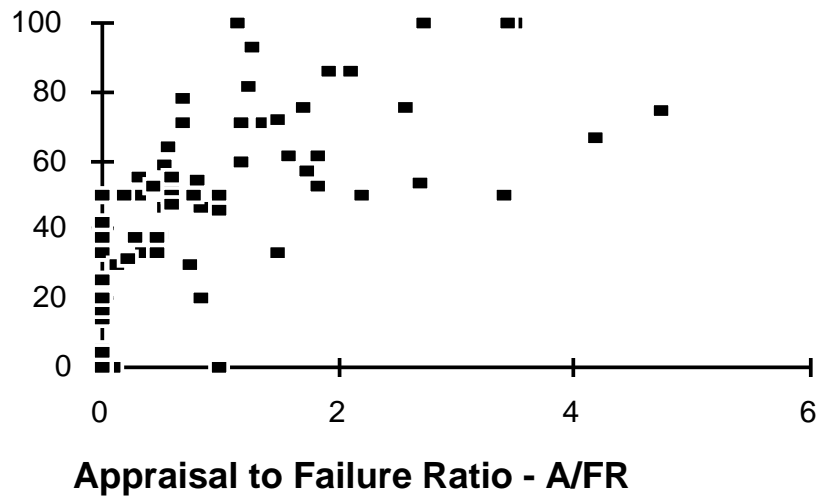
Yield vs. A/F Ratio - Student 3



Yield vs. A/FR - Student 20



Yield vs. A/FR - All Students, All Programs



Yield vs. A/FR Conclusions

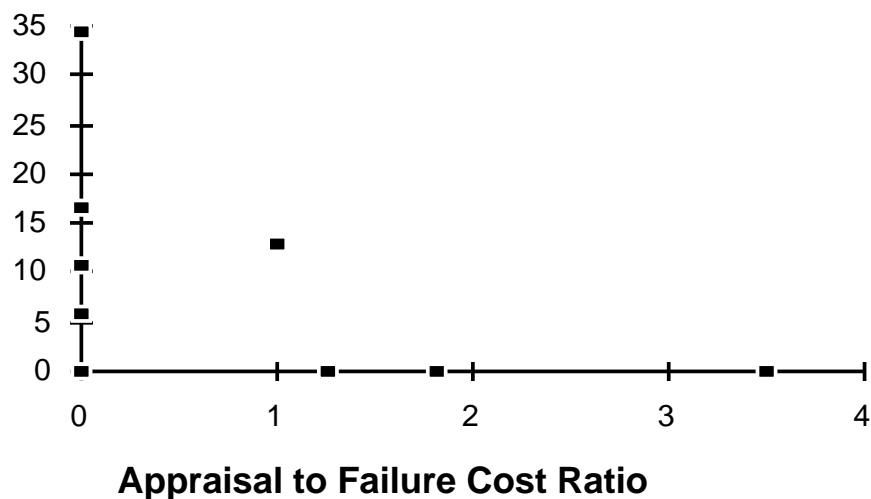
Yield and A/FR

- are closely related for these students
- there is considerable variation among students

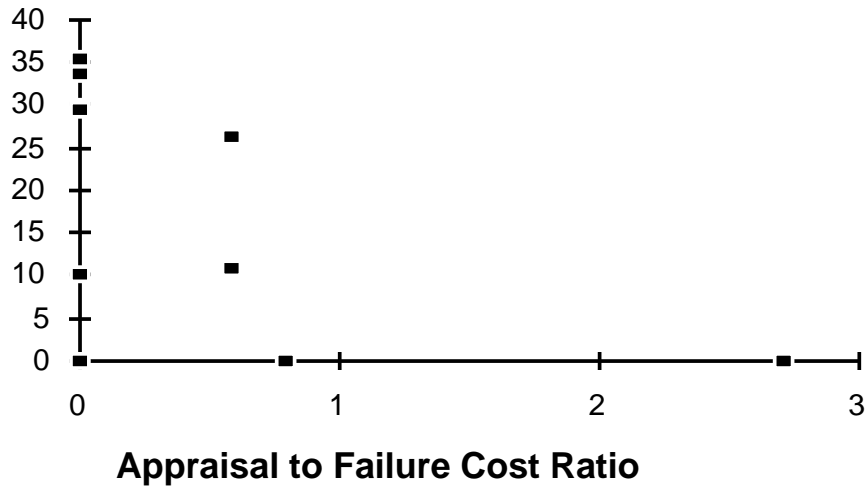
High A/FR ratios appear to lead to higher yields

- 70+% yields not achieved without A/FRs near 1.0 or above
- high A/FR does not guarantee high yield - the appraisal time must be spent effectively

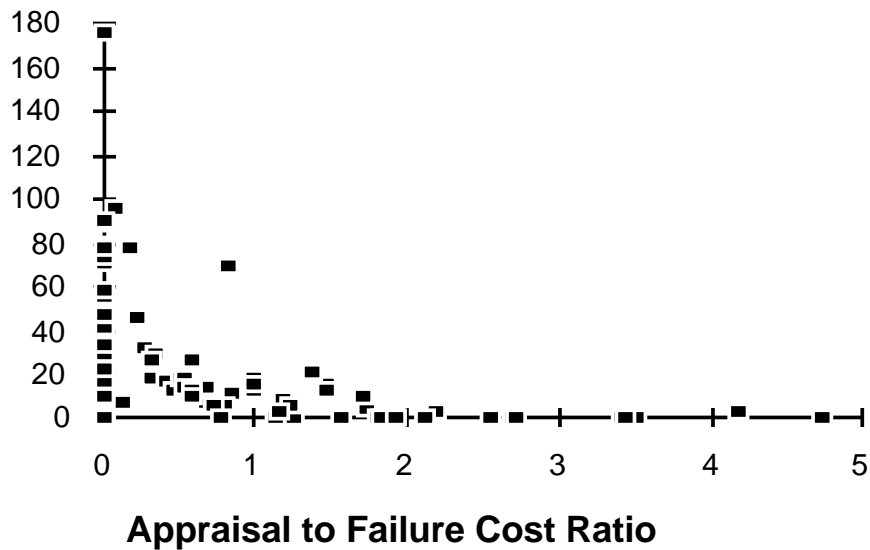
Test Defects vs. A/FR - Student 3



Test Defects vs. A/FR - Student 20



Test Defects vs. A/FR - Class



Test Defects vs. A/FR

Defects are reduced by high A/FR ratios for all students.

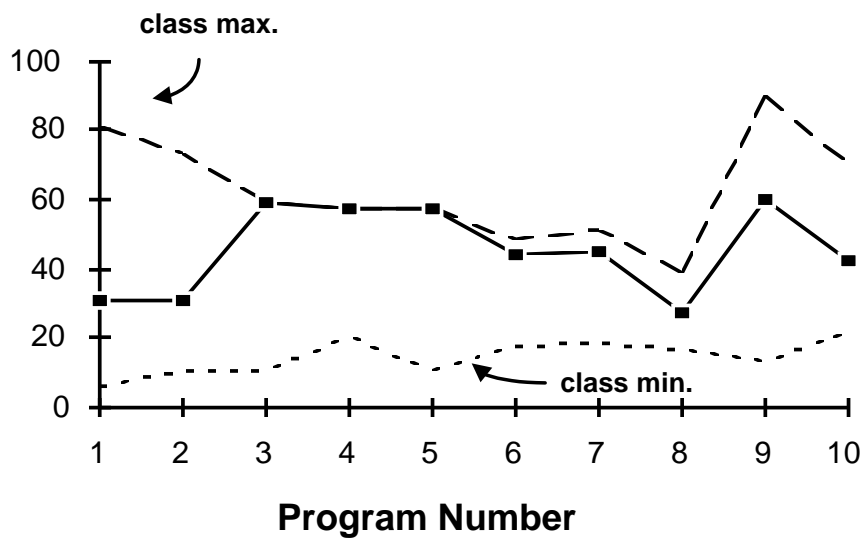
To get very low numbers of test defects, A/FR values of above 2.0 appear required.

With A/FRs between 1.0 and 2.0, low test defects are occasionally obtained.

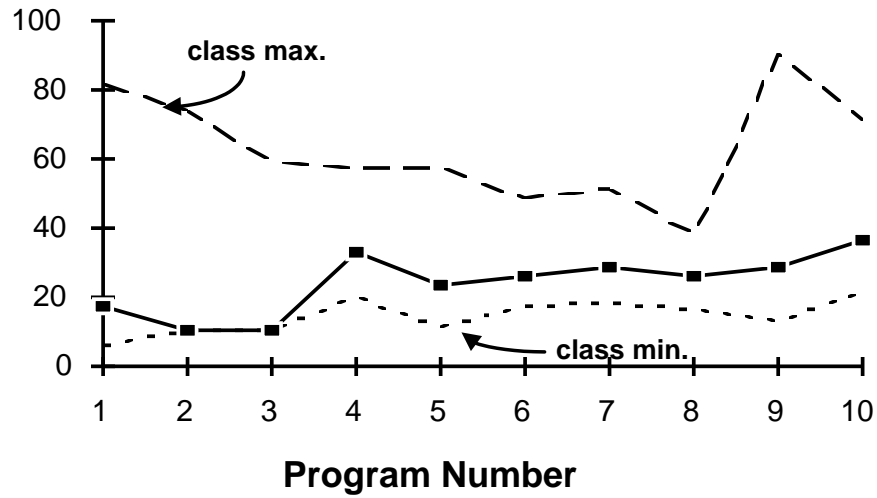
With an A/FR of below 1.0, low test defects are rare.

Productivity Trend - Student

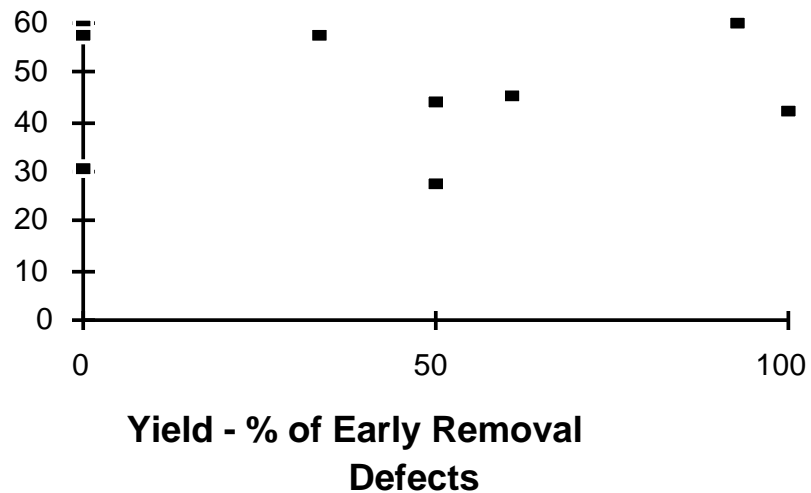
3



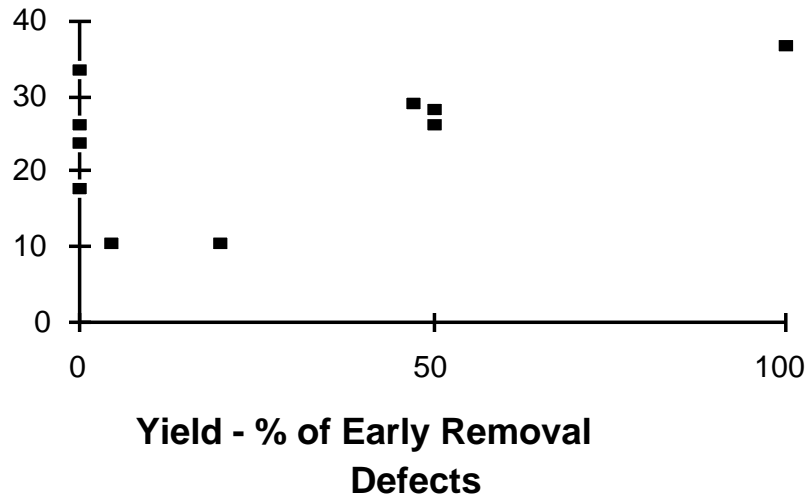
Productivity Trend - Student 20



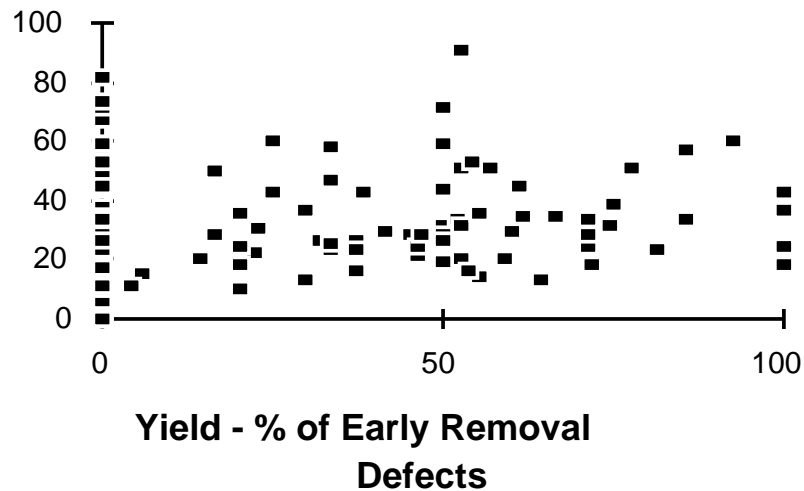
Yield vs. Productivity - Student 3



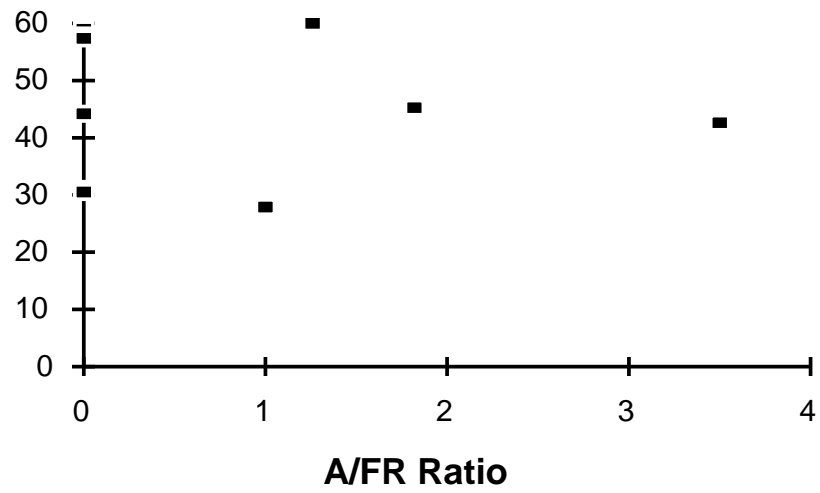
Yield vs. Productivity - Student 20



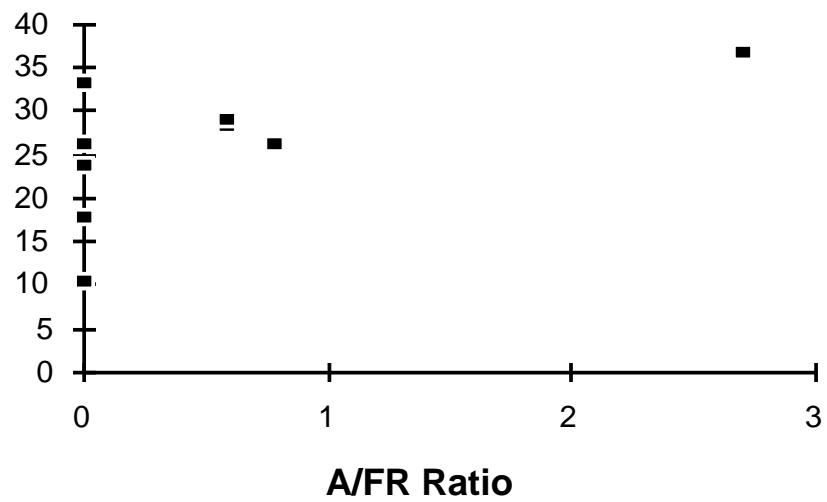
Yield vs. Productivity - All Students, All Programs



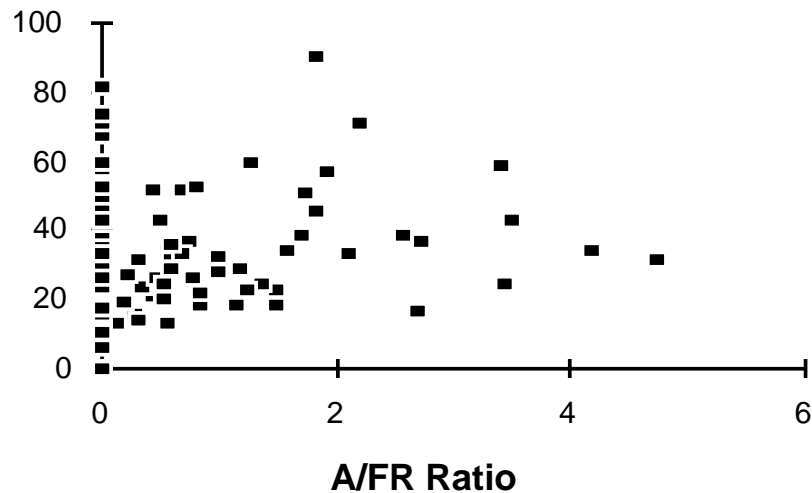
Productivity vs. A/FR - Student 3



Productivity vs. A/FR - Student 20



Productivity vs. A/F Ratio - All Students, All Programs



Yield and A/FR vs. Productivity

Productivity has considerable variation among individuals.

In some cases, high yield produces higher productivity but in others it does not.

High A/FR also sometimes results in high productivity and sometimes not.

Clearly, yield and A/FR are somewhat independent of productivity.

Benchmark Conclusions

It is desirable to have high values for

- yield
- A/FR
- productivity

Since yield and A/FR are closely related, a yield vs A/FR benchmarking chart would not be useful.

Yield vs. productivity or A/FR vs. productivity would likely be useful benchmarking charts.

Defect Removal Strategies - 1

Focus inspections and reviews on specialties

- HLD reviews - structural issues
- DLD reviews - logic correctness
- code reviews - implementation details

To save time, do not address

- system issues in DLD
- design issues in code reviews

Do the reviews thoroughly the first time and then trust them.

Defect Removal Strategies - 2

Do thorough unit tests

- **check all parameters at normal values, at limits, and outside limit values**
- **check all loops and recursions for normal and abnormal termination**
- **check all dependancies among procedures and objects**

Then do thorough system level testing

- **integration**
- **system**
- **regression**

Defect Prevention

Defect prevention is important because

- **it is always expensive to find defects**
- **if the defects can be prevented, these costs are avoided**
- **the defect prevention analysis costs are incurred once but save time on every project**

Defect prevention should follow an orderly strategy and a defined process.

For the PSP, defect prevention actions include improved design methods and prototyping.

Defect Prevention Strategy - 1

Set priorities for the defects types that are the most

- frequently found defects
- troublesome defects
- easily prevented defects

Defect Prevention Strategy - 2

The defect prevention process has the following steps:

1. follow an established schedule
2. select one or two defect types for initial action
3. track and evaluate the effectiveness of the defect prevention actions
4. make needed adjustments and continue

Defect Prevention Strategy - 3

In setting your initial priorities, consider the defect types most frequently found 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 in your process scripts, checklists, and forms.

Assignment #8

Read chapter 9 of the text

Using PSP2, write program 7A to calculate the correlation between two series of numbers and calculate the significance of that correlation.

Use program 5A to calculate the values of the t distribution.

Consult Appendix C for the PSP2 description and Appendix D for program 7A specifications.

The Correlation

The formula for calculating the correlation coefficient r is

$$r(x, y) = \frac{n \sum_{i=1}^n x_i y_i - \sum_{i=1}^n x_i \sum_{i=1}^n y_i}{\sqrt{\left[n \sum_{i=1}^n x_i^2 - \left(\sum_{i=1}^n x_i \right)^2 \right] \left[n \sum_{i=1}^n y_i^2 - \left(\sum_{i=1}^n y_i \right)^2 \right]}}$$

Where

- x and y are the two paired sets of data
- n is the number of their members

Correlation Significance

The formula for calculating the correlation significance is

$$t = \frac{|r(x, y)| \sqrt{n - 2}}{\sqrt{1 - r(x, y)^2}}$$

Where

- r is the correlation
- use program 5A to calculate the value of p for the value t from the t distribution
- the significance is indicated by $1 - p$
- > 0.2 is not significant, < 0.05 is significant

Messages to Remember from Lecture 8 - 1

- 1. Software quality starts with defects.**
- 2. If defects are not managed, more important quality issues cannot be adequately addressed.**

Messages to Remember from Lecture 8 - 2

- 3. The most effective way to manage defects is with the individual software engineer**
- 4. If you don't eliminate your own defects, they will be much more expensive and time consuming to remove later.**