Universität Zürich[UZH]

Institut für Informatik

Software Quality

[selected chapters]

Lecture 1 - Introduction

Thomas Fritz

Martin Glinz

Overview

- About the course
- About me
- Introduction to Software Quality
- Product/Code Quality and Process Quality
- Mechanisms for Improving Software Quality
- Quality Management in Agile Processes

Learning Goals

By the end of this class you should be able to:

- Describe how to succeed in this course
- Describe aspects that affect software quality
- Explain the benefits of high software quality
- Explain why we can't sufficiently measure code quality with testing alone
- Describe mechanisms for quality control and assurance
- Given a method, write a test suite that provides statement, branch or path coverage
- Explain how process models, in particular the agile process model, support quality management

Software Quality – Prerequisites

- Basic knowledge of software quality from software engineering course
  - Kapitel 2: Ziele und Qualität
  - Kapitel 7: Validierung und Verifikation
  - Kapitel 8: Testen von Software
  - Kapitel 9: Reviews
  - Kapitel 10: Messen von Software
  - Kapitel 11: Statische Analyse
  - Kapitel 19: Software-Qualitätsmanagement
  - Kapitel 20: Bewertung und Verbesserung von Prozessen und Qualität
- This basic knowledge is assumed; all chapters are available at http://www.ifi.uzh.ch/rerg/courses/hs10/software_engineering

Software Quality – Tentative Schedule

20.02	Introduction to Software Quality, Product & Process Quality
27.02	Model Checking I + II, Assignment 1
05.03	Advanced Testing Practices, Debugging I
12.03	Debugging II, Discussion Assignment 1, Assignment 2
19.03	Design, Metrics, Smells, Refactoring and Code Reviews
26.03	Human Aspects in Software Quality (Paper Based), Discussion Assignment 2, Assignment 3
02.04	Continuous Integration, UI Tests
07.05	**Exam**

Keys to Success

- Attend lectures
- Know the facts
  - Listen ***actively*** in class
  - Take notes
  - Do the readings seriously
  - Strive to identify factual information
- Practice applying the facts
  - Assignments and in-class activities are good for this

Passing the Course

- Pass the assignments
  (in groups of two if stated)

- Pass the exam

People & Resources

- Lecturers
  - Thomas Fritz fritz@ifi.uzh.ch
  - Martin Glinz glinz@ifi.uzh.ch
- TA
  - Eya Ben Charrada charrada@ifi.uzh.ch
- Website(s)
  - http://seal.ifi.uzh.ch/softwarequality/
  - http://www.ifi.uzh.ch/rerg/courses/fs12/swq.html

About me…

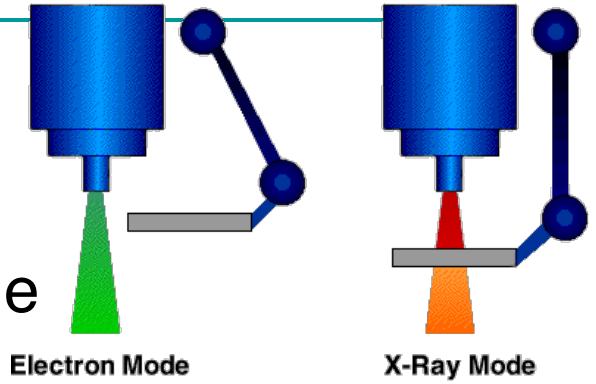
	undergrad	graduate
education	   ECOLE DES MINES DE NANTES	
experience	SIEMENS  debis Systemhaus	

About me…

- Since Nov 2011: Assistant Professor at IFI
- I'm available by appointment. Please just ask if you need extra time or privacy!
- Looking for students for bachelor and master thesis projects. Talk to me!

Software Quality

Therac-25



- Computerized radiation therapy machine
  - Shallow tissue: direct electron beam
  - Deeper tissue: electron beam converted into X-ray photons
- accidents occurred when high-energy electron-beam was activated without target having been rotated into place; machine's software did not detect this
- First case in 1984: lawsuit but manufacturer refused to believe in a malfunction of Therac-25
- Second case in 1985: display indicated “no dose” so operator repeated 5 times; patient died 3 months later
- Overall six accidents with ~100 times the intended does between 1985 and 1987; 3 patients died

See more at http://courses.cs.vt.edu/cs3604/lib/Therac_25/Therac_1.html

Therac-25: some reasons

- The design did not have any hardware interlocks to prevent the electron-beam from operating in its high-energy mode without the target in place.
- The engineer had reused software from older models. These models had hardware interlocks and were therefore not as vulnerable to the software defects.
- The hardware provided no way for the software to verify that sensors were working correctly.
- The equipment control task did not properly synchronize with the operator interface task, so that race conditions occurred if the operator changed the setup too quickly. This was evidently missed during testing, since it took some practice before operators were able to work quickly enough for the problem to occur.
- The software set a flag variable by incrementing it. Occasionally an arithmetic overflow occurred, causing the software to bypass safety checks.

taken from Stephen Dannelly

Therac-25

Many factors:

- Programming errors / race conditions
- No independent review of software
- Inadequate risk assessment together with overconfidence in software
- Therac-25 software and hardware combination never tested until assembled at the hospital
- poor human computer interaction design
- a lax culture of safety in the manufacturing organization
- management inadequacies and lack of procedures for following through on all reported incidents
- …

Patriot Missile System

- On February 25, 1991, the Patriot missile battery at Dharan, Saudi Arabia had been in operation for 100 hours, by which time the system's internal clock had drifted by one third of a second. For a target moving as fast as a Scud, this was equivalent to a position error of 600 meters.
- The radar system had successfully detected the Scud and predicted where to look for it next, but because of the time error, looked in the wrong part of the sky and found no missile. With no missile, the initial detection was assumed to be a spurious track and the missile was removed from the system. No interception was attempted, and the missile impacted on a barracks killing 28 American soldiers.

taken from Stephen Dannelly; see more at
http://www.fas.org/spp/starwars/gao/im92026.htm

Software Quality Hazards

- Ariane 5
- Embedded software in cars
- …

Poor software quality has become one of the most expensive topics in human history > $150 billion per year in U.S and > $500 billion per year world wide

Improving software quality is a key topic!

What is Quality?

Industrial definition according to ISO 9000:2000

- Quality of something is the *degree* to which a set of *inherent characteristics* comply to a set of *requirements*.

- An *inherent characteristic* exists in something or is a permanent feature of something, while an *assigned characteristic* is a feature that is attributed or attached to something. (e.g., composition vs. price)

Quality is always relative to a set of requirements.

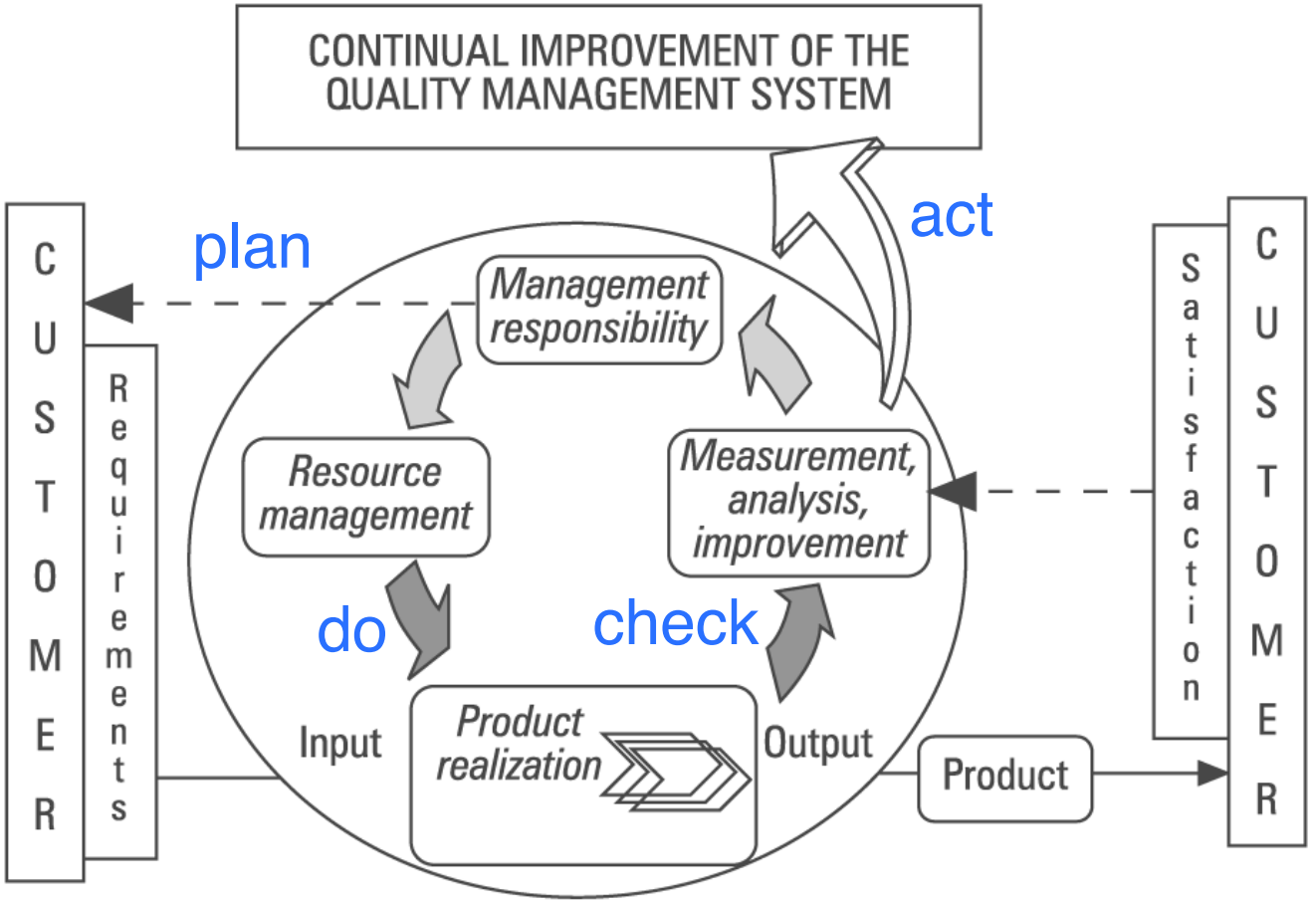
[ISO: International Organization for Standards]

How to manage quality?

- The ISO 9000 family of standards represents **an international consensus on good quality management practices**
- 8 basic principles of ISO 9000:2000 on Quality Management Systems
  - Customer focus
  - Leadership
  - Involvement of people
  - Process approach
  - System approach to management
  - Continual improvement
  - Factual approach to decision making
  - Mutually beneficial supplier relationships

http://www.iso.org/iso/iso_catalogue/management_and_leadership_standards/quality_management/qmp.htm

ISO 9000 – Process-Based Quality Management System



Quality Management Process

- Quality Planning
  - Setting quality objectives/requirements and specifying processes and resources to achieve them
- Quality Assurance
  - Set of activities to establish confidence that quality requirements will be met (prevention driven)
- Quality Control
  - Set of activities to ensure that quality requirements are met (inspection driven)
- Quality Improvement
  - Anything that enhances an organization's ability to meet requirements

http://praxiom.com/iso-definition.htm#Quality

QC vs QA

- ***Quality control*** activities are focused on the ***product*** itself.

- ***Quality assurance*** activities are focused on the ***process*** used to create the deliverable.

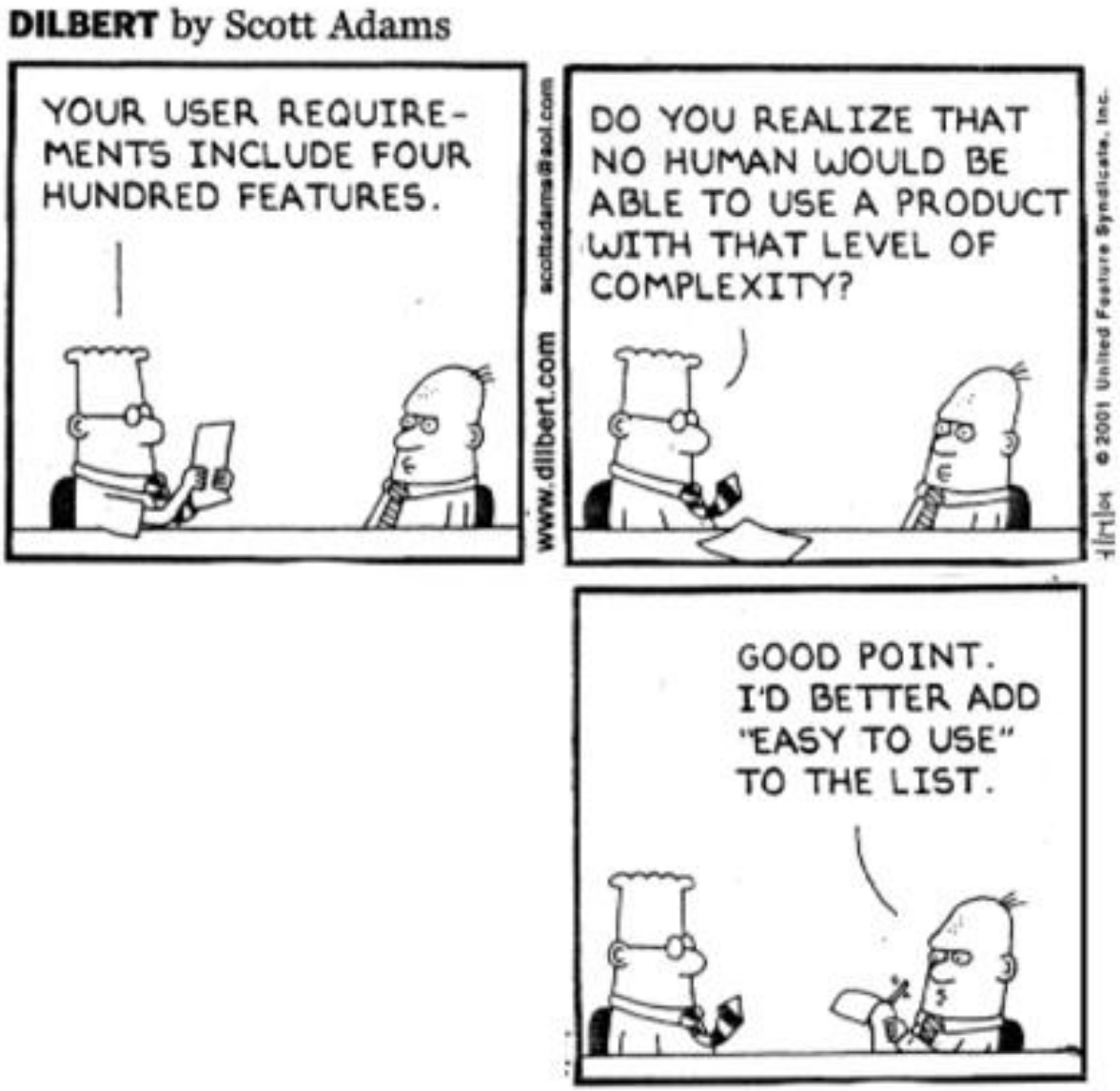
Quality Management Standard

- ISO 9001:2008 standard that provides **a set of standardized requirements for a quality management system**, regardless of what the user organization does, its size, or whether it is in the private, or public sector.
- It is the only standard in the family against which organizations can be certified – although **certification is not a compulsory requirement** of the standard.
- More than 1Mio companies are independently certified
- Better performance attributed to companies complying to the standard

What is Software Quality?

- According to IEEE
  - The degree to which a system, component or process meets the ***specified requirements***.
  - The degree to which a system, component or process meets the ***customer or user needs and expectations***.

[IEEE: Institute for Electrical and Electronics Engineers]

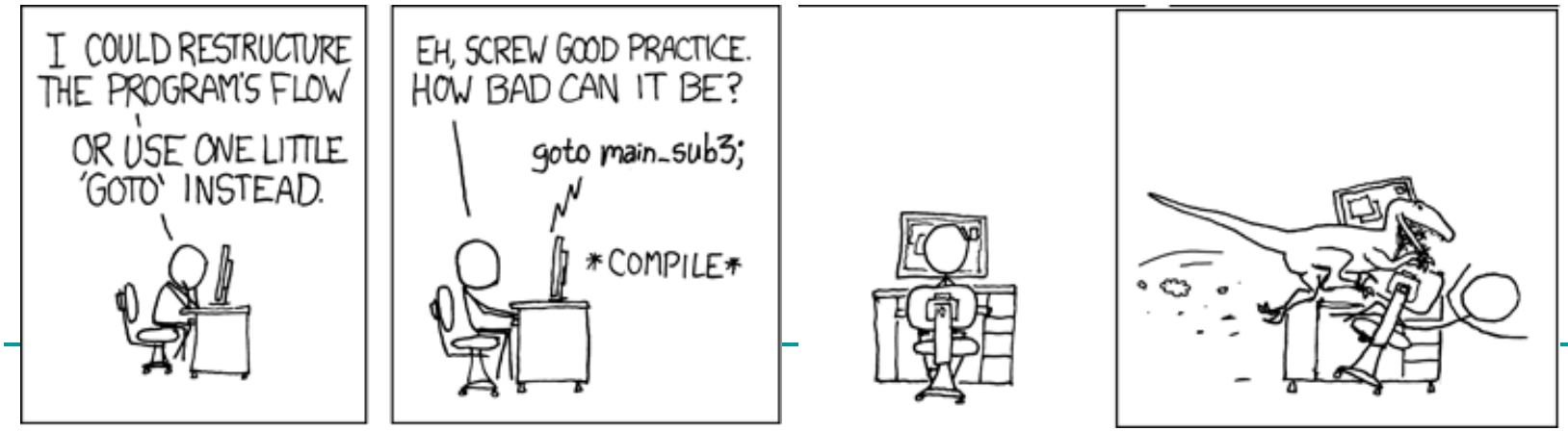

DILBERT by Scott Adams
YOUR USER REQUIREMENTS INCLUDE FOUR HUNDRED FEATURES.
DO YOU REALIZE THAT NO HUMAN WOULD BE ABLE TO USE A PRODUCT WITH THAT LEVEL OF COMPLEXITY?
GOOD POINT. I'D BETTER ADD "EASY TO USE" TO THE LIST.
www.dilbert.com scottadams@aol.com
© 2001 United Feature Syndicate, Inc.

What is Software Quality?

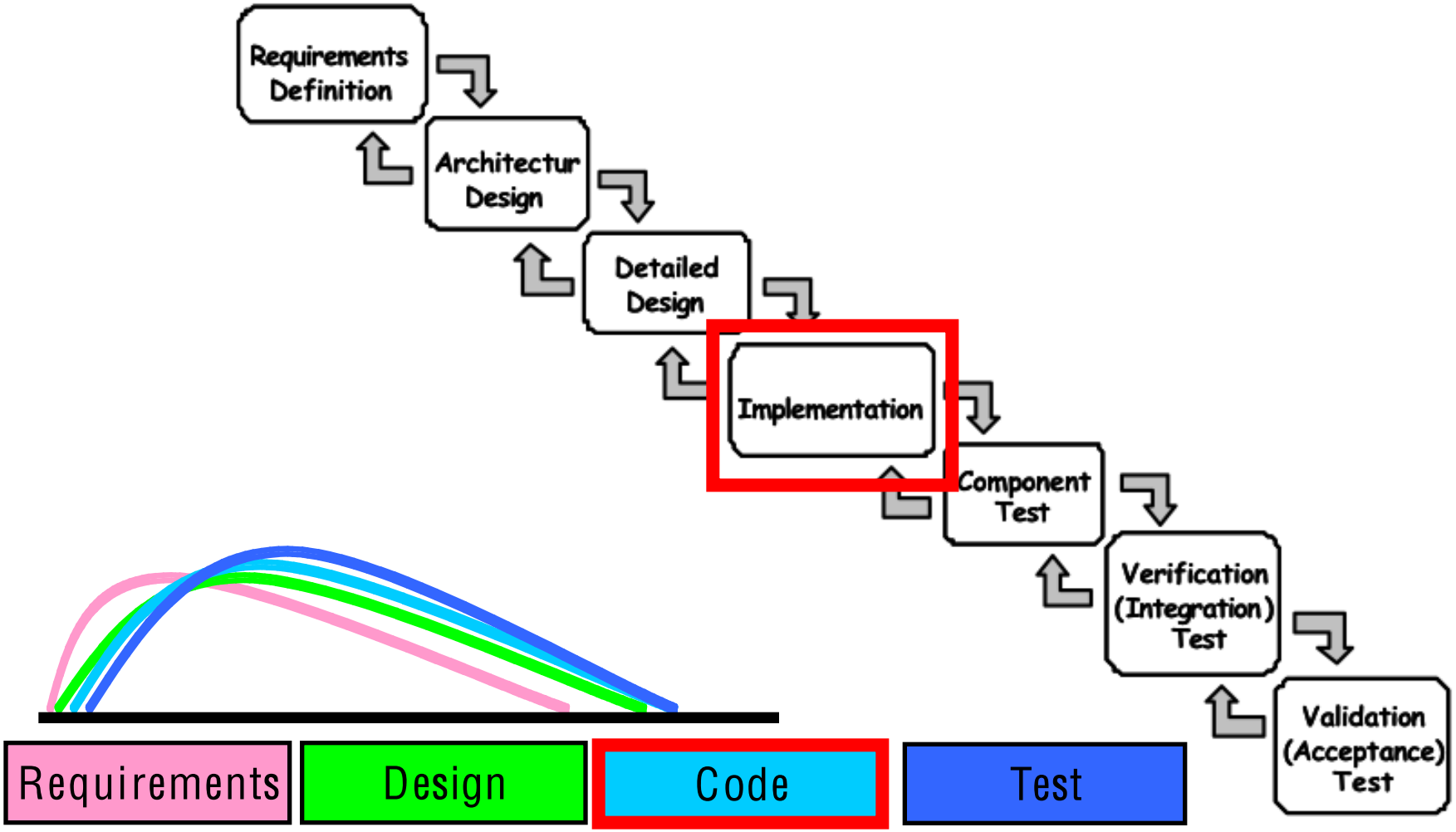
- According to Roger Pressman
  - Conformance to explicitly stated functional and performance ***requirements***, explicitly documented ***development standards***, and ***implicit characteristics*** that are expected of all professionally developed software.

Class Activity

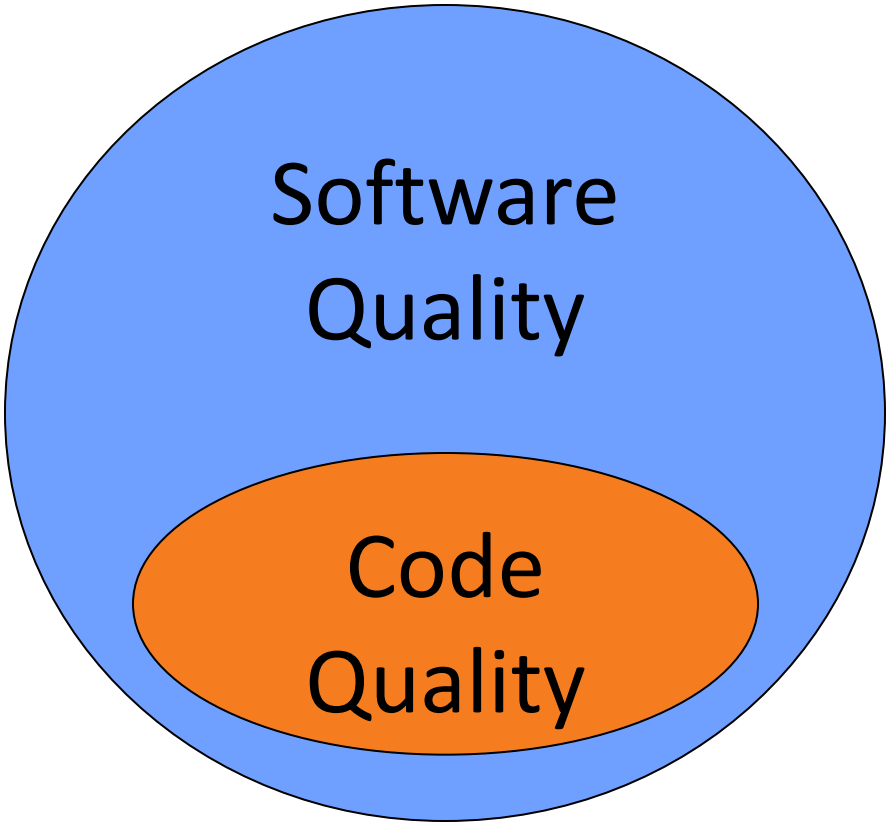
- Individually, on paper:
  - What are the possible consequences of poor software quality?
  - How can one assure/manage/improve/control software quality?
  - Write down 2 scenarios.
  - I'll give you one…



Focus often on Code Quality



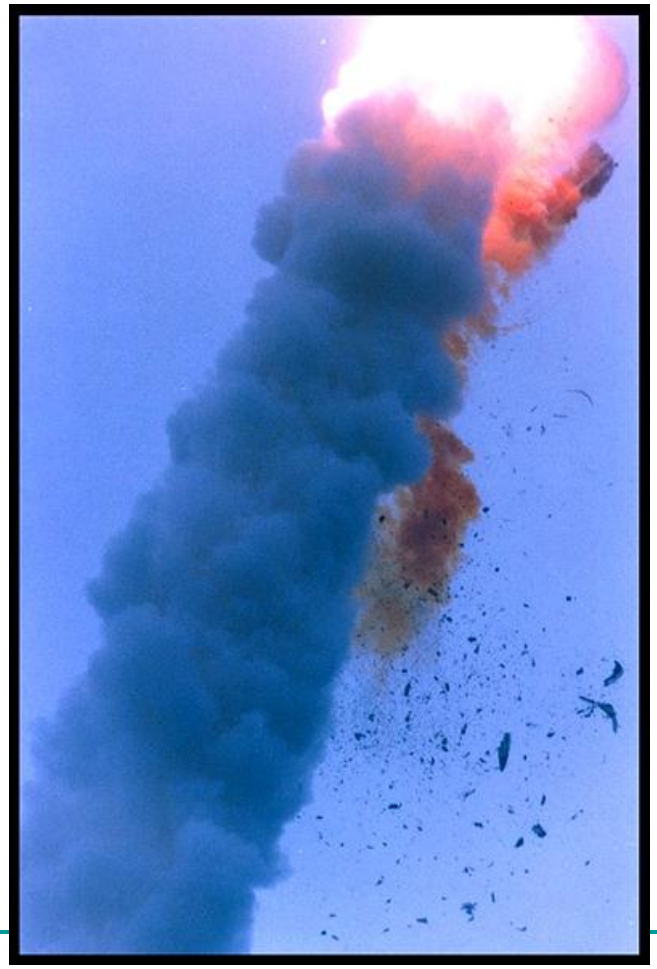
Not the only element of Software Quality



Other elements of Software Quality

- Faulty definition of requirements
- Client-developer communication failures
- Deliberate deviations from software requirements
- Logical design errors
- Shortcomings of the testing process
- Procedure errors
- Time managements problems
- …

Code Quality is Important



- A single bug can cause disastrous outcomes
- The Ariane 5 rocket's guidance computer software just threw an unchecked exception.
- Unmanned flight. Nobody was killed, but somebody probably lost his/her job…

An Example

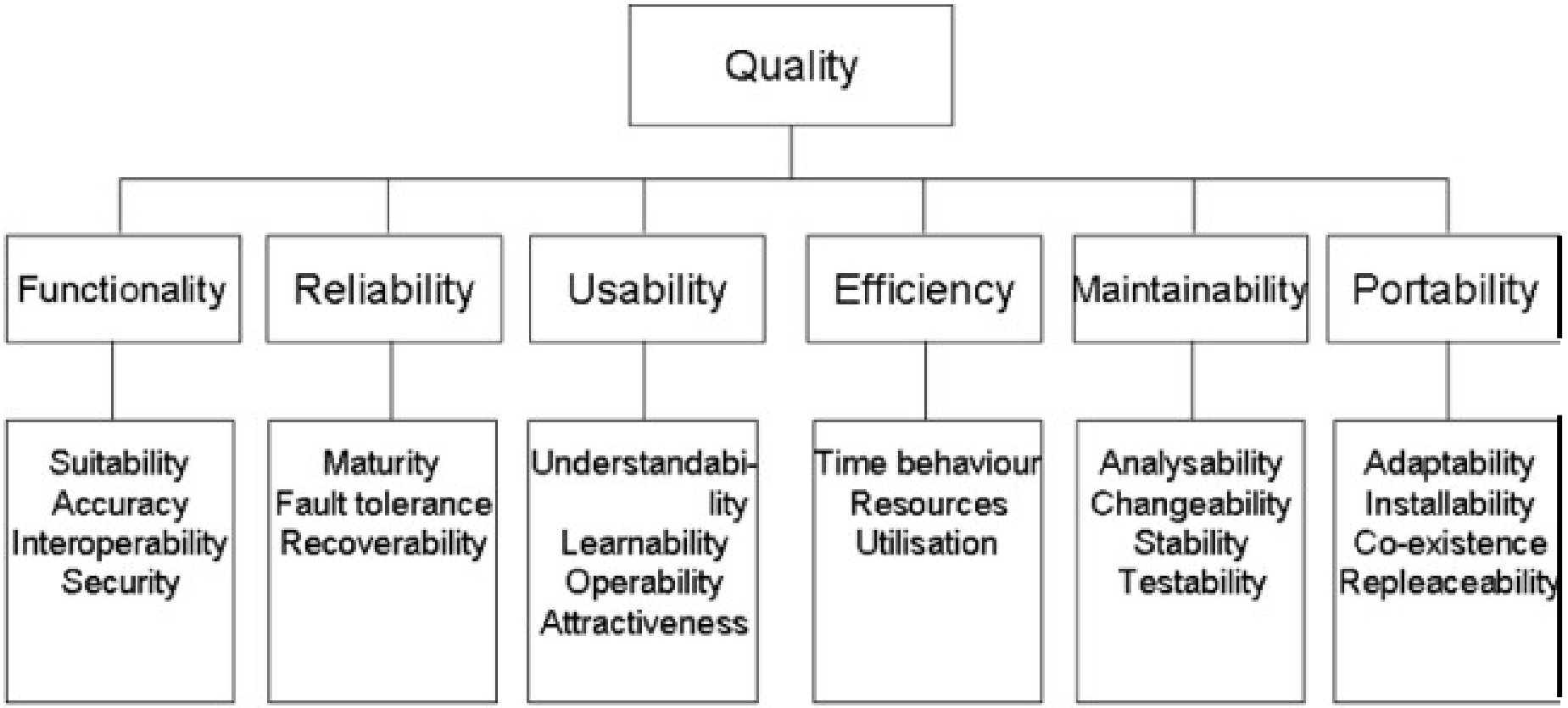
- Is there anything wrong with this code?

```
char b[2][10000],*s,*t=b,*d,*e=b+1,**p;main(int c,char**v)
{int n=atoi(v[1]);strcpy(b,v[2]);while(n--){for(s=t,d=e;*s;
s++){for(p=v+3;*p;p++)if(**p==*s){strcpy(d,*p+2);d+=strlen(
d); goto x;}*d++=*s;x:}s=t;t=e;e=s;*d++=0;}puts(t);}
```

Software Quality Attributes/Factors

- Functionality: the ability of the system to do the work for which it was intended.
- Maintainability
- Security
- Usability
- Modifiability
- Reusability
- Robustness
- Understandability
- ….

ISO/IEC 9126


Quality
Functionality
Reliability
Usability
Efficiency
Maintainability
Portability
Suitability
Accuracy
Interoperability
Security
Maturity
Fault tolerance
Recoverability
Understandabi-
lity
Learnability
Operability
Attractiveness
Time behaviour
Resources
Utilisation
Analysability
Changeability
Stability
Testability
Adaptability
Installability
Co-existence
Repleaceability

Process Quality influences Code Quality

- Software is never perfect
  - We can test it, but…
  - We cannot ensure it is free of defects
- So we need ways to assess the quality
  - Unfortunately, ***automatic methods*** to check code quality are often ***impossible*** (*Halting Problem*)
  - We are forced to link ***code quality*** to the quality of the overall ***code-writing process***.
    - *i.e.* The Capability Maturity Model (CMM) assesses the quality of a team/organization through their process.
- We will therefore talk about the ***process*** (and the mechanisms) more than the ***code***
- High code/product quality is still the final goal

Ensure High Software Quality

Things we can do to ensure we produce a high quality (low defect) product.

Product:	Add quality through removing bugs (**testing**), Prove correctness of program (verification techniques)
Process:	Build in quality from start (documenting, quality audits, reviews, training, …)

Revisiting Testing & Coverage

- Everyone (developers, managers,…) should care about testing!
- Industry averages
  - 30-85 errors per 1000 lines of code
  - 0.5-3 errors per 1000 lines of code in product (ie, not discovered before the product is delivered)

How to Test?

- Different scope
  - Classes, Subsystems, System
- Different purposes
  - No regression, suits the user
- Different tactics
  - Black box, white box

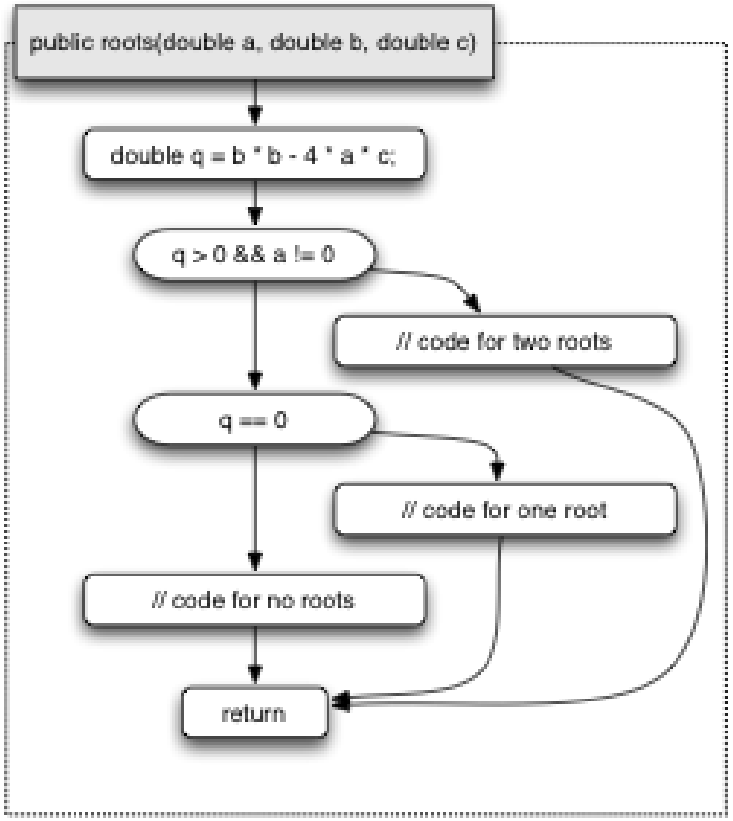
- Lot's of ways to test!

Stopping Criteria (when to stop testing)

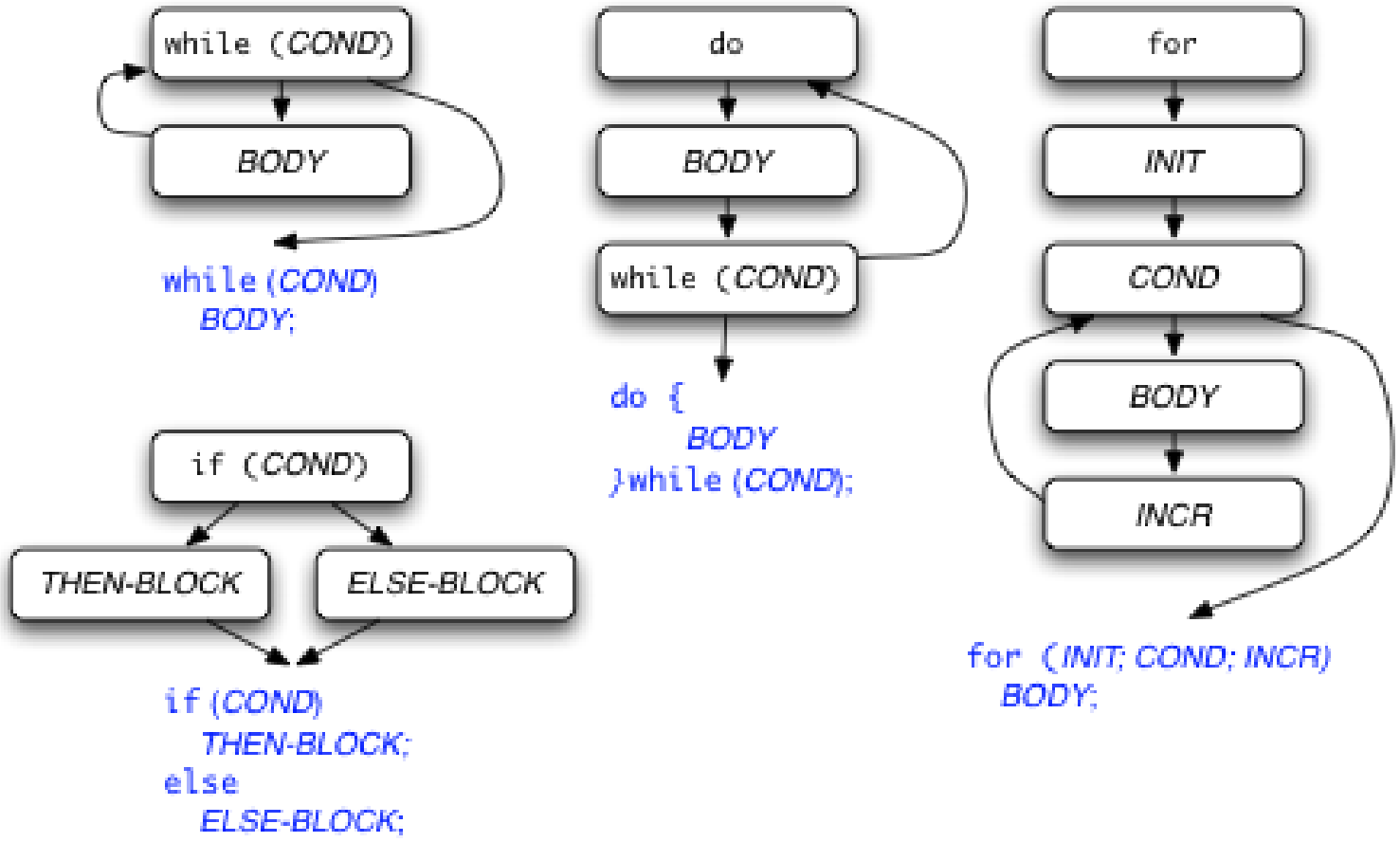
- Unit test for every method?
- Number of tests?
- Equivalence Partitioning?
- Boundary Tests?
- Coverage?
  - Each Statement?
  - Each Branch?
  - Each Path?

What is Coverage?

- The more ***parts*** are executed, the higher the chance that a test uncovers a defect.
- ***Parts*** can be nodes, edges, paths, conditions…
- Each lead to a different definition of ***coverage***.

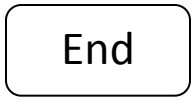


Control Flow Patterns

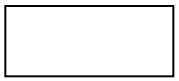


Representing Control Flow - Flowchart

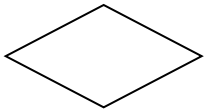
Indicates the start of the control-flow



Indicates the end of the control-flow



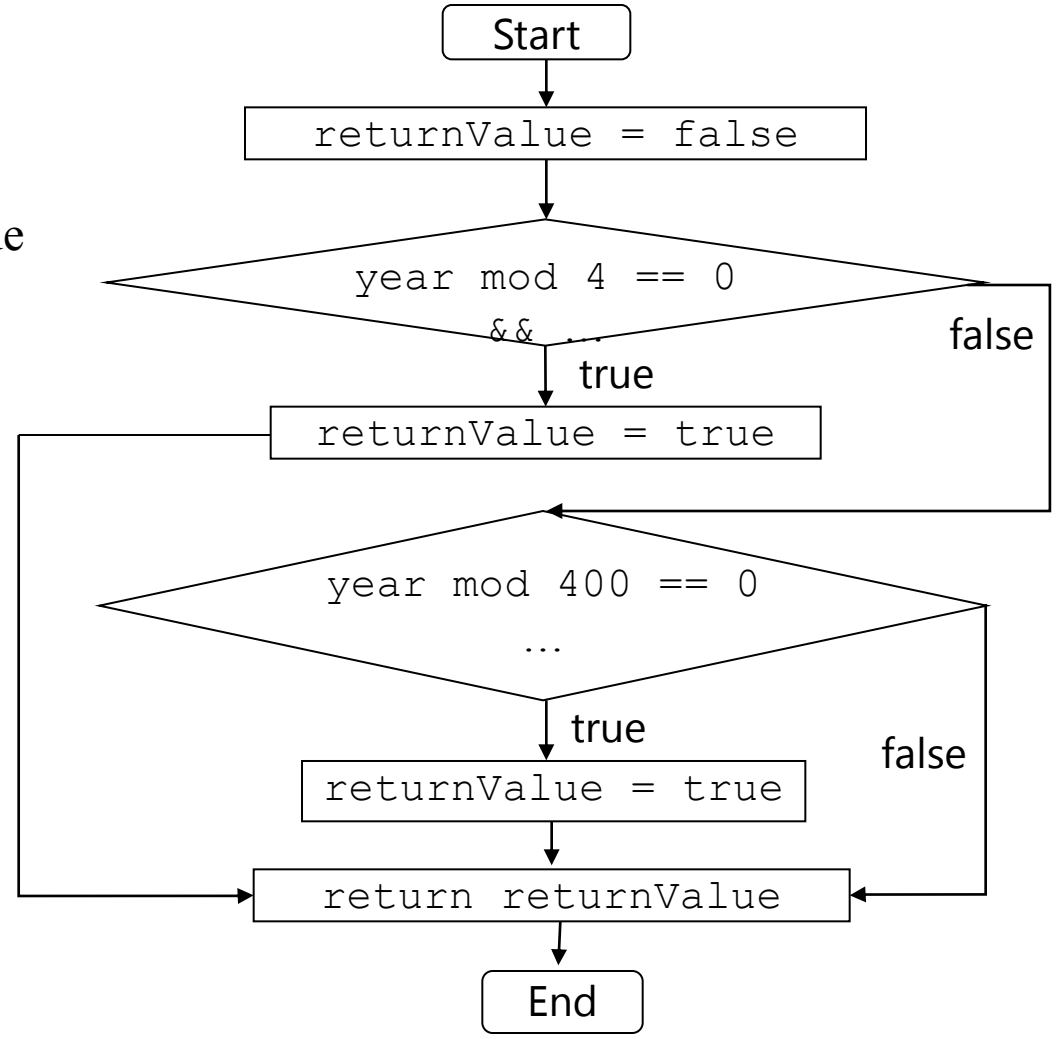
Indicates a processing step, text in the box is the code to execute



Indicates a conditional representing a yes/no question or true/false test. Two arrows emanate (one for yes/true and one for no/false) and must be labeled. The yes/true arrow typically comes out the bottom and the other out one of the sides.

Flowchart Example

```
boolean isALeapYear( int year ) {
// Declare a variable for the return value
// of the function
   boolean returnValue = false;
   // If the year is divisable by 4 and
   // not by 100 it is a leap year
   if ( ( year mod 4 == 0 ) &&
           ( year mod 100 != 0 ) )
      returnValue = true;
   else if ( year mod 400 == 0 )
      returnValue = true;
    return returnValue;
}
```



Different Types of Coverage

- Statement
  - Each “line of code” is covered once
- Branch
  - Each “condition” is covered twice: true & false
- Path
  - Each possible path through the code is covered once

Class Activity

- In groups of two, on paper
  - Write three test suites for the following code
    1. Statement coverage (but not branch nor path)
    2. Branch coverage (but not path)
    3. Path coverage
  - A test suite is :
    - A set of tests
    - And associated expected results

Class Activity (continued)

```
1. int function(boolean a, boolean b){
2.     int x;
3.     if(a)
4.         x = 1;

5.     if(b)
6.         x = 2;
7.     else
8.         x = 3;
9.     return x;
10. }
```

How good is coverage?

- The adequacy of a coverage criterion can only be defined intuitively
- Path coverage quickly becomes infeasible
  - Try it on a *while* loop!
- Coverage as a stopping criterion is good
  - But *smart* testing is always better!

More Examples of Mechanisms for improving Code Quality

- Frequent demonstration of working software to stakeholders
- Pair programming
- Coding standard
- Code reviews (or requirements / design review)
- External auditing
- Automatic tools (static analysis, code coverage, IDEs, …)
- Refactoring
- Testing (integration, regression, acceptance, …); automatic testing; test-driven development
- Component reuse
- Team building activities
- Clear division of responsibilities within the team
- Realistic estimations and up-to-date scheduling
- …

Examples of Mechanisms for a Disaster

- Ignore what the customers say they want – the developers surely must know better.
- Put in all the features that could potentially ever be useful.
- ***Do not worry about quality aspects (and ignore the related practices) until the deadline approaches.***
- Do not waste time on design or documentation – after all, code is the most important thing and time is already too short to do all that needs to be done.

Human Aspect in Software Quality

Software is built by humans. Humans

- Need to be able to understand the systems
- Have a limit for cognitive load
- Need to be aware of relevant information
- Need to be able to find the necessary information
- Need to have enough knowledge to complete tasks
- Are affected by team and its distribution

QUESTION:

How can we help humans to build better software?
Do you know existing tools/approaches that can help?

Some of the Mechanisms for Quality Assurance (QA)

- Cultural mechanisms
  - Teamwork / Team-Building
  - Organizational Values
- Human mechanisms
  - Code Reviews
  - Refactoring
- Automatic mechanisms
  - Style checkers
  - Quality Metrics

Teamwork / Team Building

- "No matter what the problem is, it's always a people problem." - *Jerry Weinberg*
- Techniques
  - Ice-breaker
  - Personality test
  - Casual meetings
  - Inclusive teams
    - Open communication
    - Transparent decision making
  - Foosball?

Organizational Values

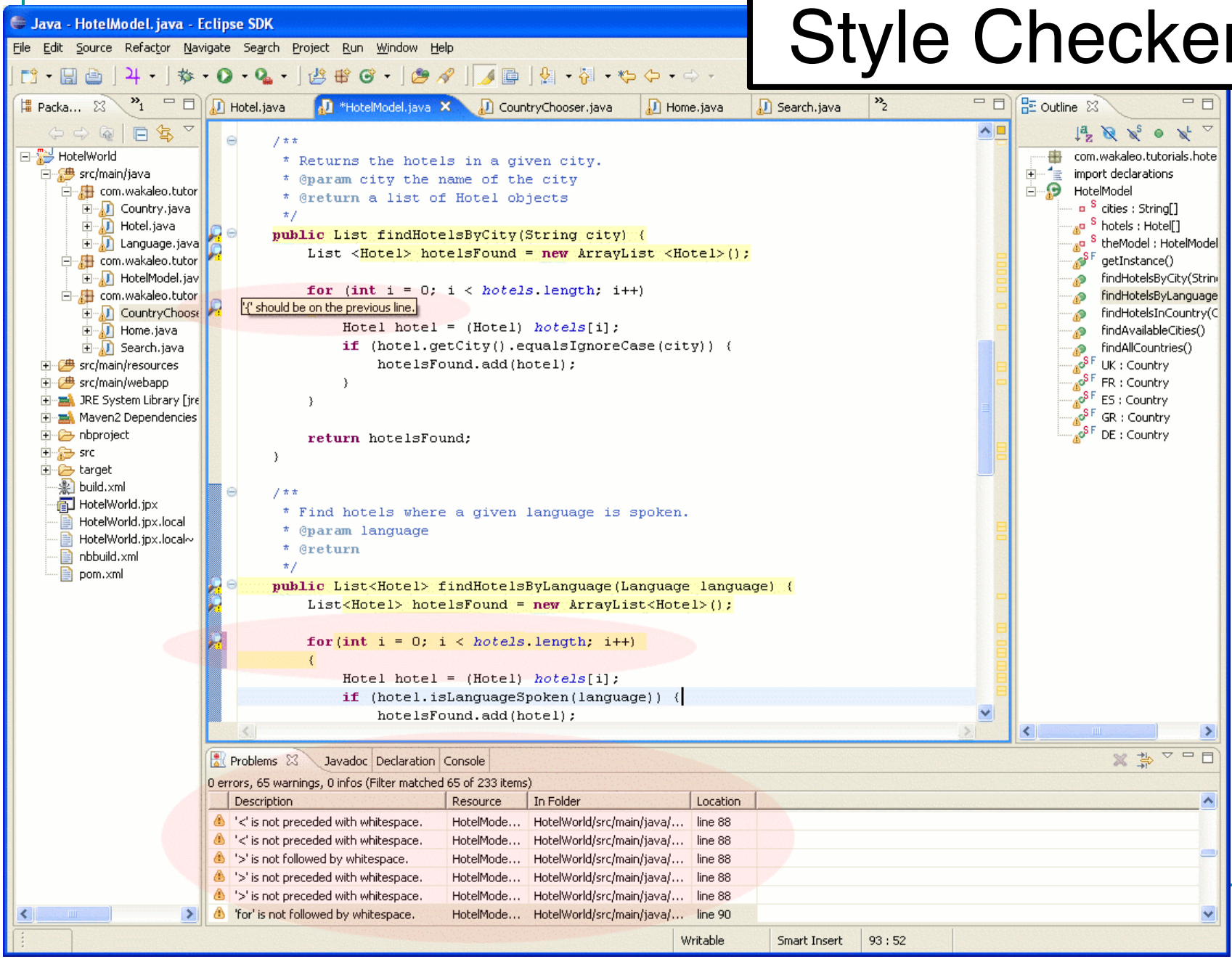
- "The structure of a computer program reflects the structure of the organization that built it."
  - Conway's Law
- Rigid hierarchical structure
  - Decisions are handed down, no ability to dispute
  - Less input into each decision, less motivation?
  - Less discussions could lead to faster decisions (although…)
- Flexible, collaborative, team-based structure
  - Better decisions through collaboration
  - Different people focus on different issues, cover all bases

Code Reviews

- Formal Inspection
  - Well defined, specific participant roles and responsibilities, documented review procedure, reporting of process…
- Less formal reviews
  - Tool-assisted code review
  - Ad-hoc review (over-the-shoulder)
  - Peer deskcheck / Email pass-around
  - Pair programming
  - etc.

See more at
http://www.atlassian.com/software/crucible/learn/codereviewwhitepaper.pdf

Style Checkers



Joel Test: 12 steps to better code

1. Do you use source control?
2. Can you make a build in one step?
3. Do you make daily builds?
4. Do you have a bug database?
5. Do you fix bugs before writing new code?
6. Do you have an up-to-date schedule?
7. Do you have a spec?
8. Do programmers have quiet working conditions?
9. Do you use the best tools money can buy?
10. Do you have testers?
11. Do new candidates write code during their interview?
12. Do you do hallway usability testing?

see http://www.joelonsoftware.com/articles/fog0000000043.html

CMMI- Software Process Improvement

- Capability Maturity Model Integration is an approach to guide organizations to improve their performance
- Various process areas are assessed and classified using maturity levels
- Note: there are other ones, e.g., SPICE / ISO 15504

CMMI – 22 Process Areas

CAR	Causal Analysis and Resolution
CM	Configuration Management
DAR	Decision Analysis and Resolution
IPM	Integrated Project Management
MA	Measurement and Analysis
OID	Organizational Innovation and Deployment
OPD	Organizational Process Definition
OPF	Organizational Process Focus
OPP	Organizational Process Performance
OT	Organizational Training
PI	Product Integration
PMC	Project Monitoring and Control
PP	Project Planning
PPQA	Process and Product Quality Assurance
QPM	Quantitative Project Management
RD	Requirements Development
REQM	Requirements Management
RSKM	Risk Management
SAM	Supplier Agreement Management
TS	Technical Solution
VAL	Validation
VER	Verification

CMMI – Maturity Levels

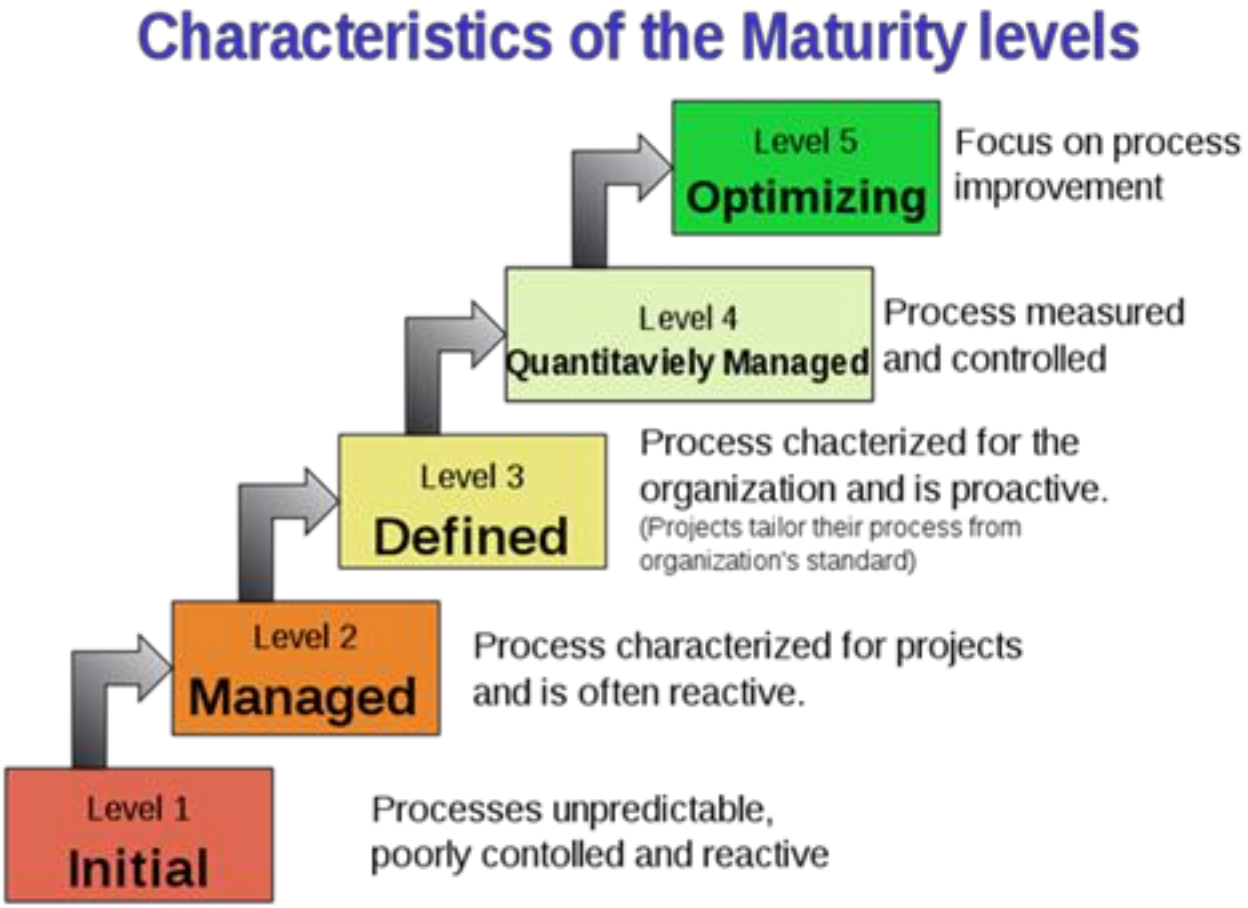


Image Source Wikipedia

Process Models and Quality

© Scott Adams, Inc./Dist. by UFS, Inc.

Reminder – Software Process

- A ***software process*** is a structured set of activities to develop a software system.
  - Defines who is doing what, when and how to reach a goal.
- Many different software processes, all include:
  - requirements elicitation & specification, design, implementation, integration, testing, …
- Goals of each activity
  - Mark out clear set of steps to perform, produce tangible item(s), allow for review of work, specify actions to perform in the next activity

Benefits of a software process

- provides an ***organizational tool***: activities cannot be forgotten
- provides a large-scale shared ***framework*** in which to work
- facilitates necessary ***communication***
- forces us to ***break down*** the problem
- provides a ***management tool***

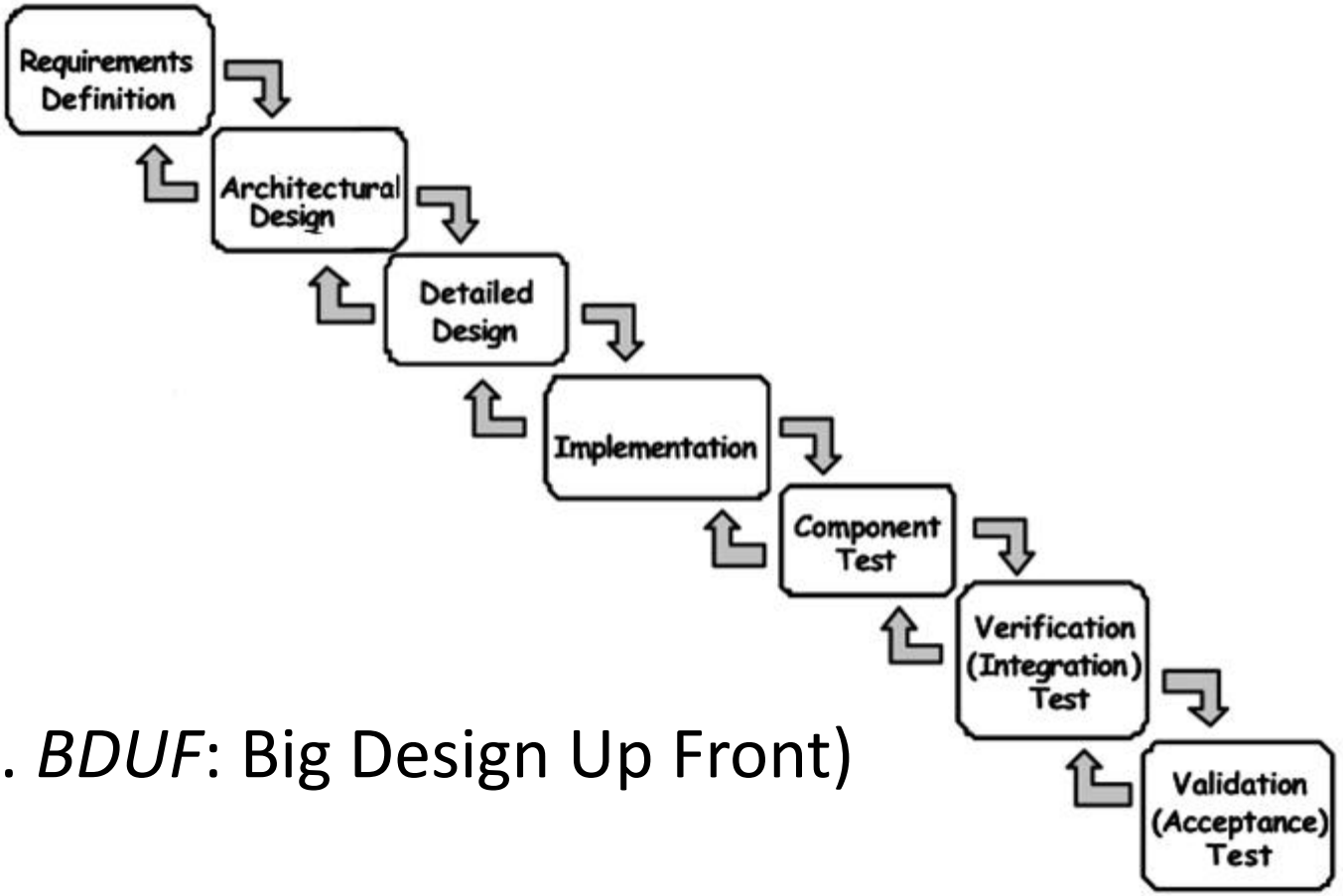
Software Process Models

A *software process model* is an abstract representation of a software process.

- Many different types; different models for different
  - types of software
  - types of companies
  - types of management
- One size does not fit all – mix of models often used in practice, tailored to environment, project, …

Waterfall Model

- separate and distinct phases of specification and development



(a.k.a. *BDUF*: Big Design Up Front)

Waterfall

- Good for well-understood but complex projects
  - Tackles all planning up front
  - No midstream changes = efficient process
- Provides support for an inexperienced team
  - Orderly, sequential, easy-to-follow model
  - Relatively slow progress
  - Reviews at each stage

Agile Models / Principles

The goal of agility: to develop software in the face of ***changing environment*** and ***constrained resources***

- Incremental and iterative
  - Development/delivery broken down into increments (parts of required functionality)
  - Requirements are prioritised and highest priority requirements are included in early increments.
- self-organizing cross-functional teams

More a **set of principles** than a fixed model; many variations of agile processes

What is Agility?

- Satisfy customer through ***early*** and ***continuous delivery***
- ***Customers***, ***developers*** and ***stakeholders*** work together daily
  - An agile process must be continually guided
- Build project around motivated ***individuals***
- High value on ***face-to-face conversations***
  - Primary mode of communication
  - Written documentation is not required

What is agility?

A sustainable process

- Teams work at a ***rate*** that can be ***maintained*** for the entire duration of the project

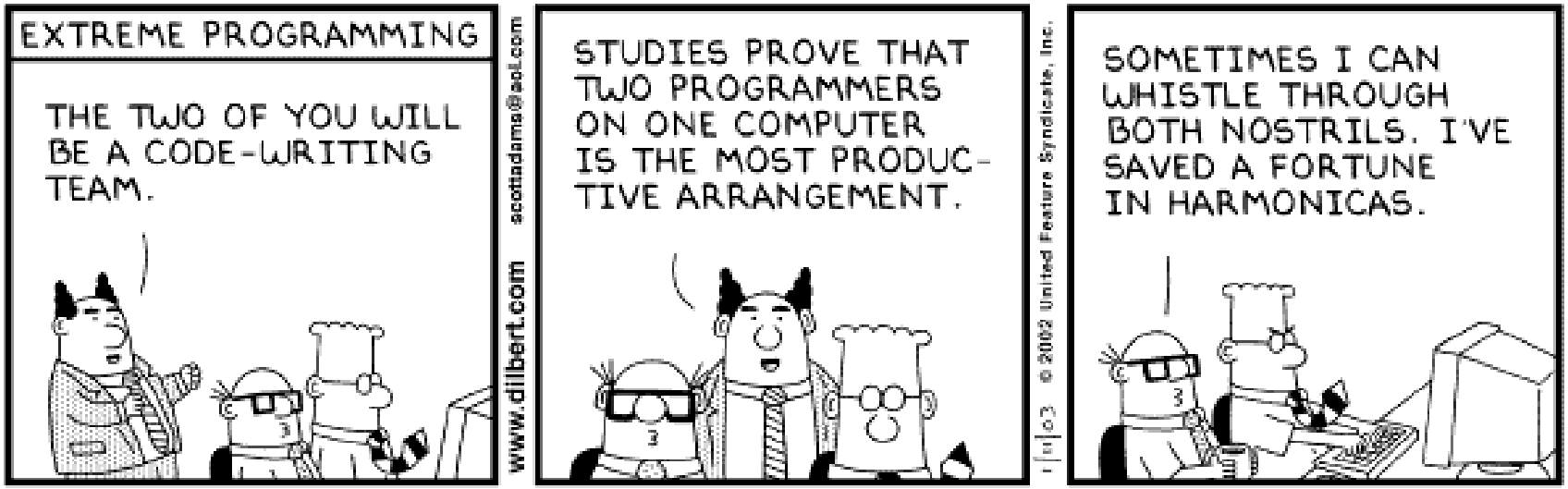
- Continuous attention to ***technical excellence***
  - High quality is the key to high speed
  - Don't hesitate to refactor

- ***Simplicity*** is essential
  - Take the simplest path that is consistent with the goals
  - Be confident that it is easy to change if needed

Extreme Programming (XP) – 12 core practices

- **Fine scale feedback**
  - Pair programming
  - Planning game
  - Test driven development
  - On-site customer / whole team
- **Continuous process**
  - Continuous integration
  - Refactoring
  - Small releases
- **Shared understanding**
  - Coding standards
  - Collective code ownership
  - Simple design
  - System metaphor
- **Programmer welfare**
  - Sustainable pace

XP - Pair Programming



Copyright © 2003 United Feature Syndicate, Inc.

Pair Programming (1)

- ***Increased discipline.*** Pairing partners are more likely to "do the right thing" and are less likely to take long breaks.
- ***Better code.*** Pairing partners are less likely to produce a bad design due to their immersion, and tend to come up with higher quality designs.
- ***Multiple developers contributing to design.*** If pairs are rotated frequently, several people will be involved in developing a particular feature. This can help create better solutions, particularly when a pair gets stuck on a tricky problem.
- ***Improved morale.*** Pair programming can be more enjoyable for some engineers than programming alone.

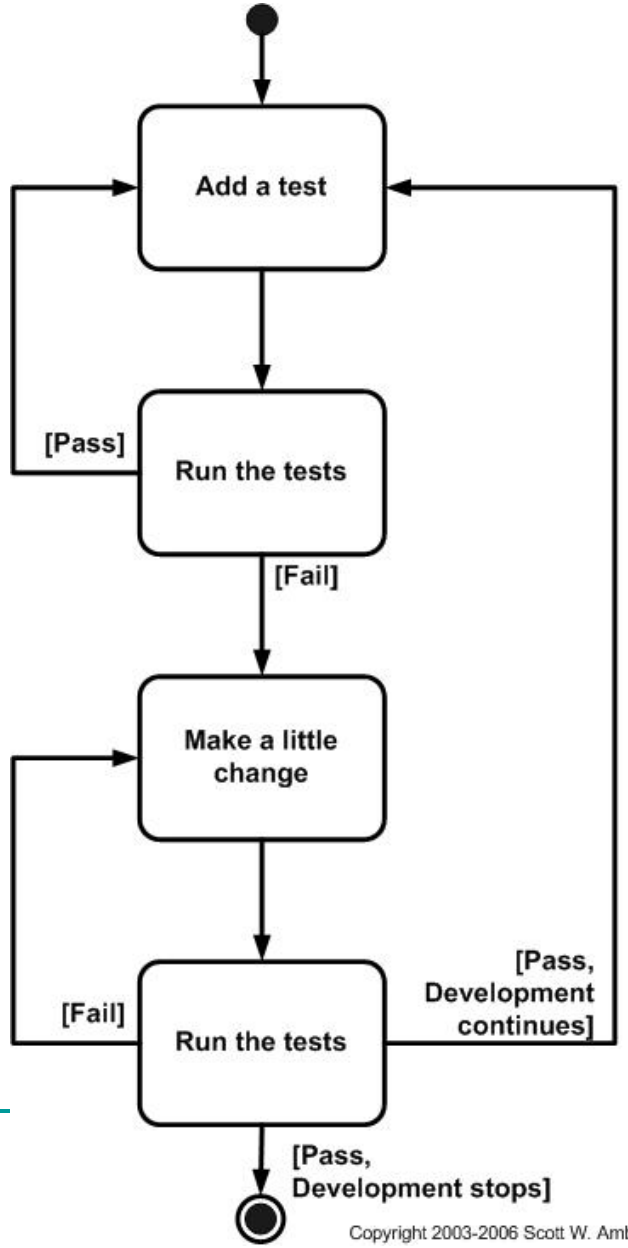
Pair Programming (2)

- ***Collective code ownership.*** When everyone on a project is pair programming, and pairs rotate frequently, everybody gains a working knowledge of the entire codebase.
- ***Mentoring.*** Everyone, even junior programmers, possess knowledge that others don't. Pair programming is a painless way of spreading that knowledge.
- ***Team cohesion.*** People get to know each other more quickly when pair programming. Pair programming may encourage team gelling.

Test-Driven Development

- Test cases are written first
  - Cover new functionality or improvement
- Then the necessary function is implemented
- Code is “complete” when all tests pass

- **Refactor** before adding feature if design could be better



Copyright 2003-2006 Scott W. Ambler

Advantages of Test-Driven

-
-
-

Drawbacks of Test-Driven

-
-
-

Class Activity

- In groups of 2 or 3, on paper:
  - Compare and contrast Waterfall vs. XP with respect to managing software quality

Summary

- Software Quality is a large problem
  - Code quality is an important part of it
- Code quality is difficult to assess directly
  - Usually associated to process quality
- Good mechanisms for these processes
  - Cultural, Human, Automatic, Pair programming, …
- CMMI to capture & improve process maturity
- Agile principles incorporate quality management