Module 4 – Topic 4.3

Software Metrics

Topic Outline

- Monitoring issues
- Goal, Question, Metric framework
- Desirable properties of metrics
- Defect analysis

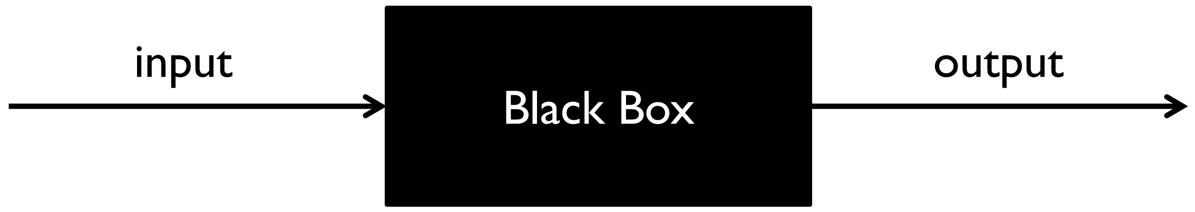
Recall Summary of Review Techniques

Method	Typical Goals	Typical Attributes
Walkthroughs	Minimal overhead Developer training Quick turnaround	Informal process Little/no preparation No measurement
Technical reviews	Well-defined requirement Robust design Improvement to code Training	Formal process Author presentation Discussion oriented Review data collected
Inspections	Detect and remove all defects efficiently and effectively	Formal process Checklists Measurements

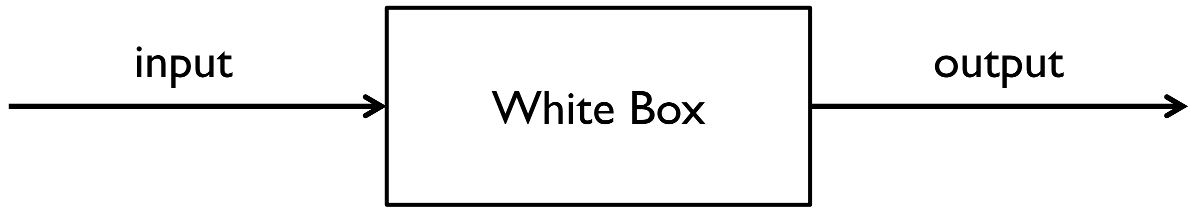
Source: Johnson P.M. (1996) Introduction to Formal Technical Reviews

Recall Software Testing Techniques

- Black-box testing: test **behaviour** of the software



- White-box testing: test **structure** of the software



Recall Monitoring, Metrics & Feedback (Topic 1.2)

- Monitoring is essential for ensuring you are delivering the right product, done right and managed right
  - Module 3 Right Product – Topic 3.5 Validating Requirements
  - Module 4 Done Right – Topic 4.3 Software Metrics
  - Module 5 Managed Right – Topic 5.2 Monitoring Project Plan
- Metrics are an effective way of monitoring that gets quantifiable results
- Monitoring and metrics help to provide feedback that suggests improvements or affirms that you are on the right track

Monitoring Issues

Finding the Time to Compile Metrics

- Agile focuses on fast and efficient development
  - Any time the developers spend doing non-coding tasks, such as compiling metrics, takes away from the entire time that they have to work on the coding
- In Agile the success of the project depends on how much can get done right now
  - It can be hard to see the value of calculating metrics

Lack of Knowledge and Industry Standard

- Those assigned the task of managing software projects do not have substantial training in measuring quality
  - It's common that project metrics are either misused or not used at all
- Some people say to track this, others say to track that
  - Studies in this area tend to be inconclusive, or not generalizable
  - Ultimately each organization has to determine what works for their situation

Lines of Code – Measuring Productivity?

- Tracks the number of lines of code written for a piece of software

- It is unreliable as a gauge of productivity
  - Depends on the language
  - Different functions have different complexities

- It will change developers behaviour
  - Whenever someone is given a score by which to measure themselves, their natural inclination will be to maximize that score

Goal, Question Metric (GQM)

Goal Question Metric

- For an organization to measure in a purposeful way
  - Goal - specify the goals for itself and its projects
  - Question - trace those goals to the data that are intended to define those goals operationally
  - Metric - provide a framework for interpreting the data with respect to the stated goal

Set a Goal

- Goals are applied to products, processes, and resources
  - Product – code, documentation
  - Process – requirement elicitation, software testing processes
  - Resource – office space, computers, salaries

- Breaking your project goals down into these three categories helps you to think about how to measure them

Define Questions to Ask

- This depends on the individual goal

- Questions should be specific enough so that the answer to the question addresses the goal

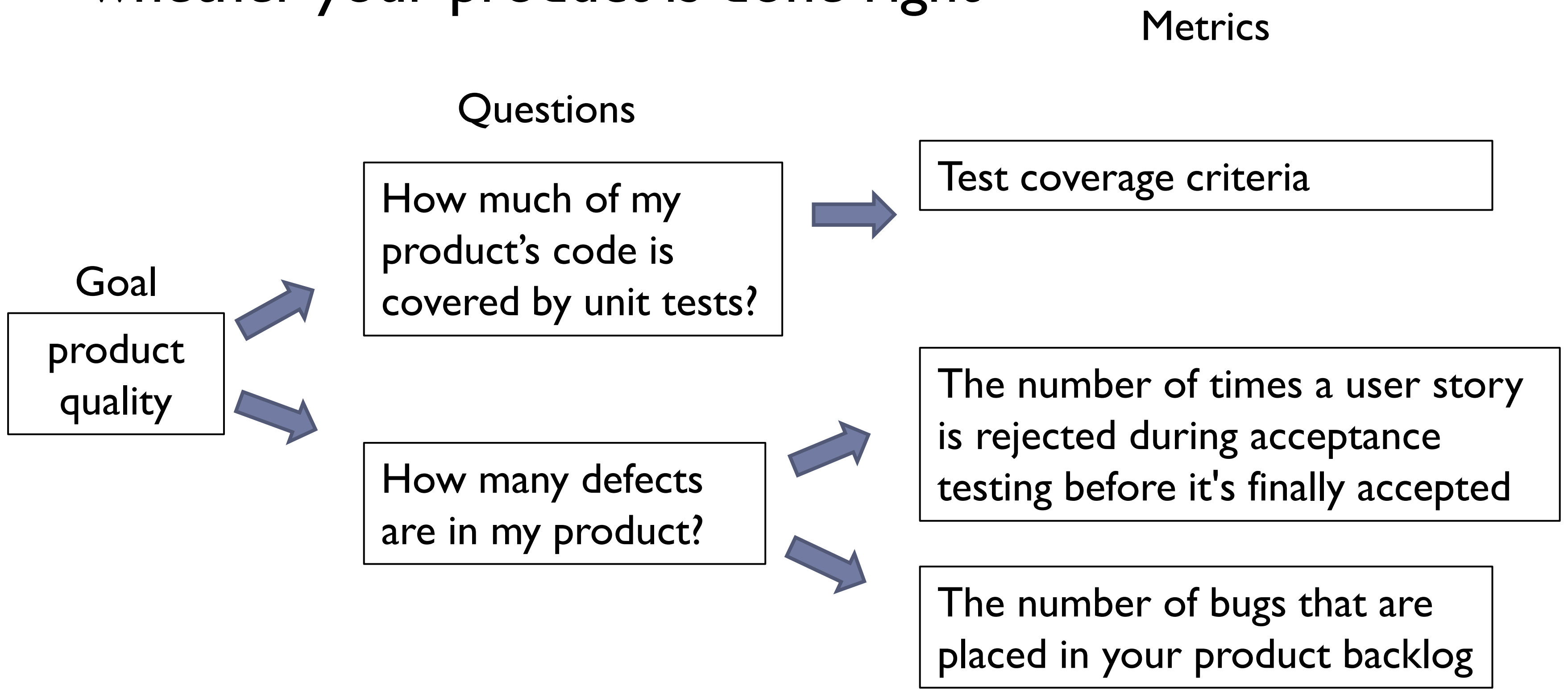
Compile Metrics

- A quantitative way in which data can be collected to address the questions you're asking

- Metrics can either be subjective or objective
  - objective – the number of product downloads, sales numbers, processing time, user stories completed
  - subjective – product ratings by user, developer morale, code quality

GQM – An Example

- You're developing product and you want to know whether your product is done right


Metrics
Questions
Goal
product quality
How much of my product's code is covered by unit tests?
How many defects are in my product?
Test coverage criteria
The number of times a user story is rejected during acceptance testing before it's finally accepted
The number of bugs that are placed in your product backlog

Quick Question

What is the purpose of GQM?

A. To ensure you are build the right product
B. To ensure that you use metrics effectively
C. To make sure that metrics use the correct data
D. To make sure that you ask the client the right questions

Desirable Properties of Metrics

Simple and Easily Computable

- The more complex a metric is, the more likely it is for the person using the metric to make a mistake

Does that mean metrics like "lines of code" or "the number of keystrokes your developers make per hour" are good metrics?

Intuitively Persuasive

- If you calculate the metric, take it to someone under your team and ask them to tell you what it means, it should make sense to them

"The number of keystrokes your developers make per hour" is a simple metric, but it just does not make much intuitive sense to track your developer's keystrokes.

Objective

- Metrics do not depend on anyone's opinion and they're based on empirical data

Can we use your developer's estimates for how productive they are in order to track developer productivity?

In Agile, story points and velocity are better objective metrics to track developer productivity.

Consistent

- Metrics should also be consistent in their use of units and dimensions

If you use story points as your metric for project velocity, then you need to be consistent in your definition of what a story point it.

Programming Language Independent

- Different programming languages vary widely in the amount of lines of code they require to implement a certain functionality

- The amount of code that is required to implement a function could vary from programmer to programmer, based on coding style alone

The lines of code metric is a great counter-example to this.

Quality Improvement

- A metric should give you the ability to improve your quality of your software project or process

You're not just tracking numbers for the sake of tracking numbers. There isn't much point in tracking them in the first place.

Good Metrics – Examples

- The number of defects found in your product per week
  - Simple – track the number of new defects reported in the form of bug reports or logged issues
  - Intuitive – defects are clearly a bad thing in any software project
  - Consistent – consistency across programming languages and units
  - Objective – if a new defect is logged, a new defect is logged
  - Quality improvement - the number of defects you observe per week should decrease over time

Good Metrics –Examples

- Maintainability
  - Complexity metrics, e.g., cyclomatic complexity
- Performance
  - Response time
- Reliability
  - A product’s up time
- Productivity
  - Velocity/story points

Quick Question

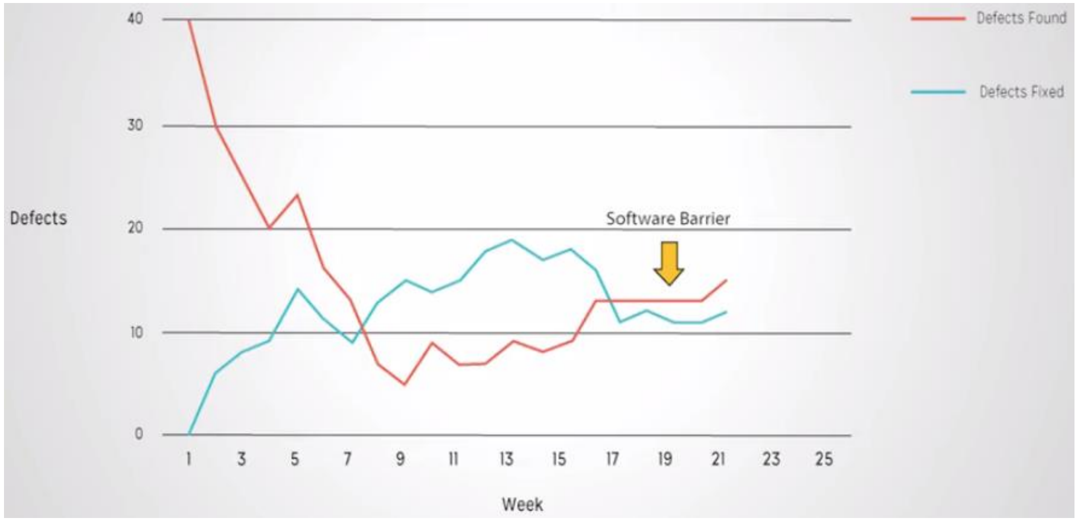
Which of the following is not considered a desirable property of metrics.

A. Improves the quality of your product or process
B. Programming language-independent
C. Consistent in their use of dimensions
D. Reduces the number of hours required to complete the project

Defect Analysis

Ratio of Defects Found vs. Fixed

- The further along in your project you get, the more defects you're fixing versus finding. And fewer defects that remain in your product



If the number of bugs found per week far outweighs the number of bugs that are being fixed each week, then it might be a good idea to invest time into fixing those problems before creating new features.

If you're creating as many defects as you're fixing you're running into what's called a software barrier

Defect Density

- Track the subsystem that is associated with each found defect to see the areas in your product that need attention
  - A ratio of the number of defects found per thousands of lines of code
  - An industry average for structured programming techniques, is about 15 to 50 defects per thousand lines of code

- This should be divided by the size of each subsystem for a fair comparison
  - Big sub systems will naturally have more defects

Quick Question

Jill has been working on some software to help her perform colour analysis on photographs. She calculates the defect density of her code pre-release. 400 defects were found in a total of 100,000 lines of code. In the post-release, however, 600 defects were found. Although her total was 10 defects per thousand lines of code, what strategies could Jill use to help lower her high post-release defect numbers? (Choose two that apply)

A. It doesn't matter, because the overall defect density is below industry average
B. Jill could compare this defect density against that of other subsystems
C. Jill could perform more pre-release testing
D. More experienced developers could be brought onto the team, who may be able to catch these problems easier