SE-3002
SOFTWARE QUALITY ENGINEERING

RUBAB JAFFAR
RUBAB.JAFFAR@NU.EDU.PK

Introduction

Overview and Basics

Lecture # 1,2 3
23,24,25 Sep

TODAY'S OUTLINE

- Administrative Stuff
- Overview of SE-3002

ADMINISTRATIVE STUFF

ABOUT ME

- Graduated BS(SE) from FJWU, RWP
- Complete Masters MS(SE) from NUST, ISB
- Research interests:
  - Software engineering
  - Object Oriented Design
  - Database systems
  - Algorithm analysis

ADMINISTRATIVE STUFF

OFFICE HOURS

- Office 6 (CS building)
- Office hours: 3:00 to 4:00 pm
  - (Tuesday, Wednesday, Thursday)

OVERVIEW OF SE-3002
ABOUT THE COURSE

- Knowledge of Software Quality Engineering concepts through standards.
- Comprehension of why do we have to engineer quality in to software.
- Comprehension and Application of Software Quality Assurance Plan (SQAP).
- Comprehension and Application of static and dynamic testing techniques for Quality Control (QC).
- Comprehension and Application of Software Quality Measurement using Models (QMM).
- A strong in class participation from the students will be encouraged and required during the discussion on these case studies(your projects)

OVERVIEW OF SE-3002

MAJOR TOPICS OF THE COURSE

- Software Quality, Software Quality Attributes, Quality Engineering,
- Testing Concepts, Issues, Software testing lifecycle. Testing Scopes, Testing Approaches,
- Requirement of software test planning, Testing documentation, Software testing techniques, Testing philosophies
- Testing strategies, Software inspections, Software reviews, Inspection checks and metrics,
- From functional to system aspects of testing, System testing, Scenarios development, Use-cases for testing, Specification-based testing, Open issues on software testing,
- Quality Models, Models for quality assessment, Product quality metrics,
- Quality Measurements, Quality improvement

OVERVIEW OF SE-3002

COURSE OUTLINE

- Basic concept of software quality and Importance of software quality
- Software Quality Standards e.g. ISO 9126
- Software Quality attributes and characteristics
- Software Quality Assurance (SQA) techniques
- Software Quality Control
- Static Testing Techniques types
- Dynamic Testing levels
- Software Quality Measurement Models

PRE-REQUISITES /KNOWLEDGE ASSUMED

- Software Analysis and Design
- Programming language concepts
- Data structure concepts
- Database system concepts

SKILLS ASSUMED

- You have the skills to code in any programming language therefore you
  - can design, implement, test, debug, read, understand and document the programs.

TENTATIVE MARK DISTRIBUTION AND GRADING POLICY

- Assignments/Class Participation: 6 %
- Mid Exams: 25 %
- Final: 50%
- Presentations + Projects: 10%
- Quizzes: 9%
- Absolute Grading Scheme

OVERVIEW OF SE-3002

COURSE ETHICS

Projects/Assignments

- Deadlines are always final
- No credit for late submissions
- One student per assignment at maximum
- Project Proposal Submission : 4th Week
- Project Presentation : Last week

Quizzes

- Announced
- Unannounced

Honesty

- All parties involved in any kind of cheating in any assessment will get zero in that assessment.

OVERVIEW OF SE-3002

PROJECT

- An advance project
  - Provides interesting problem, realistic pressures, unclear/changing
- 3 member teams
- Emphasis on good SE practice/methodology and test plan for QC
  - Homework's and deliverables tied to the project

OVERVIEW OF SE-3002

COURSE MATERIAL

- You will have Presentations of each topic and ***reference books*** in PDF format will be available on GCR.

Text Books

1. Software Quality Engineering Testing, Quality Assurance, and Quantifiable Improvement, Jeff Tian, Wiley, 2005
2. Software testing – Yogesh Singh

Reference Books

1. Software Testing, Principles and Practices, 2nd edition, Naresh Chauhan, Oxford publisher, 2016
2. Software Quality Engineering, A Practitioner's Approach, Witold Suryn, Wiley, 2014

COURSE GOALS/OBJECTIVES

- By the end of this course, you will
  - CLO-1: (C1) outline software testing, software quality assurance principles and role of QA throughout the SDLC
  - CLO-2: (C3) prepare test case and test suites for completely testing all aspects of a system under test (SUT)
  - CLO-3: (C5) compile findings of a quality assurance cycle. Quality improvement through software process quality, improvement models and approaches

HOW THIS COURSE WILL RUN

- The course is divided into four parts
- Part I (Chapters 1-5). Overview and Basics:
- Part II (Chapters 6-12). Software Testing
- Part III (Chapters 13-17). Other Quality Assurance Techniques
- Part IV (Chapters 18-22). Quantifiable Quality Improvement

SOFTWARE QUALITY ENGINEERING

- PART I
- Overview and Basics
- What is Quality?, Quality Assurance, QA in Context, Quality Engineering and Quality Challenge

SOFTWARE QUALITY ENGINEERING

- PART II
- Software Testing:
  - Models & techniques, management, automation, and integration

SOFTWARE QUALITY ENGINEERING

- PART III

Other Quality Assurance Techniques

- Defect prevention, process improvement, inspection, formal verification, fault tolerance, accident prevention, and safety assurance.

SOFTWARE QUALITY ENGINEERING

- PART IV
- Quantifiable Quality Improvement
- Analysis and feedback, measurements and models, defect analysis, risk identification, and software reliability engineering.

PART I: OVERVIEW AND BASICS

- What is Software?
- What is Quality?
- What is Engineering?

WHAT IS SOFTWARE QUALITY?

- No commonly agreed definition
- IEEE definition

"The degree to which a system, component, or process meets specified requirements (Philip Crosby)"

- *Emphasis here is on specifications*

"The degree to which a system, component, or process meets customer or user needs or expectations (Joseph M. Juran)"

- *Here, emphasis is on a satisfied customer whatever it takes.*

WHAT IS SOFTWARE QUALITY?

- Pressman believes that 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"

PEOPLE'S QUALITY EXPECTATIONS

- In general, people's quality expectations for software systems they use and rely upon are two-fold:
  - The software systems must do what they are supposed to do. In other words, they must *do* ***the right things.***
    - For example, an airline reservation system is supposed to handle reservations, not intended to fly airplanes automatically
  - They must perform these specific tasks correctly or satisfactorily. In other words, they must *do* ***the things right.***
    - The system should help travel agents or individual travelers make valid reservations within a pre-specified time limit, instead of making invalid ones

REASONS SOFTWARE QUALITY IS IMPORTANT

- Safety
- Cost
- Customer satisfaction
- Future value

SOME HISTORICAL FACTS

- Software bugs caused:
  - A massive blackout cut off electricity to 50 million people in eight US states and Canada
  - Technical problems with the baggage system at Heathrow's terminal 5 caused thousands of the passengers to wait for their baggage
- A study commissioned by the National Institute of Standards and Technology found that software bugs cost the U.S. economy about $59.5 billion per year

WHY SOFTWARE QUALITY ENGINEERING?

- Why software?
  - Because in contemporary social life software, systems and services rendered by software are ubiquitous, beginning with the watches we wear, ending with nuclear electricity plants or spaceships.
- Why quality?
  - Because if these instances of software work without the required quality we may be late, dead, or lost in space.
- Why engineering?
  - As in every technical domain, it is engineering that transforms ideas into products, it is the verified and validated set of "to-dos" that help develop the product that not only has required functionalities but also executes them correctly.

IMPORTANCE W.R.T. PROCESS

- Specifications
  - The correctness, completeness, and consistency of the requirements model will have a strong influence on the quality of all work products that follow
- Design
  - Every element of the design model should be assessed by the software team to ensure that it exhibits high quality and that the design itself conforms to requirements
- Construction
  - Source code and related work products (e.g., other descriptive information) must conform to local coding standards and exhibit characteristics that will facilitate maintainability
- Conformance
  - Thorough application of all quality assurance activities in the organization, process definition, and improvement

THE ENVIRONMENTS FOR WHICH SQA METHODS ARE DEVELOPED

- Important to note that variety of professionals, such as students, amateur developers, professionals in engineering, economics, management and other fields, software teams, departments of large and small industrial applications and all those who participate in these activities are required to deal with software quality problems ("bugs").

SOFTWARE QUALITY ATTRIBUTES(PARAMETERS)

- Portability
- Usability
- Reusability
- Correctness
- Maintainability

THE CHARACTERISTICS OF THE SQA ENVIRONMENT PROCESS

- Contractual conditions
- Subjection to customer-supplier relationship
- Requirement for teamwork
- Need for cooperation and coordination with other development teams
- Need for interfaces with other software systems
- Need to continue carrying out a project while the team changes
- Need to continue maintaining the software system for years

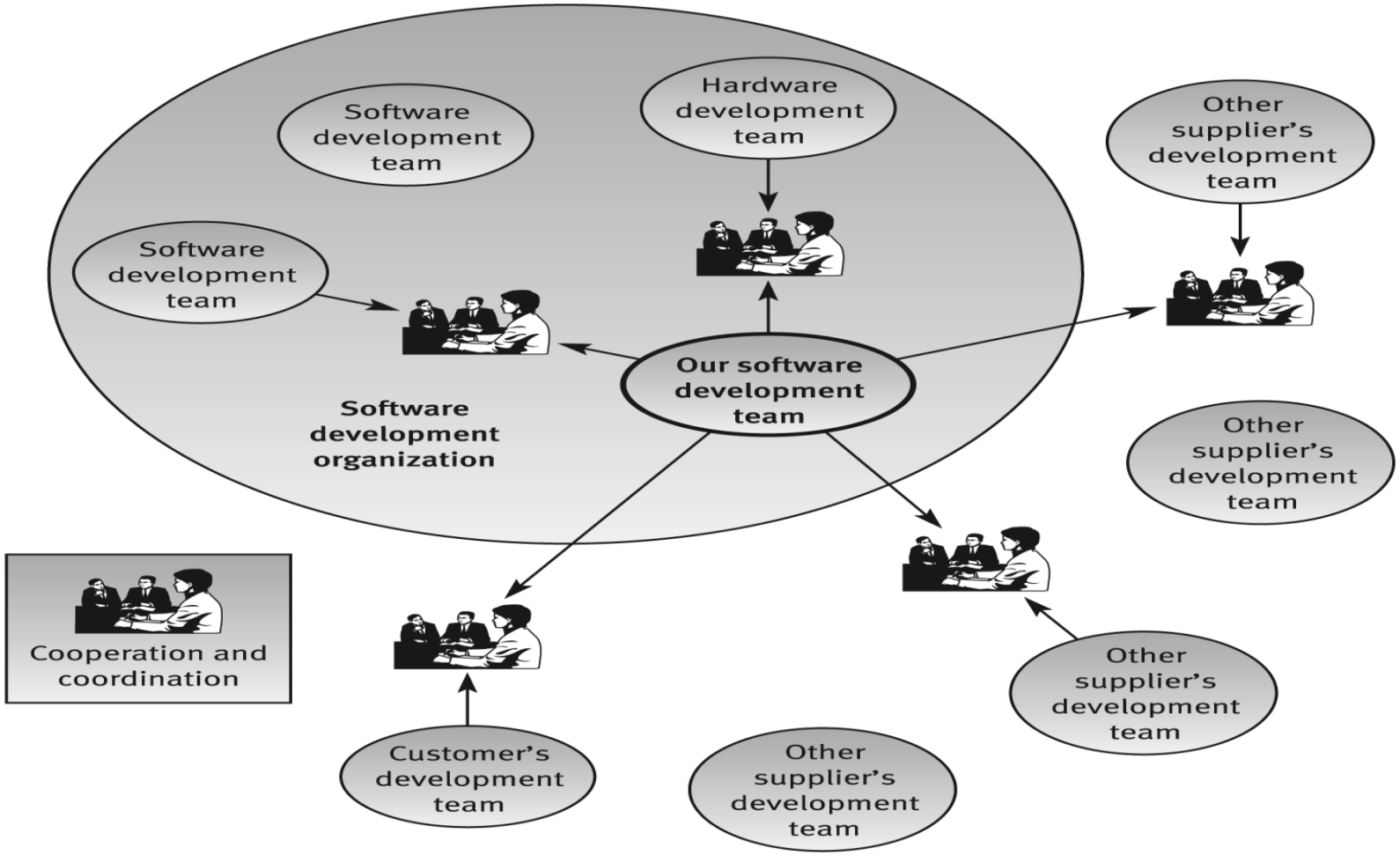
THE CHARACTERISTICS OF THE SQA ENVIRONMENT PROCESS

- Contractual Conditions
  - As a result of the commitments and conditions defined in the contract between the software developer and the customer, the activities of software development and maintenance need to cope with:
    - A defined list of functional requirements that the developed software and its maintenance need to fulfill.
    - The project budget.
    - The project timetable
- Subjection to customer-supplier relationship
  - Continuous cooperation with the customer
    - Request for changes, criticism/feedback, approval of changes
  - Such relationships do not usually exist when software is developed by non-software professionals

THE CHARACTERISTICS OF THE SQA ENVIRONMENT PROCESS

- Requirement for teamwork
  - The following factors usually motivate the establishment of a project team rather than assigning the project to one professional
    - Timetable (schedule) requirements
    - The need for a variety of specializations to carry out the project.
    - The desire to benefit from professionals' mutual support and review for the enhancement of project quality.
- Cooperation and coordination with other development teams
  - Cooperation may be required with:
    - Other software development teams in the same organization.
    - Hardware development teams in the same organization.
    - Software and hardware development teams of other suppliers.
    - Customer software and hardware development teams that take part in the project's development.

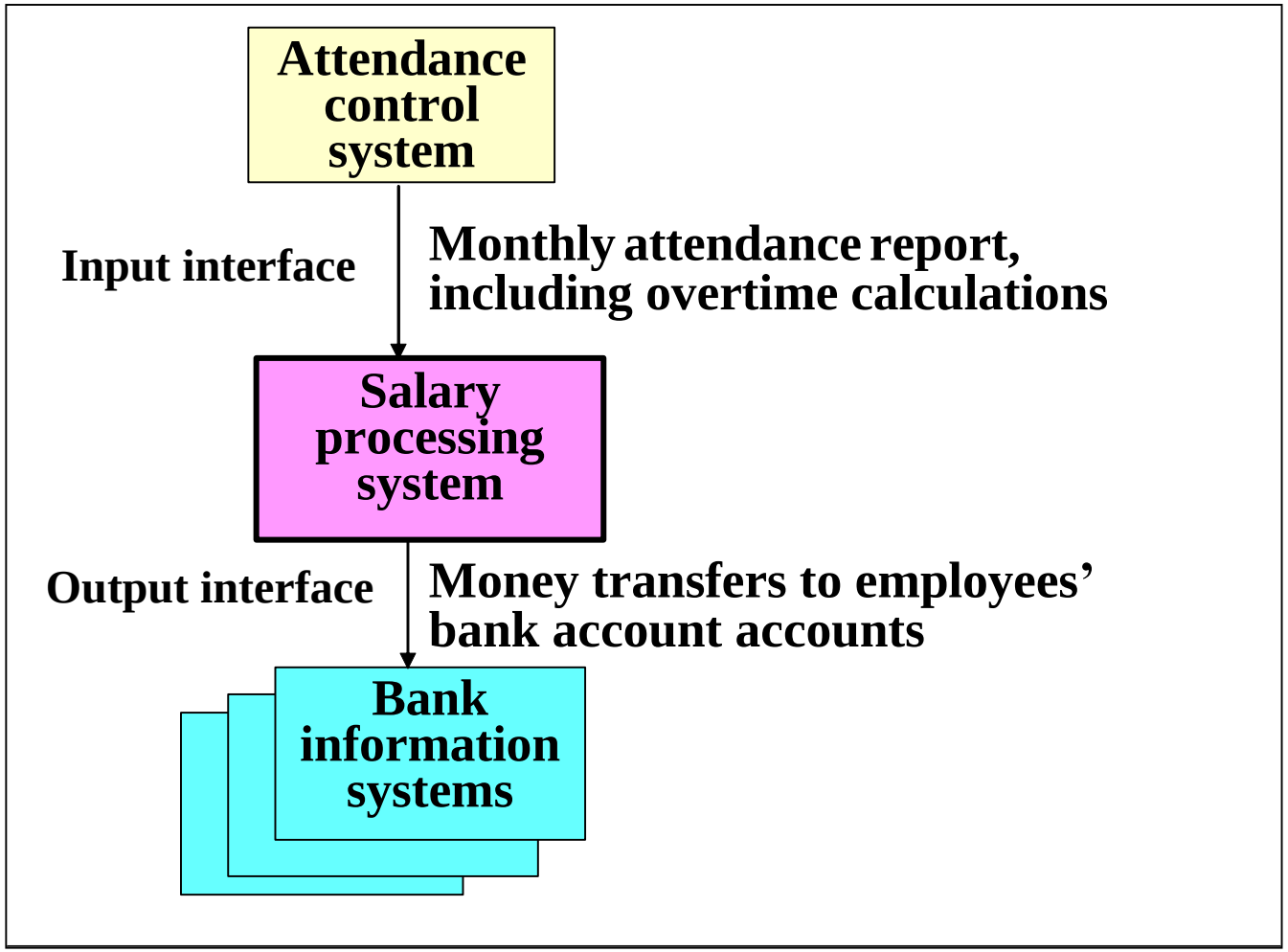
COOPERATION AND COORDINATION SCHEME FOR A SOFTWARE DEVELOPMENT PROJECT TEAM



THE CHARACTERISTICS OF THE SQA ENVIRONMENT PROCESS

- Interfaces with other software systems
  - One can identify the following main types of interfaces:
    - Input interfaces, where other software systems transmit data to your software system.
    - Output interfaces, where your software system transmits processed data to other software systems.
    - Input and output interfaces to the machine's control board, as in medical and laboratory control systems, metal processing equipment, etc.
- Need to continue carrying out a project while the team changes
  - It is quite common for team members to leave the team during the project development period. The team leader then has to replace the departing team member either by another employee or by a newly recruited employee.

THE SALARY SOFTWARE SYSTEM – AN EXAMPLE OF SOFTWARE INTERFACES



THE CHARACTERISTICS OF THE SQA ENVIRONMENT PROCESS

- Need to continue maintaining the software system for years
  - Customers who develop or purchase a software system expect to continue utilizing it for a long period, usually for 5–10 years. During the service period, the need for maintenance will eventually arise. In most cases, the developer is required to supply these services directly.
  - Internal "customers", in cases where the software has been developed in-house, share the same expectation regarding the software maintenance during the service period of the software system.

QUALITY PERSPECTIVES AND EXPECTATIONS

- Quality is a complex concept - it means different things to different people, and it is highly context dependent .
- In Kitchenham & Pfleeger (1996):
  - Transcendental view: seen/not-defined
  - User view: fitness for purpose(task context)
  - Manufacturing view: conformance to standards
  - Product view: inherent characteristics
  - Value-based view: willingness to pay, The value-based view makes a trade-off between cost and quality

PEOPLE'S ROLES AND RESPONSIBILITIES

- About software quality, people may have different views and expectations based on their roles and responsibilities
- Mainly two groups:
  - Consumers of software products and services
    - Customers, users, non-human users (other systems)
    - External view (black box)
    - Quality expectations:
      - Can be different for different users (usability for novice and reliability for sophisticated users)
      - Correctness, performance, maintainability
  - Producers of software products and services
    - anyone involved with the development, management, maintenance, marketing, and service of software products
    - Internal view (white box)
    - Quality expectations:
      - Fulfil contractual obligations
      - Design, size, complexity, change
      - Can be different according to roles for example modifiability may be paramount for people involved with software service, maintainability for maintenance personnel, portability for third-party or software packaging service providers

ACTIVITY

- Consider any two your daily activities in which you use the computers and underlying software. Perform an overall assessment on how important software quality is to your daily activities.

QUALITY FRAMEWORKS AND ISO-9126

QUALITY FRAMEWORKS AND ISO-9126

- ISO-9126—hierarchical framework for quality definition, organized into quality characteristics and sub-characteristics with six top-level quality characteristics/dimensions:
  - Functionality: what is needed?
    - Suitability, Accuracy, Interoperability, Security
  - Reliability: function correctly
    - Maturity, fault tolerance, recoverability
  - Usability: effort to use
    - Understandability, learnability, operability
  - Efficiency: resource needed
    - Time, resource
  - Maintainability: correct/improve/adapt
    - Analyzability, changeability, stability, testability
  - Portability: one environment to another
    - Adaptability, installability, conformance, replaceability

ALTERNATIVE FRAMEWORKS AND FOCUS ON CORRECTNESS

- Quality measure for IBM,s software products
  - CUPRIMDS (capability, usability, performance, reliability, installation, maintenance, documentation, and service)
- Quality attributes for Web-based applications
  - reliability, usability, and security—primary attributes
  - availability, scalability, maintainability, and time to market—secondary attributes
    - performance (or efficiency) and reliability would take precedence over usability and maintainability for real-time software products. On the contrary, it might be the other way round for mass market products for end users.
- Correctness is typically the most important aspect of quality for situations where daily life or business depends on the software

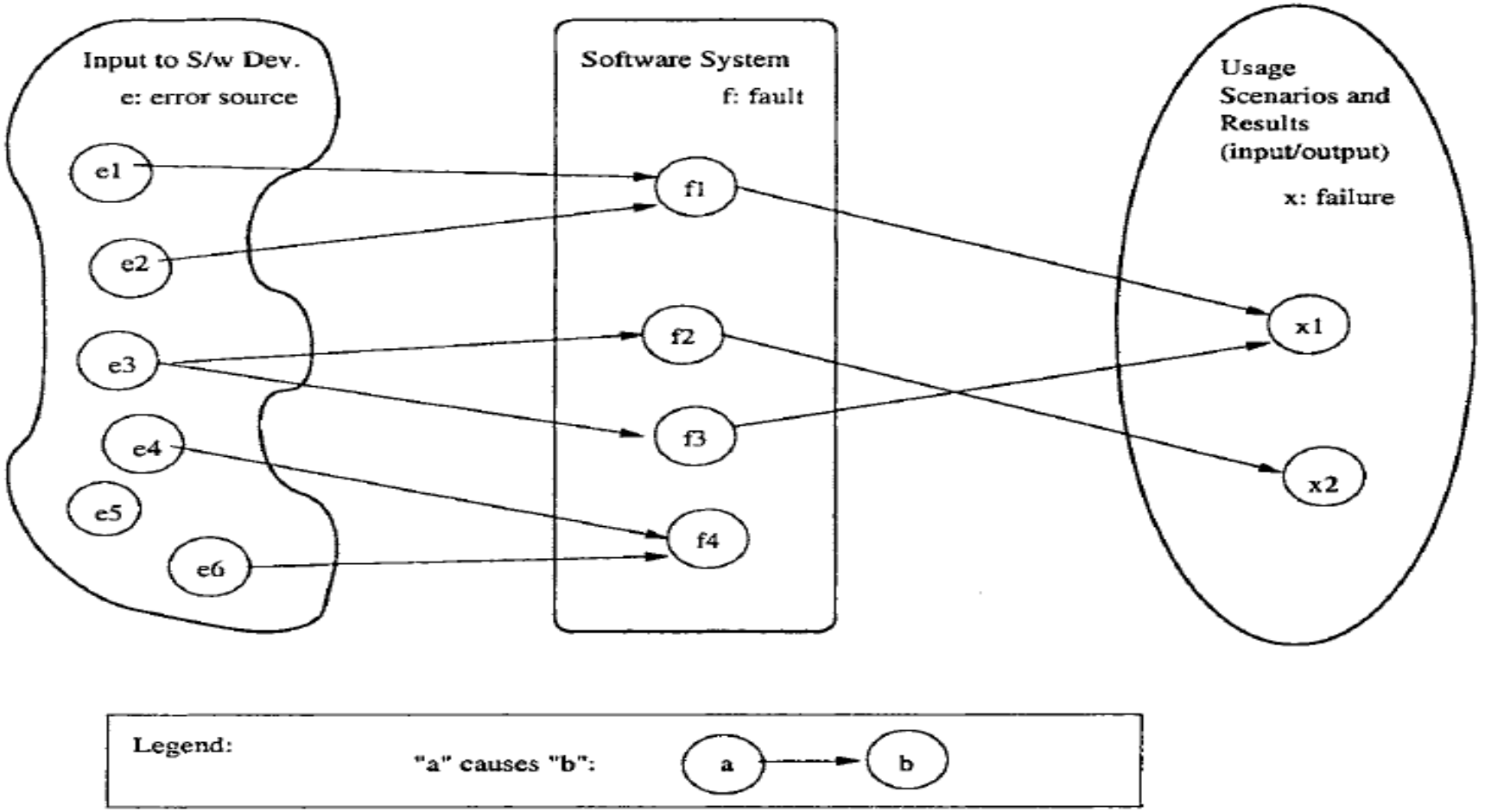
CORRECTNESS AND DEFECTS

- Error:
  - incorrect/missing human action – conceptual mistakes
- Fault:
  - An incorrect step, process, or data definition in a computer program.
- Failure:
  - The inability of a system or component to perform its required functions within specified performance requirements.
- Defect (bug):
  - All the above three terms collectively

CORRECTNESS AND DEFECTS

- We also extend errors to include error sources or the root causes for the missing or incorrect actions, such as human misconceptions, misunderstandings, etc.
- Failures, faults, and errors are collectively referred to as defects in literature.
- Software problems or defects, are also commonly referred to as "bugs". However, the term bug is never precisely defined
- Moreover, we use defect detection and removal for the overall concept and activities related to what many people commonly call "debugging".
  - Specific activities related to defect discovery, including testing, inspection, etc.
  - Specific follow-up activities after defect discovery, including defect diagnosis, analysis, fixing, and re-verification

ILLUSTRATION OF CONCEPTS RELATED TO DEFECTS



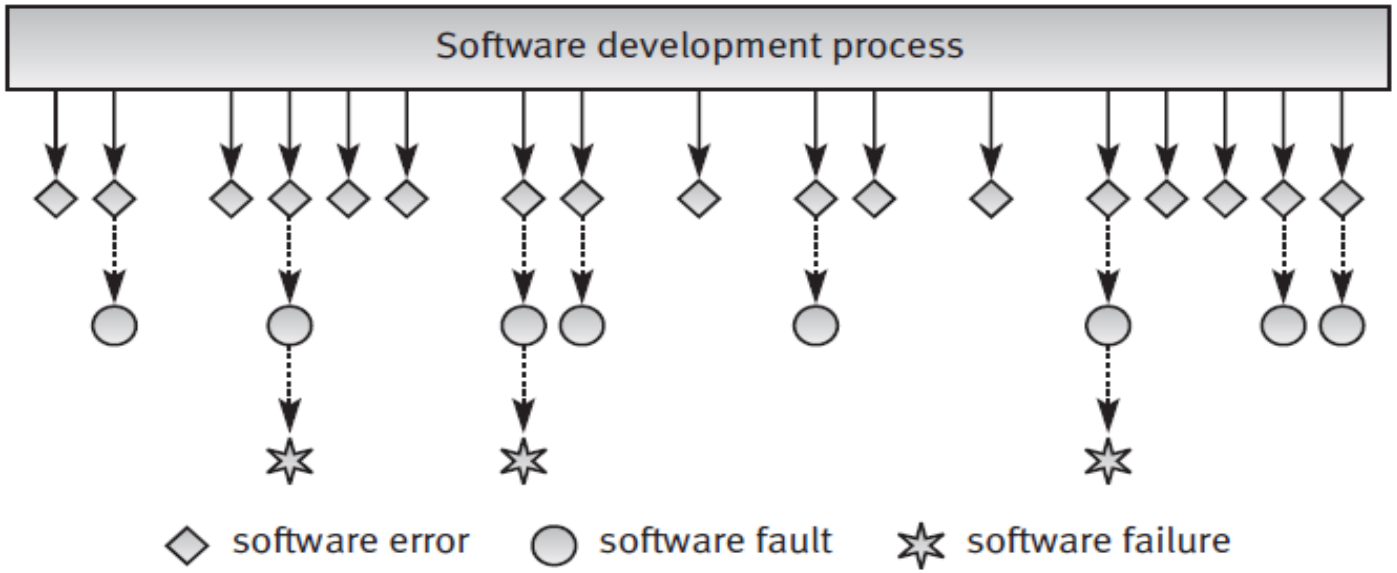
CONCEPTS RELATED TO DEFECTS

errors → faults → failures

- The relationship is not necessarily 1-to- 1:
- A single error may cause many faults, such as in the case that a wrong algorithm is applied in multiple modules and causes multiple faults, and a single fault may cause many failures in repeated executions.
- Conversely, the same failure may be caused by several faults, such as an interface or interaction failure involving multiple modules, and the same fault may be there due to different errors.
- Sometimes, an error source, such as e5, may not cause any fault injection, and a fault, such as f4, may not cause any failure, under the given scenarios or circumstances. Such faults are typically called dormant or latent faults, which may still cause problems under a different set of scenarios or circumstances.

CONCEPTS RELATED TO DEFECTS

- It should be noted that only a portion of the software faults, and in some cases only a small portion of them, will turn into software failures in either the early or later stages of the software's application.
- Other software faults will remain hidden, invisible to the software users, yet capable of being activated when the situation changes.



CORRECTNESS ORIENTED PROPERTIES ACCORDING TO QUALITY VIEWS AND ATTRIBUTES

View	Attribute	
	Correctness	Others
Consumer/ External (user & customer)	Failure-related properties	Usability Maintainability Portability Performance Installability Readability etc. (-ilities)
Producer/ Internal (developer, manager, tester, etc.)	Fault-related properties	Design Size Change Complexity, etc.

CORRECTNESS ORIENTED PROPERTIES AND MEASUREMENTS

- Failure properties and direct failure measurement
  - Failure properties include information about the specific failures, what they are, how they occur, etc. These properties can be measured directly by examining failure count, distribution, density etc.
- Failure likelihood and reliability measurement
  - How often or how likely a failure is going to occur is of critical concern to software users and customers. This likelihood is captured in various reliability measures, where reliability can be defined as the probability of failure-free operations for a specific time period or for a given set of inputs

CORRECTNESS ORIENTED PROPERTIES AND MEASUREMENTS

- Failure severity measurement and safety assurance
  - Accidents, which are defined to be failures with severe consequences, need to be avoided, contained, or dealt with to ensure the safety for the personnel involved and to minimize other damages

DEFECTS IN THE CONTEXT OF QA AND QUALITY ENGINEERING

- Three generic ways to deal with defects include:
  - defect prevention
  - defect detection and removal
  - defect containment
- Quality engineering can also be viewed as defect management. In addition to the execution of the planned QA activities, quality engineering also includes:
  - quality planning before specific QA activities are carried out
  - measurement, analysis, and feedback to monitor and control the QA activities

CAUSES OF SOFTWARE ERRORS

- Faulty definition of requirements
  - Erroneous definition of requirements
  - Incomplete definition of requirements
  - Inclusion of unnecessary requirements, functions that are not expected to be needed in the near future
- Client-developer communication failures
  - Misunderstanding of (i) client's instructions as stated in the requirement documents, (ii) requirement changes, (iii) client's response to design problem
- Deliberate deviation from requirements
- Logical design errors
  - Process definitions that contain sequencing errors
  - Erroneous definition of boundary conditions
  - Omission of required software states
  - Omission of definitions concerning reactions to illegal operation of the software system.

CAUSES OF SOFTWARE ERRORS

- Coding errors
  - misunderstanding the design documentation, linguistic errors in the programming languages, errors in the application of CASE and other development tools, errors in data selection, and so forth.
- Non-compliance with documentation and coding instructions
  - Team members who need to coordinate their own codes with code modules developed by "non-complying" team members can be expected to encounter more than the usual number of difficulties when trying to understand the software developed by the other team members.
  - Individuals replacing the "non-complying" team member (who has retired or been promoted) will find it difficult to fully understand his or her work.
  - The design review team will find it more difficult to review a design prepared by a non-complying team.
- Shortcomings of the testing process
  - Incomplete test plans; failure to report, document and correct detected errors;
- Procedure errors
- Documentation errors
- Omission of software functions

That is all