Defect Removal Metrics

Objectives

- Understand some basic defect metrics and the concepts behind them
  - Defect density metrics
  - Defect detection and removal effectiveness
  - etc.
- Look at the uses and limitations of quality metrics
- Build some intuition to help balance investment in quality against the cost of poor quality
  - Cost of Quality ←→ Cost of Poor Quality

Defect Removal Metrics: Concepts

- All defect removal metrics are 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 metric and 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"
  - For example: 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 is asymptotic over time
    - We find fewer defects as time goes along, especially after release
  - So metrics that require "total defects" information will change over time, but hopefully converge eventually
- The later in the lifecycle we compute the metric, the more meaningful the results, but also the less useful for the current project
- If and when we use these metrics, we must be aware of this lag 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
    - Are included libraries counted?
    - 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” (FP’s)
    - 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 FP’s
- Be careful with using “feature count” in agile processes

Defect Density

- Most common metric: 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 defect densities per phase, per increment, per component, etc.
  - Useful to identify “problem” components that could use rework or deeper review
    - Heuristic: Defects tend to cluster
  - Note that problem components will typically be high-complexity code at the heart of systems
    - Focus early increments on complex functionality to expose defects and issues early

Using Defect Density

- Defect densities (and most other metrics) vary a lot by domain
  - Can only compare across similar projects
- Very useful as measure of organizational capability to produce defect-free outputs
  - Can be compared with other organizations in the same application 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 or increment
  - If much lower, may indicate defects not being found
  - If much higher, may indicate poor quality of work
  - (Need to go behind the numbers to find out what is really happening – Metrics can only provide triggers)

Defect Density: Limitations

- Size estimation has problems of reliability and validity
- "Total Defects" problem: Can only count the defects you detect
- Criticality and criticality assignment
  - Combining defects of different criticalities reduces validity
  - Criticality assignment is itself subjective
- Defects may not equal reliability
  - Users experience failures, not defects
- Statistical significance when applied to phases, increments, and components
  - Actual number of defects may be so small that random variation can mask significant variation

Defect Removal Effectiveness

- Percentage of defects removed during a phase or increment
  - (Total Defects found) / (Defects found during that phase + Defects not found)
  - Approximated by:
    - (Defects found) / (Defects found during that phase + Defects found later)

(Can only count the defects you detect)

- Includes defects carried over from previous phases or increments
- 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 Found	Cum. Found
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 Injected	15	38	98	8	5	1	1	166	
Cum. Injected	15	53	151	159	164	165	166		

Phase Found

(Illustrative example, not real data)

Phase of Origin

DRE Table Example: Increments

Increment of Origin

Increment Found

	I-1	I-2	I-3	I-4	I-5	I-6	Field	Total Found	Cum. Found
I-1	5							5	5
I-2	2	14						16	21
I-3	3	9	49					61	82
I-4	0	2	22	8				32	114
I-5	0	3	5	0	5			13	127
I-6	1	3	16	0	0	1		21	148
Field	4	7	6	0	0	0	1	18	166
Total Injected	15	38	98	8	5	1	1	166	
Cum. Injected	15	53	151	159	164	165	166		

Increment Found

(Illustrative example, not real data)

Increment of Origin

Requirements Phase DRE Example

- In the requirements phase, 5 requirements defects were found and removed
- But additional requirements defects were found in later phases. The total number of found requirements defects at the end of all phases (plus field operations) is 15
    - 15 total requirements defects injected
- DRE in requirements phase is 5/15 (# found / # available to find)

Phase Found

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

Total defects found in requirements phase = 5

Total requirements defects injected = 15

(Illustrative example, not real data)

Phase of Origin

Design Phase DRE Example

- In the design phase, 14 design defects were found and removed, plus 2 requirements defects were found and removed.
    - Total defects found and removed: (14+2) = 16
- Additional design defects were found in later phases: 38 total design defects injected

Total defects removed prior to design phase = 5

Total defects found in design phase = 16

Total design defects injected = 38

Phase Found

	Req	Des	Code	UT	IT	ST	Field	Total Found	Cum. Found
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	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]1	148
Field	4	7	6	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]8	166
Total Injected	15	38	98	8	5	1	1	166	
Cum. Injected	15	53	151	159	164	165	166		

Total defects available to find = 48
(Cum. injected – Cum. Found in prior phases)

- To compute removal effectiveness in the design phase, we need to count how many defects (requirements and design) were still in the system (we do not count those already found and removed in the requirements phase)
    - There were 15 requirements defects total injected, but 5 had already been found and removed in the requirements phase ➔ 10 requirements defects available to find
    - There were 38 total design defects injected, and 14 of those 38 were found
    - So, in design phase
        - (2+14) defects found
        - (10 + 38) defects available to find
        - Design phase DRE = (2+14)/(10+38) = 16/48

Coding Phase DRE Example

Coding phase DRE = 61/130

	Req	Des	Code	UT	IT	ST	Field	Total Found	Cum. Found
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				21	148
Field	4	7	6	0	0	0	1	18	166
Total Injected	15	38	98	8	5	1	1	166	
Cum. Injected	15	53	151	159	164	165	166		

Total defects removed prior to coding phase = 21

Total defects found in coding phase = 61

Total coding defects injected = 98

Total defects available to find = 130

Phase Found

Phase of Origin

(Illustrative example, not real data)

Unit Test Phase DRE Example

Unit Test phase DRE = 32/77

	Req	Des	Code	UT	IT	ST	Field	Total Found	Cum. Found
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				
Total Injected	15	38	98	8	5	1	1	166	
Cum. Injected	15	53	151	159	164	165	166		

Total defects found in unit test phase = 32

Total defects available to find = 77

Phase Found

Phase of Origin

(Illustrative example, not real data)

DRE Value

- Compute effectiveness of tests and reviews:
  - Actual defects found / defects present at entry to review/test
  - (Phasewise Defect Removal Effectiveness: PDRE)
  - For incremental development, Increment Defect Removal Effectiveness: IDRE
- Compute overall defect removal effectiveness:
  - 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
- Shows pattern of defect removal
  - Where defects originate ("injected"), where they get removed

DRE Implications

- 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

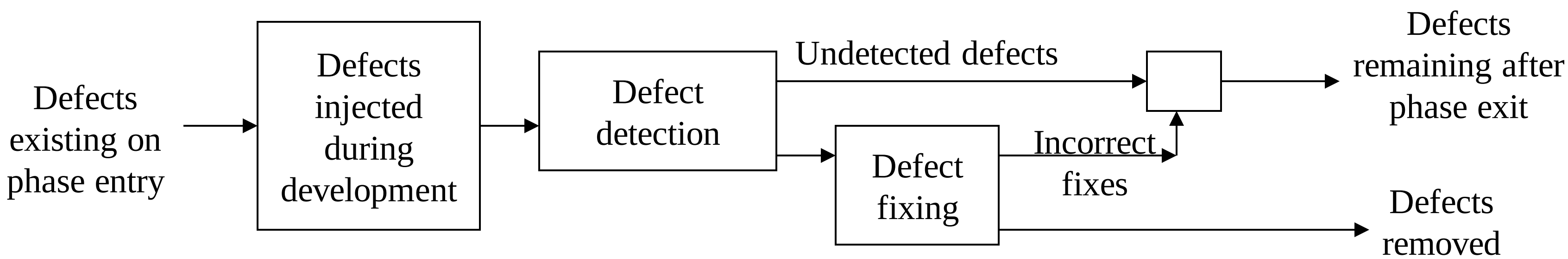
- Statistical significance:
  - Note how small the numbers in each box are
  - Hard to draw conclusions from data about one project
    - At best a crude indicator of which phases and reviews worked better
  - Organization-wide data has far more validity
  - Remember that when the numbers are small, better to show the raw numbers
    - Even if you show DRE percentages, include actual defect count data in each box
    - (DRE = 32/77 preferred to DRE = 42%)
- 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
    - For example, 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, tools, checklists, etc.
  - Similar for Increment Defect Injection Rate

Defect Removal Model

(From Kan Text)



- 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 finding defects (through reviews and testing)
  - Phasewise costs of fixing defects
- Can decide whether it is worthwhile to reduce fault injection rates, by providing additional training, adding more processes and 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 problems
  - Inspection effort:
    - (Preparation + meeting + tracking) / size
  - Inspection preparation time:
    - Make sure preparation happens
    - Avoid overloading others on team
- Inspection effectiveness is still the bottom line
  - These are just helping with optimizing inspections

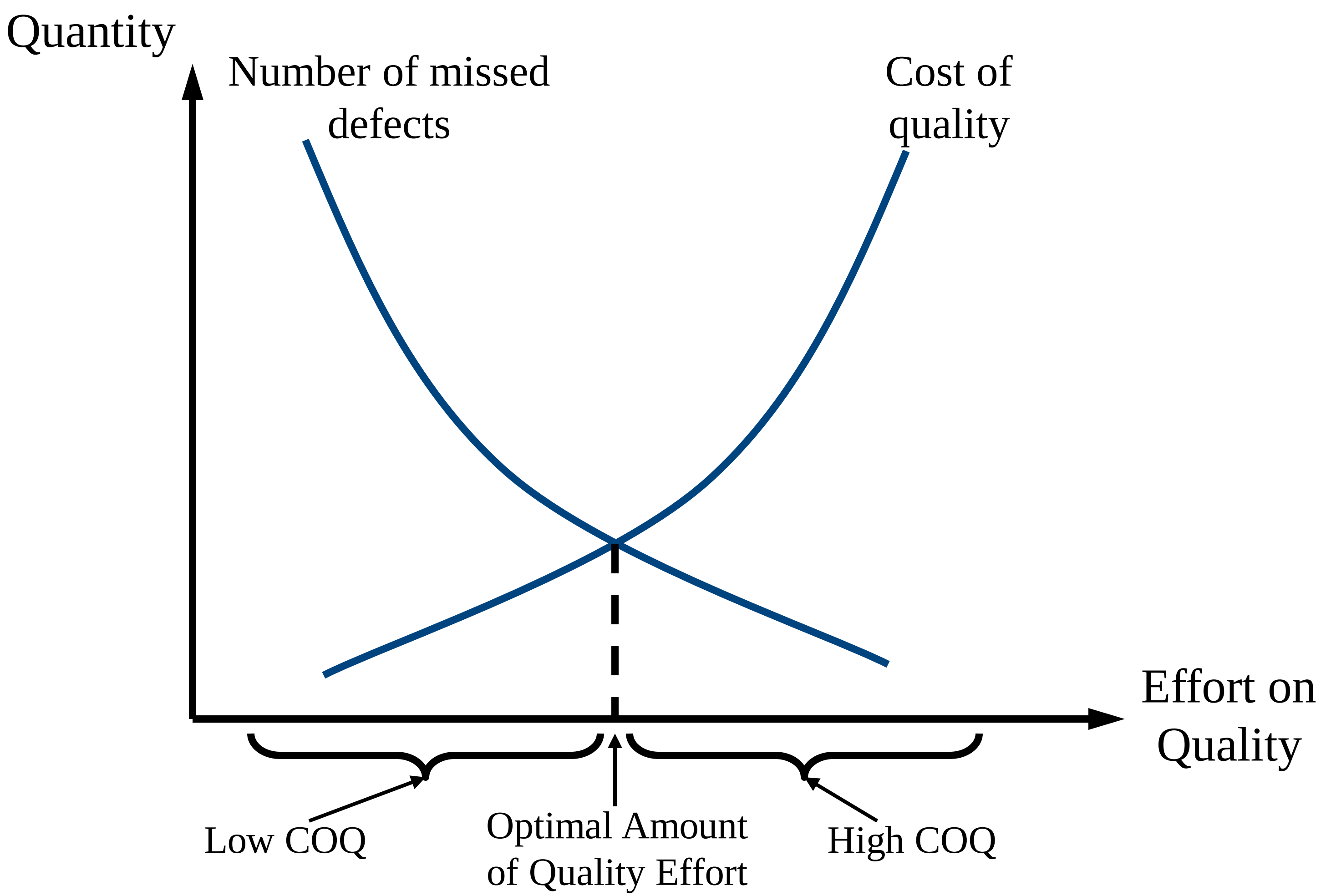
Cost of Quality (COQ)

- Total effort put into quality-related activities:
  - Testing and test development effort
  - Inspections and reviews
  - Quality assessments and preparation
- COQ is a percentage of project effort
  - Pure number, suitable for comparisons across projects and 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 affected documentation
  - Cost of re-testing, re-inspecting
  - Cost of patches & patch distribution
  - Cost of tracking defects
- Percentage of project effort, a 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) and defect prevention reduce COPQ
- Here, COPQ is cost of rework. Also consider cost of re-deployment (patches), customer dissatisfaction, etc.

Quality Sweet Spot



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 and other re-work
- Defect prevention and superior quality approaches (better test methodologies, more effective reviews, etc.) 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 the numbers 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 until product has been in the field to get “truer” COPQ numbers
- Should COPQ include “customer expectation management?”
  - It is more expensive to gain a customer than to keep a customer

Limitations of COQ/COPQ - Cont'd

- 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

Conclusions

- Defect densities tell us a lot about the quality of the product
- 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
- DRE and similar charts help 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: COQ vs. COPQ
- Notice how all these fancy metrics come from just the basic review and test reports!
  - Don’t gather too much data; focus on meaningful analysis