Activity Metrics
Activity Metrics Overview

- Metrics that indicate how well we are performing various activities:
  - Requirements, Design, Coding, Testing, Maintenance, Configuration Management, Quality Engineering, etc.

- Most of these are relatively crude indicators:
  - Outlier values indicate possible problems
  - Good values are not conclusive indicators of goodness
  - Most do not measure actual quality of output
    - Process quality does not necessarily imply product quality
    - Just provide detection of some kinds of problems
  - Sort of like MS-Word’s green lines to indicate grammar problems

- Many metrics can be generated by tools and don’t require additional effort or process changes
  - Cheap ways to get some additional useful feedback
  - But don’t ignore the cost of analyzing and reacting
Requirements

- Requirements volatility:
  - Average # of changes per requirement
  - Requirements changes grouped by source

- Requirements density:
  - Number of requirements per function point or KLOC
  - Indicator of granularity of requirements capture

- Number of variations per use case:
  - Indicator of coverage of exception situations

- Requirements defects classification
Requirements Failure

- How the customer explained it
- How the requirements person designed it
- How the programmer wrote it
- How it performed under load
- What the customer really needed

http://projectcartoon.com
Used per Creative Commons License
Requirements Defects Classification

- Can classify requirements defects:
  - Requirements discovery: missed requirements, misunderstood requirements
    - Indicators of elicitation effectiveness
  - Requirements errors: consistency, completeness, ambiguity, etc.
    - Effectiveness of requirements analysis & specification
  - Requirements updates and enhancements identified in design activities:
    - Effectiveness of architecture & component design practices
    - Effectiveness of requirements specification
    - Such as cases not considered
  - Customer-originated updates:
    - Can’t control → opportunities for improving elicitation
- Can do this classification and removal for any of the activities
  - Same concept as DRE
Design Metrics

- Cohesion
- Coupling
- Fan-in / fan-out:
  - Number of methods called by/calling each method
  - Keep within control limits
    - Low fan-out indicates too much hierarchy
    - High fan-out indicates too many dependencies
  - Not absolute rules at all!
- Complexity
Object-Oriented Design Metrics

- Average method size: less is good
- Number of methods per class: within control limits
- Number of instance variables per class: within limits
- Class hierarchy nesting level: < 7 (guideline)
- Number of subsystem/subsystem relationships
  - Less is good? Control limits?
- Number of class/class relationships within subsystem
  - High (relative to subsystem relationships) is good – indicates higher cohesion
- Instance variable grouping among methods
  - May indicate possibility of splits
Code Complexity Metrics

- Comment density
  - Does not tell you quality of comments!
  - Are comments code smells?

- Cyclomatic complexity:
  - Number of branches/decisions
  - Number of operators / line or procedure
  - Useful to estimate complexity of software and expected error rates
    - Most applicable to method and data structure complexity

- Software science:
  - A set of equations that try to derive parametric relationships among different software parameters, and create estimates of “difficulty,” expected effort, faults, etc.
  - Not really proven empirically, and of unclear value?
Historical Perspective

- Much of the early work in metrics was on code complexity and design complexity
  - Of rather limited value, since it quickly gets prescriptive about coding practices, and its outputs are indicators at best
    - Runs easily into various religious arguments

- Even now, this is what some people think of when you mention metrics

- Metrics has now moved on to measuring:
  - Customer view of product
  - Aspects that give you clearer insight into improving development

- Many practitioners have not caught up with this yet
Even So ...

- What “metrics” are implied by the “code smells” that drive refactoring patterns?
- What “metrics” are implied by the need to apply design patterns and architecture styles?
- Since reuse is so important, how do you evaluate the “design quality” of a design and code base you are considering adapting to your use?
Test Metrics: Coverage

- **Black box:**
  - Requirements coverage: test cases per requirement
    - Works with use cases / user stories / numbered requirement
  - Equivalence class coverage
    - Extent of coverage of equivalence classes of input parameters
  - Combinations of equivalence class coverage
    - This is the real challenge

- **Glass box:**
  - Function coverage
  - Statement coverage
  - Path coverage

- There are tools that automatically generate coverage statistics
  - And even create test cases and scripts!
Test Progress

- S-curve
  - Histogram of number of test cases attempted / successful per week of project

- Test defects arrival rate
  - Similar to reliability growth curves

- Test defect backlog curve:
  - Cumulative defects not yet fixed
  - Shows productivity of resolving defects
  - Distinct from defect removal productivity
    - Throughput vs. delay

- Number of severe defects (crashes, freezes, wrong output, etc.) over time
  - Similar to reliability curve, but not as formal
Maintenance Metrics

- Fix backlog:
  - Age of open and closed problems
  - Backlog management index: closed rate / arrivals rate
  - Fix response time: mean time from open to closed

- Fixing effectiveness: \((1 - \% \text{ of bad fixes})\)

- Fixing delinquency: \% closed within acceptable response time
Configuration Management

- Defect classification can provide insight into sources of CM problems

- Also, “Configuration Status Accounting” (CSA):
  - Tool-based cross-check of expected progress
  - As project moves through different phases or increments, would expect different documents to be generated / modified
  - CSA reports which files are being modified
    - Powerful, advanced technique
    - If pre-configured which expected modifications, can flag discrepancies
    - Can go deeper and look at extent of modifications
    - Also useful to monitor which files are modified during defect fixes, hence which regression tests need to run
Quality Engineering

- Assessment results
  - Red/yellow/green on practices in each area
    - For example, requirements, planning, CM etc.

- Classifying defects: Defects related to “not following process”

- Shape of various curves
  - For example, wide variations in estimation accuracy or defect injection rates might show non-uniform practices
In-Process Metrics

- Metrics can help us determine whether projects went well and where problems are.
- Some metrics are most meaningful after the project is done, such as productivity or cycletime.
- Other metrics can be used to diagnose problems while the project is in progress, or to ensure that activities are done right:
  - Most activity metrics are used as in-process metrics.
  - Defect density, even DRE defect removal patterns can be used as in-process metrics, but need to be careful.
  - Many metrics are not fully available until the end of project, but can monitor how the metric evolves as project proceeds.
- Most in-process metrics are like dashboard gauges: out-of-range values indicate problems, but “good” values do not guarantee health.
Summary

- Activity metrics help us to gauge the quality of activity execution:
  - Most are useful as indicators, but crude and inconclusive
  - Cheap to generate, so good benefit/cost
  - Don’t “work to the metrics”!

- People are constantly coming up with new ones