SOFTWARE DESIGN
COSC 4353/6353

Dr. Raj Singh

SOFTWARE QUALITY

Functional Quality

- How well it complies with or conforms to a given design, based on functional requirements or specifications.

Structural Quality

- Refers to how it meets non-functional requirements that support the delivery of the functional requirements, such as robustness or maintainability, the degree to which the software was produced correctly.

WHY CARE ABOUT CODE QUALITY?

You can't be Agile if your Code sucks

Customers prefer quality software products

A product of poor quality is very difficult to maintain

Its just like a ceramic plate that must not drop to the ground

Updates to bad code is a nightmare which if not handled carefully can break the whole product

QUALITY IMPACTS BUSINESS

- Software quality drives predictability.
- Do it once and do it right
- There will be less re-work, less variation in productivity and better performance overall
- Products get delivered on time, and they get built more productively
- Poor quality is much more difficult to manage

QUALITY IMPACTS REPUTATION

Some companies have a reputation for building quality software.

A good, solid reputation is hard to establish and easy to lose, but when your company has it, it's a powerful business driver.

A few mistakes and that reputation can be gone, creating major obstacles to sales, and consequently, your bottom line.

QUALITY IMPACTS EMPLOYEE

The most productive and happy employees have pride in their work.

Enabling employees to build quality software will drive a much higher level of morale and productivity.

On the other hand, poor products, lots of re-work, unhappy customers and difficulty making deadlines have the opposite effect, leading to expensive turnover and a less productive workforce.

QUALITY IMPACTS CUSTOMERS

A quality product satisfies the customer.

A satisfied customer comes back for more and provides positive referrals.

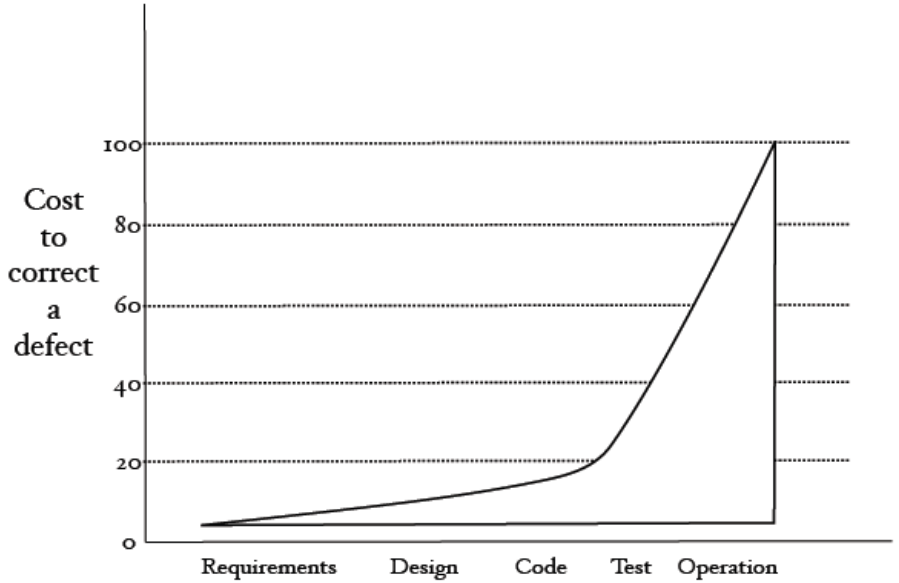
Customer loyalty is heavily driven by the quality of the software you produce and service you provide.

Explosion of social media channels such as Twitter and Facebook, positive referrals can spread quickly.

Poor quality and dissatisfaction can also be communicated quickly, if not even quicker than the good ones.

COST OF DEFECTS

- It's less costly to fix a defect if it's discovered early in the project
- As project moves forward cost increases
- The graph shows the cost impact of fixing defects



SOFTWARE DEFECT REDUCTION TOP 10 LIST

- Finding, fixing problem in production is 100 times more expensive than during requirements/design phase.
- 40-50% of effort on projects is on avoidable rework.
- ~80% of avoidable rework comes from 20% of defects.
- ~80% of defects come from 20% of modules; about half the modules are defect free.
- ~90% of downtime comes from at most 10% of defects.
- Peer reviews catch 60% of defects.
- Perspective-based reviews catch 35% more defects than non-directed reviews.
- Disciplined personal practices can *reduce defect introduction rates* by up to 75%.
- It costs 50% more per source instruction to develop high-dependability software product.
- ~40-50% of user programs have nontrivial defects.

WHY SHOULD I CARE, THERE'S QA?

Can't QA take care of quality, why should developers care?

QA shouldn't care about quality of design and implementation

”

They should care about acceptance, performance, usage, and relevance of the application

Give them a better quality software so they can really focus on that

PAY YOUR TECHNICAL DEBT

Technical debt are activities like

refactoring, upgrading a library, conforming to some UI or coding standard

These will hamper your progress if left undone for a longer time

You'll be more productive if quality is better

MEASURING QUALITY

Some quality attributes are objective, and can be measured accordingly.

Some are subjective, and are therefore captured with more arbitrary measurements.

QUALITY ATTRIBUTES

Quality attributes can be external or internal.

External: Derived from the relationship between the environment and the system (or the process). (To derive, the system or process must run)

e.g. Reliability, Robustness

Internal: Derived immediately from the product or process description (To derive, it is sufficient to have the description)

Underlying assumption: internal quality leads to external quality (cfr. metaphor manufacturing lines)

e.g. Efficiency

CORRECTNESS, RELIABILITY, ROBUSTNESS

Correctness

A system is correct if it behaves according to its specification

An absolute property (i.e., a system cannot be "almost correct")

Reliability

The user may rely on the system behaving properly

Reliability is the probability that the system will operate as expected over a specified interval

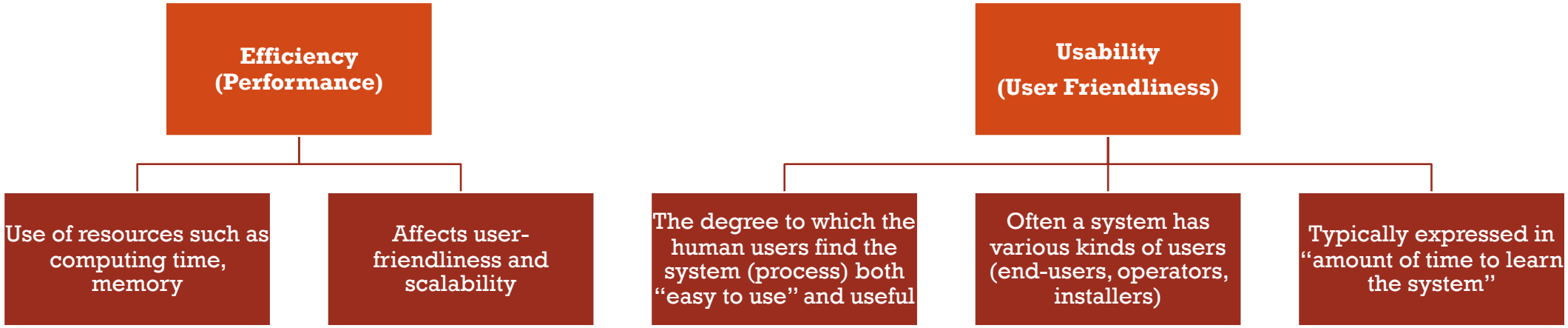
A relative property (a system has a mean time between failure of 3 weeks)

Robustness

A system is robust if it behaves reasonably even in circumstances that were not specified

A vague property (once you specify the abnormal circumstances they become part of the requirements)

EFFICIENCY, USABILITY



MAINTAINABILITY

How easy it is to change a system after its initial release

software entropy ⇒ maintainability gradually decreases over time

Repairability

How much work is needed to correct a defect

Evolvability (Adaptability)

How much work is needed to adapt to changing requirements (both system and process)

Portability

How much work is needed to port to new environment or platforms

Verifiability, Understandability

Verifiability:

How easy it is to verify whether desired attributes are there?

- internally: verify requirements, code inspections
- externally: testing, efficiency

Understandability:

How easy it is to understand the system?

- internally: contributes to maintainability
- externally: contributes to usability

Productivity

- Amount of product produced by a process for a given number of resources
- productivity among individuals varies a lot

Timeliness

- Ability to deliver the product on time
- often a reason to sacrifice other quality attributes

Visibility (Transparency)

- Current process steps and project status are accessible
- important for management

PRODUCTIVITY, TIMELINESS, VISIBILITY

WAYS TO IMPROVE QUALITY

- Start early
- Don’t Compromise
- Schedule time to lower your technical debt
- Make it work; make it right)right away*
- Requires monitoring and changing behavior
- Be willing to help and be helped
- Devise lightweight non-bureaucratic measures

INDIVIDUAL EFFORTS

- What can you do?
- Care about design of your code
- Keep it Simple
- Write tests with high coverage
- Run all your tests before checkin
- Learn your language
- Court feedback and criticism

TEAM EFFORTS

- Avoid shortcuts
- Take collective ownership — team should own the code
- Promote positive interaction
- Provide constructive feedback
- Constant code review

CODE REVIEW

Code review is by far the proven way to reduce code defects and improve code quality

Code review does not work if it's not done right.

Do you get together as a team, project code, and review?

Don't make it an emotionally draining

SEEKING AND RECEIVING FEEDBACK

- We've used code review effectively
- Code reviewed by one developer right after task is complete (or anytime before)
- Rotate reviewer for each review
- Say positive things, what you really like
- Constructively propose changes
- Instead of "that's lousy long method" say, "why don't you split that method..."
- Review not only code, but also tests
- Do not get picky on style, instead focus on correctness, readability, and design.

VALUE OF REVIEW

Rigorous inspection can remove up to 90 percent of errors before the first test case is run.

Reviews are both technical and sociological, and both factors must be accommodated.

Code review makes me smarter.

I learn ways to improve my code by looking at somebody's code.

CODE QUALITY FACTORS

- Cohesion – single responsibility principle
- Extensibility and Flexibility – OOP
- Triangulation – generalization
- Cost of Change – code that does many things is hard to maintain
- Code Coverage – how much code needs testing
- Complexity – large classes and methods are hard to maintain
- Code Size – too much or too little
- Code Duplication – why are you repeating?

CODE ANALYSIS

Analyzing code to find bugs

Look for logic errors, coding guidelines violations, synchronization problems, data flow analysis, …

IDEs

Automated tools

Other tools:

[Java] PMD, FindBugs, JLint
[.NET] VS, FxCop, …
[C++] VS, Lint, …

Code smell

It's a feeling or sense that something is not right in the code

You can't understand it

Hard to explain

Does some magic

Common code smells

- Duplication
- Unnecessary complexity
- Useless/misleading comments
- Long classes, methods, poor naming
- Code that's not used
- Improper use of OOP
- Tight coupling
- Design Pattern overuse

Things to remember for quality code

Deal with code smells	Refactor frequently
Don't rush	take time to write tests
Commenting and self documenting code	Just enough comments. Don't write stories.
Capture errors	Surround code that might be troublesome with try catch blocks

Things to remember for quality code

- Practice tactical peer code review
- Consider untested code is unfinished code
- Make your code coverage and metrics visible
- Use tools to check code quality
- Treat warnings as errors
- Keep it small and simple

HOMEWORK

Review class notes.

Additional reading:

Why quality matters?

Start a discussion on Google Groups to clarify your doubts.