Course Structure

1. Software Quality Assurance
2. Testing Fundamentals
3. Code-based Techniques
4. Specification-based Techniques
5. Inspection Technique
6. Test Tools

7.2 Measuring Software Quality

8. TDD

Measuring Software Quality

- ROI
- Estimate Remaining defects
- Estimate Test Effort
- When to Stop testing?

Learning Objectives

- estimate **ROI**
- estimate **remaining defects** in program
- estimate **testing effort**
- determine **when to stop testing**

Calculating Return On Investment (ROI)

Look at the **cost of quality** – the cost of defect detection and repair prior to release, as opposed to post-release.

Then measure the efficiency of the test team by studying this differential.

Example:
- it costs $500 on average to find and remove a defect prior to release,
- $15000 to find and remove a defect post-release, then
- every defect that the test team finds prior to release will save the organization $14500.

Where to Improve?

Cost of testing is generally at its highest during the test execution process.

Anything we can do to **shorten** the test execution process will reduce the overall testing process and the length of the project.

Look at the defect detection % for high priority defects versus all defects. If the testing is properly focused, we should *find a higher percentage for high priority defects* than for all defects.

Example: we find a high rate of defects in the field, identify the holes in our test coverage.

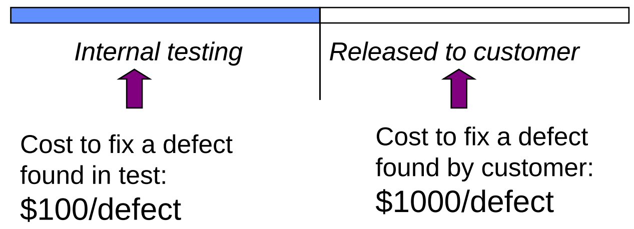
If we find excessive costs associated with our testing efforts, determine how best our efficiency can be enhanced.

- A **quick fix** is always to push defect discovery process earlier in the lifecycle.
- Look at unit testing, code reviews, design reviews to see what can be improved.

ROI from Improvement (1)

Suppose

- Current testing cost: $20000
- DRE: 70%



ROI from Improvement (2)

- If we can invest $30000 to improve the testing process and increase DRE to 80%.

Before improvement:		After improvement:
DRE = 70%	⇒	DRE = 80%

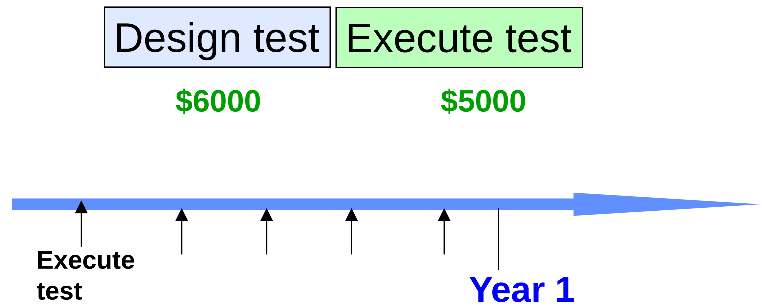
Is this a good investment?

What is the ROI? Assume a typical project has 1000 defects.

ROI from Automation (1)

Suppose

- Cost to design test cases: $6000
- Cost to execute a full cycle of test: $5000



Number of test cycles/year = 5

ROI from Automation (2)

ROI
Est R Defects
Est T Effort
Stop Testing

Suppose we can invest $5000 to buy a test automation tool and the cost to implement automation is $11000

Then, the cost to execute a full cycle of test after automation is reduced to $1000.

Is this a good investment?

Test execution

$5000 --> $1000

Before automation *After*

What is the ROI?

Question

ROI
Est R Defects
Est T Effort
Stop Testing

When we report testing progress, we may be asked

- How many defects left in the system?

How do we answer this question?

Estimating Remaining Defects

ROI
Est R Defects
Est T Effort
Stop Testing

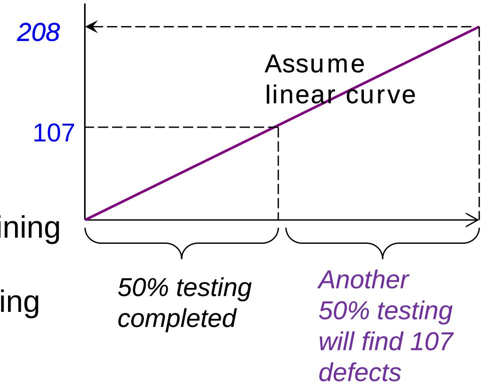
Method 1

Use the total number of defect found, and the % test completed, estimate the remaining defects.

<u>Example:</u>

107 defects found
50% testing done
Thus, 50% testing remaining

⇒ Estimate 107 remaining defects



Estimating Remaining Defects

ROI
Est R Defects
Est T Effort
Stop Testing

<u>Improvement 1</u>

We should not care too much about minor defects.

Focus on <u>high severity defects</u>

First, identify each testing task, work out the % test and severe defects found,

Predict the remaining defect by test task.

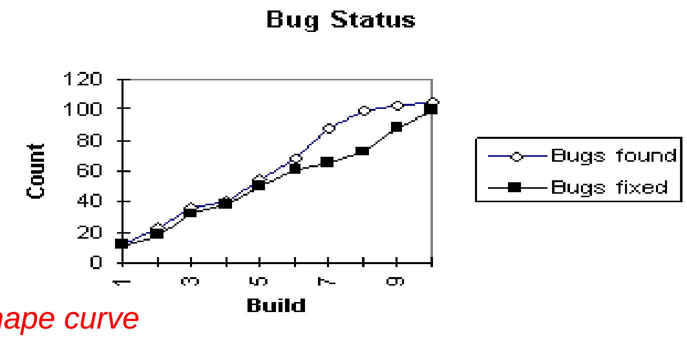
Task	%Test	Severe defect found	Predicted defect
Printing	40	12	30
Scenario	50	14	28
Integration	30	3	10
Interface	60	78	130
Total		107	198

=> Estimated remaining <u>severe defect</u> = 198-107 = 91

Estimating Remaining Defects

Improvement 2

- The cumulative defect curve is like an S curve.
- Better use a curve fitting model (outside the scope of this course) rather than a linear curve.



S shape curve

Estimating Remaining Defects

Improvement 3

Use past projects to predict the current number of defects.
Find out the defect rate for previous projects (per LOC, per function point).
Need to adjust the numbers to reflect the differences of this project from the previous projects.

From historical data:

Project	Size (KLOC)	Defect	Defect density
A	40	85	2.1
B	10	31	3.1
C	90	201	2.3
Total	**140**	**317**	**2.3**

Our new project has a size of 160 KLOC
We can estimate the number of defects to be 2.3*160 = 368

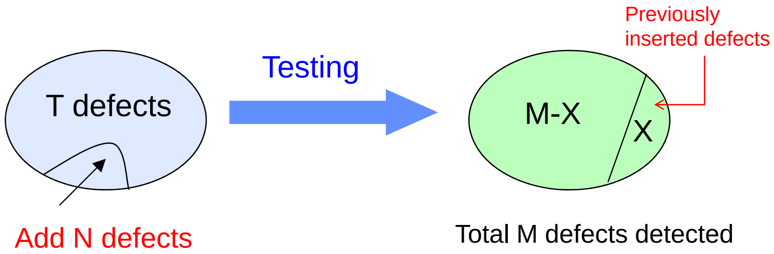
Method 2: Insert defects into the program

➔ Suppose we deliberately introduce/insert N defects into the program, which has a total of T defects.

➔ Our testing reveals M defects, of which X were the *inserted defects*.

➔ We discovered (M-X)/T of the **unknown defects**.

➔ We discovered X/N of the ***inserted defects***.



Method 2

If these proportions are equal

$$\frac{(M-X)}{T} = \frac{X}{N}$$

Then, total defect, $T = \frac{(M-X)*N}{X}$

➔ Remaining defects

R=T-real defects

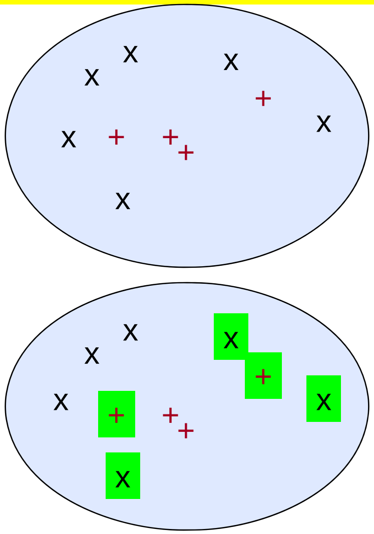
=T-(M-X) = T-M+X

Assumptions:

➔ Types and proportions of real defects match the inserted defects

➔ The inserted defects do not interact with the real ones.

Example



x are the original defects in the system

+ are the <u>inserted defects</u>

We inserted 4 defects.

Suppose we detected 5 defects: 2 are the inserted defects.

N=4

M=5

X=2

T = (M-X)*N/X = (5-2)*4/2 = 6

R = 6 – 5 + 2 = 3

Method 3: Capture-Recapture Estimation

This method is used in estimating animal populations.

- *Suppose we catch 30 fish from a pond, tag them, and release them.*
- *A few days later, we catch 25 fish from the pond. 5 of them have tags.*
- *How many fish are in the pond?*

$$\frac{\text{30 tagged fish (in pond)}}{\text{T total fish (in pond)}} \cong \frac{\text{5 tagged fish (in sample)}}{\text{25 total fish (in sample)}}$$

$$T \cong \frac{\text{30 tagged fish (in pond)} \times \text{25 total fish (in sample)}}{\text{5 tagged fish (in sample)}} \cong 150$$

Estimating Remaining Defects

- We want to determine the total number of defects in a product.
  - ✓ The **product** is the "Pond".
  - ✓ **Defects** are the "fish".

Capture recapture method works best when:

- The number of defects found is not too small
- The defect detection is good at finding all of the defect types that are being predicted.

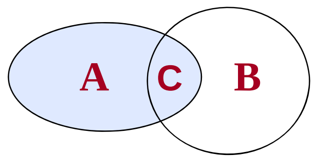
Example: 2 engineers conduct independent testing (or inspection) of a product and record all the defects they find.

- Those found by the first engineer are "tagged" defects.
- The second engineers typically finds some of the same "tagged" defects, plus others.
- Just as with the fish, we can estimate the remaining defects (the ones not found by either engineer).

The Capture-Recapture Method

1. Count the number of defects recorded by the first engineer (A)
2. Count the number of defects recorded by the second engineer (B)
3. Count the number of defects found by both engineers (C)

$$\frac{A}{T} = \frac{C}{B}$$



4. total defects, T = (A*B) /C
5. defects found, F = (A+B) –C
6. remaining defects, R = T - F = (A*B /C) – (A+B –C)

With Multiple Inspectors/Testers

If we have more than 2 engineers doing the testing or inspection:

- Count the defects recorded by the engineer who found the most unique defects (A).
- Combine the defect data from the rest of the inspectors (B). When multiple engineers find the same defect, count it just once.
- The rest of the calculations remain as for the two engineer case.

Key assumptions:

- Inspectors work independently
- Defects have an equal probability of being detected
- Inspectors all have equal ability to find the defects.

Example

3 product engineers identified a total of 7 defects in a product;

Defect Number	Engineer Larry	Engineer Curly	Engineer Moe	"Column A"	"Column B"	"Column C"
1	✓			✓		
2	✓			✓		
3			✓		✓	
4	✓	✓		✓	✓	✓
5	✓			✓		
6	✓		✓	✓	✓	✓
7		✓			✓	
Totals	5	2	2	5	4	2

In Column A, the defects by the engineer who found the most unique defects are identified. Larry found the most unique defects.

In Column B, each defect that was found by all of the other engineers is identified. The defects found by Curly and Moe are identified.

In Column C, each defect that was found in both Column A and Column B are identified (e.g., the intersection of these two columns).

Example

The estimated number of probable defects in the product is **(A * B) / C** = **(5 * 4) / 2 =10**

The number of defects found by the participants is:
A + B - C = **5 + 4 - 2 = 7**

The number of defects remaining is the difference between the probable number of defects (10) and the found defects (7).
((A * B) / C)-(A + B - C)=**((5 *4)/2)-(5+4- 2)=3**

We estimated that 70% of the defects in the product were identified, and that 30% of those defects remain.

Estimating Testing Effort

How much effort required to do testing?

Method 1 (quick method):
Ask the expert

Method 2: Use COCOMO or other estimation methods to estimate the development effort, then use a % for testing (25-45%)

E.g. Total development effort is estimated to be 100 person-days,
Testing effort is 25% or 25 person-days

Estimating Testing Effort

ROI
Est R Defects
Est T Effort
Stop Testing

Method 3. Ratio method

- Estimate the ratio between developer to tester
- Can vary from 6:1 to 1:1
- Collect data from previous projects.
- Adjust the ratio a bit based on the specific characteristics of the current project.

Example:

Previous projects have 4 developers to 1 tester.
Our current project involves integration with many external systems; thus, likely to require more testing.
=> Use the ratio 3.5: 1 for developer: tester

Estimating Testing Effort

ROI
Est R Defects
Est T Effort
Stop Testing

Method 4. FP method

Collect data from previous projects:

Project	*FP*	*Test effort (day)*
P01	*100*	*40*
P02	*150*	*50*
P03	*400*	*88*
P04	*200*	*55*
Total	*850*	*233*

Average Test effort per FP = 0.27
New project estimate: 350 FP =>
Test effort = 350*0.27=94.5 days

Estimating Testing Effort

ROI
Est R Defects
Est T Effort
Stop Testing

Method 5. Work breakdown structure (WBS) Method

- Involve the business users, test team and developers.
- Work out the test strategy
- Work out the test plan (with all the key **work items**: construct the test environment, develop test cases, etc)
- Estimate the effort for each work item and dependences
- Sum and add extra for contingency.

We need historical data to do accurate estimate.
Collect data from completed projects to give a rough benchmark.

WBS Task	Description	Effort estimate
1	design and build test program (write tests)	40
2	set up and configure the test system	16
3	install specialized test tools, customize the tools	20
4	program and execute each test	40
5	verify defect fixes for failed tests (rerun tests)	10
6	prepare a test report and summary documents	4
	Total (hours)	130

Example: Test Effort Estimation

	Test
Test cases/screen (low risk)	2-4
Test cases/screen (medium risk)	10-15
Test cases/screen (high risk)	20-25
Test cases/report	1-2
Test cases/message	1-2

Test case development: 15 test/day
Test case execution: 25 test/day
Retest: 25 test/day

Example:

Screens		Test
1	Low risk	2
2	High risk	25
3	Medium risk	10
Messages	15	15
Reports	20	20
Total test case:		72

Test planning: 72/15 = 5 days
Test execution: 72/25 = 3 days
Debug and fix: 1 day
Total = *9 days*

Some Rules for Effort Estimation

ROI
Est R Defects
Est T Effort
Stop Testing

Rule 1: Estimation shall be always based on the software requirements
All estimation should be based on what would be tested, i.e., software requirements, which shall be understood by testing team.

Rule 2: Estimation shall be based on previous projects
All estimation should be based on previous projects.
If a new project has similar requirements from a previous one, the estimation is based on that project.

Some Rules for Effort Estimation

ROI
Est R Defects
Est T Effort
Stop Testing

Rule 3: Estimation shall be based on expert judgment
Classifies the requirements in 3 categories:

- **Critical**: development team <u>has little knowledge</u> in how to implement it;
- **High**: development team <u>has good knowledge</u> in how to implement it but it is not an easy task;
- **Normal**: development team has good knowledge in how to implement.

The experts in each requirement should say how long it would take for testing them. The categories help the experts in estimating the effort for testing the requirements.

Some Rules for Effort Estimation

Rule 4: Estimation shall be recorded
All decisions should be recorded. It is important because if requirements change for any reason, the records would help the testing team to estimate again.

Rule 5: Estimation shall be supported by tools
Use tools (e.g. spreadsheet) that help to reach the estimation quickly. The tool calculates automatically the costs and duration for each testing phase.

When to Stop Testing?

ROI
Est R Defects
Est T Effort
Stop Testing

Some useful **completion criteria**:

1 **The successful use of specific test case design methodologies**
Finish running all test cases in the test plan and they all pass.

2 **A % of coverage for each coverage category**
- ➔ All branches coverage (control flow)
- ➔ All functions coverage

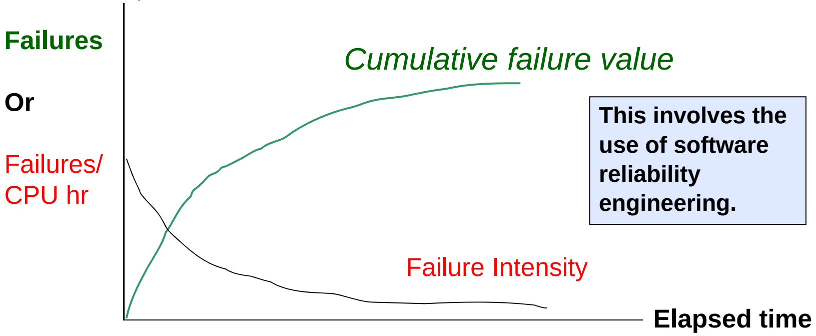
Finish testing when the coverage satisfies the coverage criteria (e.g., 100% branch executed)

When to Stop Testing?

ROI
Est R Defects
Est T Effort
Stop Testing

3 When the failure intensity falls below a specified threshold (e.g. less than 0.01 failure/CPU hr)

Cumulative Failure value will increase in value as time elapses.
Failure Intensity (measured as failure/CPU hour) should decrease as time elapses.



We live in a non-linear world!

When to Stop Testing?

ROI
Est R Defects
Est T Effort
Stop Testing

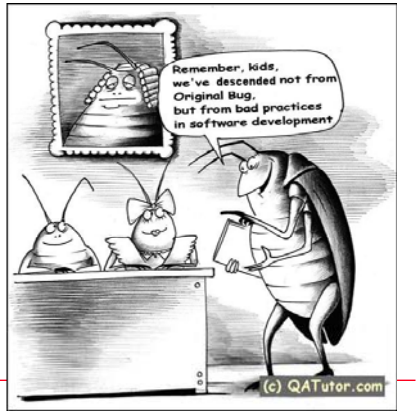
4 The detection of a specific number of errors based on historical data

Suppose, from historical data:

Project	Defect density (defect/KLOC)
P01	3.7
P02	4.1
P03	3.8
P04	4.2
Average	4.0

Suppose, the current project has Size = 40KLOC
⇒ Expected defects = 40x4 = 160
⇒ Stop testing when we detect close to 160 defects.

Supplementary Notes



Another Example of Capture/Recapture

We have a total of 7 inspectors.

Product Size Reviewed (Requirements Statements)		**357**	
Major Defects Found		**42**	
Defect Projection (Major Defects Only)			
	Inspector (s) Finding Most Unique Defects	**1 & 2**	
	Unique Defects Found by Inspectors 1 & 2	**11**	
	Total Defects Found by Inspectors 1 & 2	**31**	A
	Total Defects Found by the Other Inspectors (3-7)	**31**	B
	Defects Found by Both Groups	**20**	C
	Total Defects Projected	**48**	
	Total Defects Found	**42**	
	Remaining Defects Projected	**6**	

Total defect = A*B/C = 31*31/20 = 48
Unique defect found = 31+31-20 = 42

Another Example of C/R

Defect Projection (Counting only Major Defects)			
	Total Defects Projected	48	
	Total Defects Found	42	
	Remaining Defects Projected	6	
Defect Projection (Counting only Major Defects)			
	Defects found by Inspector 3	23	
	Defects found by Inspector 4	12	
	Defects found by Inspector 5	1	
	Defects found by Inspector 6	13	
	Defects found by Inspector 1 & 2	31	
	Defects found by Inspector 7	3	
	Yield by Inspector 3	48	Percent
	Yield by Inspector 4	25	Percent
	Yield by Inspector 5	2	Percent
	Yield by Inspector 6	27	Percent
	Yield by Inspector 1 & 2	65	Percent
	Yield by Inspector 7	6	Percent
	Total Inspection Yield	87	Percent

Example Metrics (from Questcon)

Base metrics

- Gathered by tester throughout the testing effort.
- Used to provide project status reports to the Test Lead and PM

Every project tracks these metrics:

- # of test cases
- # of tc executed
- # of tc passed
- # of tc failed
- # of tc under investigation
- # of tc blocked
- # of tc re-executed
- # of first run failures
- Total executions
- Total passes
- Total failures
- Test case execution time
- Test execution time

Example Metrics (from Questcon)

Derived metrics

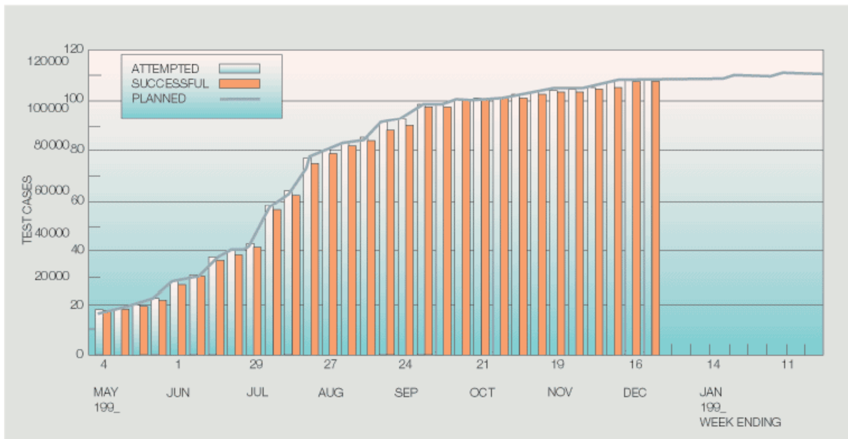
Tracked at many different levels (by module, tester, or project):

- % complete
- % test coverage
- % test cases passed
- % test cases blocked
- 1st run failures
- % failures
- % defects fixed
- % rework
- % test effectiveness
- % test efficiency
- Defect discovery rate
- Defect removal cost

Also, plot test case passes against time, and Open defects against time

Example Metrics (from IBM)

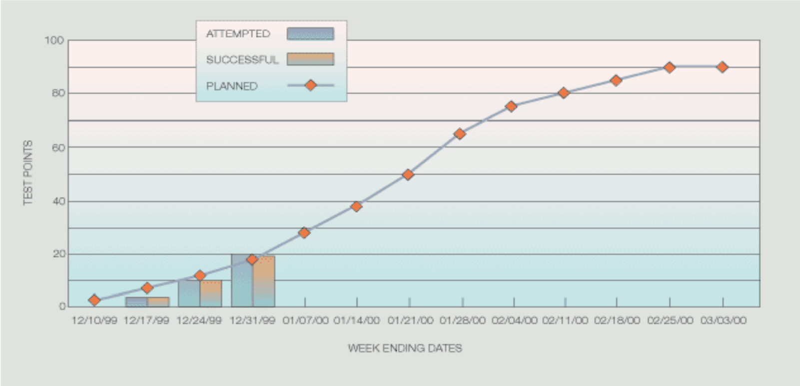
Figure 2 Testing progress S curve example



Track actual testing progress against plan and support proactive action when early indications that testing activity is falling behind.

Example Metrics (from IBM)

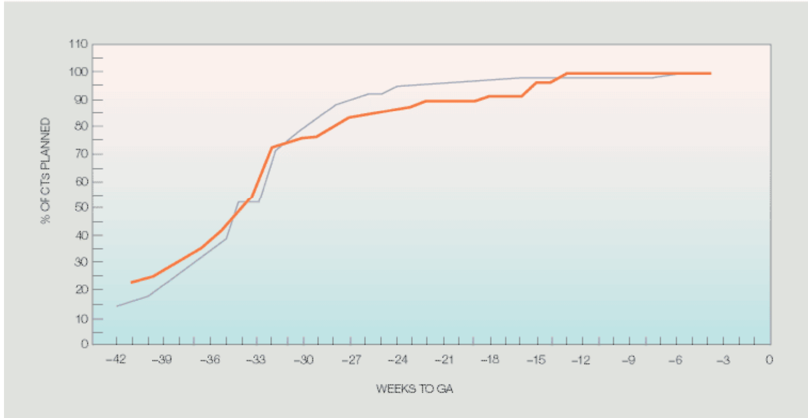
Figure 3 Testing progress S curve—test points tracking



Assign points to test cases. 10 for the most important test cases, and 1 for the least.
15-20% between attempted and planned test cases should trigger additional actions to catch up.

Example Metrics (from IBM)

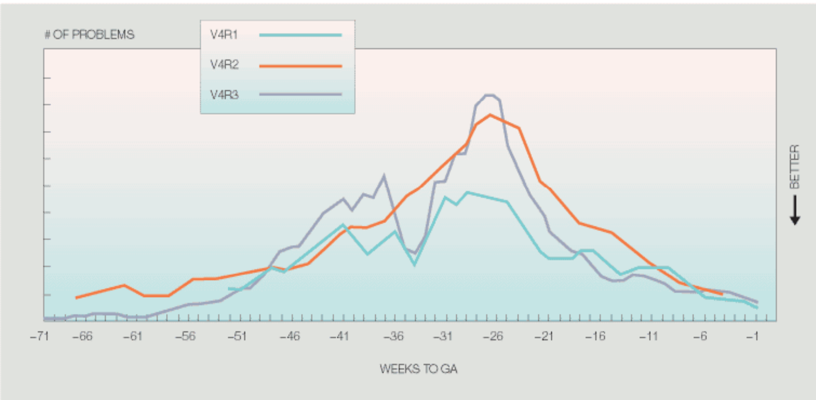
Figure 4 Testing plan curve—release-to-release comparison



Use weeks before product general availability (GA) as the time unit (x-axis).
Show 2 releases, one before and the other current release.

Example Metrics (from IBM)

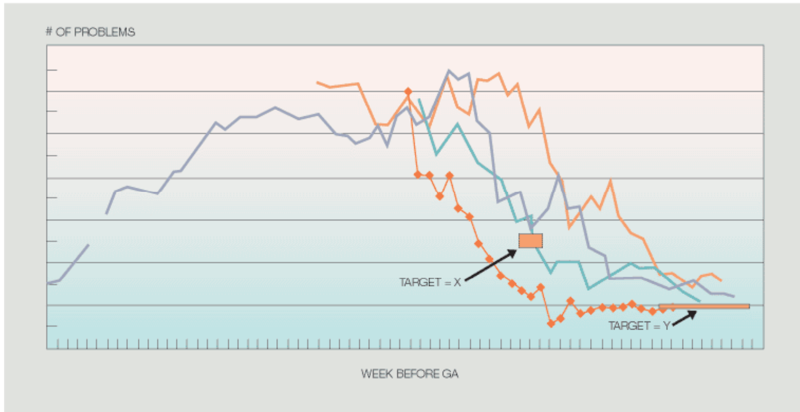
Figure 5 Testing defect arrivals metric



Track the test defect arrival pattern over time.

Example Metrics (from IBM)

Figure 6 Testing defect backlog tracking



Backlog is the accumulated difference between problem arrivals and problem closed.
Check release-to-release comparisons and actuals versus targets.