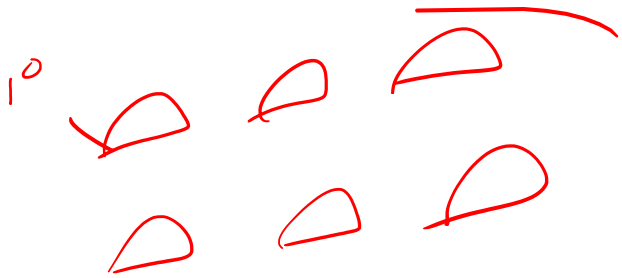
Test Management Part 1

Test Exit Criteria

Ira A. Fulton Schools of Engineering
Arizona State University

Objective



Objective

Select test exit criteria

When to stop testing?

1. Out of time / $
2. coverage
3. Reliability models
4. curves / graphs
5. analytics found all defects
6. customer is satisfied
7. test objectives

System Test Exit Criteria

The best criteria for stopping testing is when the test objectives have been met

Typical approaches for determining that the system is ready for release include:

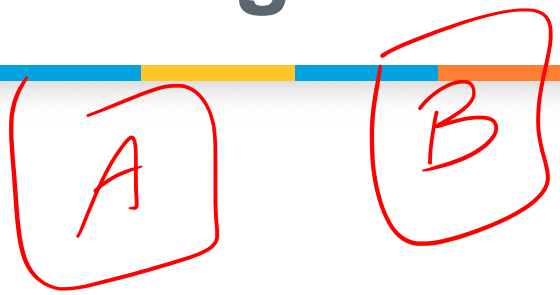
- Measuring defect density
- Defect pooling
- Defect seeding
- Trend analysis
- Reliability modeling

Measuring Defect Density

- **Defect density is defined as number of defects per thousand lines of code**
- **If available, historical defect density data might be used to predict expected number of defects to be found in system test**
- **Actual number might be compared against expected number to determine if system is ready for release**

- **Problems with this approach include:**
  - Lack of historical data
  - Defect injection rate for current system might be different than historical data
  - Defect containment for previous phases might be different than historical data
  - Defects found may not impact customer's perception of the system

Defect Pooling



- Appropriate to apply when operational usage scenarios are being executed
- Requires defect reports to be broken into 2 groups which can be tracked separately
- Assumes that each group of defects reflects independent testing of whole system based on operational usage
- Can be implemented many ways such as:
  - Splitting the system testers into 2 groups
  - Collecting defect data from 2 independent beta test groups

Defect Pooling Approach

Calculate the following assuming:

- $Defects_A$: is the number of defects found by Group A
- $Defects_B$: is the number of defects found by Group B
- $Defects_{A+B}$ is the number of defects found by both Group A and B

Unique Defects =
$(Defects_A + Defects_B) - Defects_{A+B}$

Estimated Total Defects =
$(Defects_A \times Defects_B) / Defects_{A+B}$

Estimated Remaining Defects =
Estimated Total Defects - Unique Defects

Defect Pooling Example

Assume:

- Group A found 30 defects
- Group B found 40 defects
- 20 defects were common to both Group A and Group B

$Defects_A = 30$ $Defects_B = 40$ $Defects_{A+B} = 20$

Unique Defects = (30 + 40) - 20 = 50

Estimated Total Defects = (30 x 40) / 20 = 60

Estimated Remaining Defects = 60 - 50 = 10

Defect Seeding

- Controversial approach involving "seeding" defects into the system
- Assumes that defects can be inserted into the system that are representative of defects that customers will encounter
- Assumes ability to detect remaining defects is equivalent to ability to detect seeded defects
  - Estimated Total Defects = (seeded defects planted / seeded defects found) x (normal defects found)

Defect Seeding Example

Assume:

- 20 defects are seeded
- 10 seeded defects are found by test
- 50 additional defects are found by test

Estimated Total Defects = (20 / 10) x 50 = 100

Estimated Remaining Defects = 100 - 50 = 50

Trend Analysis

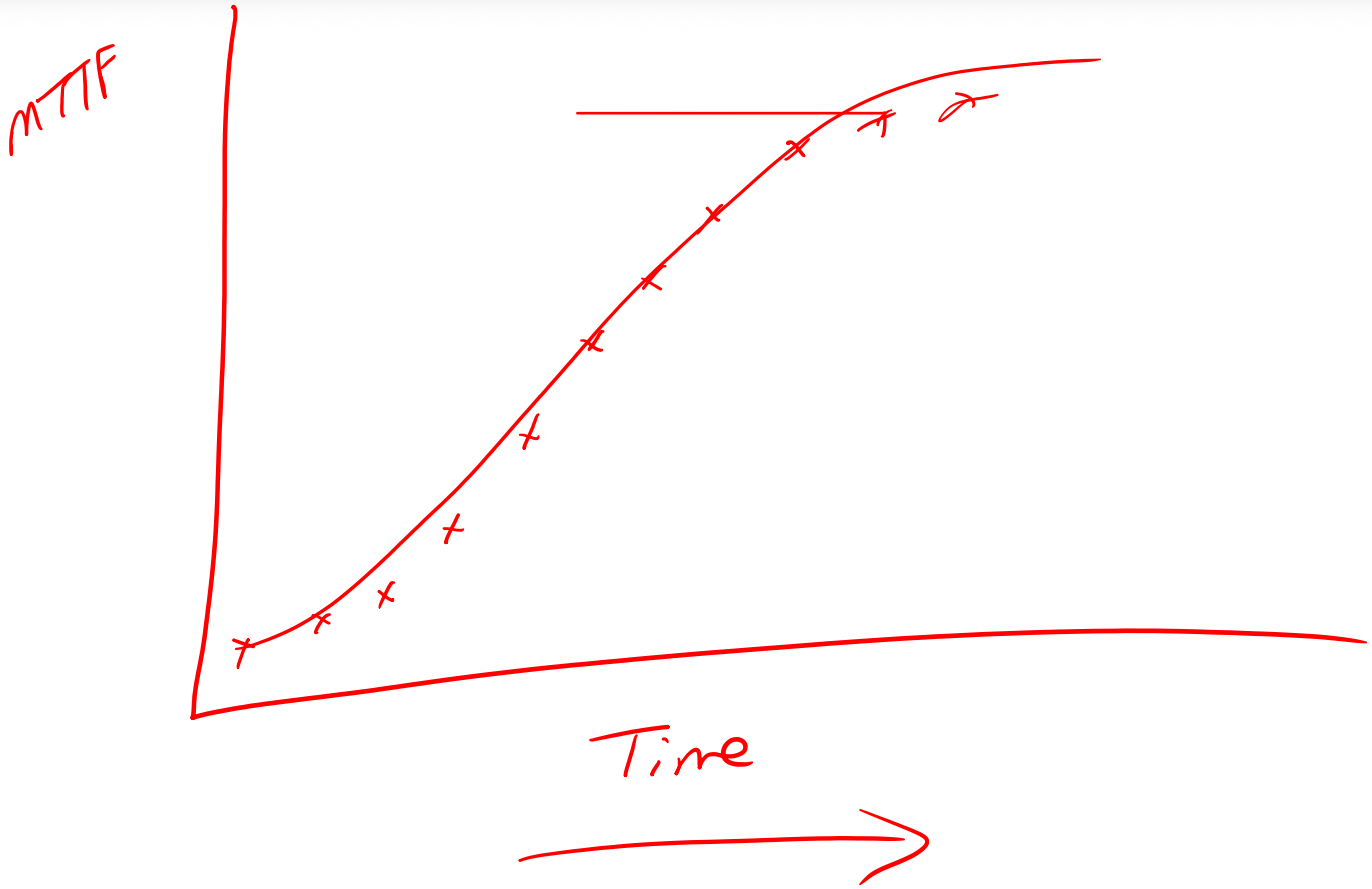
Software readiness may be assessed via analysis of trend data:

- Time to failure
- Cumulative number of failures
- Number of failures per unit of time (failure intensity)

One of three trends can be identified:

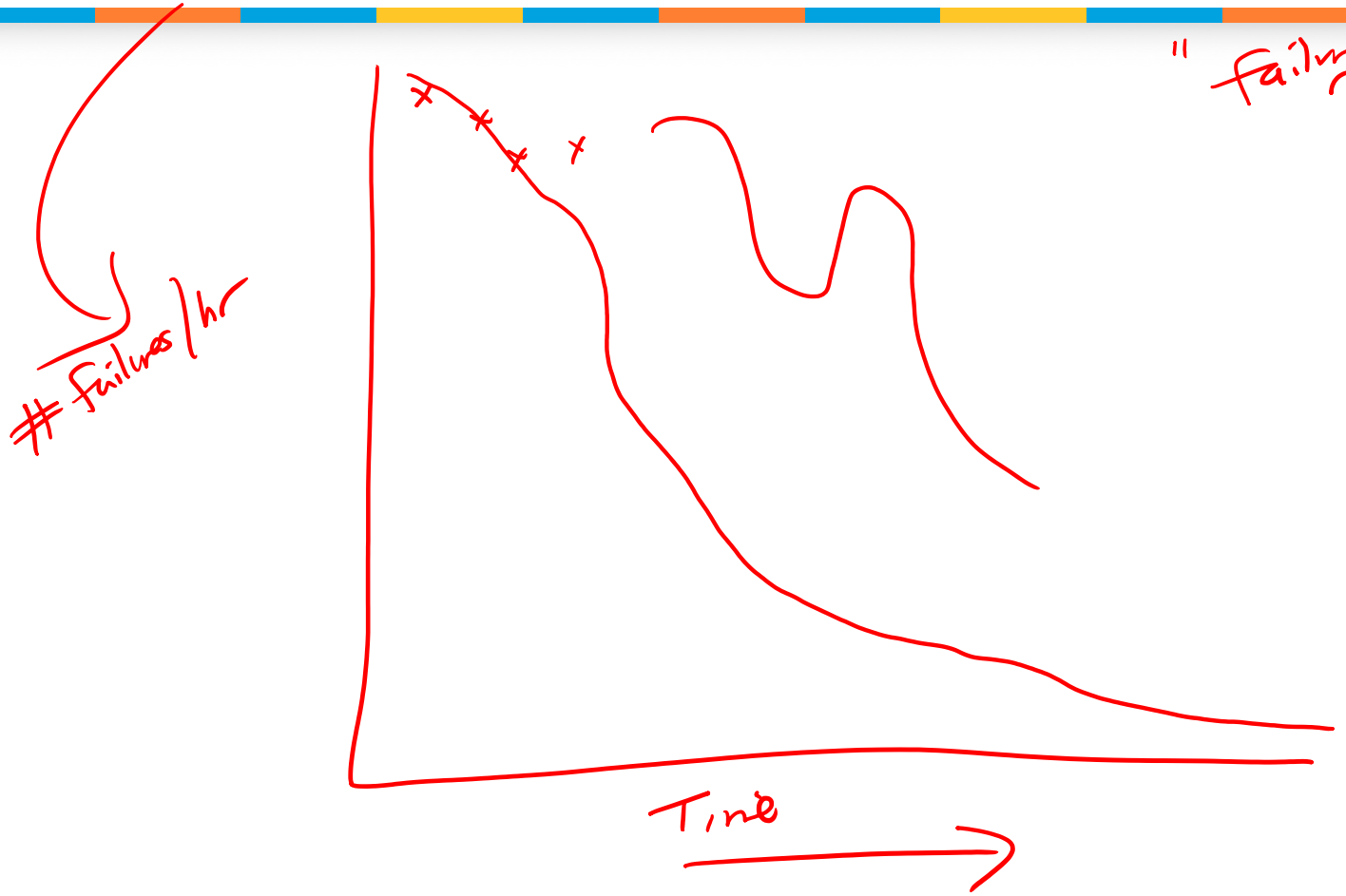
- Decreasing reliability
- Increasing reliability
- Stable reliability

Time to Failure



Cumulative Number of Failures

Number of Failures Per Unit of Time



Trend Analysis (Decreasing Reliability)

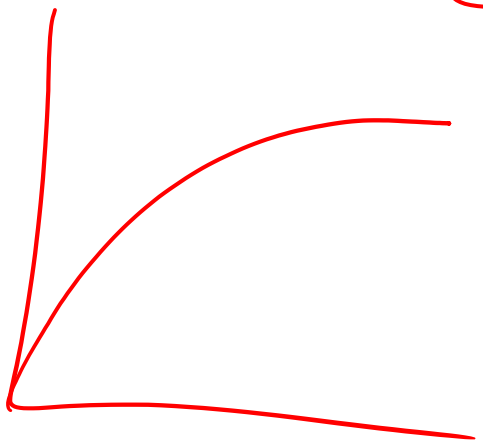
Expected at the start of a new activity

- New testing phase
- New type of testing
- Different user profile

Long duration signals significant software problems

Trend Analysis (Increasing Reliability)

- Normally good news



- Sudden increase may, however, be due to:
  - Changing test effort
  - Test burnout
  - Unrecorded failures

Summary