Purpose

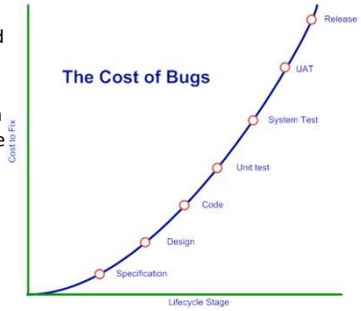
1. To give you some background on the **cost of quality** and provide a motivation to measure quality
2. To provide an annotated context to the topic of **measurement**

Cost of Correcting Defects

- Relative costs to find and repair a defect increase dramatically:
  - From prevention to detection
  - From internal vs. external

The Cost of Defects

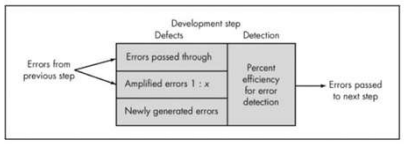
- Software is produced in a development cycle.
- Defects (or bugs) can be found at any stage in the lifecycle
- The cost of fixing bugs grows over the course of the development lifecycle.



The Cost of Defects

- Why is the graph the shape it is?
  - If a bug is found in requirements (before any other work is carried out) the only change needed may be to a document or a user story.
  - If the same bug is found in user acceptance testing, a substantial rewrite of software may be required, costing substantially more.
  - It is sometimes suggested that the cost of fixing errors goes up by ten times from one stage to another.
- One large insurance company estimates a fifty-fold increase at each stage. (Geoff Thompson, Experimentus)

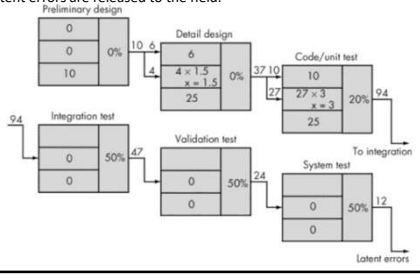
Defect Amplification Model



- A box represents a software development step.
  - During the step, errors may be inadvertently generated.
  - Review may fail to uncover newly generated errors and errors from previous steps, resulting in some number of errors that are passed through.
  - In some cases, errors passed through from previous steps are amplified (amplification factor, x) by current work.
- Box subdivisions represent each of these characteristics and the percent of efficiency for detecting errors, a function of the thoroughness of the review.

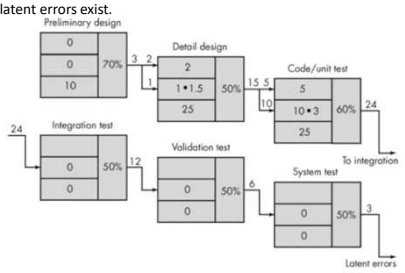
Example: Defect Amplification with No Reviews

- Each test step uncovers and corrects 50% of all incoming errors without introducing any new errors.
- 10 preliminary design defects are amplified to 94 errors before testing
- 12 latent errors are released to the field.



Example: Defect Amplification with Reviews

- Same conditions except that design and code reviews are conducted as part of each development step.
- 10 initial preliminary design errors are amplified to 24 errors before testing
- Only 3 latent errors exist.



Costs of software quality

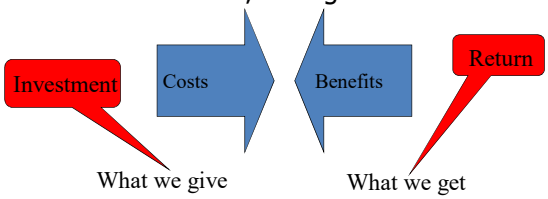
- The BIG questions for us…
  - Does higher quality mean **more cost** OR it is an investment that pays you back?
  - Does higher quality mean a **longer time** to develop OR can development efforts can be reduced?
  - How much do we need to be aware of the *lifetime* costs?
  - Is there any relation between high quality and business performance?

9

Return on Investment

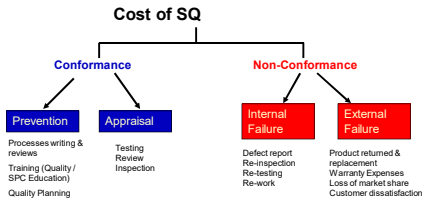
Return on Investment

- Treating quality assurance, process improvement, as an investment.
- For every dollar put into these activities what is the income, savings?



Cost of Software Quality

- Cost of software quality represents all costs that an organization incurs from having to repeat a process in order to complete the work as per requirements.



12

Components of Cost of SQ

- **Prevention Cost** are the costs incurred to prevent failure in meeting the requirements.
  - Processes writing & reviews
  - Training (Quality / SPC Education)
  - Quality Planning
- **Appraisal Cost** are the costs incurred to detect failure in meeting the requirements.
  - Testing
  - Review
  - Inspection
- **Internal Failure Cost** is any cost incurred as a result of failure to meet the requirements and software has not been transferred to customer.
  - Pre-release defect management
    - Defect report
    - Re-inspection
    - Re-testing
    - Re-work
- **External Failure Cost** is any cost incurred as a result of failure to meet requirements and software has been transferred to customer.
  - Product returned & replacement
  - Warranty Expenses
  - Loss of market share
  - Customer dissatisfaction
  - Defect Resolution

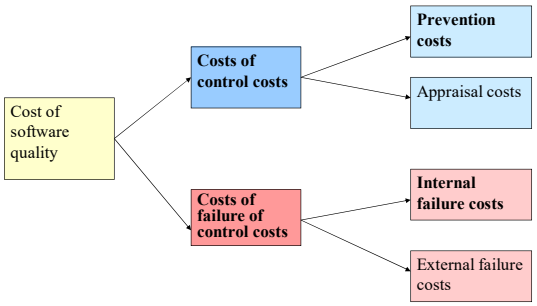
13

Economics of Software Quality

- You must put a **quality** product into test before you can expect to get out one, otherwise rework will be very high.
- Software quality largely concerned with defect prevention and appraisal.
- Finding a requirement error in test can be very expensive. Finding a code error during code review will generally cost much less.
- Recent studies have supported the importance of better processes and increased investment in the early stages of SDLC to improve the quality of software and the productivity of development.
- Experiences in manufacturing fields relating to the cost and ROI of quality improvements suggest that there are diminishing returns to quality expenditures after a certain point.

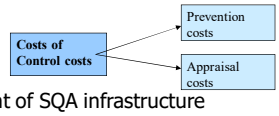
14

The classic model of cost of software quality



16

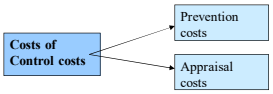
Prevention costs



- Investments in development of SQA infrastructure components
  - Procedures and work instructions
  - Support devices: templates, checklists etc
  - Software configuration management system
  - Software quality metrics
- Regular implementation of SQA preventive activities:
  - Instruction of new employees in SQA subjects
  - Certification of employees
  - Consultations on SQA issues to team leaders and others
- Control of the SQA system through performance of:
  - Internal quality reviews
  - External quality audits
  - Management quality reviews

17

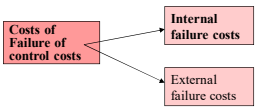
Appraisal costs



- Costs of reviews:
  - Formal design reviews (DRs)
  - Peer reviews (inspections and walkthroughs)
  - Expert reviews
- Costs of software testing:
  - Unit, integration and software system tests
  - Acceptance tests (carried out by customers)
- Costs of assuring quality of external participants

18

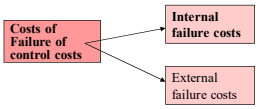
Internal failure costs



- Costs of redesign or design corrections subsequent to design review and test findings
- Costs of re-programming or correcting programs in response to test findings
- Costs of repeated design review and re- testing (regression tests)

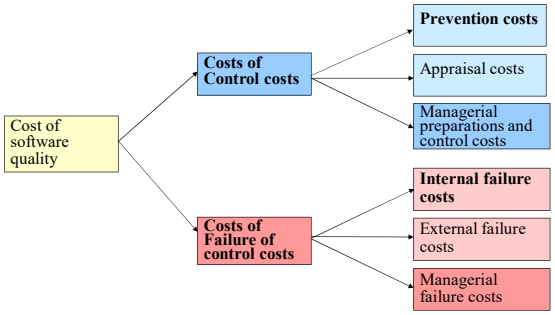
19

External failure costs



- Typical external failure costs cover:
  - Resolution of customer complaints during the warranty period.
  - Correction of software bugs detected during regular operation.
  - Correction of software failures after the warranty period is over even if the correction is not covered by the warranty.
  - Damages paid to customers in case of a severe software failure.
  - Reimbursement of customer's purchase costs.
  - Insurance against customer's claims.
- Typical examples of hidden external failure costs:
  - Reduction of sales to customers that suffered from software failures.
  - Severe reduction of sales motivated by the firm's damaged reputation.
  - Increased investment in sales promotion to counter the effects of past software failures.
  - Reduced prospects to win a tender or, alternatively, the need to under-price to prevent competitors from winning tenders.

Galin's extended model for cost of software quality



21

Managerial preparation and control costs

- Costs of carrying out contract reviews
- Costs of preparing project plans, including quality plans
- Costs of periodic updating of project and quality plans
- Costs of performing regular progress control
- Costs of performing regular progress control of external participants' contributions to projects

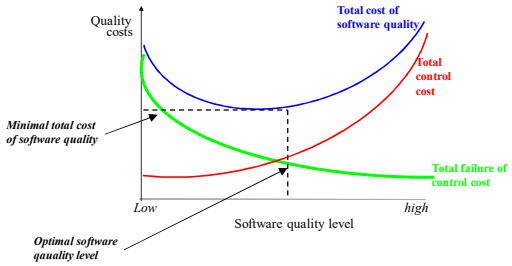
22

Managerial failure costs

- Unplanned costs for professional and other resources, resulting from underestimation of the resources in the proposals stage.
- Damages paid to customers as compensation for late project completion, a result of the unrealistic schedule in the Company's proposal.
- Damages paid to customers as compensation for late completion of the project, a result of management's failure to recruit team members.
- Domino effect: Damages to other projects planned to be performed by the same teams involved in the delayed projects. The domino effect may induce considerable hidden external failure costs.

23

Cost of software quality balance by quality level



24

Cost of software quality metrics - Objectives

- In general – it enables management to achieve **economic control** over SQA activities and outcomes. The specific objectives are:
  - Control organization-initiated costs to prevent and detect software errors.
  - Evaluation of the economic damages of software failures as a basis for revising the SQA budget.
  - Evaluation of plans to increase or decrease of SQA activities or to invest in SQA infrastructure on the basis of past economic performance.

25

Cost metrics for evaluating SQA systems - examples

- Percentage of cost of software quality out of total software development costs.
- Percentage of software failure costs out of total software development costs.
- Percentage of cost of software quality out of total software maintenance costs.
- Percentage of cost of software quality out of total sales of software products and software maintenance.

26

Problems in the application of cost of software quality metrics

- General problems
  - Inaccurate and/or incomplete identification and classification of quality costs.
  - Negligent reporting by team members
  - Biased reporting of software costs, especially of "censored" internal and external costs.
  - Biased recording of external failure costs - "camouflaged" compensation of customers for failures.
- Problems arising when collecting data on managerial costs:
  - Contract review and progress control activities are performed in a "part-time mode". The reporting of time invested is usually inaccurate and often neglected.
  - Many participants in these activities are senior staff members who are not required to report use of their time resources.
  - Difficulties in determination of responsibility for schedule failures.
  - Payment of overt and formal compensation usually occurs quite some time after the project is completed, and much too late for efficient application of the lessons learned.

27

Software measurement

29

Why Measurement?

When you can **measure** what you are speaking about, and express it into numbers, you **know** something about it;

But when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind:

It may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the stage of science.

Lord Kelvin (1824-1904) - measurement

Software measurement

- Software measurement is concerned with deriving a numeric value for an **attribute of a software product or process**.
- This allows for **objective comparisons** between techniques and processes.
- Although some companies have introduced measurement programmes, most organisations still don't make **systematic** use of software measurement.
- There are few established standards in this area.

31

Software metric

- Any type of measurement which relates to a software system, process or related documentation
  - Lines of code in a program, the Fog index, number of person-days required to develop a component.
- Allow the software and the software process to be quantified.
- May be used to predict product attributes or to control the software process.
- Product metrics can be used for general predictions or to identify anomalous components.

32

Problems with measurement in industry

- It is difficult / impossible to quantify the return on investment of introducing an organizational metrics program.
- There are no standards for software metrics or standardized processes for measurement and analysis.
- In many companies, software processes are not standardized and are poorly defined and controlled.
- Most work on software measurement has focused on code-based metrics and plan-driven development processes. However, more and more software is now developed by configuring ERP systems or COTS.
- Introducing measurement adds additional overhead to processes.

33

Measurement is Difficult

- Measurement process is difficult to define.
  - Measuring colors, intelligence is difficult.
  - Measurement accuracy, margin of errors.
  - Measurement units, scales.
  - Drawing conclusions from measurements is difficult.
- However: the correct interpretation of appropriate metrics leads to increased visibility, understanding, control.

Measurement is Difficult

- Measurement process is difficult to define.
  - Measuring colors, intelligence is difficult.
  - Measurement accuracy, margin of errors.
  - Measurement units, scales.
  - Drawing conclusions from measurements is difficult.
- However: the correct interpretation of appropriate metrics leads to increased visibility, understanding, control.
- Measurement = *Direct* quantification of an attribute.
- Calculation = *Indirect*, a combination of measurements used to understand some attribute. (E.g. Overall scores in decathlon).

Why measure? The differing perspectives

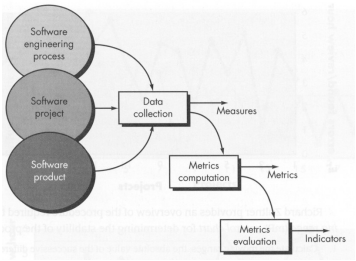
- There are as many perspectives as there are stakeholders
- Developers perspective
  - Completeness of requirements, quality of design, testing readiness.
- Managers perspective.
  - Delivery readiness, budget and scheduling issues.
- Customers perspective.
  - Compliance with requirements, quality.

Measurement in Software Engineering

- Applicable to all activities: managing, costing, planning, modeling, analyzing, specifying, designing, implementing, verifying, validating, maintaining.
  - Engineering implies understanding and control.
  - Computer science provides theoretical foundations for building software, software engineering focuses on controlled and scientifically sound implementation process.
- Considered somewhat a luxury?!?
  - Weakly defined targets: "Product will be user-friendly, reliable, maintainable".
    - Gilb's Principle of Fuzzy Targets: "Projects without clear goals will not achieve their goals clearly."
  - Estimation of costs.
    - Cost of design, cost of testing, cost of coding…
  - Predicting product quality.
  - Considering technology impacts.

Integrating Metrics with Software Process

- Many software developers do not collect measures
- Without measurement it is impossible to tell whether a process is improving or not
- Baseline metrics data should be collected from a large, representative sampling of past software projects
- Getting this historic project data is very difficult, if the previous developers did not collect data in an on-going manner



Software Measurement Objectives

- Managers angle.
  - Cost: Measure **time** and **effort** of various processes (elicitation, design, coding, test).
  - Staff **productivity**: Measure staff time, size of artifacts. Use these for predicting the impact of a change.
  - Product **quality**: Record faults, failures, changes as they occur. Cross compare different projects.
  - **User satisfaction**: Response time, functionality.
  - **Potential for improvement**.
- Engineers angle.
  - Are **requirements testable**?
    - Instead of requiring "reliable operation", state the expected mean time to failure.
  - Have all major **faults** been **found**?
    - Use models of expected detection rates.
  - **Meeting** product or process **goals**.
    - <20 failures per beta-test site in a month.
    - No module contains more than x lines (standards).
  - What the **future** holds?
    - Predict product size from specification size.

Process Measurement

- Is when we **measure the efficacy of a software process indirectly**.
- That is, we derive a set of metrics based on process outcomes.
  - Outcomes include
    - measures of errors uncovered before release of the software
    - defects delivered to and reported by end-users
    - work products delivered (productivity)
    - human effort expended
    - calendar time expended
    - schedule conformance
    - other measures.
- We also derive process metrics by measuring the characteristics of specific software engineering tasks.
- Process Metrics
  - majority focus on quality achieved as a consequence of a repeatable or managed process
  - statistical SQA data: error categorization & analysis
  - defect removal efficiency: propagation from phase to phase
  - reuse data

Project Metrics

- Effort / time per SE task
- Errors uncovered per review hour
- Scheduled vs. actual milestone dates
- Changes (number) and their characteristics
- Distribution of effort on SE tasks

Product & Project Metrics

- Focus on the quality of deliverables
- measures of analysis model
- complexity of the design
  - internal algorithmic complexity
  - architectural complexity
  - data flow complexity
- code measures (e.g., Halstead, McCabe)
- measures of process effectiveness
  - e.g., defect removal efficiency

- Typical Project Metrics
  - Effort/time per software engineering task
  - Errors uncovered per review hour
  - Scheduled vs. actual milestone dates
  - Changes (number) and their characteristics
  - Distribution of effort on software engineering tasks

Product metrics

- A quality metric should be a predictor of product quality.
- Classes of product metric
  - Dynamic metrics which are collected by measurements made of a program in execution;
  - Static metrics which are collected by measurements made of the system representations;
  - Dynamic metrics help assess efficiency and reliability
  - Static metrics help assess complexity, understandability and maintainability.

45

Software Measurement

- Measurements
  - Direct - Cost and effort applied
    - LOC - lines of code
    - execution speed
    - defects reported over time period
  - Indirect - functionality, quality, complexity, efficiency, reliability
- Normalization for Metrics
- Normalized data are used to evaluate the process and the product
  - Size-oriented normalization
    - line-of-code approach
  - Function-oriented normalization
    - function point approach

- Size-Oriented Metrics
  - errors per KLOC (thousand lines of code)
  - defects per KLOC
  - € per LOC
  - page of documentation per KLOC
  - errors per person-month
  - LOC per person-month
  - € per page of documentation
- Function-Oriented Metrics - Function Points
  - errors per FP
  - defects per FP
  - € per FP
  - pages of documentation per FP
  - FP per person-month

Program size

- Many programmers count lines of code (LOC)
  - Not as simple metric as it may sound
  - Assumes linear relationship between system size and volume of documentation
- Is used in many metrics, for example in defect rate: defects per KLOC (thousand lines of code) or LOC inspected
- Many problems
  - The ambiguity of the counting – meaning is not the same with Assembler or high-level languages
  - What to count? Blank lines, comments, data definitions, only executable lines..?
    - SLOC, NCSLOC…
  - Problematic for productivity studies: the amount of LOC is negatively correlated with design efficiency

Measuring programming progress by lines of code is like measuring aircraft building progress by weight Bill Gates

Lines of code (LOC)

- LOC is wrongly used as a measure of software complexity.
- This metric is just as good as source listing weight if we assume consistency w.r.t. paper and font size.
- Makes as much sense (or nonsense) to say:
  - "This is a 2 kilo program" as it is to say "This is a 100,000 line program"

Hello World – C Vs COBOL

```
#include <stdio.h>

int main(void) {

  printf("Hello World");
  return 0;
}
```

Lines of code: 5
(excluding whitespace)

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. HELLOWORLD.
000300
000400*
000500 ENVIRONMENT DIVISION.
000600 CONFIGURATION SECTION.
000700 SOURCE-COMPUTER. RM-COBOL.
000800 OBJECT-COMPUTER. RM-COBOL.
000900
001000 DATA DIVISION.
001100 FILE SECTION.
001200
100000 PROCEDURE DIVISION.
100100
100200 MAIN-LOGIC SECTION.
100300 BEGIN.
100400     DISPLAY " " LINE 1 POSITION 1 ERASE EOS.
100500     DISPLAY "Hello world!" LINE 15 POSITION 10.
100600     STOP RUN.
100700 MAIN-LOGIC-EXIT.
100800     EXIT.
```

Lines of code: 17
(excluding whitespace)

Function points

- Function Points measure software size by quantifying the functionality provided to the user **based solely on logical design and functional specifications**
- Has gained acceptance in terms of productivity (for example FP/year) and quality (defects/FP)
- The IFPUG counting practices committee (http://www.ifpug.org) is the de facto standard for counting methods

Function Point Analysis (FPA)

- Number of function points - a measure of the development resources (human resources) required to develop a program, based on the functionality specified for the software system.
- AKA - Albrecht Function Point Analysis
  - Developed by Allan Albrecht in IBM 1979
  - He wanted to develop some way to estimate the functional size of programs, independently of the programming languages they were developed in
- Based on a combination of program characteristics
  - external inputs and outputs
  - user interactions
  - external interfaces
  - files used by the system
- A weight is associated with each of these
- The function point count is computed by multiplying each raw count by the weight and summing all values
- FPA with agile software development?

Sample function points calculation form

Software system components	Complexity level									Total FP
	Simple			average			complex			
	Count	Weight Factor	Points	Count	Weight Factor	Points	Count	Weight Factor	Points	
	A	B	C= AxB	D	E	F= DxE	G	H	I= GxH	
User inputs		3			4			6		
User outputs		4			5			7		
User online queries		3			4			6		
Logical files		7			10			15		
External interfaces		5			7			10		
Total FP										

Lines of code paradox

- **Paradox:**
  - If you unroll a loop, you reduce the complexity of your software …
- Studies show that there is a linear relationship between LOC and error rates for small programs (*i.e.,* LOC < 100).
- The relationship becomes non-linear as programs increases in size.
- So… program complexity is an issue

The Issue Is…

- Is software quality metrics all about size?
- Size does matter, but…
- There is a classification of software quality metrics where size is only one small aspect

Examples

Product metrics

- External Product Attributes
  - Reliability, maintainability, understandability (of documentation), usability, integrity, efficiency, reusability, portability, interoperability…
- Internal Product Attributes
  - Size, modularity, complexity…
- Some example metrics…
  - Error severity metrics (e.g avg. severity of code errors)
  - Timetable metrics (e.g. timetable observance)
  - Error removal effectiveness metrics (e.g. development errors removal effectiveness)
  - Process productivity metrics (e.g. development productivity)

Process metrics – Error Density Metrics

NCE = The number of code errors detected by code inspections and testing.

NDE = total number of development (design and code) errors detected in the development process.

Code	Name	Calculation formula
CED	**Code Error Density**	$CED = \frac{NCE}{KLOC}$
DED	**Development Error Density**	$DED = \frac{NDE}{KLOC}$

Product metrics categories

- HD quality metrics:
  - HD calls density metrics - measured by the number of calls.
  - HD calls severity metrics - the severity of the HD issues raised.
  - HD success metrics – the level of success in responding to HD calls.
- HD productivity metrics.
- HD effectiveness metrics.
- Also have Corrective maintenance quality metrics & Corrective maintenance productivity and effectiveness metrics.

HD calls density metrics

Code	Name	Calculation Formula
HDD	**HD calls density**	$HDD = \frac{NHYC}{KLMC}$
WHDD	**Weighted HD calls density**	$WHYC = \frac{WHYC}{KLMC}$
WHDF	**Weighted HD calls per function point**	$WHDF = \frac{WHYC}{NMFP}$

NHYC = the number of HD calls during a year of service.
KLMC = Thousands of lines of maintained software code.
WHYC = weighted HD calls received during one year of service.
NMFP = number of function points to be maintained.

Key points

- Software measurement can be used to gather quantitative data about software and the software process.
- You may be able to use the values of the software metrics that are collected to make inferences about product and process quality.
- Product quality metrics are particularly useful for highlighting anomalous components that may have quality problems.
  - These components should then be analyzed in more detail.
- Software analytics is the automated analysis of large volumes of software product and process data to discover relationships that may provide insights for project managers and developers.

83