Defect Removal Metrics

Defect removal metrics: concepts

- All defect removal metrics computed from the measurements identified last time
  - Inspection reports, test reports, field defect reports
- Used to get different views on what's going on
  - Each metric can be used to tell us something about the development process or results
    - Many are amazingly useful, though all have limitations
  - Need to learn how to use each tool effectively
- For most defect metrics, filter out minor and cosmetic defects
  - Can easily make many metrics look good by finding more or fewer cosmetic problems (level of nitpicking)

Measuring "total number of defects"

- Many metrics have parameters such as "total number of defects" e.g. total number of requirements defects
- Clearly, we only ever know about the defects that are found
  - So we never know the "true" value of many of these metrics
- Further, as we find more defects, this number will increase
  - Hopefully, finding defects in asymptotic over time i.e. we find fewer defects as time goes along, esp. after release
  - So metrics that require "total defects" type info will change over time, but hopefully converge eventually
- The later in the lifecycle we compute the metric, the more meaningful the results
- If and when we use these metrics, we must be aware of this effect and account for it.

Measuring size

- Many defect metrics have “size” parameters
  - The most common size metric is KLOC (thousands of lines of code)
    - Depends heavily on language, coding style, competence
    - Code generators may produce lots of code, distort measures
    - Does not take “complexity” of application into account
    - Easy to compute automatically and “reliably” (but can be manipulated)
  - An alternative size metric is “function points”
    - See http://www.qpmg.com/fp-intro.htm
    - A partly-subjective measure of functionality delivered
    - Directly measures functionality of application: number of inputs and outputs, files manipulated, interfaces provided etc.
    - More valid but less reliable, more effort to gather

- We use KLOC in our examples, but works just as well with FPs

Defect density

- Number of defects / size
- Defect density in released code (“defect density at release”) is a good measure of organizational capability
  - Defects found after release / size of released software
- Can compute phasewise and component-wise defect densities
  - Useful to identify “problem” components that could use rework or deeper review
  - Note that problem components will typically be high-complexity code at the heart of systems
- Defect densities (and most other metrics) vary a lot by domain
  - Can only compare across similar projects

Using defect density

- Very useful as measure of organizational capability to produce defect-free outputs
  - Can be compared with other organizations in the same domain
- Outlier information useful to spot problem projects and problem components
- Can be used in-process, if comparison is with defect densities of other projects in same phase
  - If much lower, may indicate defects not being found
  - If much higher, may indicate poor quality of work
  - (In other words, need to go behind the numbers to find out what is really happening. Metrics can only provide triggers)

Defect Density: Limitations

- Size estimation itself has problems
  - We will discuss this in next class
- “Total defects” problem
- Criticality and criticality assignment
  - Combining defects of different criticalities reduces validity
  - Criticality assignment is itself subjective
- Defects != reliability (user experience) problem
- Statistical significance when applied to phases and components
  - Actual number of defects may be so small that random variation can mask significant variation

Defect Removal Effectiveness

- % of defects removed during a phase
  - (Defects found) / (Defects found during that phase + defects not found)
  - Approximated by
    - (defects found) / (defects found during that phase + defects found later)
- Includes defects carried over from previous phases
- Good measure of effectiveness of defect removal practices
  - Test effectiveness, inspection effectiveness
- Correlates strongly with output quality
- Other terms: *Defect removal efficiency*, *error detection efficiency*, *fault containment* etc.

DRE table example

Phase of Origin

Phase found

	Req	Des	Code	UT	IT	ST	Field	Total	Cum.
Req	5							5	5
Des	2	14						16	21
Code	3	9	49					61	82
UT	0	2	22	8				32	114
IT	0	3	5	0	5			13	127
ST	1	3	16	0	0	1		21	148
Field	4	7	6	0	0	0	1	18	166
Total	15	38	98	8	5	1	1	166	
Cum.	15	53	151	159	164	165	166		

Phase found

(Illustrative example, not real data) Phase of Origin

DRE computation

- http://www.westfallteam.com/defect_removal_effectiveness.pdf
- See above link for an example of DRE computations
- The text has a good example on p.166

DRE value

- Compute effectiveness of tests and reviews
  - Actual defects found / defects present at entry to review/test
  - (Phasewise Defect Removal Efficiency: PDRE)
- Compute overall defect removal efficiency
  - Problems fixed before release / total originated problems
- Analyze cost effectiveness of tests vs. reviews
  - Hours spent per problem found in reviews vs. tests
  - Need to factor in effort to fix problem found during review vs. effort to fix problem found during test
  - To be more exact, we must use a *defect removal model*
    - Discussed later
- Shows pattern of defect removal
  - Where defects originate (“injected”), where they get removed

Counter-intuitive implication

If testing reveals lots of bugs,
Likely that final product will be very buggy too

Not true that "we have found and fixed a lot of problems, now our software is OK"

We can only make this second assertion if testing reveals lots of bugs early on, but the latter stages of testing reveal hardly any bugs – and even then, only if you are not simply repeating the same tests!

DRE Limitations

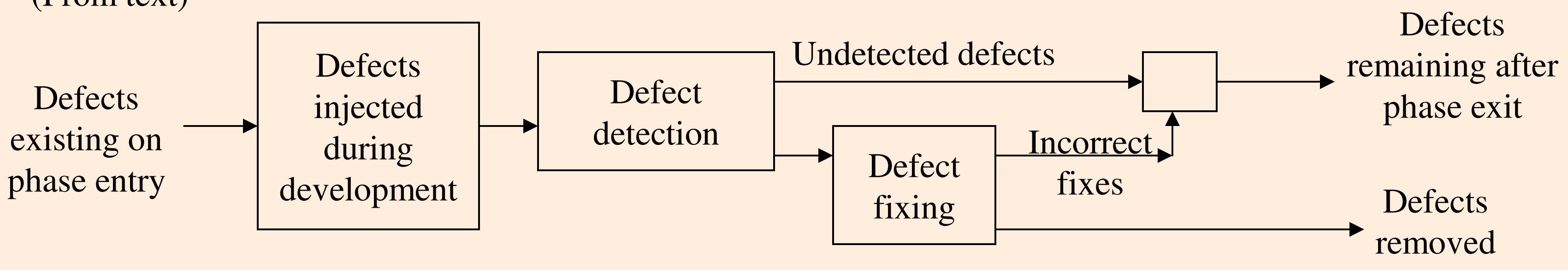
- Statistical significance
  - Note how small the numbers in each box are
  - Hard to draw conclusions from data about 1 project
    - At best a crude indicator of which phases & reviews worked better
  - Organizational data has far more validity
  - Remember that when the numbers are small, better to show the raw numbers
    - Even if you showing DRE %s, include actual defects data in each box
- Full picture only after project completion
- Easily influenced by underreporting of problems found

Other related metrics

- Phase containment effectiveness
  - % of problems introduced during a phase that were found within that phase
    - E.g. Table 1 design PCE = 14/38 = 0.37 (37%)
  - PCE of 70% is considered very good
- Phasewise Defect Injection Rate
  - Number of defects introduced during that phase / size
  - High injection rates (across multiple projects) indicate need to improve the way that phase is performed
    - Possible solutions: Training, stronger processes, checklists

Defect Removal Model

(From text)


Defects existing on phase entry
Defects injected during development
Defect detection
Undetected defects
Defect fixing
Incorrect fixes
Defects remaining after phase exit
Defects removed

- Can predict defects remaining, given
  - Historical data for phasewise defect injection rates
  - Historical data for rates of defect removal
  - Historical data for rates of incorrect fixes
  - Actual phasewise defects found
- Can statistically optimize defect removal, given (in addition to rates)
  - Phasewise costs of fixing defects
  - Phasewise costs of finding defects (through reviews and testing)
- Can decide whether it is worthwhile to improve fault injection rates, by providing additional training, adding more processes & checklists etc.

This is "statistical process control". But remember all the disclaimers on its validity

Additional metrics for inspections

- Several simple (secondary) metrics can be tracked and managed within control limits
  - Inspection rates
    - Size / Duration of inspection meeting
    - Very high or very low rates may indicate problem
  - Inspection effort
    - Preparation + meeting + tracking / size
  - Inspection preparation time
    - Avoid overloading others on team, make sure preparation happens
- Inspection effectiveness is still the bottom line
  - These are just helping with optimizing inspections

Cost of Quality (COQ)

- Total effort put into quality-related activities
  - Testing and test development effort
  - Inspections and reviews
  - Quality assessments and preparation
- COQ is a % of project effort
  - Pure number, suitable for comparisons across projects, organizations
- Can observe relationships between COQ and defect removal efficiency, COQ and release defect density
  - Note that defect prevention reverses the normal relationships: reduces both COQ and release defect density
    - Will NOT show up in defect removal effectiveness!

Cost of Poor Quality (COPQ)

- Total effort put into rework
  - Cost of fixing defects
  - Cost of revising/updating docs
  - Cost of re-testing, re-inspecting
  - Cost of patches & patch distribution, tracking defects
- % of project effort – pure number
- Would generally correlate well with defect densities
  - If there are fewer defects, less rework needed
- COPQ < 10% is very good
- Note that early defect detection (inspections) reduces COPQ

Optimizing quality efforts

- Normally, there is a balance between COQ and COPQ
  - To reduce rework, need to spend more effort on quality upfront
  - Note that high COPQ increases COQ, because of re-testing
- Defect prevention and superior quality approaches (better test methodologies, more effective reviews) cut both COQ and COPQ
- Objective is to have a lower COQ while maintaining good (low) COPQ and low release defect density
  - More quality efforts will always improve quality, but there is a point of diminishing returns
    - COPQ, release defect density within targets -> adequate quality

Limitations of COQ/COPQ

- Assumes the numbers are accurate
  - That they fairly reflect all defect removal activities and all rework activities
- COPQ easily distorted if there is one requirements or design bug that creates a large amount of rework
- Balancing COQ / COPQ is an organizational-level activity
  - Improves statistical significance, averages out variations
  - Evens out distortions in COPQ due to a couple of high-rework bugs
  - Need to wait till product has been in the field to get “truer” COPQ numbers
- Can Use COQ / COPQ at the project level as indicators
  - But need to go behind the numbers to interpret better
- Hard for COQ to account properly for unit tests if developers do it along with coding
- Inspections often have additional hidden effort because developers will go through their code extra carefully before submitting it for inspection

Levels of sophistication in quality efforts (yes, this is a repeat)

- Systematic testing approaches

- Multi-stage approach to quality: inspections, unit & integration testing
- Basic metrics: defect densities

- DRE chart
  - Identifying problem components
  - Inspection & test effectiveness
- Checklists and quality tools

- Systematic defect prevention
- Formal methods
- Optimizing quality efforts
  - Minimizing COQ & COPQ
  - Statistical process control using defect removal models

Summary

- Need multiple stages of defect removal
  - Inspections are well-known to be cost-effective
  - Early detection of defects saves work
  - More expensive to fix bugs late in lifecycle
- Release defect densities tell us about the quality of the product
- DRE chart helps us to
  - compute inspection and test effectiveness
  - predict field defect rates
  - see pattern of defect removal
- Defect removal metrics can also help optimize effort spent on quality activities
- Notice how all these fancy metrics come from just the basic review and test reports! Don't gather lots of data, focus on meaningful analysis

Exercises

- From DRE table, compute
  - Effectiveness of the code inspections (“*how good are the code inspections*?”)
  - Effectiveness of unit testing (“*how good are the unit tests*?”)
  - Requirements PCE (“*how well are req problems found and fixed early*?”)
- Given that the codesize is 30 KLOC, compute
  - Release defect density (“*how good is the product*?”)
  - Coding defect injection rate (“*how many mistakes made during coding*?”)
- The system has 6 modules, with sizes and defects found shown below. Which of the modules do you think may need redesign / redevelopment?

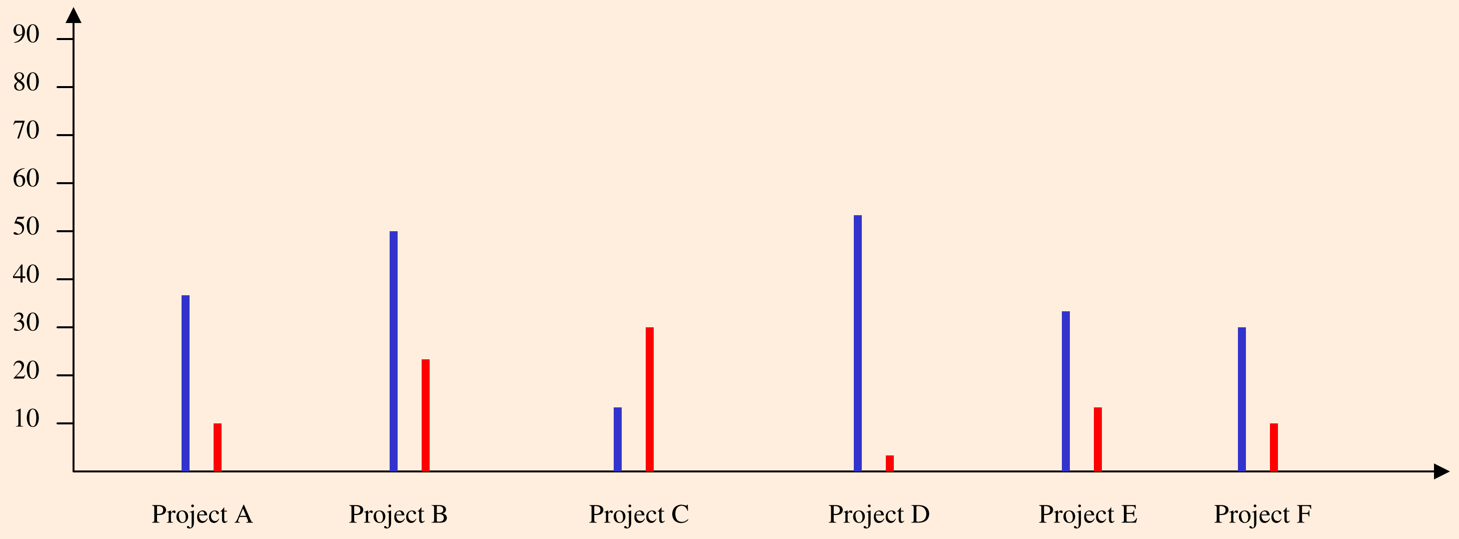
	Module 1	Module2	Module3	Module4	Module5	Module6
Size (KLOC)	1	15	8	1	3	2
Req defects	1	2	3	5	1	3
Des defects	6	8	6	8	7	3
Code defects	16	26	12	19	11	14

Exercises

- Your organization has executed many projects over the past few years, inspects all documents and code, and fills out test and inspection reports regularly. Given this data, how would you figure out
  - Whether your requirements techniques need improvement?
  - Whether your configuration management techniques need improvement?
  - Whether your design inspections are effective?
  - Whether you are doing a good job of delivering quality code?
  - Whether coding mistakes are more likely in large modules, or in complex modules (i.e. complex functionality)?
  - How effective the coding standards and checklists that you introduced two years ago have been in reducing coding mistakes?
  - Whether teams consisting largely of highly experienced people are better at avoiding requirements and design errors that teams of less experienced people?

Exercises

- The graph below shows COQ & COPQ for 6 “similar” projects in a company.
  - What observations can you make about each project from the data?
  - What (all) inferences can you make about what might be happening on each project?
  - What all does this tell you about the organization?
  - What are the assumptions and limitations underlying your inferences?



Exam Review

- Sample questions
  - If you see that an organization is at CMM level 3, what does it tell you about their quality systems?
  - If another organization tell you it is at CMM level 5, what does it tell you about their quality systems, relative to the first? What about their output quality?
  - If you wanted to compare the quality of two products from different organizations, how and to what extent can you do that using metrics?
  - How can you use metrics and quality tools to determine the extent to which requirements changes are a major reason for late delivery of projects?
  - Your organization often gets complaints from customers that “customer support reps are rude / unhelpful”. How would you use the quality tools to systematically address this problem?