UNIT 8
Planning and Monitoring the Process

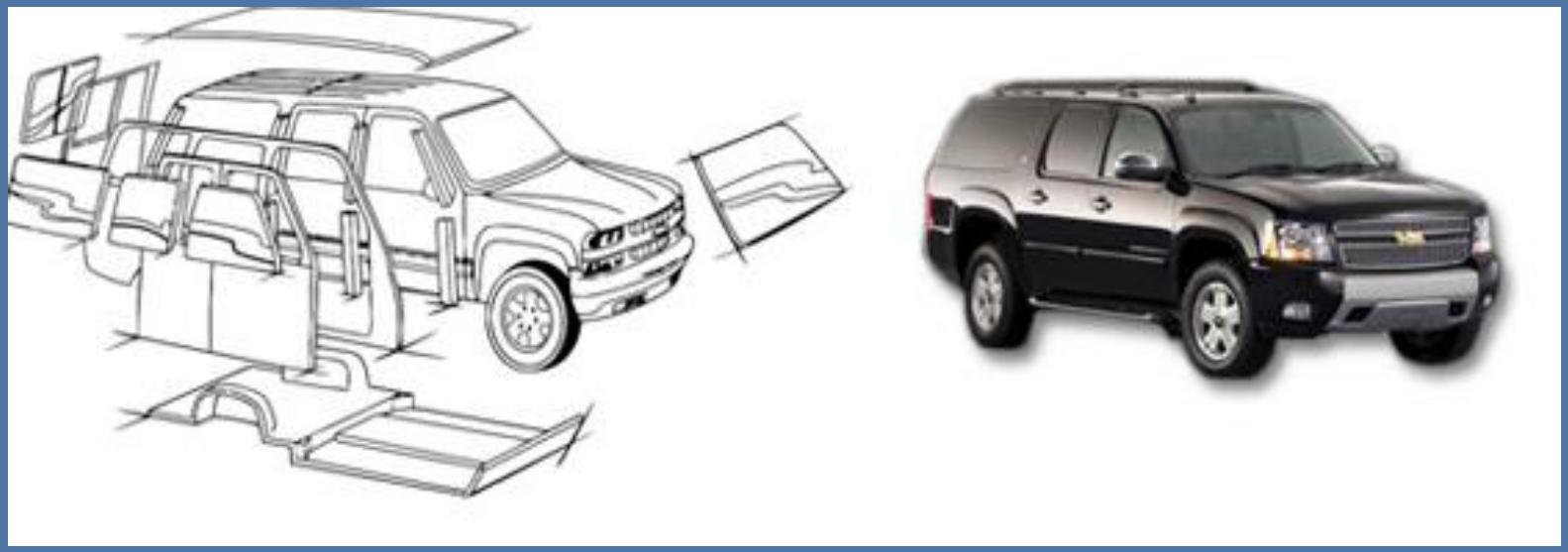
By
Mr. C. R. Belavi
Asst. Professor, HSIT, NIDASOSHI

TOPICS IN UNIT 8

- **Planning and Monitoring the Process, Documenting Analysis and Test:** Quality and process, Test and analysis strategies and plans, Risk planning, Monitoring the process, Improving the process, The quality team, Organizing documents, Test strategy document, Analysis and test plan, Test design specifications documents, Test and analysis reports.

Contents

- Overview
- Quality and Process
- Test and Analysis strategies



Overview

- Any complex process requires planning and monitoring.

- Planning:
  - Scheduling activities (what steps? in what order?)
  - Allocating resources (who will do it?)
  - Devising unambiguous milestones for monitoring
- Monitoring: Judging progress against the plan
  - How are we doing?

A good plan must have *visibility* :
- Ability to monitor each step, and to make objective judgments of progress
- Counter wishful thinking and denial

Quality plan

- It is one aspect of project planning
- It begins at the inception of a project and developed with overall project plan.
- It is developed incrementally {beginning from feasibility study, development, delivery}
- Plan formulation involve risk analysis
- Allocation of responsibility among team members is crucial and difficult part of planning.

Quality and process

- A software plan involves many intertwined concerns from schedule to cost to usability and dependability.
- Ex: Architectural design involving various testability to review structure and build order.
- Cost of detecting and repairing a fault increases as function of time between committing the error and detecting the resultant faults.

• Quality process: Set of activities and responsibilities

– focused primarily on ensuring adequate dependability
– concerned with project schedule or with product usability

• A framework for

– selecting and arranging activities
– considering interactions and trade-offs

• Follows the overall software process in which it is embedded

– Example: waterfall software process –-> “V model”: unit testing starts with implementation and finishes before integration

– Example: XP and agile methods –-> emphasis on unit testing and rapid iteration for acceptance testing by customers

Verification Steps for Intermediate Artifacts

- Internal consistency checks
  – compliance with structuring rules that define "well-formed" artifacts of that type
  – a point of leverage: define syntactic and semantic rules thoroughly and precisely enough that many common errors result in detectable violations
- External consistency checks
  – consistency with related artifacts
  – Often: conformance to a "prior" or "higher-level" specification
- Generation of correctness conjectures
  – Correctness conjectures: lay the groundwork for external consistency checks of other work products
  – Often: motivate refinement of the current product

Strategies vs Plans

	Strategy	*Plan*
Scope	Organization	Project
Structure and content based on	Organization structure, experience and policy over several projects	Standard structure prescribed in strategy
Evolves	Slowly, with organization and policy changes	Quickly, adapting to project needs

Test and Analysis Strategy

- Lessons of past experience
  – an organizational asset built and refined over time

- Body of explicit knowledge
  – more valuable than islands of individual competence
  – amenable to improvement
  – reduces vulnerability to organizational change (e.g., loss of key individuals)

- Essential for
  – avoiding recurring errors
  – maintaining consistency of the process
  – increasing development efficiency

Considerations in Fitting a Strategy to an Organization

- Structure and size

– example

Distinct quality groups in large organizations, overlapping of roles in smaller organizations• greater reliance on documents in large than small organizations

- Overall process

– example

Cleanroom requires statistical testing and forbids unit testing

– fits with tight, formal specs and emphasis on reliability

XP prescribes “test first” and pair programming

– fits with fluid specifications and rapid evolution

• Application domain

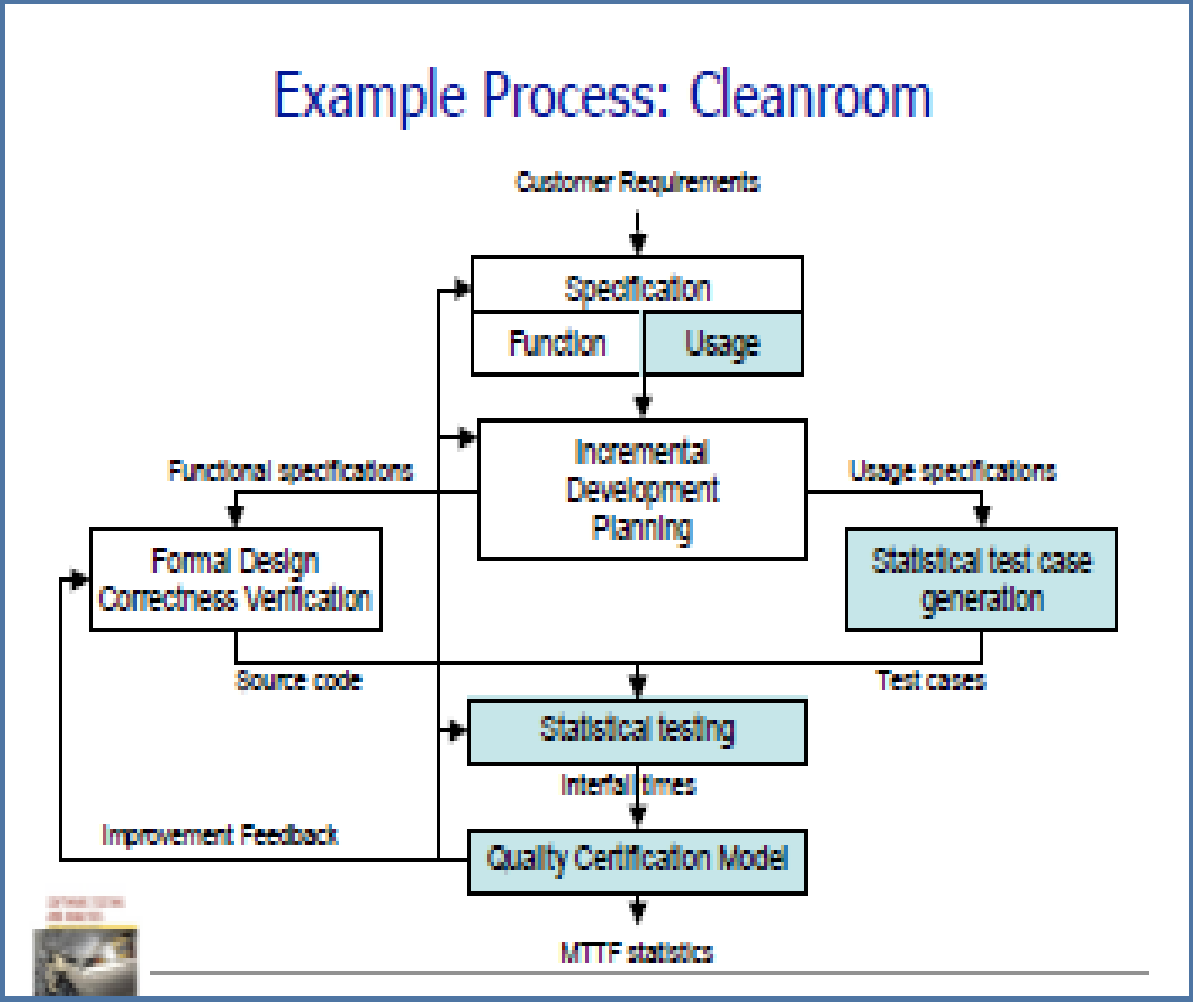
– example

Safety critical domains may impose particular quality objectives and require documentation for certification (e.g,RTCA/DO-178B standard requires MC/DC coverage

Contents

- Cleanroom process Model
- SRET
- Extreme Programming

Cleanroom process Model



Cleanroom process

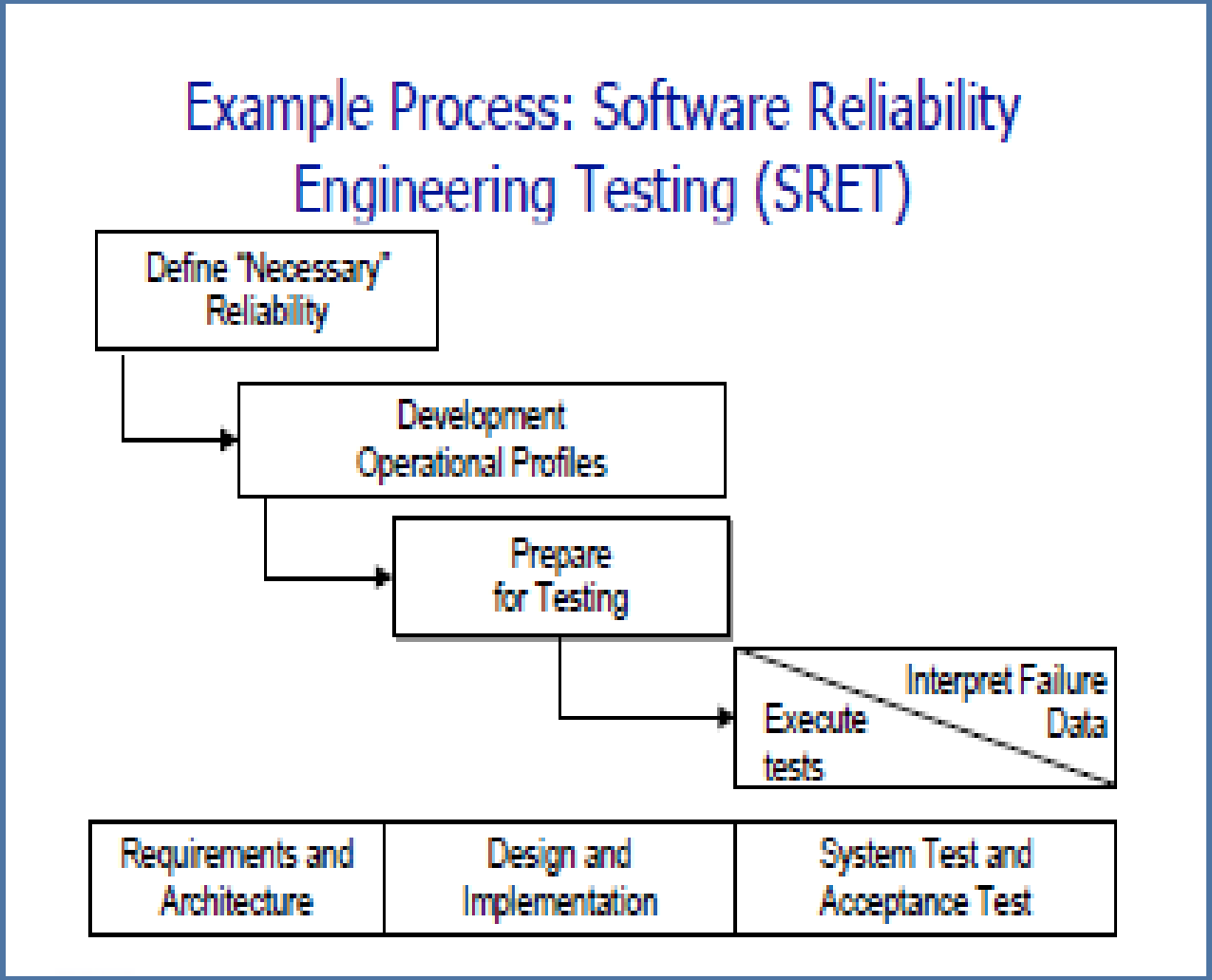
- Introduced by IBM in the year 1980s.
- Cleanroom process involves two teams

1)Development Team.
2)Quality Team.

- Five major activities

1)Specification
2)Planning
3)Design and verification
4)Quality certification
5)feedback

SRET

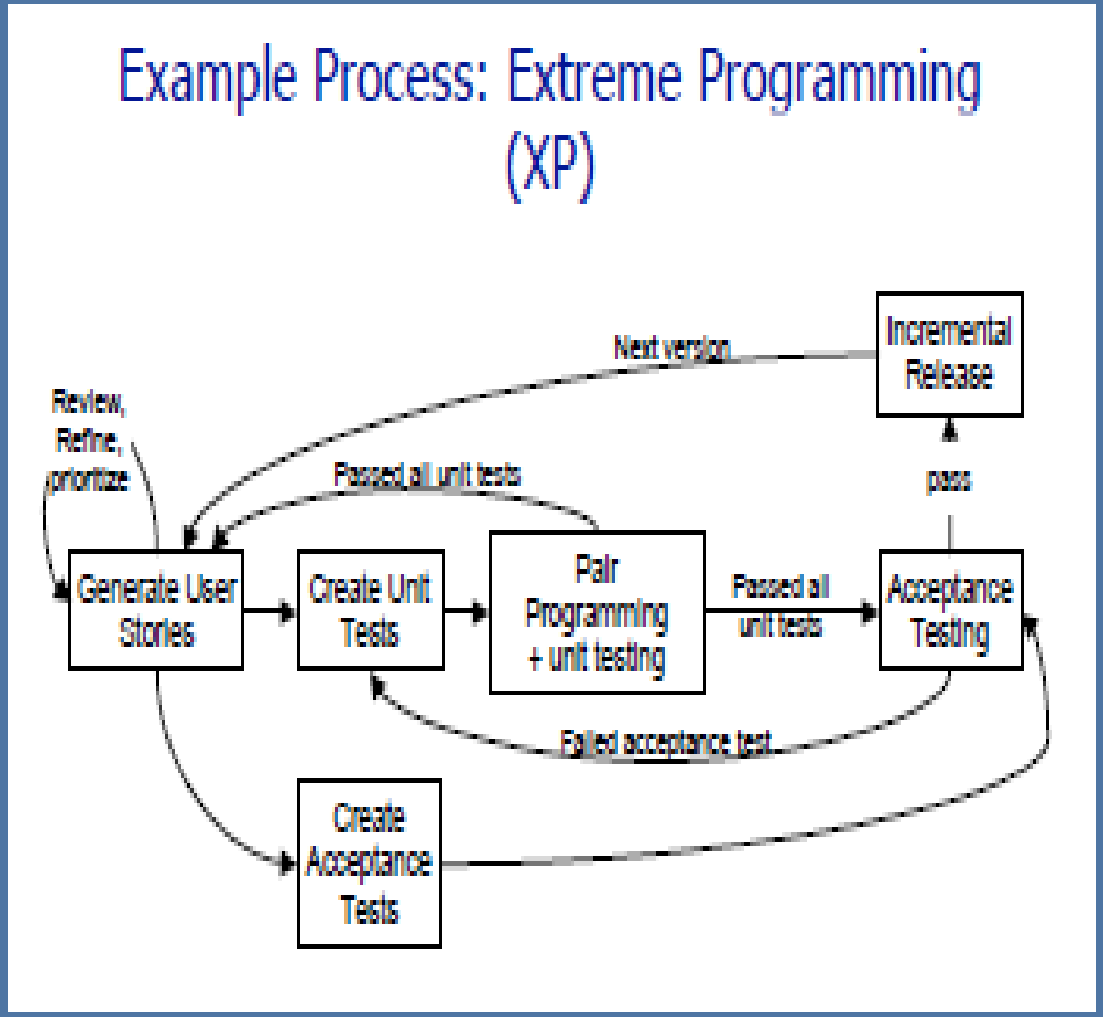

Example Process: Software Reliability Engineering Testing (SRET)
Define "Necessary" Reliability
Development Operational Profiles
Prepare for Testing
Execute tests
Interpret Failure Data
Requirements and Architecture
Design and Implementation
System Test and Acceptance Test

- Software reliability engineering test(SRET)
- Develop at AT&T in 1990's
- Assumes spiral development process after each coil of the spiral rigorous testing activities is done.
- SRET identifies two main types of testing

1)Development testing 2)Certification testing

- SRET includes seven main steps: Two decision Making and Five core steps : Define necessary reliability, develop operational profile, prepare for testing, Execute test, Interpret failure data.

Extreme Programming



Extreme programming XP

- It includes requirement analysis and acceptance testing.

Contents

- Test and Analysis Plans
- Risk Planning

Test and Analysis Plan

- Answer the following questions:
- What quality activities will be carried out?
- What are the dependencies among the quality activities and between quality and other development activities?
- What resources are needed and how will they be allocated?
- How will both the process and the product be monitored?

Main Elements of a Plan

- Items and features to be verified
  - Scope and target of the plan
- Activities and resources
  - Constraints imposed by resources on activities
- Approaches to be followed
  - Methods and tools
- Criteria for evaluating results

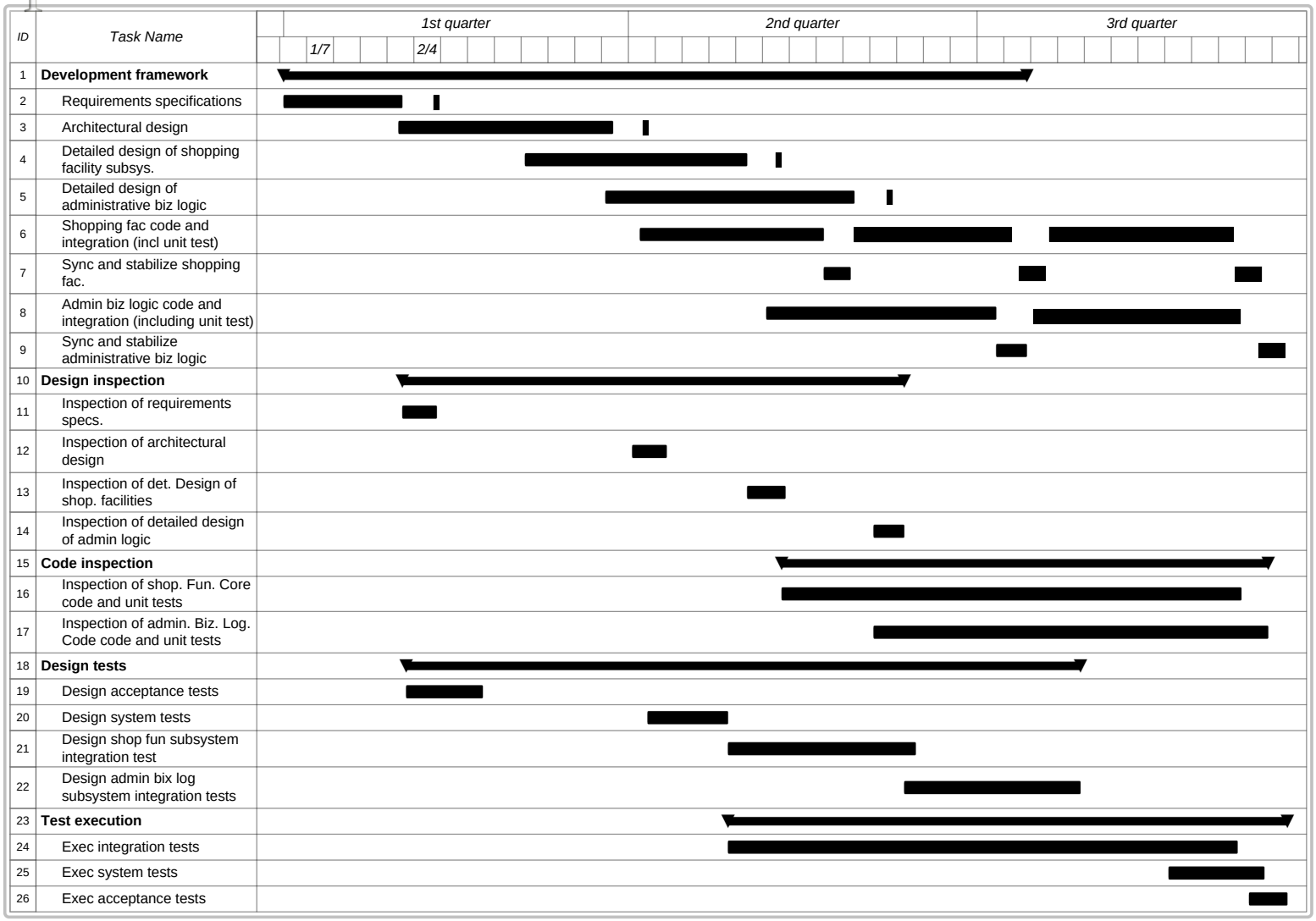
Quality Goals

- Expressed as properties satisfied by the product
  - must include metrics to be monitored during the project
  - *example:* before entering acceptance testing, the product must pass comprehensive system testing with no critical or severe failures
  - not all details are available in the early stages of development
- Initial plan
  - based on incomplete information
  - incrementally refined

Task Schedule

- Initially based on
  - quality strategy
  - past experience
- Breaks large tasks into subtasks
  - refine as process advances
- Includes dependencies
  - among quality activities
  - between quality and development activities
- Guidelines and objectives:
  - schedule activities for steady effort and continuous progress and evaluation without delaying development activities
  - schedule activities as early as possible
  - increase process visibility (how do we know we're on track?)

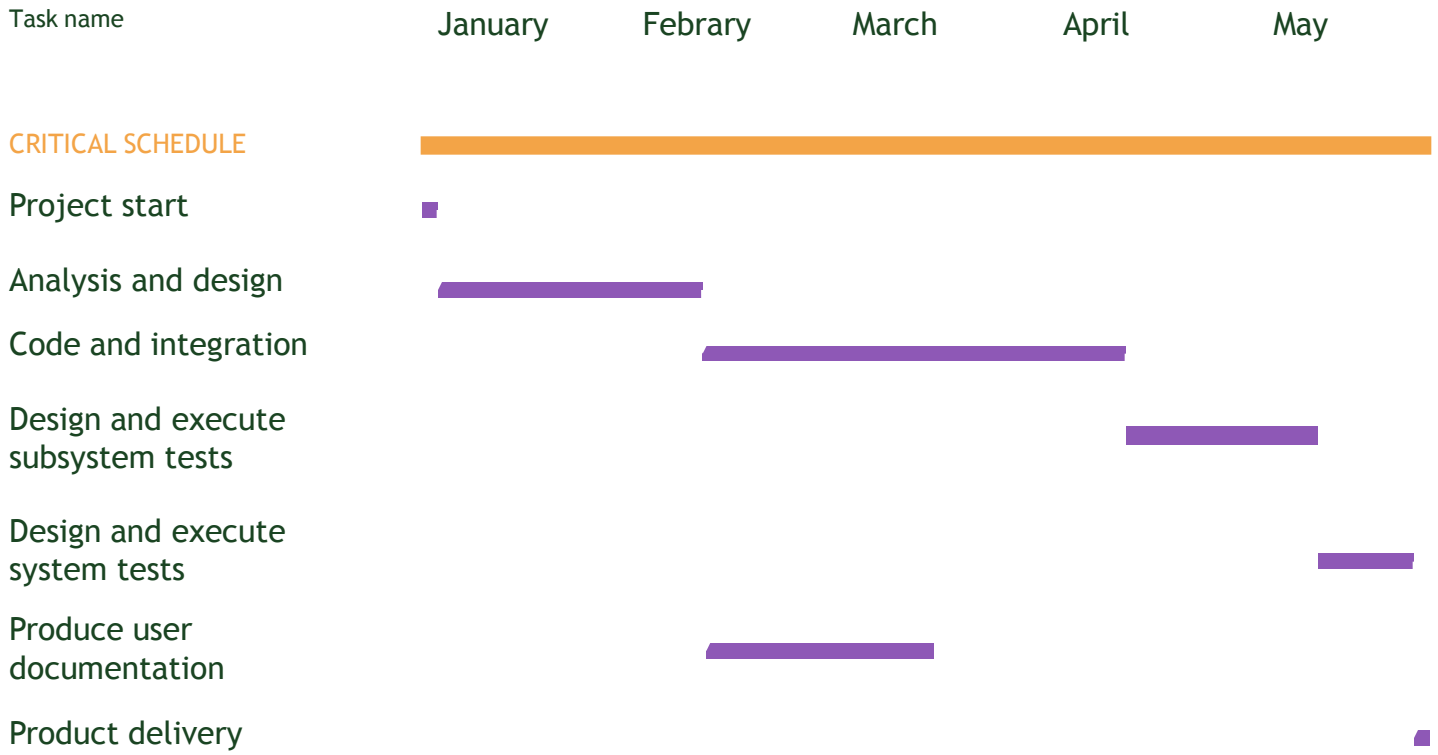
Sample Schedule



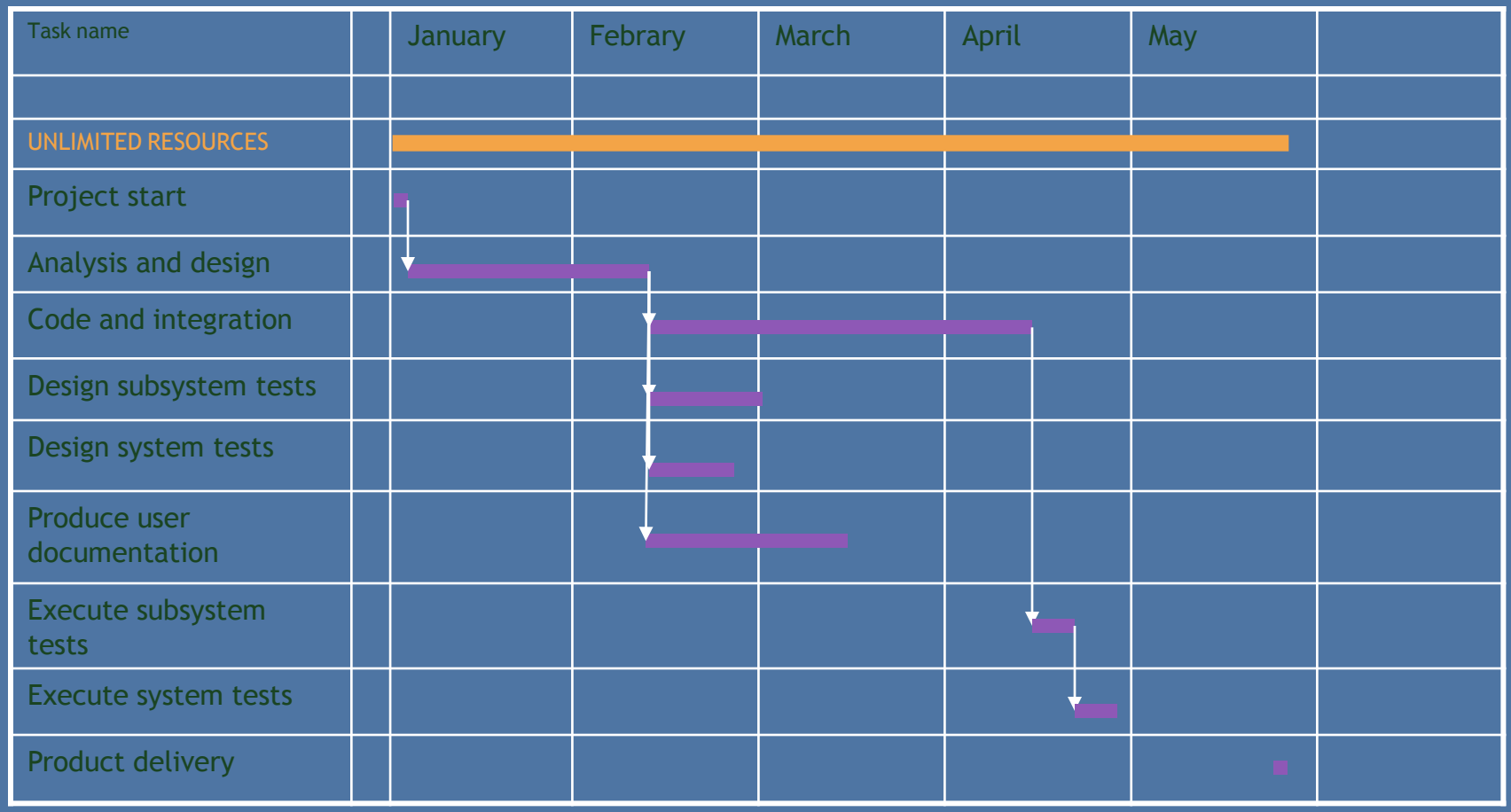
Schedule Risk

- *critical path* = chain of activities that must be completed in sequence and that have maximum overall duration
  - Schedule critical tasks and tasks that depend on critical tasks as early as possible to
    - provide schedule slack
    - prevent delay in starting critical tasks
- *critical dependence* = task on a critical path scheduled immediately after some other task on the critical path
  - May occur with tasks outside the quality plan (part of the project plan)
  - Reduce critical dependences by decomposing tasks on critical path, factoring out subtasks that can be performed earlier

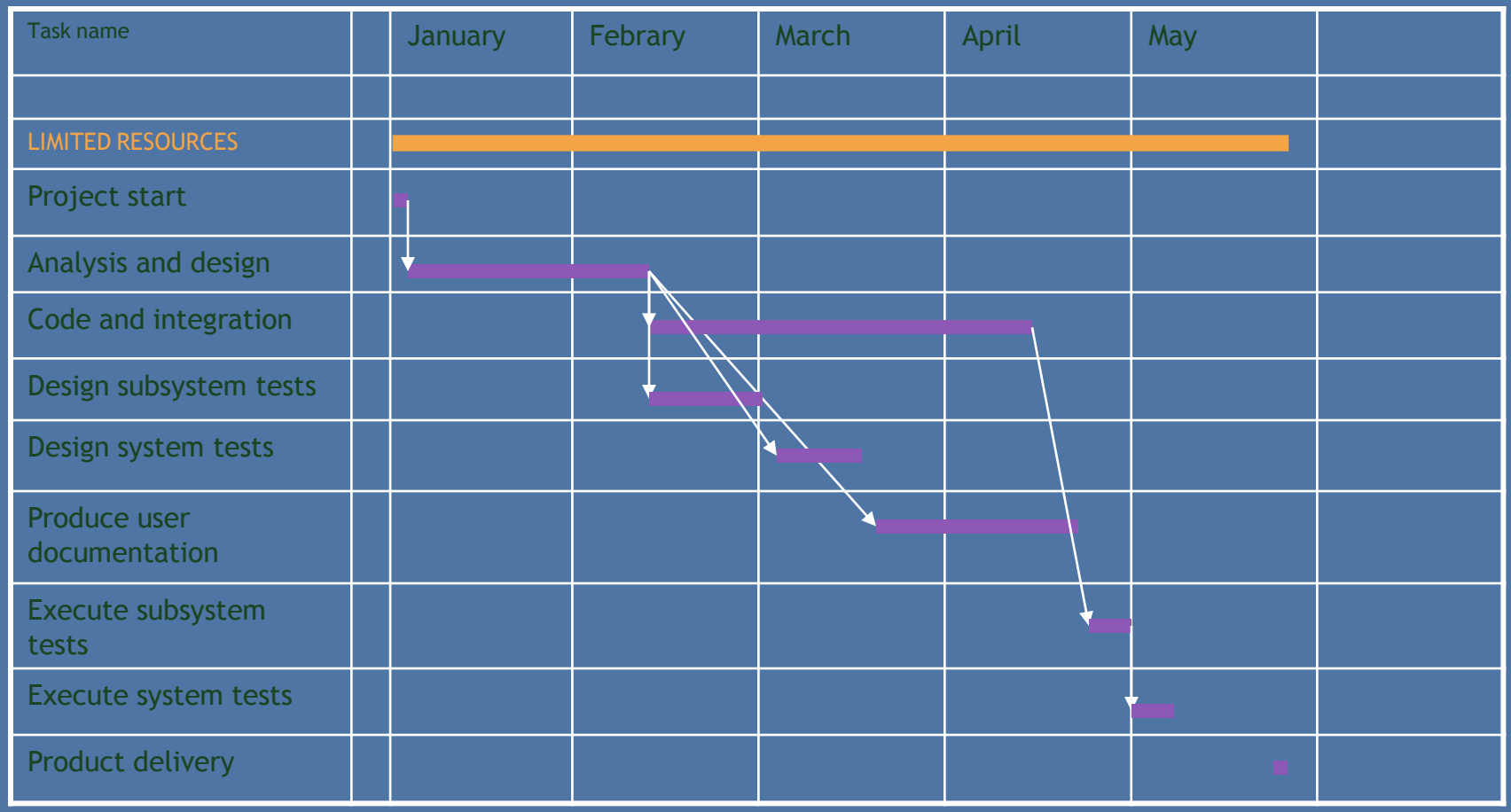
Reducing the Impact of Critical Paths



Reducing the Impact of Critical Paths



Reducing the Impact of Critical Paths



Contents

- Risk planning
- Monitoring the process
- Improving the Process
- The Quality Team

Risk Planning

- Risks cannot be eliminated, but they can be assessed, controlled, and monitored
- Generic management risk
  - personnel
  - technology
  - schedule
- Quality risk
  - development
  - execution
  - requirements

Personnel

Example Risks

- Loss of a staff member
- Staff member under-qualified for task

Control Strategies

- cross training to avoid over-dependence on individuals
- continuous education
- identification of skills gaps early in project
- competitive compensation and promotion policies and rewarding work
- including training time in project schedule

Technology

Example Risks

- High fault rate due to unfamiliar COTS component interface
- Test and analysis automation tools do not meet expectations

Control Strategies

- Anticipate and schedule extra time for testing unfamiliar interfaces.
- Invest training time for COTS components and for training with new tools
- Monitor, document, and publicize common errors and correct idioms.
- Introduce new tools in lower-risk pilot projects or prototyping exercises

Schedule

Example Risks

- Inadequate unit testing leads to unanticipated expense and delays in integration testing
- Difficulty of scheduling meetings makes inspection a bottleneck in development

Control Strategies

- Track and reward quality unit testing as evidenced by low fault densities in integration
- Set aside times in a weekly schedule in which inspections take precedence over other meetings and work
- Try distributed and asynchronous inspection techniques, with a lower frequency of face-to-face inspection meetings

Development

Example Risks

- Poor quality software delivered to testing group
- Inadequate unit test and analysis before committing to the code base

Control Strategies

- Provide early warning and feedback
- Schedule inspection of design, code and test suites
- Connect development and inspection to the reward system
- Increase training through inspection
- Require coverage or other criteria at unit test level

Test Execution

Example Risks

- Execution costs higher than planned
- Scarce resources available for testing

Control Strategies

- Minimize parts that require full system to be executed
- Inspect architecture to assess and improve testability
- Increase intermediate feedback
- Invest in scaffolding

Requirements

Example Risk

- High assurance critical requirements increase expense and uncertainty

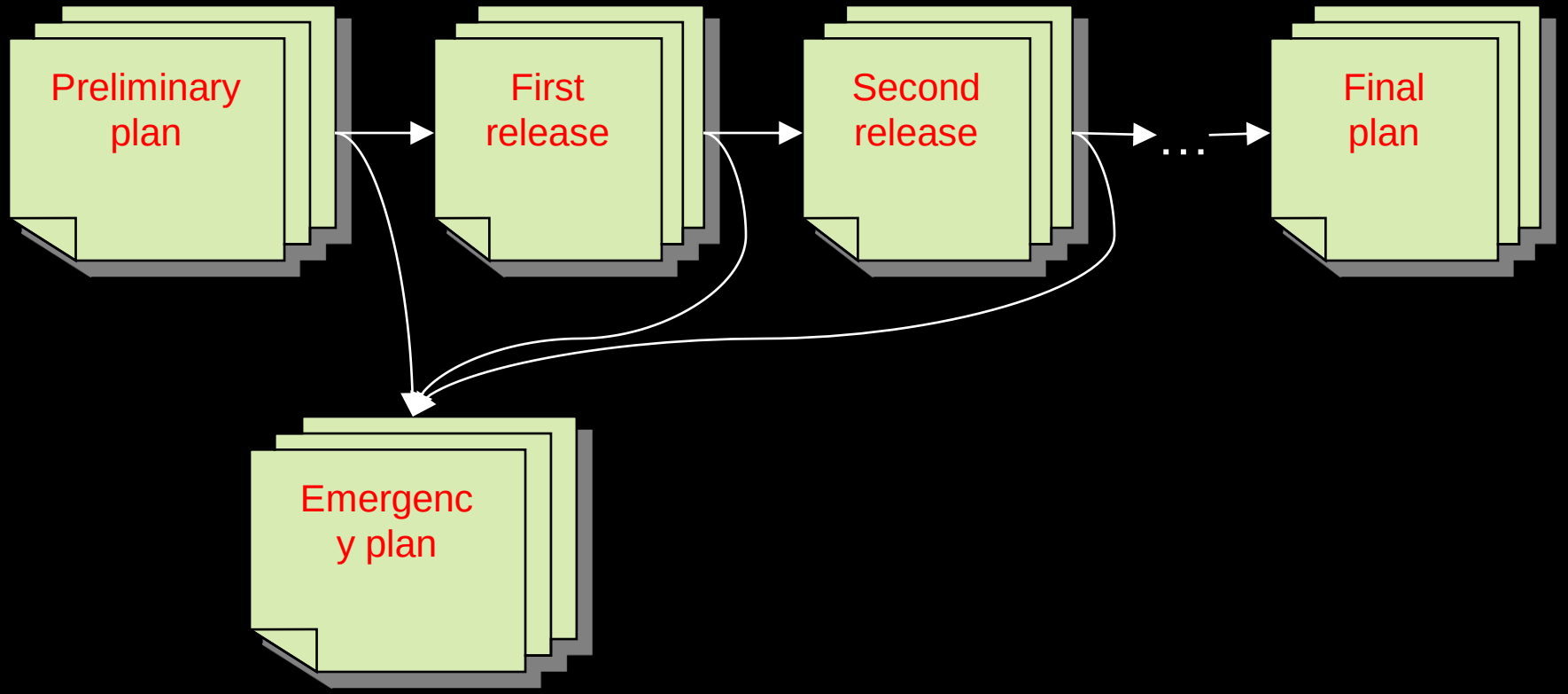
Control Strategies

- Compare planned testing effort with former projects with similar criticality level to avoid underestimating testing effort
- Balance test and analysis
- Isolate critical parts, concerns and properties

Contingency Plan

- Part of the initial plan
  - What could go wrong? How will we know, and how will we recover?
- Evolves with the plan
- Derives from risk analysis
  - Essential to consider risks explicitly and in detail
- Defines actions in response to bad news
  - Plan B at the ready (the sooner, the better)

Evolution of the Plan



Process Monitoring

- Identify deviations from the quality plan as early as possible and take corrective action
- Depends on a plan that is
  - realistic
  - well organized
  - sufficiently detailed with clear, unambiguous milestones and criteria
- A process is *visible* to the extent that it can be effectively monitored

Evaluate Aggregated Data by Analogy

Orthogonal Defect Classification (ODC)

- Accurate classification schema
  - for very large projects
  - to distill an unmanageable amount of detailed information
- Two main steps
  - Fault classification
    - when faults are detected
    - when faults are fixed
  - Fault analysis

ODC Fault Classification

When faults are detected

- *activity* executed when the fault is revealed
- *trigger* that exposed the fault
- *impact* of the fault on the customer

When faults are fixed

- *Target:* entity fixed to remove the fault
- *Type:* type of the fault
- *Source:* origin of the faulty modules (in-house, library, imported, outsourced)
- *Age* of the faulty element (new, old, rewritten, re-fixed code)

ODC activities and triggers

- Review and Code Inspection
  - Design Conformance:
  - Logic/Flow
  - Backward Compatibility
  - Internal Document
  - Lateral Compatibility
  - Concurrency
  - Language Dependency
  - Side Effects
  - Rare Situation
- Structural (White Box) Test
  - Simple Path
  - Complex Path
- Functional (Black box) Test
  - Coverage
  - Variation
  - Sequencing
  - Interaction
- System Test
  - Workload/Stress
  - Recovery/Exception
  - Startup/Restart
  - Hardware Configuration
  - Software Configuration
  - Blocked Test

ODC impact

- Installability
- Integrity/Security
- Performance
- Maintenance
- Serviceability
- Migration
- Documentation
- Usability
- Standards
- Reliability
- Accessibility
- Capability
- Requirements

ODC Fault Analysis (example 1/4)

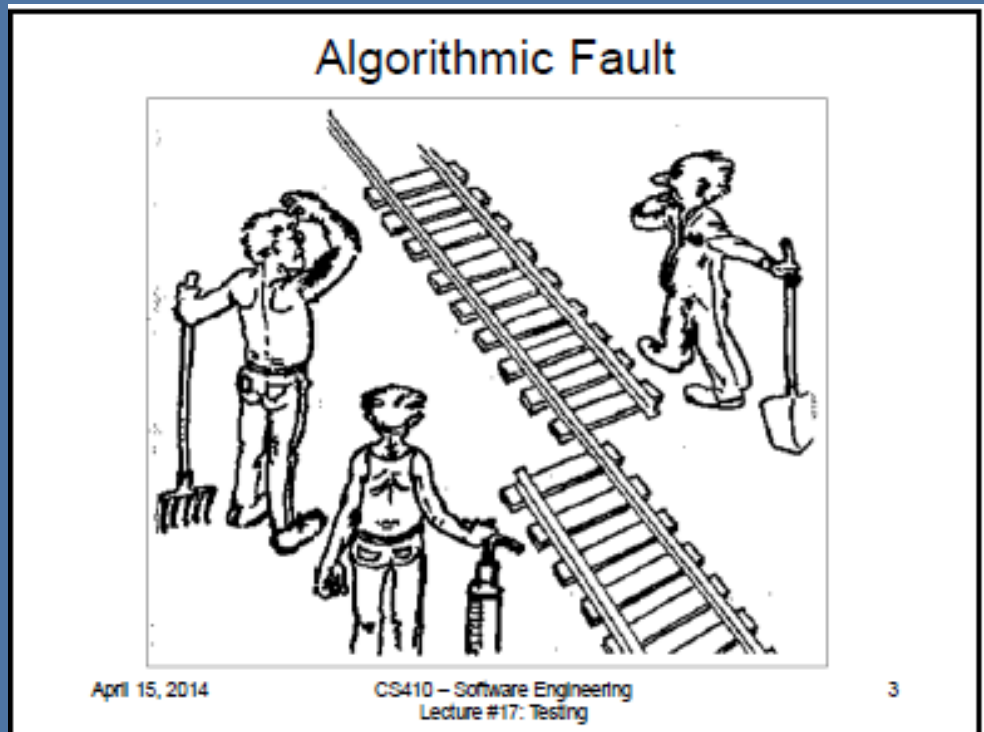
- Distribution of fault types versus activities
  - Different quality activities target different classes of faults
  - example:
    - algorithmic faults are targeted primarily by unit testing.
      - a high proportion of faults detected by unit testing should belong to this class
    - proportion of algorithmic faults found during unit testing
      - unusually small
      - larger than normal

      ⇒ unit tests may not have been well designed
    - proportion of algorithmic faults found during unit testing unusually large

    ⇒ integration testing may not focused strongly enough on interface faults

Algorithmic Faults

- Missing initialization
- Branching errors (too soon, too late)
- Missing test for nil



ODC Fault Analysis (example 2/4)

- Distribution of triggers over time during field test
  - Faults corresponding to simple usage should arise early during field test, while faults corresponding to complex usage should arise late.
  - The rate of disclosure of new faults should asymptotically decrease
  - Unexpected distributions of triggers over time may indicate poor system or acceptance test
    - Triggers that correspond to simple usage reveal many faults late in acceptance testing

    ⇒ The sample may not be representative of the user population
    - Continuously growing faults during acceptance test

    ⇒ System testing may have failed

ODC Fault Analysis (example 3/4)

- Age distribution over target code
  - Most faults should be located in new and rewritten code
  - The proportion of faults in new and rewritten code with respect to base and re-fixed code should gradually increase
  - Different patterns

  ⇒ may indicate holes in the fault tracking and removal process

  ⇒ may indicate inadequate test and analysis that failed in revealing faults early
  - Example
    - increase of faults located in base code after porting

    ⇒ may indicate inadequate tests for portability

ODC Fault Analysis (example 4/4)

- Distribution of fault classes over time
  - The proportion of missing code faults should gradually decrease
  - The percentage of extraneous faults may slowly increase, because missing functionality should be revealed with use
    - increasing number of missing faults

    ⇒ may be a symptom of instability of the product
    - sudden sharp increase in extraneous faults

    ⇒ may indicate maintenance problems

Improving the Process

- Many classes of faults that occur frequently are rooted in process and development flaws
  - examples
    - Shallow architectural design that does not take into account resource allocation can lead to resource allocation faults
    - Lack of experience with the development environment, which leads to misunderstandings between analysts and programmers on rare and exceptional cases, can result in faults in exception handling.
- The occurrence of many such faults can be reduced by modifying the process and environment
  - examples
    - Resource allocation faults resulting from shallow architectural design can be reduced by introducing specific inspection tasks
    - Faults attributable to inexperience with the development environment can be reduced with focused training

Improving Current and Next Processes

- Identifying weak aspects of a process can be difficult
- Analysis of the fault history can help software engineers build a feedback mechanism to track relevant faults to their root causes
  - Sometimes information can be fed back directly into the current product development
  - More often it helps software engineers improve the development of future products

Root cause analysis (RCA)

- Technique for identifying and eliminating process faults
  - First developed in the nuclear power industry; used in many fields.
- Four main steps
  - *What* are the faults?
  - *When* did faults occur? When, and when were they found?
  - *Why* did faults occur?
  - *How* could faults be prevented?

What are the faults?

- Identify a class of important faults
- Faults are categorized by
  - severity = impact of the fault on the product
  - Kind
    - No fixed set of categories; Categories evolve and adapt
    - Goal:
      - Identify the few most important classes of faults and remove their causes
      - Differs from ODC: Not trying to compare trends for different classes of faults, but rather *focusing* on a few important classes

Fault Severity

Level	Description	Example
Critical	The product is unusable	The fault causes the program to crash
Severe	Some product features cannot be used, and there is no workaround	The fault inhibits importing files saved with a previous version of the program, and there is no workaround
Moderate	Some product features require workarounds to use, and reduce efficiency, reliability, or convenience and usability	The fault inhibits exporting in Postscript format. Postscript can be produced using the printing facility, but with loss of usability and efficiency
Cosmetic	Minor inconvenience	The fault limits the choice of colors for customizing the graphical interface, violating the specification but causing only minor inconvenience

Pareto Distribution (80/20)

- Pareto rule (80/20)
  - in many populations, a few (20%) are vital and many (80%) are trivial
- Fault analysis
  - 20% of the code is responsible for 80% of the faults
    - Faults tend to accumulate in a few modules
      - identifying potentially faulty modules can improve the cost effectiveness of fault detection
    - Some classes of faults predominate
      - removing the causes of a predominant class of faults can have a major impact on the quality of the process and of the resulting product

Why did faults occur?

- Core RCA step
  - trace representative faults back to causes
  - objective of identifying a “root” cause
- Iterative analysis
  - explain the error that led to the fault
  - explain the cause of that error
  - explain the cause of that cause
  - …
- Rule of thumb
  - “ask why six times”

Example of fault tracing

- Tracing the causes of faults requires experience, judgment, and knowledge of the development process
- example
  - most significant class of faults = memory leaks
  - cause = forgetting to release memory in exception handlers
  - cause = lack of information: “Programmers can't easily determine what needs to be cleaned up in exception handlers”
  - cause = design error: “The resource management scheme assumes normal flow of control”
  - root problem = early design problem: “Exceptional conditions were an afterthought dealt with late in design”

How could faults be prevented?

- Many approaches depending on fault and process:
- From lightweight process changes
  - example
    - adding consideration of exceptional conditions to a design inspection checklist
- To heavyweight changes:
  - example
    - making explicit consideration of exceptional conditions a part of all requirements analysis and design steps

Goal is not perfection, but cost-effective improvement

The Quality Team

- The quality plan must assign roles and responsibilities to people
- assignment of responsibility occurs at
  - strategic level
    - test and analysis strategy
    - structure of the organization
    - external requirements (e.g., certification agency)
  - tactical level
    - test and analysis plan

Roles and Responsibilities at Tactical Level

- balance level of effort across time
- manage personal interactions
- ensure sufficient accountability that quality tasks are not easily overlooked
- encourage objective judgment of quality
- prevent it from being subverted by schedule pressure
- foster shared commitment to quality among all team members
- develop and communicate shared knowledge and values regarding quality

Alternatives in Team Structure

- Conflicting pressures on choice of structure
  - example
    - autonomy to ensure objective assessment
    - cooperation to meet overall project objectives
- Different structures of roles and responsibilities
  - same individuals play roles of developer and tester
  - most testing responsibility assigned to a distinct group
  - some responsibility assigned to a distinct organization
- Distinguish
  - oversight and accountability for approving a task
  - responsibility for actually performing a task

Roles and responsibilities pros and cons

- Same individuals play roles of developer and tester
  - potential conflict between roles
    - example
      - a developer responsible for delivering a unit on schedule
      - responsible for integration testing that could reveal faults that delay delivery
  - requires countermeasures to control risks from conflict
- Roles assigned to different individuals
  - Potential conflict between individuals
    - example
      - developer and a tester who do not share motivation to deliver a quality product on schedule
  - requires countermeasures to control risks from conflict

Independent Testing Team

- Minimize risks of conflict between roles played by the same individual
  - Example
    - project manager with schedule pressures cannot
      - bypass quality activities or standards
      - reallocate people from testing to development
      - postpone quality activities until too late in the project
- Increases risk of conflict between goals of the independent quality team and the developers
- Plan
  - should include checks to ensure completion of quality activities
  - Example
    - developers perform module testing
    - independent quality team performs integration and system testing
    - quality team should check completeness of module tests

Managing Communication

- Testing and development teams must share the goal of shipping a high-quality product on schedule
  - testing team
    - must not be perceived as relieving developers from responsibility for quality
    - should not be completely oblivious to schedule pressure
- Independent quality teams require a mature development process
  - Test designers must
    - work on sufficiently precise specifications
    - execute tests in a controllable test environment
- Versions and configurations must be well defined
- Failures and faults must be suitably tracked and monitored across versions

Testing within XP

- Full integration of quality activities with development
  - Minimize communication and coordination overhead
  - Developers take full responsibility for the quality of their work
  - Technology and application expertise for quality tasks match expertise available for development tasks
- Plan
  - check that quality activities and objective assessment are not easily tossed aside as deadlines loom
  - example
    - XP “test first” together with pair programming guard against some of the inherent risks of mixing roles

Outsourcing Test and Analysis

- (Wrong) motivation
  - testing is less technically demanding than development and can be carried out by lower-paid and lower-skilled individuals
- Why wrong
  - confuses test execution (straightforward) with analysis and test design (as demanding as design and programming)
- A better motivation
  - to maximize independence
    - and possibly reduce cost as (only) a secondary effect
- The plan must define
  - milestones and delivery for outsourced activities
  - checks on the quality of delivery in both directions

Documenting Analysis and Test

Learning objectives

- Understand the purposes and importance of documentation
- Identify some key quality documents and their relations
- Understand the structure and content of key quality documents
- Appreciate needs and opportunities for automatically generating and managing documentation

Why Produce Quality Documentation?

- Monitor and assess the process
  - For internal use (*process visibility*)
  - For external authorities (certification, auditing)
- Improve the process
  - Maintain a body of knowledge reused across projects
  - Summarize and present data for process improvement
- Increase reusability of test suites and other artifacts within and across projects

Major categories of documents

- Planning documents
  - describe the organization of the quality process
  - include organization *strategies* and project *plans*
- Specification documents
  - describe test suites and test cases

    (as well as artifacts for other quality tasks)
  - test design specifications, test case specification, checklists, analysis procedure specifications
- Reporting documents
  - Details and summary of analysis and test results

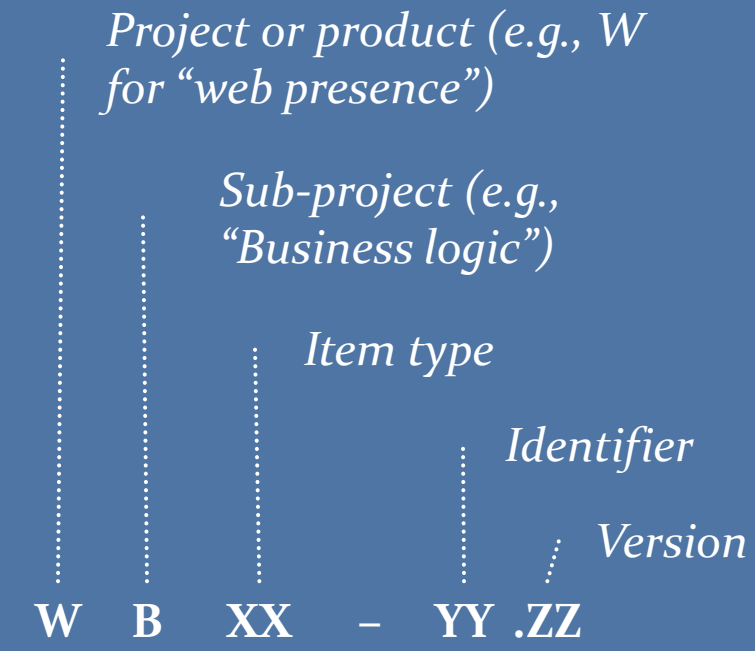
Metadata

- Documents should include *metadata* to facilitate management
  - **Approval**: persons responsible for the document
  - **History** of the document
  - **Table of Contents**
  - **Summary**: relevance and possible uses of the document
  - **Goals**: purpose of the document– Who should read it, and why?
  - **Required documents and references**: reference to documents and artifacts needed for understanding and exploiting this document
  - **Glossary**: technical terms used in the document

Naming conventions

- Naming conventions help people identify documents quickly
- A typical standard for document names include keywords indicating
  - general scope of the document (project and part)
  - kind of document (for example, test plan)
  - specific document identity
  - version

Sample naming standard



example:

W B 12 – 22 .04

Might specify version 4 of document 12-22 (quality monitoring procedures for third-party software components) of web presence project, business logic subsystem.

Analysis and test strategy

- Strategy document describes quality guidelines for sets of projects
  (usually for an entire company or organization)
- Varies among organizations
- Few key elements:
  common quality requirements across products
- May depend on business conditions - examples
  - safety-critical software producer may need to satisfy minimum dependability requirements defined by a certification authority
  - embedded software department may need to ensure portability across product lines
- Sets out *requirements on other quality documents*

Analysis and Test Plan

- Standardized structure see next slide
- Overall quality plan comprises several individual plans
  - Each individual plan indicates the items to be verified through analysis or testing
  - Example: documents to be inspected, code to be analyzed or tested, …
- May refer to the whole system or part of it
  - Example: subsystem or a set of units
- May not address all aspects of quality activities
  - Should indicate features to be verified and excluded
    - Example: for a GUI– might deal only with functional properties and not with usability (if a distinct team handles usability testing)
  - Indication of excluded features is important
    - omitted testing is a major cause of failure in large projects

Test Design Specification Documents

- Same purpose as other software design documentation:
  - Guiding further development
  - Preparing for maintenance
- Test design specification documents:
  - describe complete test suites
  - may be divided into
    - unit, integration, system, acceptance suites (organize by granularity)
    - functional, structural, performance suites (organized by objectives)
    - …
  - include all the information needed for
    - initial selection of test cases
    - maintenance of the test suite over time
  - identify features to be verified (cross-reference to specification or design document
  - include description of testing procedure and pass/fail criteria (references to scaffolding and oracles)
  - includes (logically) a list of test cases

Test case specification document

- Complete test design for individual test case
- Defines
  - test inputs
  - required environmental conditions
  - procedures for test execution
  - expected outputs
- Indicates
  - item to be tested (reference to design document)
- Describes dependence on execution of other test cases
- Is labeled with a unique identifier

Test and Analysis Reports

- Report test and analysis results
- Serve
  - Developers
    - identify open faults
    - schedule fixes and revisions
  - Test designers
    - assess and refine their approach see chapter 20
- Prioritized list of open faults: the core of the fault handling and repair procedure
- Failure reports must be
  - consolidated and categorized to manage repair effort systematically
  - prioritized to properly allocate effort and handle all faults

Summary reports and detailed logs

- Summary reports track progress and status
  - may be simple confirmation that build-and-test cycle ran successfully
  - may provide information to guide attention to trouble spots
- Include summary tables with
  - executed test suites
  - number of failures
  - breakdown of failures into
    - repeated from prior test execution,
    - new failures
    - test cases that previously failed but now execute correctly
- May be prescribed by a certifying authority

Isn't this a lot of work?

- Yes, but
  - Everything produced by hand is actually used
    - Always know the *purpose* of a document. Never expend effort on documents that are not used.
  - Parts can be automated
    - Humans make and explain decisions. Let machines do the rest.
- Designing effective quality documentation
  - Work backward from use, to output, to inputs
    - and consider characteristics of organization and project
  - Capture decisions and rationale at cheapest, easiest point and avoid repetition

Summary

- Mature software processes include documentation standards for all activities, including test and analysis
- Documentation can be inspected to
  - verify progress against schedule and quality goals
  - identify problems
- Documentation supports process visibility, monitoring, and standardization