Software Quality Assurance: Introduction

Dr. Holger Giese
Software Engineering Group
Room E 3.165
Tel. 60-3321
Email: hg@upb.de

Outline

I Introduction

[Galin2004]

I.1 Motivation

The Software Crisis:

- IBM Consulting group estimates that 55% of large distributed systems projects cost more than expected, 68% overrun their schedules, and 88% require redesign.
- The Standish group estimated the cost of ‘bad software’ for US businesses at $85 billion for 1998.
- The Y2K problem was estimated to cost $1 to $2 trillion.
- W.W Gibbs, in “Software's Chronic Crisis” in the Scientific American, September 1994 estimates that the average software project overshoots its schedule by half.

⇨ How to develop quality software?

Software's Chronic Crisis

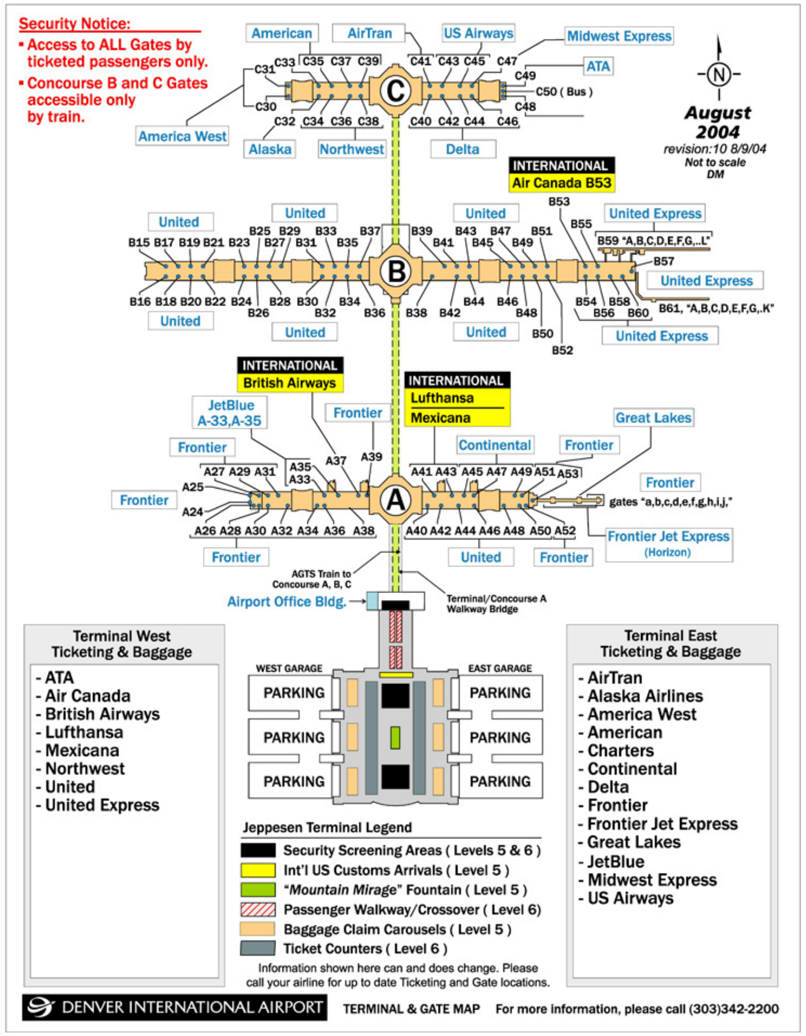
Denver's new international air port was to be the pride of the Rockies, a wonder of modern engineering. Twice the size of Manhattan, 10 times the breadth of Heathrow, the airport is big enough to land three jets simultaneously-in bad weather. Even more impressive than its girth is the airport's subterranean baggage-handling system. Tearing like intelligent coal-mine cars along 21 miles of steel track, 4,000 independent "telecars" route and deliver luggage between the counters, gates and claim areas of 20 different airlines. A central nervous system of some 100 computers networked to one another and to 5,000 electric eyes, 400 radio receivers and 56 bar-code scanners orchestrates the safe and timely arrival of every valise and ski bag.

At least that is the plan. For nine months, this Gulliver has been held captive by Lilliputians- errors in the software that controls its automated baggage system. Scheduled for takeoff by last Halloween, the airport's grand opening was postponed until December to allow BAE Automated Systems time to flush the gremlins out of its $193-million system. December yielded to March. March slipped to May. In June the airport's planners, their bond rating demoted to junk and their budget hemorrhaging red ink at the rate of $1.1 million a day in interest and operating costs, conceded that they could not predict when the baggage system would stabilize enough for the airport to open.

…

W. W. Gibbs. *Software's chronic crisis*. Scientific American, 271(3):72--81, September 1994.
http://www.cis.gsu.edu/~mmoore/CIS3300/handouts/SciAmSept1994.html

Denver International Airport Baggage System


Security Notice:
• Access to ALL Gates by ticketed passengers only.
• Concourse B and C Gates accessible only by train.
American
AirTran
US Airways
Midwest Express
ATA
America West
Alaska
Northwest
Delta
C
C50 (Bus)
August 2004
revision:10 8/9/04
Not to scale
DM
INTERNATIONAL
Air Canada B53
United
United Express
B
B59 "A,B,C,D,E,F,G,..L"
B61, "A,B,C,D,E,F,G,.K"
INTERNATIONAL
British Airways
INTERNATIONAL
Lufthansa
Mexicana
JetBlue A-33,A-35
Frontier
Great Lakes
Continental
A
gates "a,b,c,d,e,f,g,h,i,j,"
Frontier Jet Express (Horizon)
AGTS Train to Concourse A, B, C
Airport Office Bldg.
Terminal/Concourse A Walkway Bridge
Terminal West Ticketing & Baggage
- ATA
- Air Canada
- British Airways
- Lufthansa
- Mexicana
- Northwest
- United
- United Express
WEST GARAGE
EAST GARAGE
PARKING
Terminal East Ticketing & Baggage
- AirTran
- Alaska Airlines
- America West
- American
- Charters
- Continental
- Delta
- Frontier
- Frontier Jet Express
- Great Lakes
- JetBlue
- Midwest Express
- US Airways
Jeppesen Terminal Legend
Security Screening Areas (Levels 5 & 6)
Int'l US Customs Arrivals (Level 5)
"Mountain Mirage" Fountain (Level 5)
Passenger Walkway/Crossover (Level 6)
Baggage Claim Carousels (Level 5)
Ticket Counters (Level 6)
Information shown here can and does change. Please call your airline for up to date Ticketing and Gate locations.
DENVER INTERNATIONAL AIRPORT
TERMINAL & GATE MAP
For more information, please call (303)342-2200

The Denver International Airport has a modern, automated baggage-handling system designed by BAE Automated Systems, Inc. (In June, 2003 G & T Conveyor Company, Inc. acquired BAE)

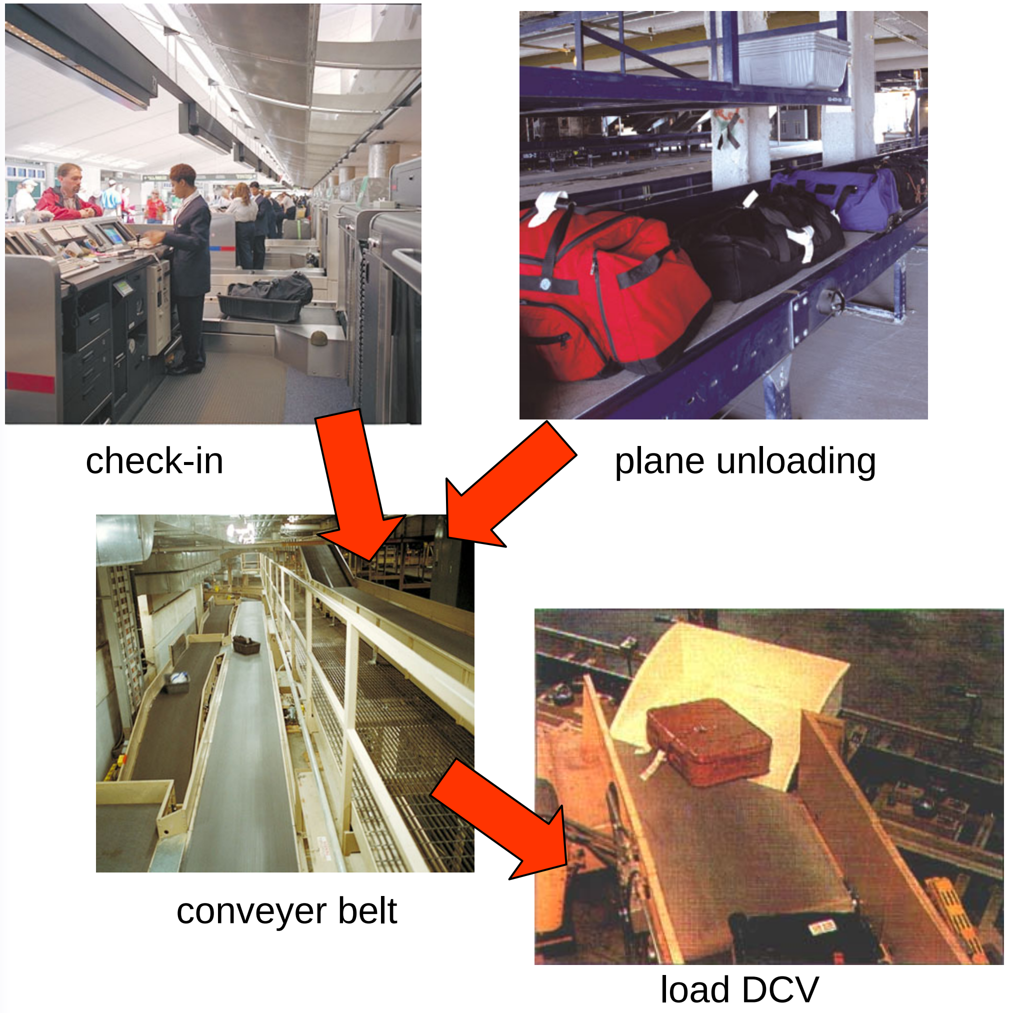
See:

Schloh, Michael. *Analysis of the Denver International Airport baggage system* http://www.csc.calpoly.edu/~dstearns/SchlohProject/csc463.html

Eipe, Rohit. The importance of software architecture: Denver International Airport's automated baggage handling system: A report http://wiki.cs.uiuc.edu/cs427/Essay+by+Rohit+Eipe:+Denver+Intnl+Airport:+Baggage+System

Nice, Karim. *How Baggage Handling Works*, HowBIZWorks http://biz.howstuffworks.com/baggage-handling.htm

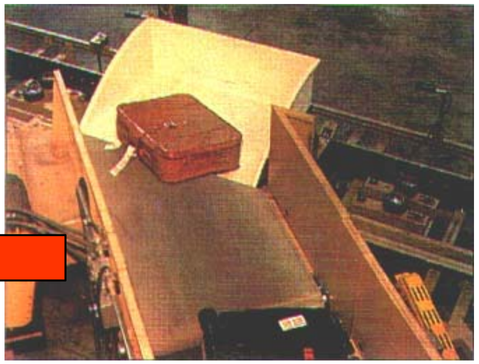
Automated Baggage-Handling System (1/3)

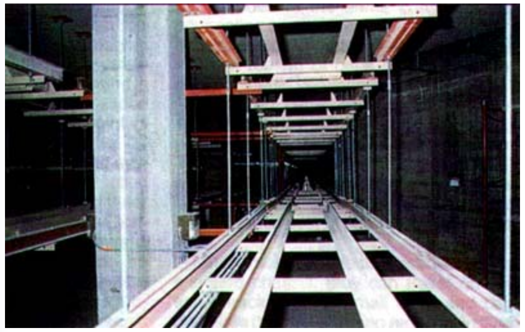


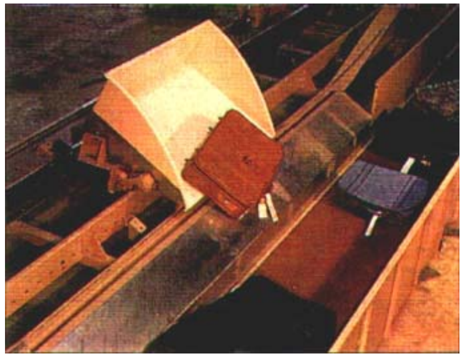
- The baggage handling system at an airport plays a crucial role. It makes the difference in an airport's ability to attract or keep a major airline **hub** ("an airport that serves as a central connecting point)
- A baggage-handling system has three main jobs:
  - Move bags from the check-in area to the departure gate
  - Move bags from one gate to another during transfers
  - Move bags from the arrival gate to the baggage-claim area
- **Conveyors** equipped with **junctions** and **sorting machines** automatically route the bags to the gate.

Automated Baggage-Handling System (2/3)

a DCV (telecar)

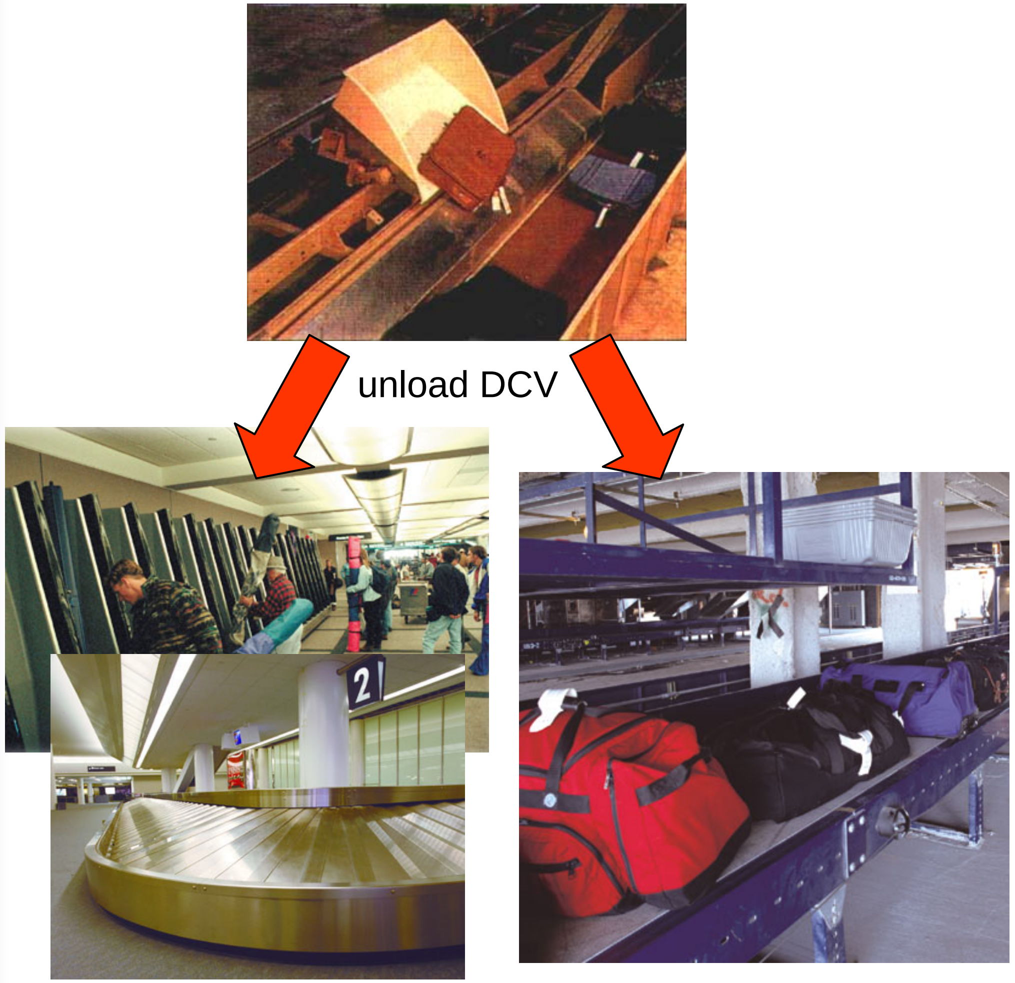

load DCV


tunel system


unload DCV

- **Destination-coded vehicles** (DCVs), unmanned carts (also named telecars)
  - propelled by linear induction motors mounted to the tracks
  - can load bags (without stopping)
  - can unload bags (without stopping)

Automated Baggage-Handling System (3/3)



DIA has:

- More than 5 miles (8 km) of conveyor belts
- More than 19 miles (30 km) of DCV tracks
- About 4,000 DCVs

⇨ enabling it to handle over 1,000 bags per minute.

Observation: First System Test (March 1994)

- Telecars jumped tracks, crashed into each other; pieces of baggage were flung off telecars, in some cases chewed to pieces.
- Baggage was sent to the wrong places.
- Line balancing algorithm problems arose, where there were bags lined up waiting for telecars to come by, and yet empty telecars would just go by without picking up these bags.
- In peak conditions, the system quickly became overloaded with tracking all the telecars in transit.
- Head of line blocking occurred at popular telecar intersections in the tunnel network, and rerouting of approaching telecars didn't always take place.
- Poorly printed baggage tags led to the laser scanners at one point rejecting 70% of all baggage tags, sending them to manual baggage stations.

Remark 1: This famous demo of the system in which hundreds of bags were destroyed was not done by BAE, but was done by the City of Denver when BAE told them not to do it because the system was not working. BAE claimed that the city violated the contract many times, making it nearly impossible for them to finish the job. Eventually the city quit managing the project and United Airlines took over, and BAE was able to finish the project.

Observation: Timeline (1/2)

- The project was begun in the mid 80s by US Transportation Secretary Fredrico Pena, the previous Denver mayor.
- Construction was supposed to begin in September 1989 with an opening date set for October 31st 1993.
- On March 2nd, 1993 the first delay was announced, changing the date to December 19th, 1993.
- October that year was the second delay.
- March 1994 was the third.
- After this Logplan was brought in, yet the fourth delay was announced on May 2nd, 1994.
- DIA finally began operations on February 28th, 1995.

Observation: Timeline (2/2)

- DIA finally began operations **sixteen** months behind schedule, with a reduced number of gates operational, concourse A being postponed indefinitely, the only having been implemented in concourse B, an inability to handle transfer flights, and with a capacity for handling only 30 bags a minute.
- As of November 1996, DIA's automatic baggage handling system was still not working. The airport was not even increasing its air traffic; in 1995 the number of passengers dropped by 2 million from when Stapleton was running, in 1994. High fees associated with Denver charging each airline a $20 fee per passenger, which must have been passed on to the customer, are supposedly the reason for this disappointing performance.

Observation: Financial Damages

Automated baggage handling system:

- Originally planed budget was $193 million
- the amount finally spent was close to $311 million (included the cost of installing a manual system)

DIA

- planed DIA costs were $1.7 billion
- the final cost was more than 2.5 times that, at $4.5 billion

Delay costs:

- estimated costs for every month that the DIA remained closed was about $33.3 million

Cause: Lack of Management

- The baggage handling system has been implemented almost as an afterthought! Thus the system of tunnels was not designed with a particular baggage handling system in mind, and as it turned out, the small tunnels and many sharp turns made BAE's job rather difficult.
- The airport designers didn't even design their system with a baggage handling system in mind, so BAE had to work around all kinds of problems.
- This was the reason that the management team never saw the baggage handling system, of all things, to be so important that it could have such a profound economic impact on the project.
- The planning of the system was done in a completely haphazard way. Little communication took place between DIA's airport designers, the city officials, the airlines, and BAE.
- The planning instead was driven by higher level management with little or no understanding of the complexity of the system, and by consultants contracted to develop a specification of the system, but who had no direct responsibility to the team(s) that would actually develop the system.

Cause: Impossible Schedule/Plan

- One ‘reason’, if it might be called that, that BAE didn’t perform a review of their own design was that they were being made to work under an impossible tight schedule.
- BAE claim that they protested at the outset that the timeline Denver city officials proposed for the completion of the project was utterly unrealistic.
- Denver on the other hand claims that BAE committed to delivering the system within a certain period of time, and should have delivered a bug free system in that time.
- Whoever the culprit, one thing is for sure, the time required to fully understand and do justice to a system of this size and complexity was grossly underestimated, and this had direct bearing on the efficiency of the design that was put forth, and ultimately implemented.
- identify problems with the design, and take decisions about them early, instead of reacting to those problems after they were uncovered in testing
- BAE and Denver didn’t heed the advice of others, and grossly underestimated the time for developing and testing the system **(**The Munich airport, on which DIA was modeled, spent two years testing the system and six months of running the system 24x7 before opening)

Cause: Complexity Problems

- As it turned out, when a change needed to be made to one part of the system, it was not clearly understood how that change would impact the rest of the system. When the system was eventually test-run, BAE found it incredibly difficult to even understand why things were going wrong.
- BAE's design was such that the number of components working together was very high, and they were very tightly coupled, and there was little redundancy.
- sheer size of the system made it so complex that not even the people who designed it were able to understand it very well, leave alone a third party
- to weed out problems at the design stage than to implement a design that had rather glaring faults, and then try to perform frantic firefighting

Cause: Many Interfaces

- One extremely difficult problem that BAE faced was the task of conversing with United Airlines'Apollo reservation computers.
- BAE's computers had to speak the same software language as each of the airlines
- The process of language translation that was necessitated was painful and bug-filled at best.

DIA Conclusions

- Missing communication between the stakeholders
- The schedule didn’t allow enough time to fully design the system before jumping into implementation.
- The design of the baggage handling system was not integrated into the design of the airport.

Not entirely inappropriately, the DIA was coined as being DOA – dead on arrival!

How can we circumvent such disasters?

Observation:

- Quality problems prevent the system from functioning
- Management decisions results in wrong reactions
  - Missing design reviews
  - Fixing rather than analyzing
  - Erratic debugging rather than systematic testing

  Therefore even larger delays result

Systematic approach for the development of high quality software is required

I.2 Definitions & Terminology

- Software
  - Definition & characterization
  - Why do the classical approach to QA not apply?
- Software quality
  - Definition
  - Quality models
  - Quality attributes
- Measure quality
  - Reliability, Availability, safety, maintainability,
  - Defect counts
  - Software errors, faults and failures
  - Classification of the causes of software errors

Basic Terminology

Software is:

Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system.

[IEEE_Std_610.12-1990]

[Galin2004]

What is special about Software?

- Invisibility of the product
- Limited opportunities to detect defects ("bugs")
- Often new demanding functionality has to be realized
- Often software has to realize extraordinary high complexity

Classical Quality Assurance (Hardware)

- **Requirements:** complete set of external quality characteristics ⇨complete set of internal quality characteristics and a detailed and complete set of requirements and specifications
- **Design:**
  - Design that satisfies the requirements and specifications
  - design for reliability (wear out)
  - design for manufacturability
  - design for maintainability (e.g., self-diagnosis)
- **Manufacturing:** statistical production process control with acceptance sampling
- **Operation:** collect failure data for continuous improvement and predictions (intelligent maintenance)

Software Quality Assurance?

- **Requirements:**
  - Completeness is hard to achieve (complexity)
- **Design:**
  - design for reliability only in special cases
  - design for manufacturability is not required
  - design for maintainability in special cases
  - Focal area of software quality assurance!
- **Manufacturing ⇨ Implementation:**
  - limited success with statistical process control (metrics)
- **Operation:**
  - Fixing found bugs

[Sommerville2004]

What is quality?

- Quality, simplistically, means that a product should meet its specification.
- This is problematical for software systems
  - There is a tension between customer quality requirements (efficiency, reliability, etc.) and developer quality requirements (maintainability, reusability, etc.);
  - Some quality requirements are difficult to specify in an unambiguous way;
  - Software specifications are usually incomplete and often inconsistent.

Approaches to Tackle Quality

- **Transcendental view**: quality is universally identifiable, absolute, unique and perfect
- **Product view**: the quality of a product is measurable in an objective manner
- **User view**: quality is fitness for use
- **Manufacturing view**: quality is the result of the right development of the product
- **Value-based view (Economic)**: quality is a function of costs and benefits

Software Quality – IEEE View

Software quality is:

(1) The degree to which a system, component, or process meets specified requirements.

(2) The degree to which a system, component, or process meets customer or user needs or expectations.

[IEEE_Std_610.12-1990]

Software Quality

Software quality is :

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

[Pressman2003]

Software Quality

Quality – the degree of excellence of something. We measure the excellence of software via a set of attributes.

[Glass1992]

(≠ “satisfying requirements”)!

- Quality is absolute or at least independent of the requirements underlying the product.
- It is part of the software development to “get the right requirements”.

(≠ “user satisfaction”)!

- McDonald‘s restaurants are popular, but …

Quality Models

- Such general definitions of software quality are not sufficient in practice
- Thus, software quality is described by specific quality models
- „causal relationship from intangible quality views to tangible software measures"

[ISO/IEC 9126]

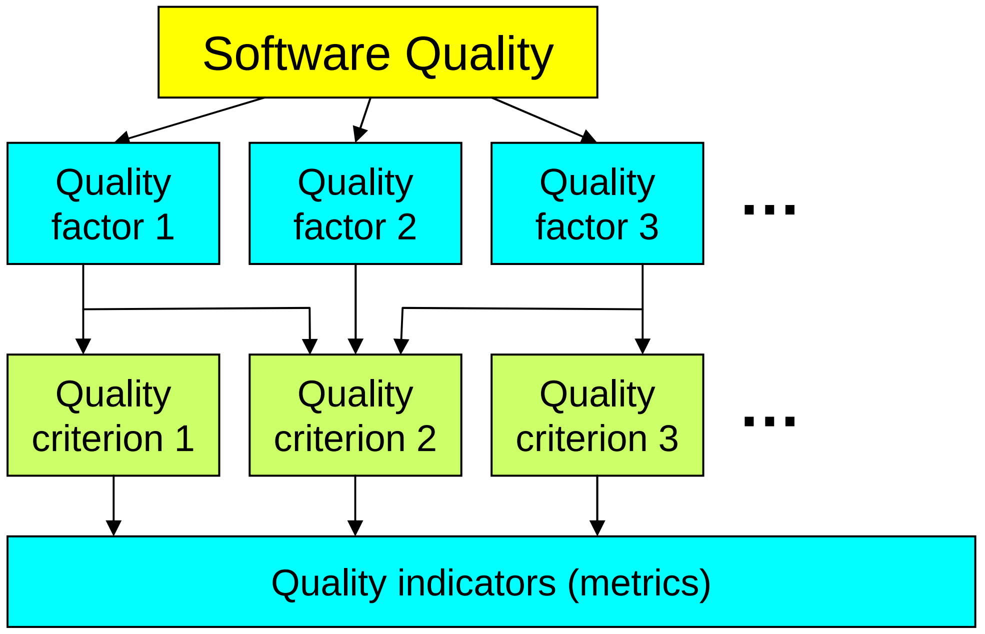
Two main approaches:

- Standard Models:
  - McCall
  - ISO/IEC 9126
- Application or company specific quality models
  - FURPS
  - GQM Approach

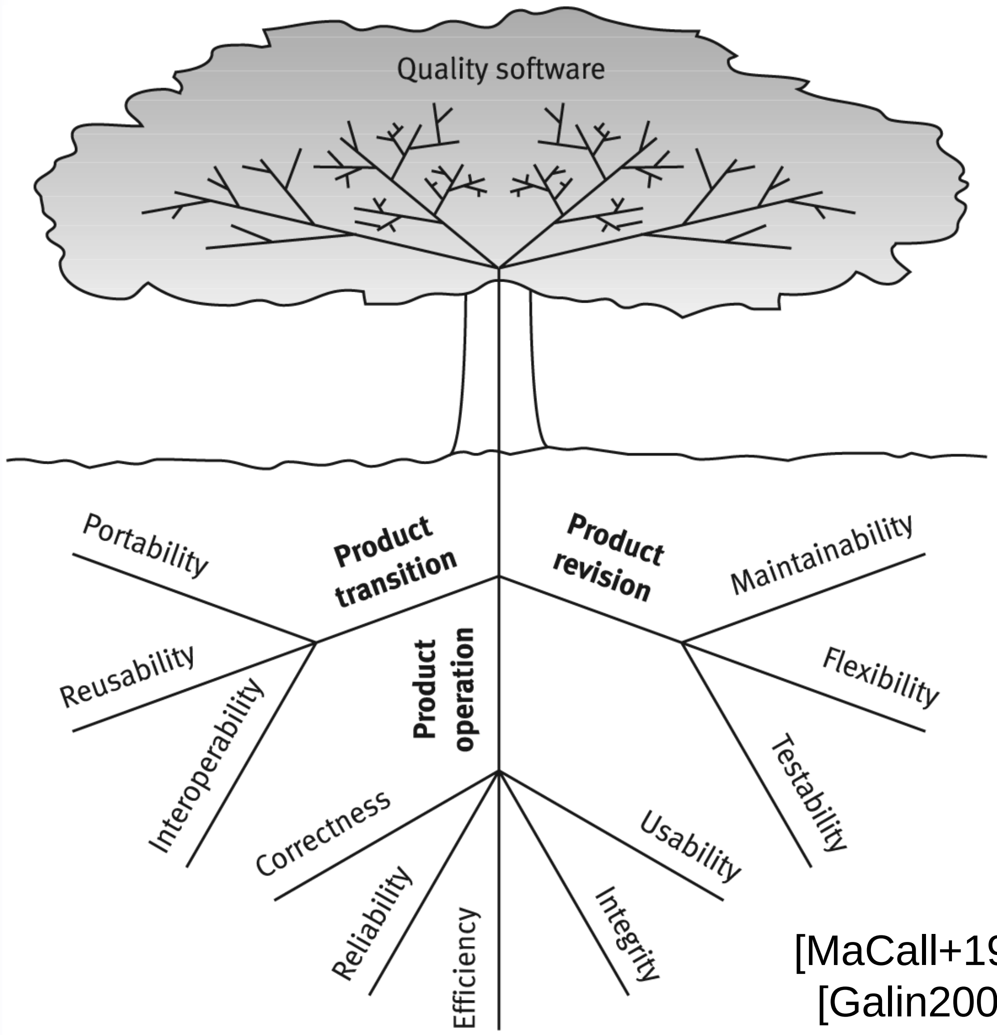
Factor-Criteria-Metrics-Model

Classification into :

- **Factors** (to specify): They describe the external view of the software, as viewed by the users.
- **Criteria** (to build): They describe the internal view of the software, as seen by the developer.
- **Metrics** (to control): They are defined and used to provide a scale and method for measurement.



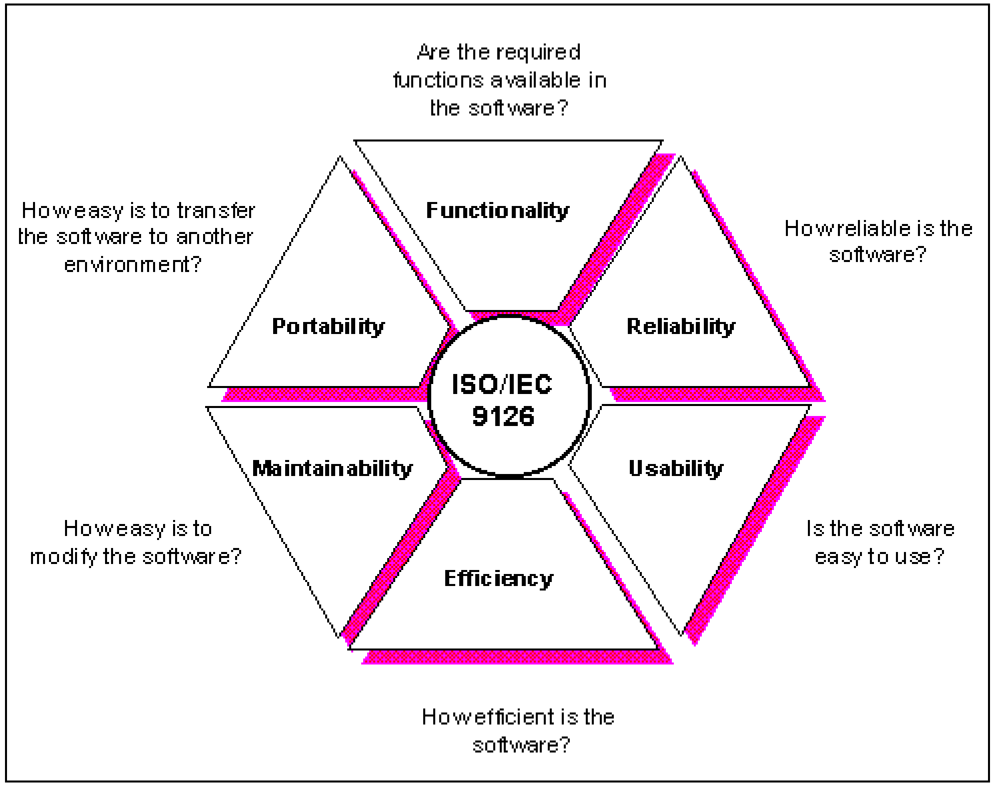
McCall's Factor Model Tree



[MaCall+1977]
[Galin2004]

- A **quality factor** represents a behavioral characteristic of the system.
  - Operation
  - Revision
  - Transition
- A **quality criterion** is an attribute of a quality factor that is related to software production and design.
- A **quality metric** is a measure that captures some aspect of a quality criterion.

The Six Quality Characteristics of a Software (ISO/IEC 9126)


Are the required functions available in the software?
How easy is to transfer the software to another environment?
How reliable is the software?
Functionality
Portability
Reliability
ISO/IEC 9126
Maintainability
Usability
Efficiency
How easy is to modify the software?
Is the software easy to use?
How efficient is the software?

[ISO/IEC 9126]

- **Software quality:** *The totality of features and characteristics of a software product that bear on its ability to satisfy stated or implied needs.* (ISO 9126: 1991, 3.11)
- **Software quality characteristics:** *A set of attributes of a software product by which its quality is described and evaluated. A software quality characteristic may be refined into multiple levels of sub-characteristics.* (ISO 9126: 1991, 3.13)

- Each characteristic is refined to a set of sub-characteristics
- Each sub-characteristic is evaluated by a set of metrics.
- Some metrics are common to several sub-characteristics.

Characteristics	Subcharacteristics	Definitions
Functionality	Suitability	Attributes of software that bear on the presence and appropriateness of a set of functions for specified tasks.
	Accurateness	Attributes of software that bear on the provision of right or agreed results or effects.
	Interoperability	Attributes of software that bear on its ability to interact with specified systems.
	Compliance	Attributes of software that make the software adhere to application related standards or conventions or regulations in laws and similar prescriptions.
	Security	Attributes of software that bear on its ability to prevent unauthorized access, whether accidental or deliberate, to programs or data.
Reliability	Maturity	Attributes of software that bear on the frequency of failure by faults in the software.
	Fault tolerance	Attributes of software that bear on its ability to maintain a specified level of performance in case of software faults or of infringement of its specified interface.
	Recoverability	Attributes of software that bear on the capability to re-establish its level of performance and recover the data directly affected in case of a failure and on the time and effort needed for it.
Usability	Understandability	Attributes of software that bear on the users' effort for recognizing the logical concept and its applicability.
	Learnability	Attributes of software that bear on the users'effort for learning its application.
	Operability	Attributes of software that bear on the users'effort for operation and operation control.

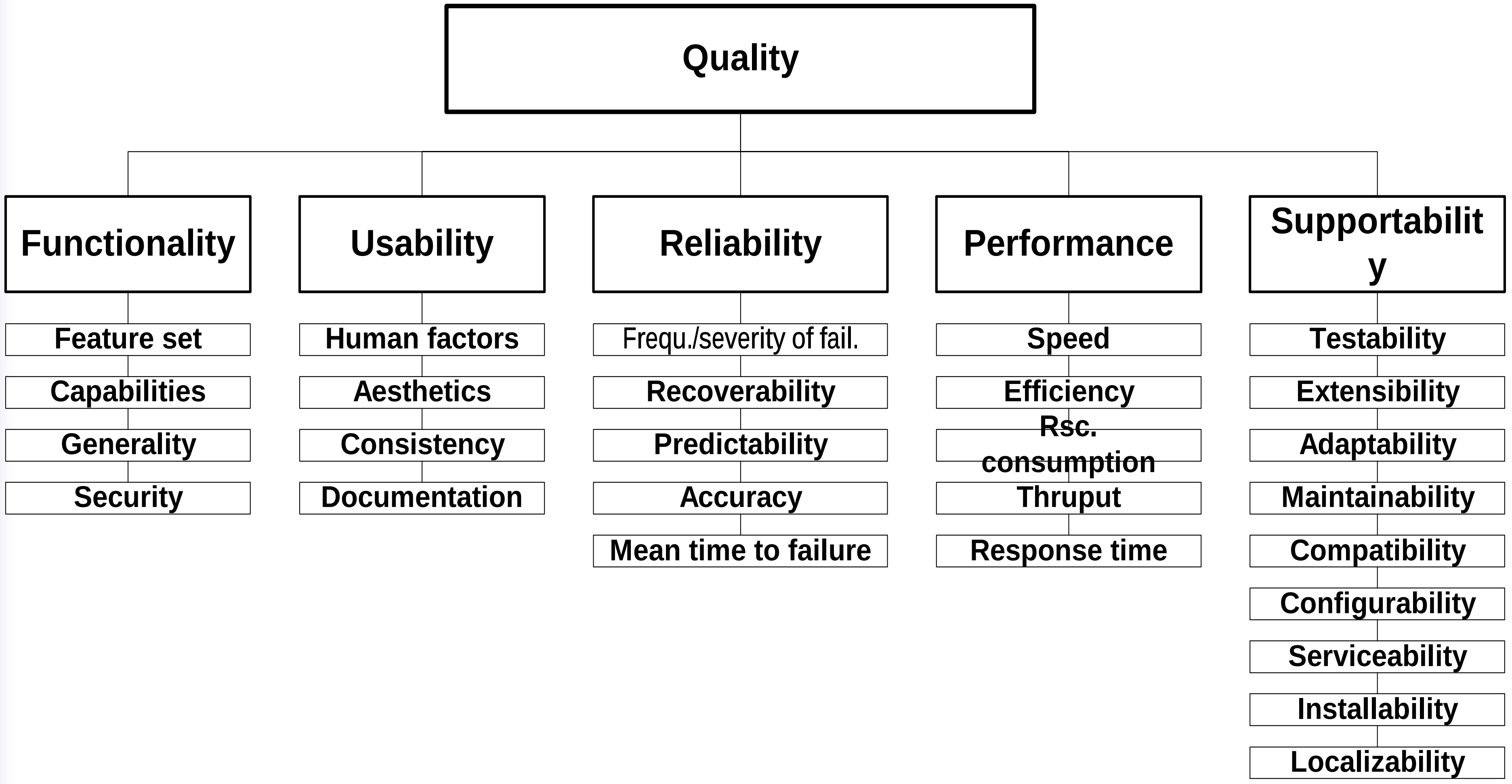
Characteristics	Subcharacteristics	Definitions
Efficiency	Time behaviour	Attributes of software that bear on response and processing times and on throughput rates in performances its function.
	Resource behavior	Attributes of software that bear on the amount of resource used and the duration of such use in performing its function.
Maintainability	Analyzability	Attributes of software that bear on the effort needed for diagnosis of deficiencies or causes of failures, or for identification of parts to be modified.
	Changeability	Attributes of software that bear on the effort needed for modification, fault removal or for environmental change.
	Stability	Attributes of software that bear on the risk of unexpected effect of modifications.
	Testability	Attributes of software that bear on the effort needed for validating the modified software.
Portability	Adaptability	Attributes of software that bear on the opportunity for its adaptation to different specified environments without applying other actions or means than those provided for this purpose for the software considered.
	Installability	Attributes of software that bear on the effort needed to install the software in a specified environment.
	Conformance	Attributes of software that make the software adhere to standards or conventions relating to portability.
	Replaceability	Attributes of software that bear on opportunity and effort using it in the place of specified other software in the environment of that software.

Hewlett Packard: F.U.R.P.S. (1/2)

- Result of a statistical project survey at Hewlett Packard 1987 to improve its products:

  Factors:
  - **F**unctionality: functions it performs, their generality and security
  - **U**sability: aesthetics, consistency, documentation
  - **R**eliability: frequency and severity of failure, accuracy of output
  - **P**erformance: response time, resource consumption
  - **S**upportability: can it be extended, adapted, corrected?
- FURPS is originally a company specific quality model

Hewlett Packard: F.U.R.P.S. (2/2)

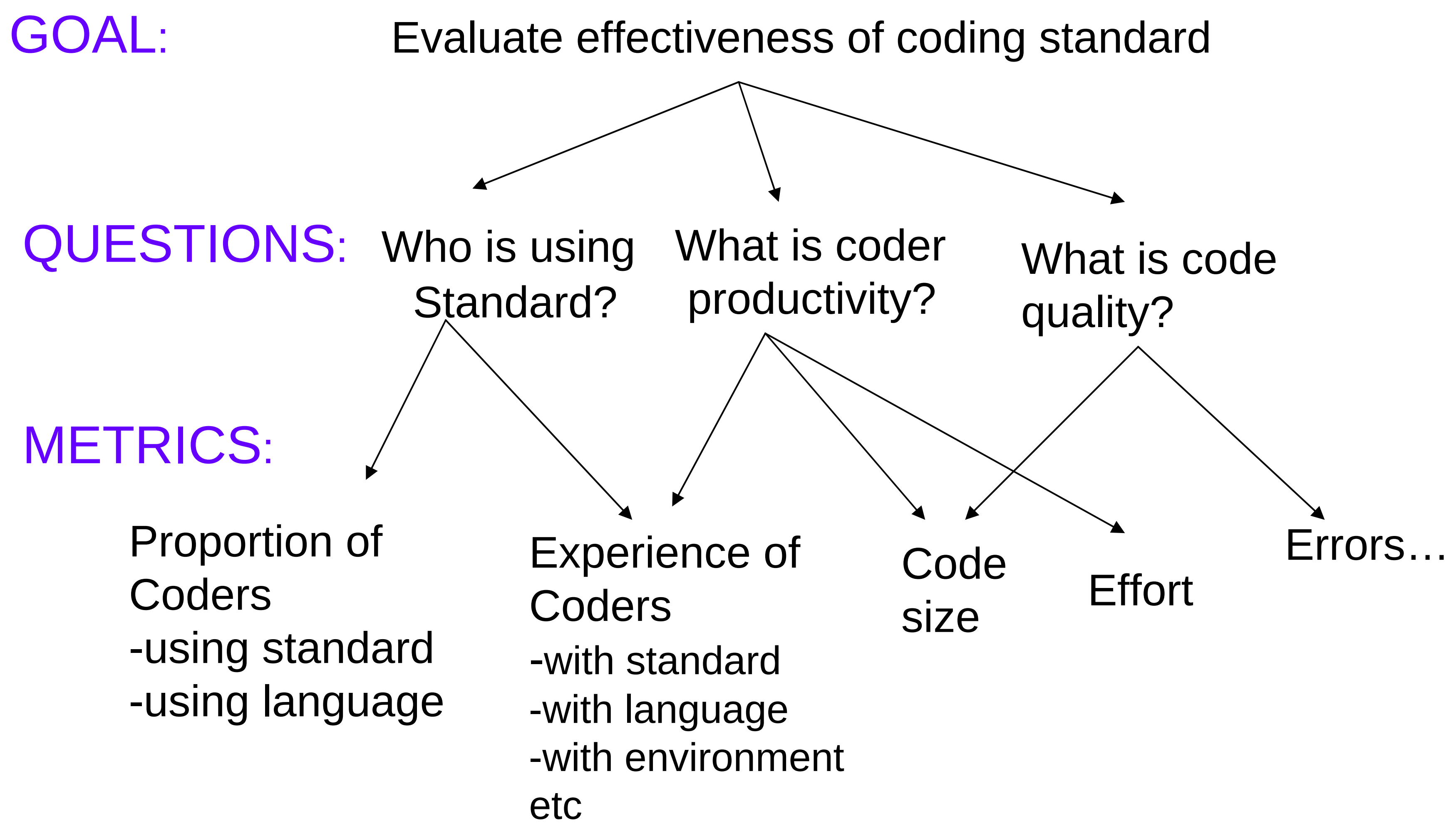


GQM: Goal-Question-Metric

- A measurement program can be more successful if designed with the goals in mind.
- GQM approach provides a framework with 3 steps:
  1. List the major goals of the development/maintenance project
  2. Derive from each goal the questions that must be answered to determine if the goals are being met
  3. Decide what must be measured to answer the questions adequately

[Basil&Rombach1988]

Example


GOAL:
Evaluate effectiveness of coding standard
QUESTIONS:
Who is using Standard?
What is coder productivity?
What is code quality?
METRICS:
Proportion of Coders
-using standard
-using language
Experience of Coders
-with standard
-with language
-with environment
etc
Code size
Effort
Errors…

GQM Discussion

- Benefits :
  - generates only those measures relevant to the goal
  - Several measurements may be needed to answer a single question.
  - A single measurement may apply to more than one question.
  - The goal provides the purpose for collecting the data.
- Not evident from the GQM
  - The model needed to combine the measurement in a sensible way so that the question can be answered.
  - It must be supplemented by one or more models that express the relationship among the metrics. (equation definition is not clear)
- Disadvantages:
  - Additional efforts to derive the goals and metrics
  - Error prone compared to standard models

Measuring Quality?

- User‘s view:
  - Reliability (number of failures over time)
  - Availability
  - Usability etc.
  - Often directly measurable
- Manufacturer‘s view:
  - Defect counts
  - Rework costs

Reliability

Reliability is the probability of a component, or system, functioning correctly over a given period of time under a given set of operating conditions.

Quantitative:

Reliability *R*(*t*) is the probability that the system will conform to its specification throughout a period of duration *t*.

- Note that t is important
- If a system only needs to operate for ten hours at a time, then that is the reliability target.

Availability

The **availability** of a system is the probability that the system will be functioning correctly at any given time.

Quantitative:

Availability *A* (or *V*) is the percentage of time for which the system will conform to its specification.

- Literally, readiness for service

Safety

[Storey1996]

- **Safety** is a property of a system that it will not endanger human life or the environment.

- A **safety-related system** (**safety-critical system**) in one by which the safety of equipment or plant is assured.

Maintainability

- **Maintenance** is the action taken to retain a system in, or return a system to, its designed operating conditions. **Maintainability** is the ability of a system to be maintained (ability to undergo repairs and modifications).

- Mean time to repair (MTTR)

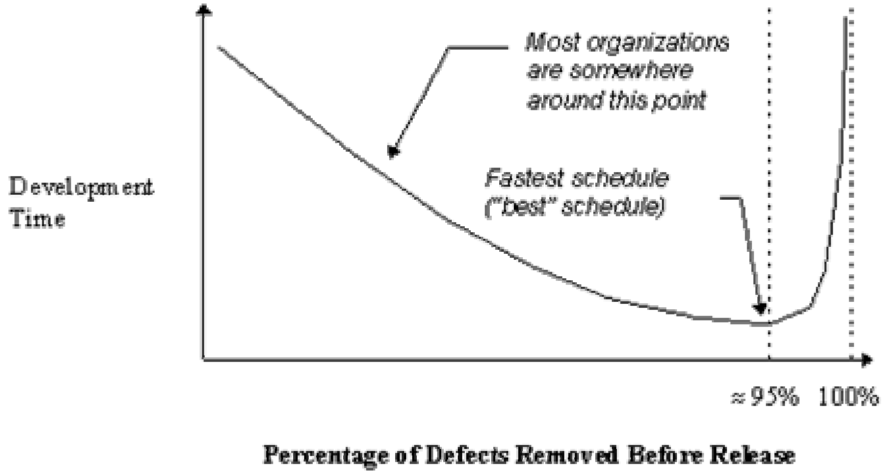
Problem: Maintenance induced failures

Defect Counts

- „software errors are the cause of poor software quality“
  [Galin2004]

- In software, higher quality (in the form of lower defect rates) and reduced development time go hand in hand.
  [McConnell 1996]

- Unlike the low-defect kind of quality, attention to this kind of quality tends to lengthen the development schedule.
  [McConnell 1996]



- Intuitively, the occurrence of defects is negatively related to functionality and reliability. Defects also interfere, to some degree, with other dimensions of quality.

Relative Cost of Correcting Defects

[Pressman2003]

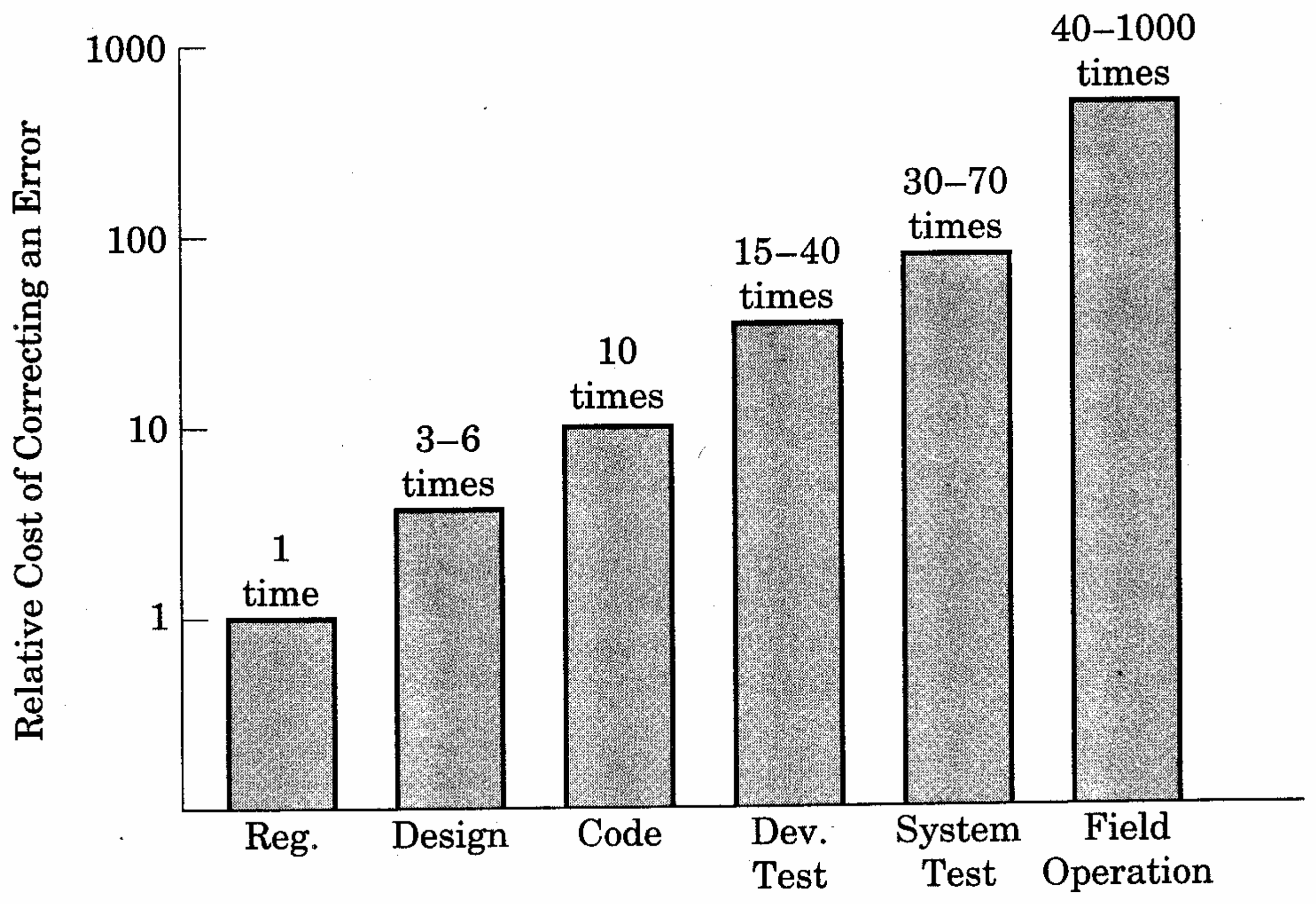


FIGURE 8.1.
Relative cost of correcting an error.

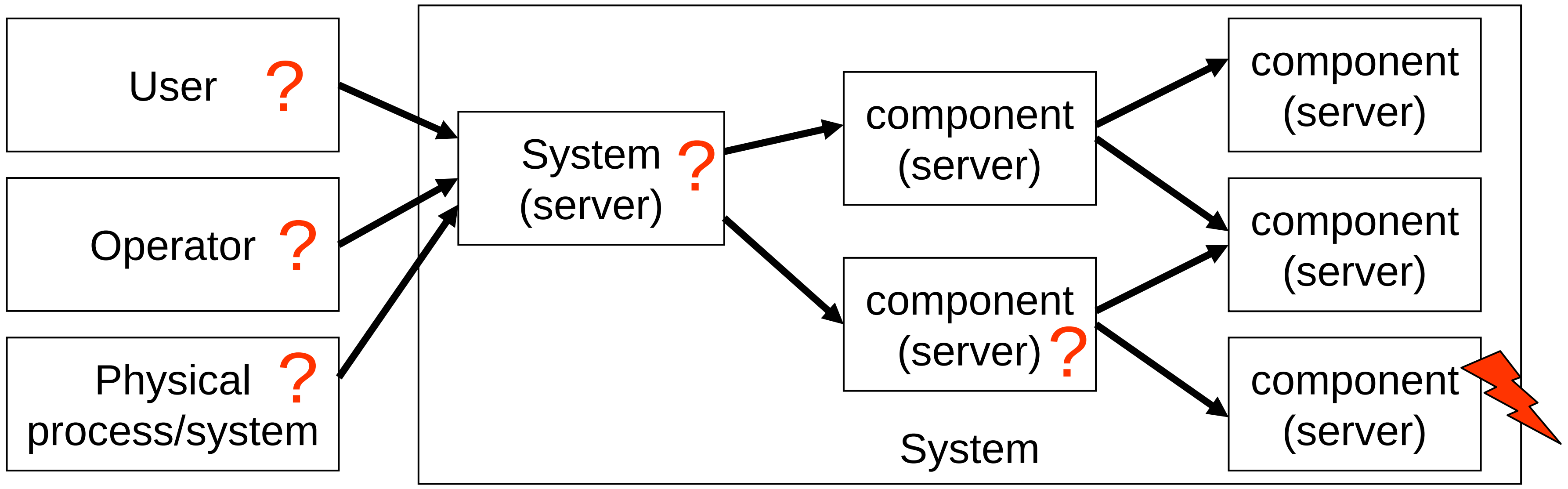
Faults, Errors and Failures

- A **fault** is a defect within the system.
  (Error cause, German: Fehlerursache)
- An **error** is a derivation of the required operation of the system or subsystem.
  (Manifestation of a fault in a system, German: Fehlzustand)
- **software errors** are a human action which results in software containing a fault
- A **system failure** occurs when the system fails to perform its required function.
  (Transition to incorrect service delivery, German: Ausfall)
- If the distinction between fault and failure is not critical, **defect** can be used as a generic term.

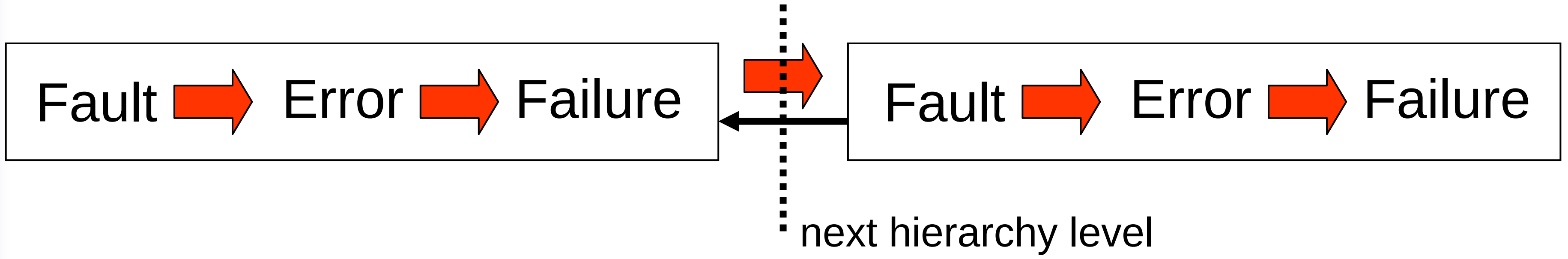
[Storey1996,Liu1996]

Recursive Nature of Faults

- A fault in a component can “infect” all other components which depend on it.
- Chain of Faults/Failures



Fault/Failure Chain



Fault ⇨ error

- a fault which has not been activated by the computation process is *dormant*
- a fault is *active* when it produces an error

Error ⇨ failure

- an error is *latent* when it has not been recognized
- an error is *detected* by a detection algorithm/mechanism

Failure ⇨ fault

- a failure occurs when an error “passes through” and affects the service delivered
- a failure results in a fault for the system which contains or interacts with the component

Examples for Fault/Failure Chain

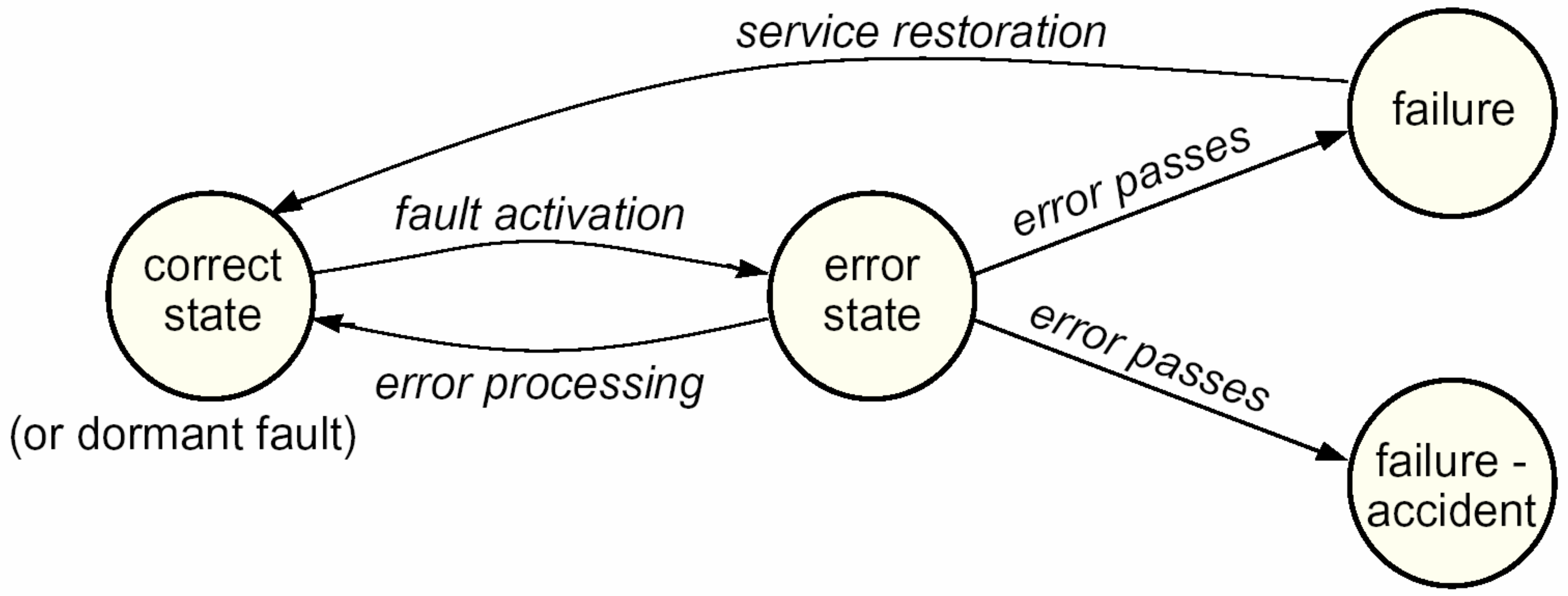
Program error (software):

- a dormant *fault* in the written software (instruction or data)
- upon activation the fault becomes active and produces an *error* (system state)
- if the erroneous data affects the delivered service, a *failure* occurs

Electromagnetic interference (hardware):

- leads to *faulty* input value (either digital or analog)
- by reading the input the fault becomes active and produces an *error*
- if the erroneous input value is processed and becomes visible at the interface a *failure* occurs

Fault/Failure state transitions



Fault Classification

Nature (Critical distinction):

- random/hardware faults
- logical/systematic/design faults

E.g., Degradation (wear-out) vs. design faults

Duration:

- permanent, transient, intermittent

Extent:

- localized, global

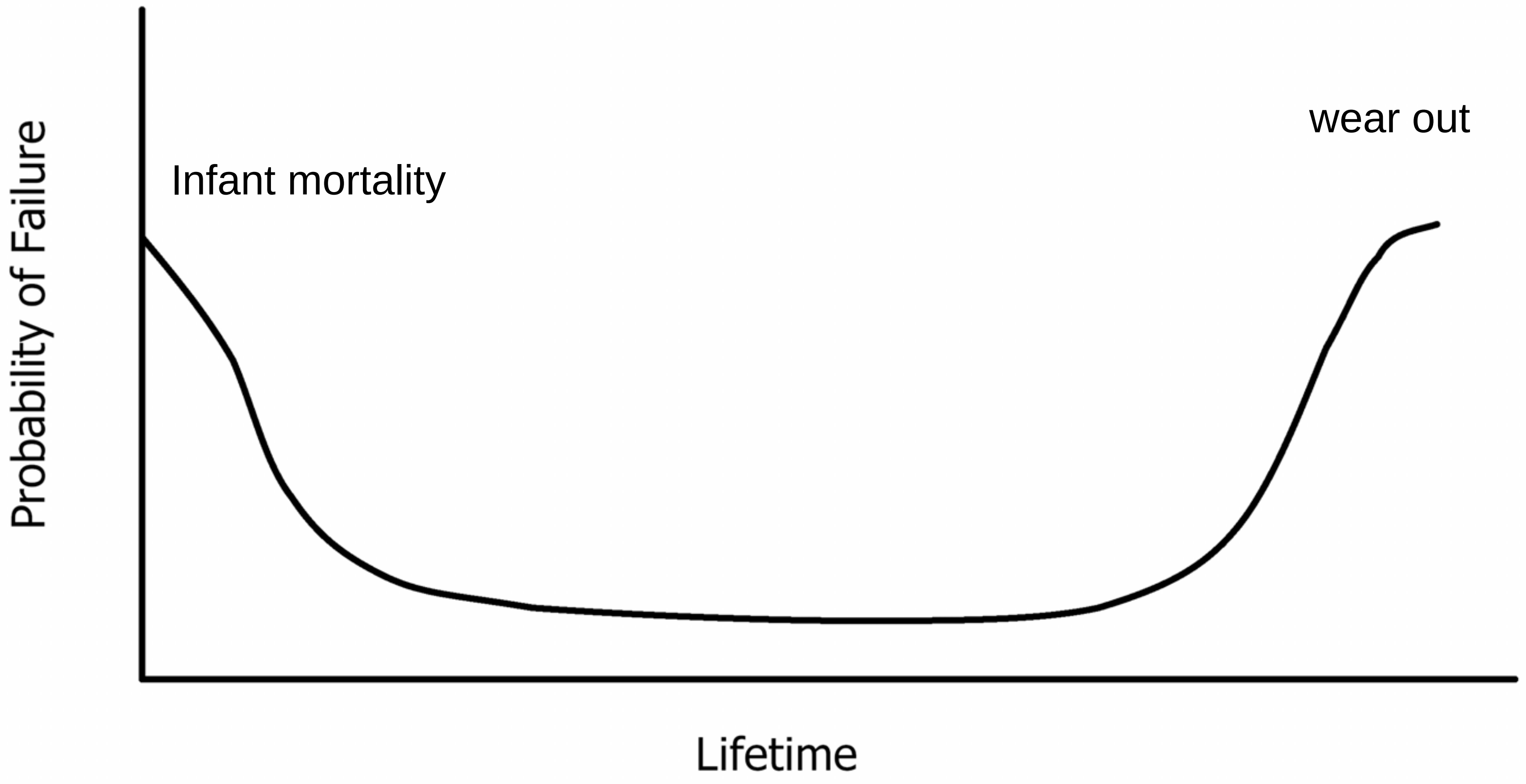
Nature: Degradation Faults

- The system used to work but now it does not
- Something broke for some reason:
  - Infant mortality
  - End of useful life—wear out
  - Physical damage—vibration, humidity, temperature

Examples:

- Light bulb burns out
- Disk head crashes
- Power supply fails

Nature: Failure “Bathtub Curve”



Nature: Design Faults

- The system never worked
- Nothing broke
- The design is flawed

Examples:

- Incorrect electrical wiring
- **Software defect!**

Dealing With Faults

Fault Prevention:

- Development techniques that prevent the introduction of faults

Fault Elimination:

- Development techniques that find and remove faults already introduced

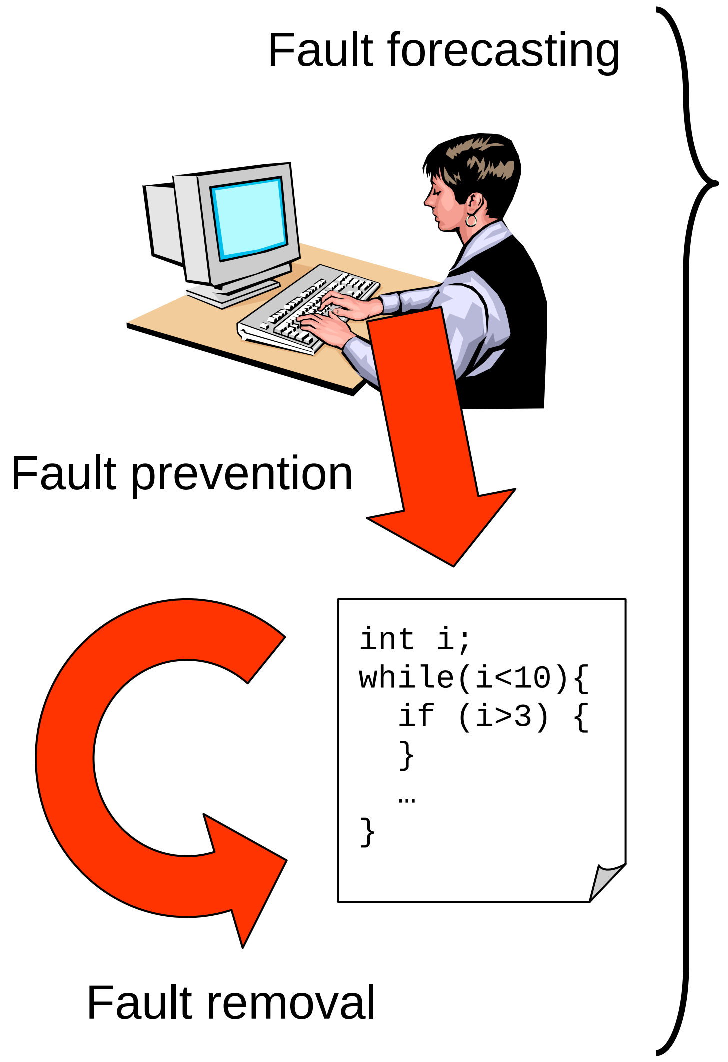
Fault Forecasting:

- Estimate current number, future incidence and likely consequences

Fault Tolerance (not considered here):

- Execution-time techniques that cope with the effects of faults

Fault Avoidance



```
int i;
while(i<10){
  if (i>3) {
  }
  …
}
```

Activities:

- Preventing the introduction or occurrence of faults **(fault prevention):**
  - SWT, formal methods, high level languages, CASE tools, …
- Reducing the number and seriousness of faults **(fault removal):**
  - Analysis ⇨ diagnosis ⇨ correction
- Evaluating the present number future incidents, and severity of faults (**fault forecasting):**
  - Statistics & management

Main causes for software faults:

1. Faulty requirements definition
2. Client-developer communication failures
3. Deliberate deviations from software requirements
4. Logical design errors
5. Coding errors
6. Non-compliance with documentation and coding instructions
7. Shortcomings of the testing process
8. User interface and procedure errors
9. Documentation errors

[Galin2004]