Software Testing

Test Planning, Strategies and the Joys of Monitoring

Strategies, Planning and Monitoring?

- Strategies:
  - What you're going to do and why.
  - Ideally traces to company/organizational objectives.

- 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

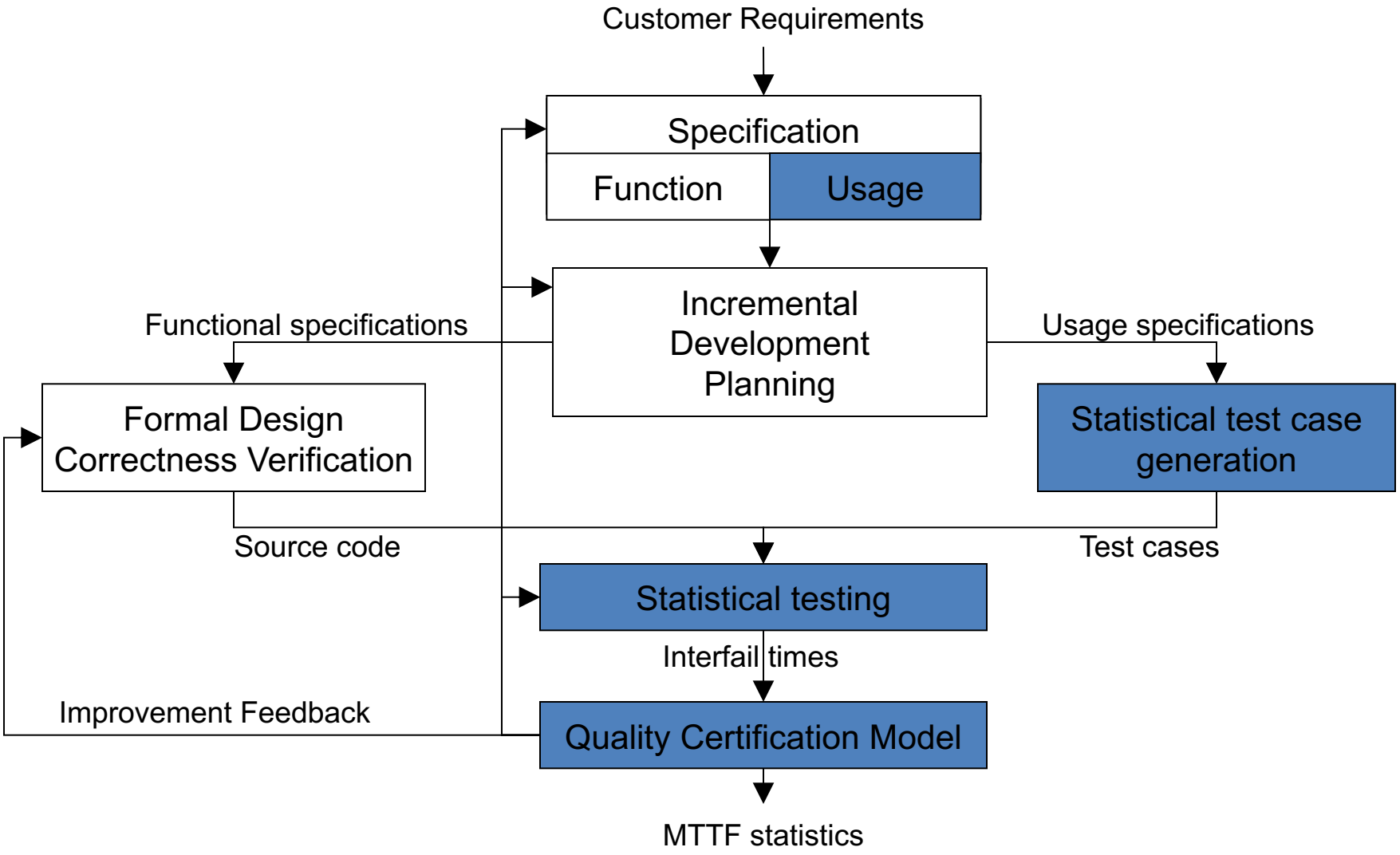
Planning/Monitoring Tools

- Basecamp
- Zoho
- Trello (Kanban board)
- Jira
- Asana
- … loads more…

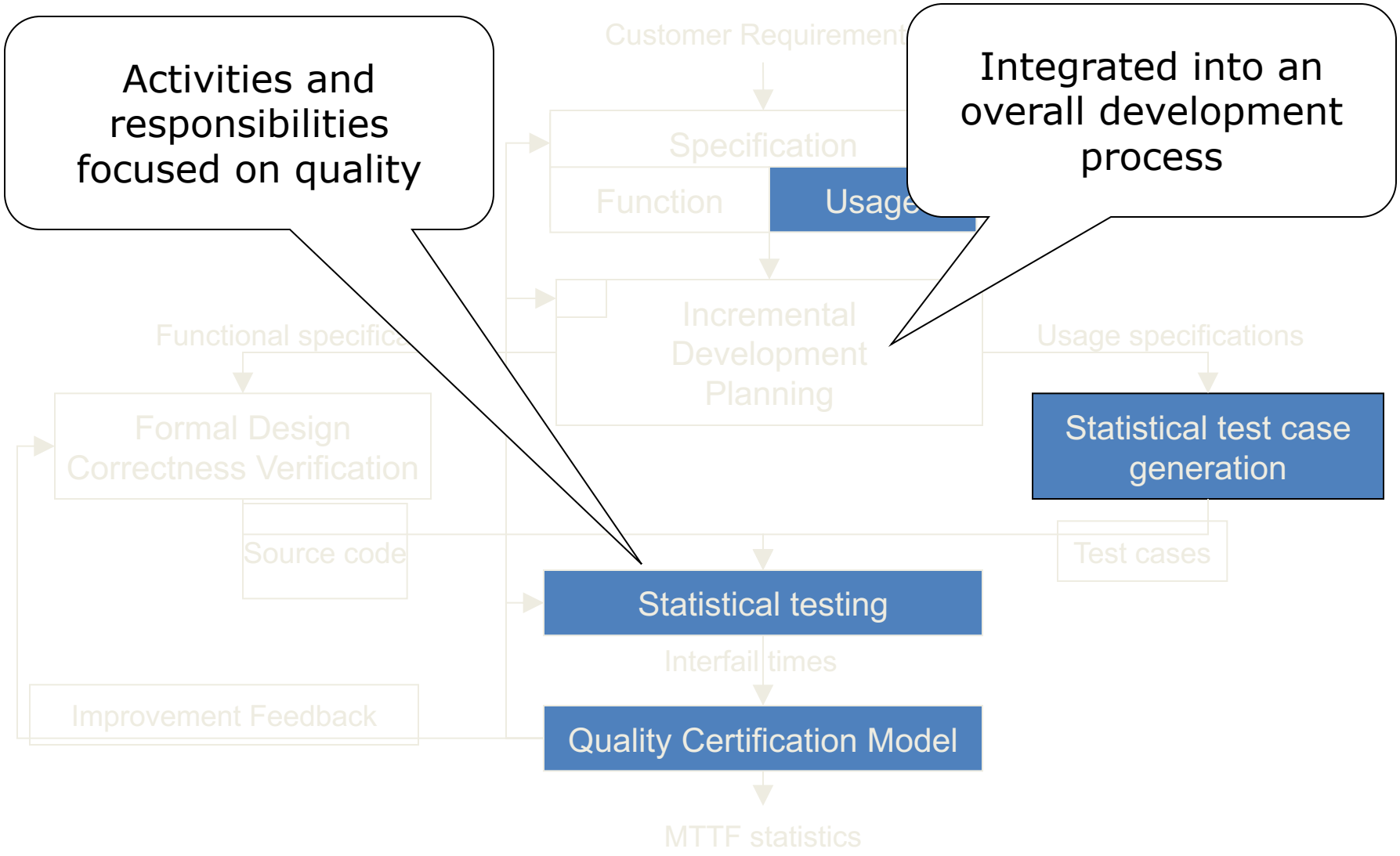
Quality and Process

- 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 (especially testing)
  - considering interactions and trade-offs (e.g., more testing, less review)
- 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

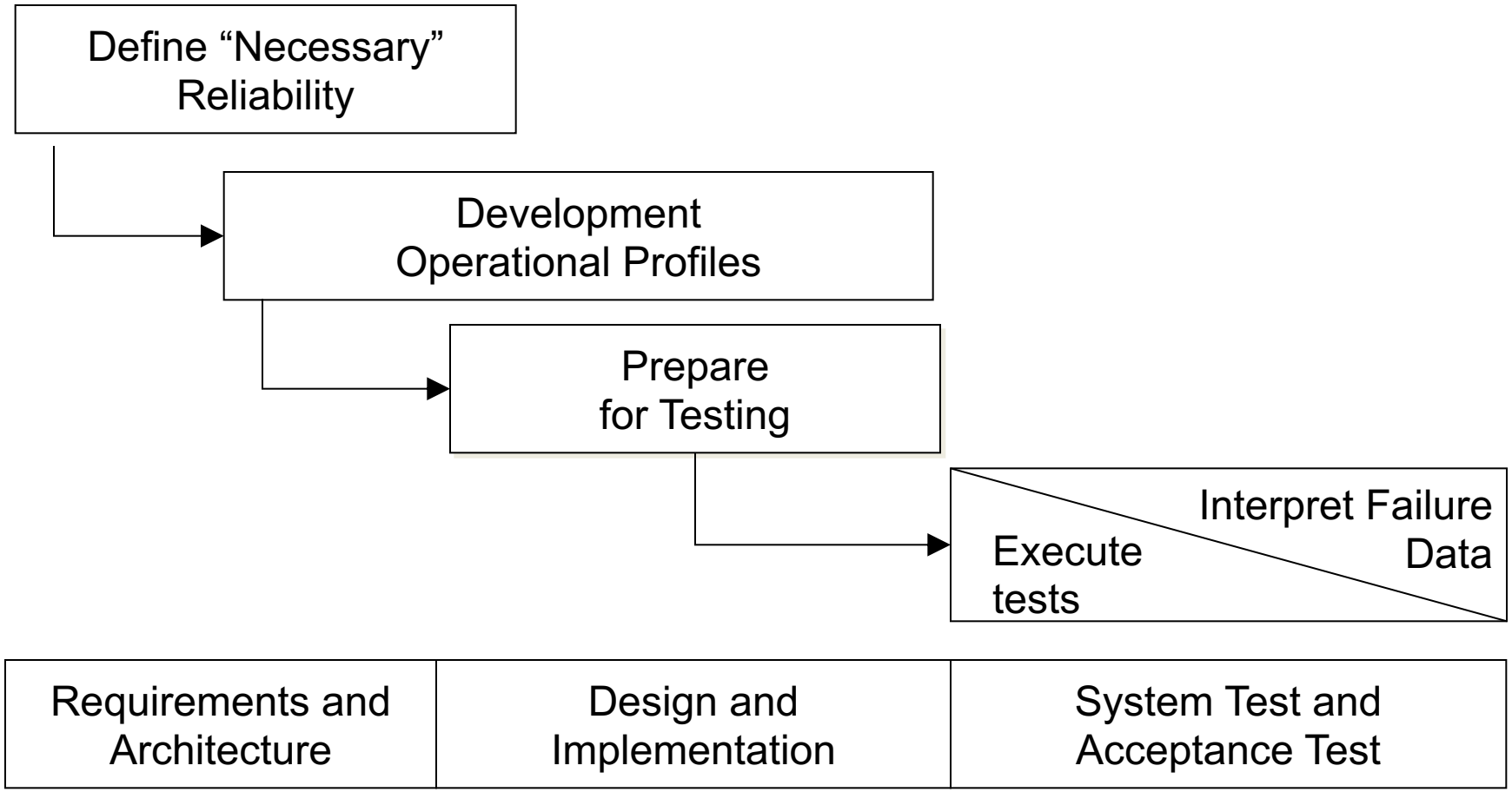
Example Process: Cleanroom



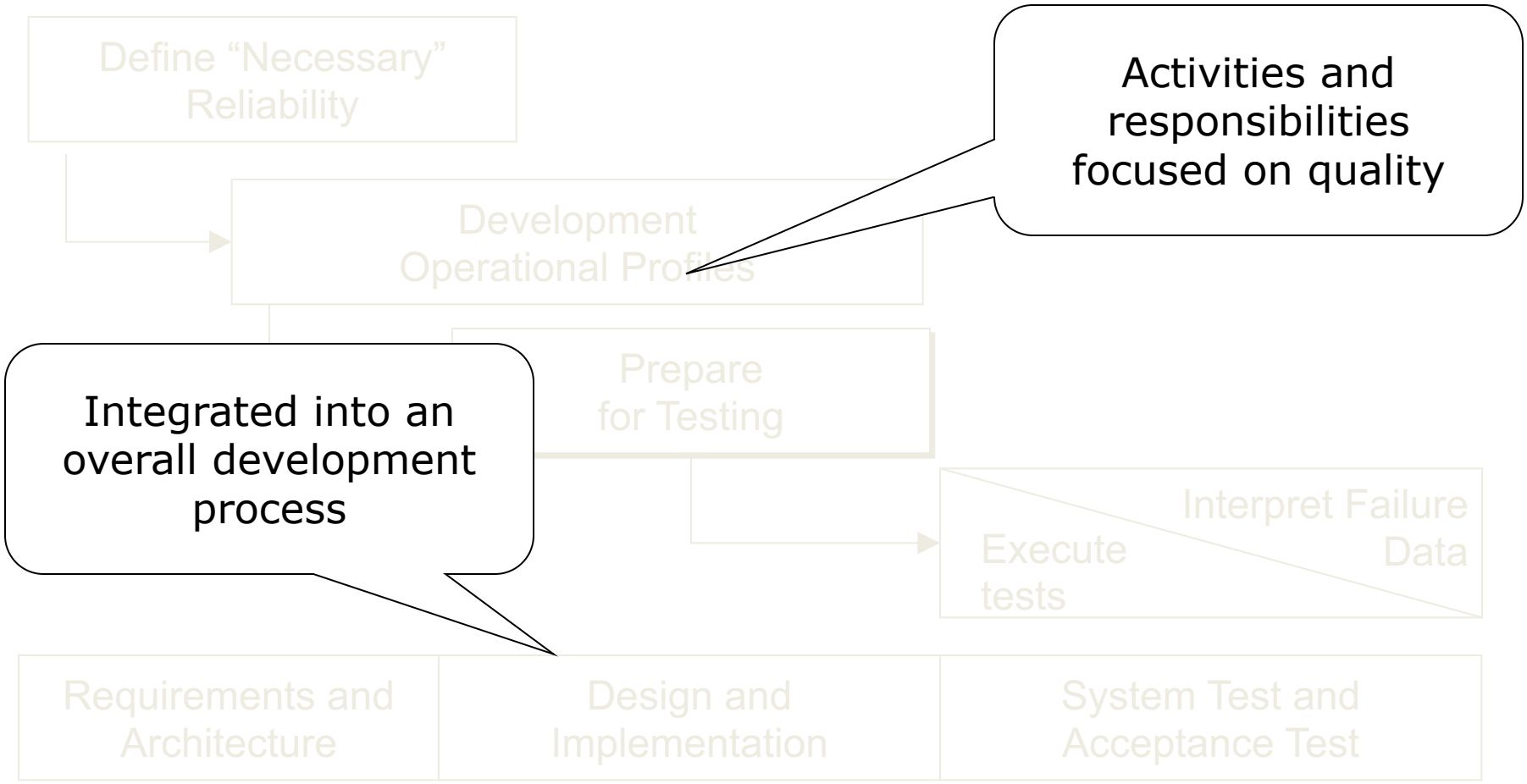
Example Process: Cleanroom



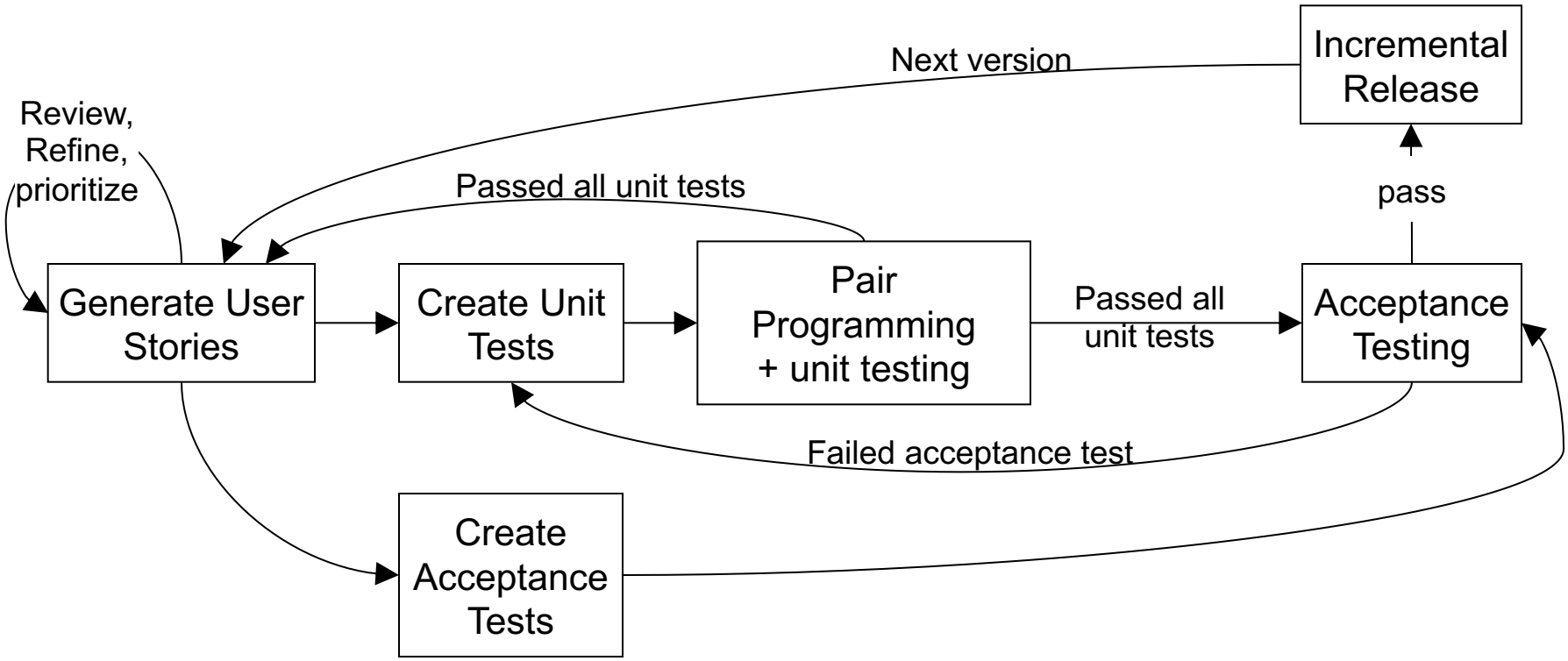
Example Process: Software Reliability Engineering Testing (SRET)



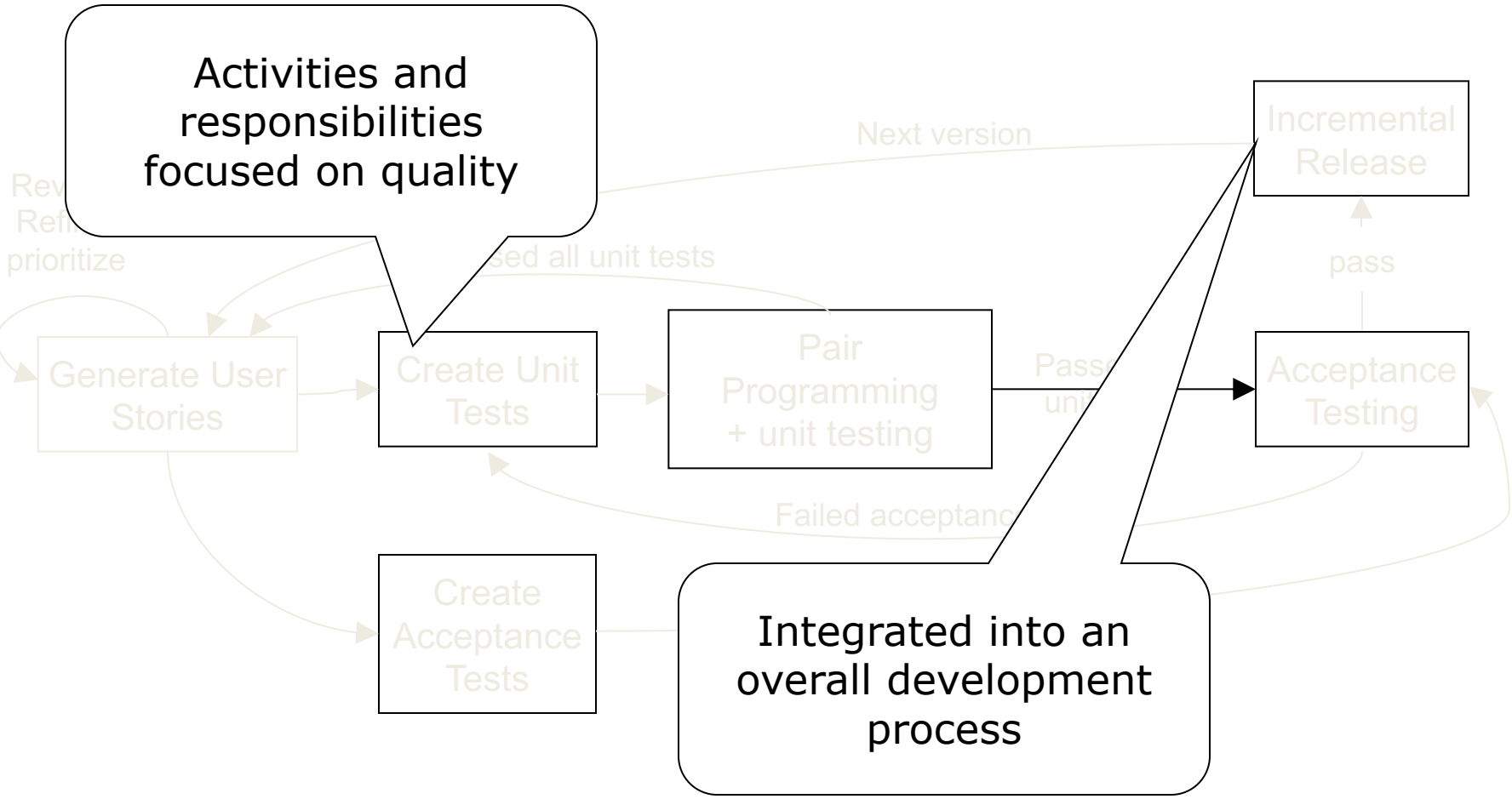
Software Reliability Engineering Testing (SRET)



Example Process: Extreme Programming (XP)



Extreme Programming (XP)



Overall Organization of a Quality Process

- Key principle of quality planning
  - *the cost of detecting and repairing a fault increases as a function of time between committing an error and detecting the resultant faults*
- therefore ...
  - an efficient quality plan includes matched sets of *intermediate* validation and verification activities that *detect most faults within a short time* of their introduction
- and ...
  - V&V steps depend on the intermediate work products and on their *anticipated defects*

Detect faults quickly!

Allen Holub
@allenholub

I do wonder why anyone thinks that it's a good idea to not just fix a bug the moment you become aware of it. Bug-tracking systems have always struck me as weird. Don't track them; fix them.

10:31 PM · Feb 22, 2022 · TweetDeck

Class exercise

- Time for some speculation!
- Answer these questions:
  - Do you believe that cost of repair for faults increases as development time progresses?
  - Are there exceptions?
  - Can you choose/design your engineering process to help detect faults earlier?

- Take about 2-3 minutes on this

An Aside: End Products

- Ultimately we want to deliver an end product of high quality.
- But it takes time and steps to get there.
- Intermediate artifacts are developed.
  - Early designs, prototypes, test plans, …
- If these are of poor quality, then is it likely that the end product will be of lower quality than we might like.

Aside: 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 small ones
    - 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)

Elements of a Test Strategy

- Common quality requirements that apply to all or most products
  - unambiguous definition and measures
- Set of documents normally produced during the quality process
  - contents and relationships
- Activities prescribed by the overall process
  - standard tools and practices
- Guidelines for project staffing and assignment of roles and responsibilities

Test Strategies

- The objective of testing is to decrease the risks inherent in computer systems
- The strategy must address the risks and present a process that can reduce those risks
  - Hence, the system risks establish the objectives for the test process
- The two components of the testing strategy are the test factors and the test phase:
  - Test factor: The risk or issue that needs to be addressed as part of the test strategy
  - Test phase: The phase of the systems development life cycle in which testing will occur

Factors in Test Strategies

- **Test factors:** Correctness, File integrity, Authorization, Audit trail, Availability, Access control, Compliance, Reliability, Ease of use, Maintainability, Portability, Coupling, Performance, Ease of operation, ...
- Not all test factors will be applicable to a software system
- The development team will need to select and rank the test factors for the specific software system being developed
- Once selected and ranked, the strategy for testing will be partially defined
- The test phase will vary based on the testing methodology used

So how do you build a strategy?

1. **Select and rank test factors:** The customers/key users of the system in conjunction with the test team should select and rank the test factors
2. **Identify the system development phases:** This is normally obtained from the system development methodology
3. **Identify the business risks associated with the system under development:** The developers, key users, customers, and test personnel should brainstorm the risks associated with the software system
4. **Place risks in Test factor/test Phase:** The risk team should determine the test phase in which the risk needs to be addressed by the test team, and the test factor to which the risk is associated

Test Strategy – like this!

Test Phase / Test Factors	Requirements	Arch. Design	Design	Coding	Unit Testing	Integration Testing	Sys./Accp. Testing	Maintenance
Factors			Risks					

Test (and Analysis) Plan

Answers 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 Test 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

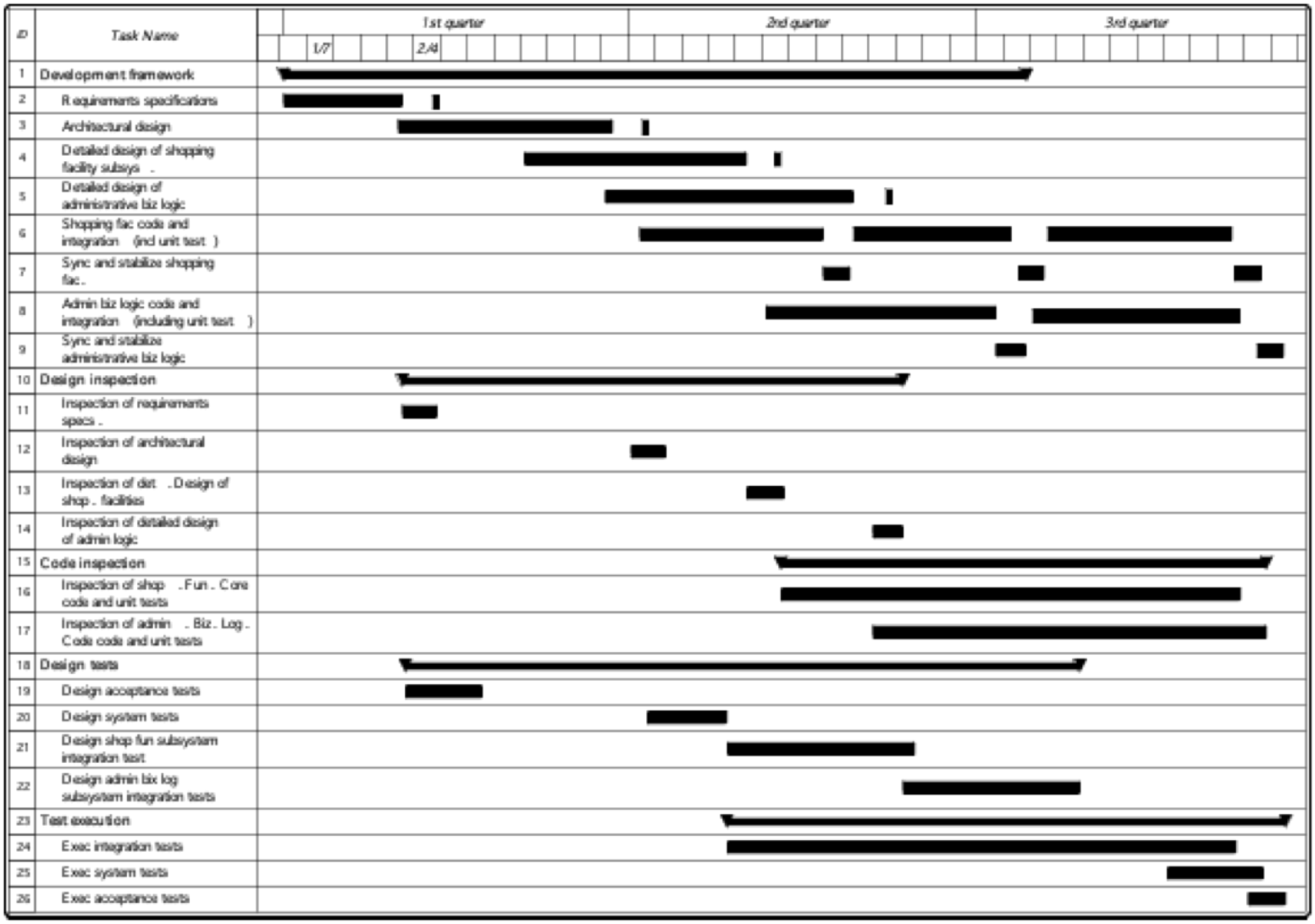
Quality Goals

- 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
  - test 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?)

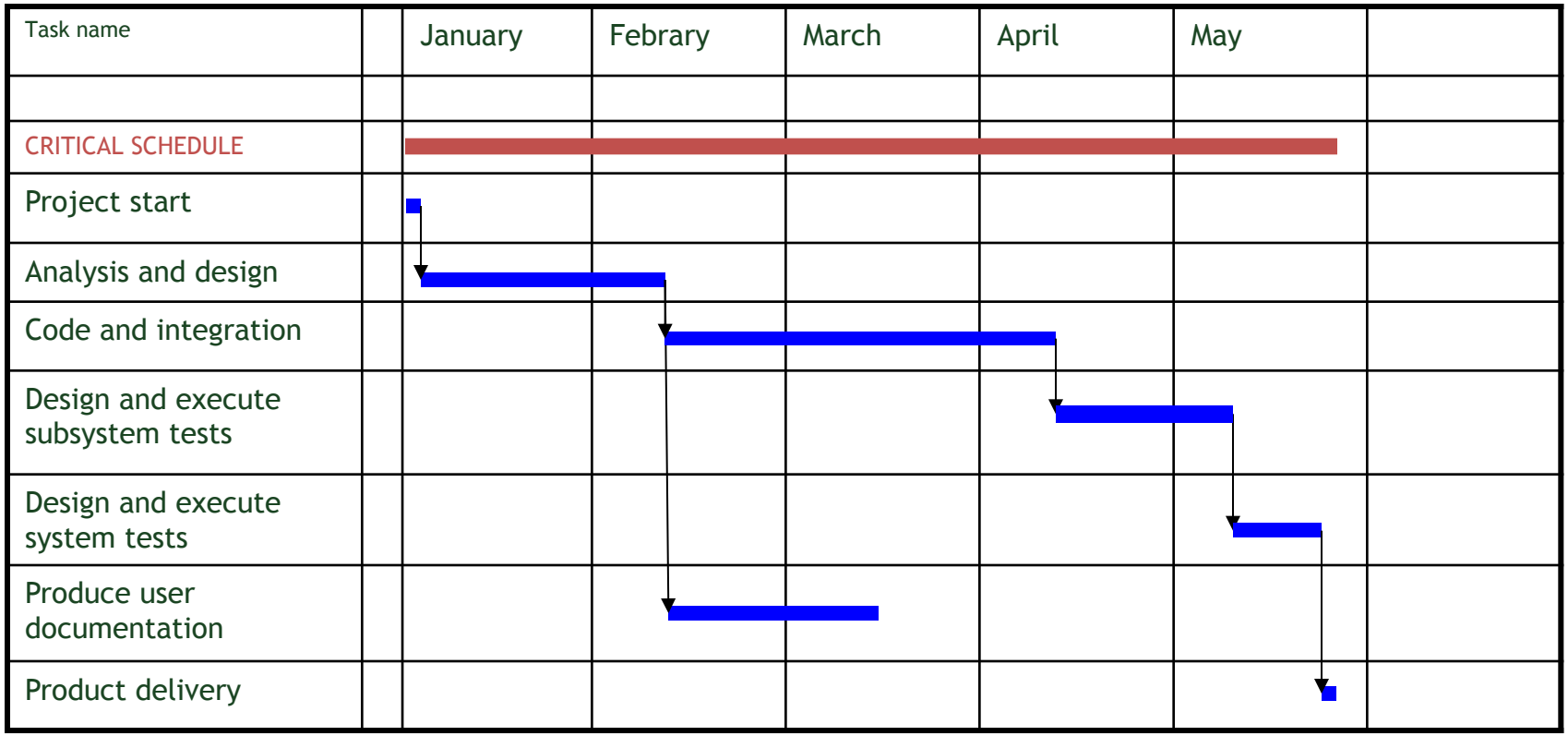
Sample Schedule



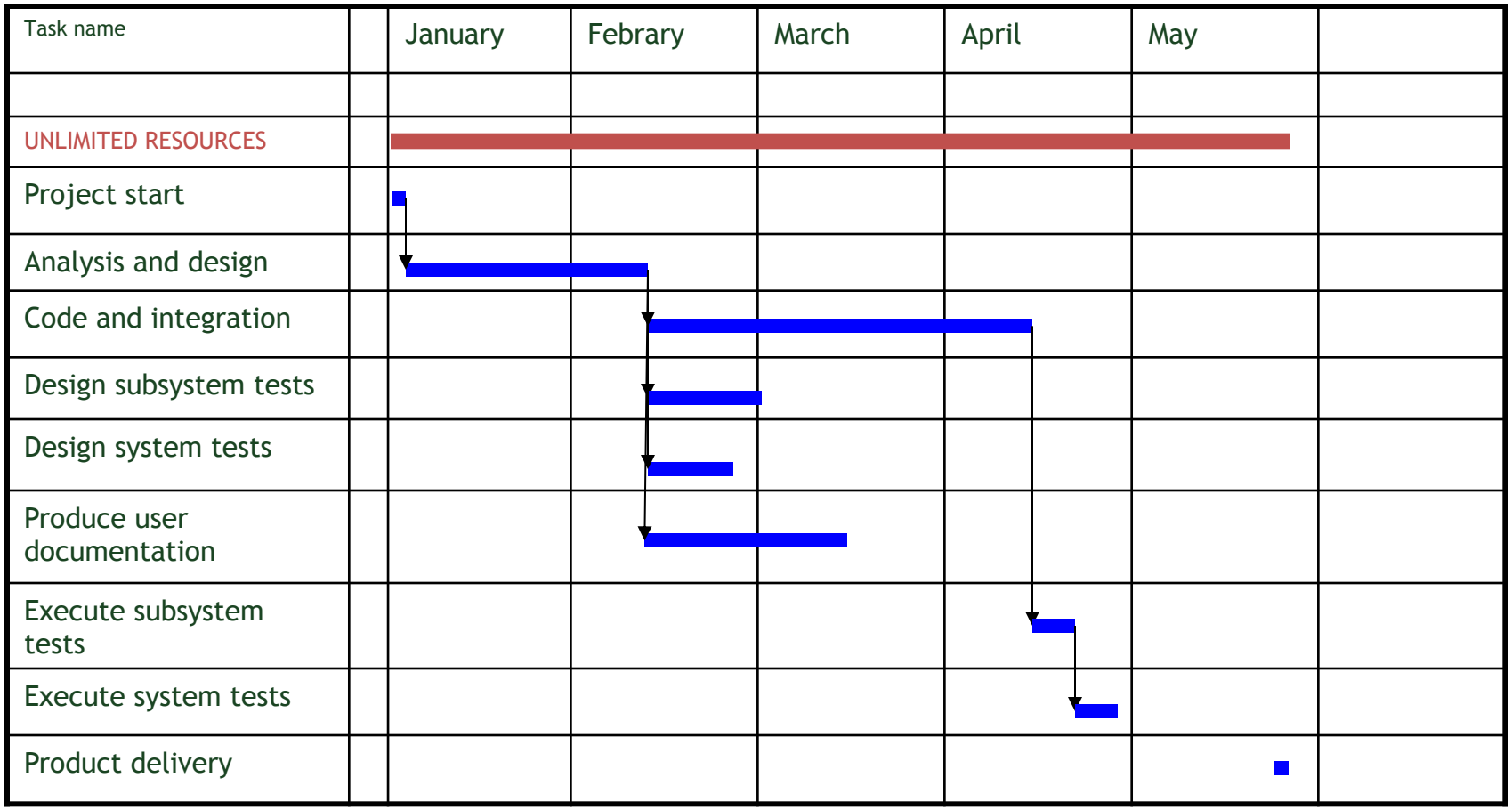
Schedule Risk

- *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

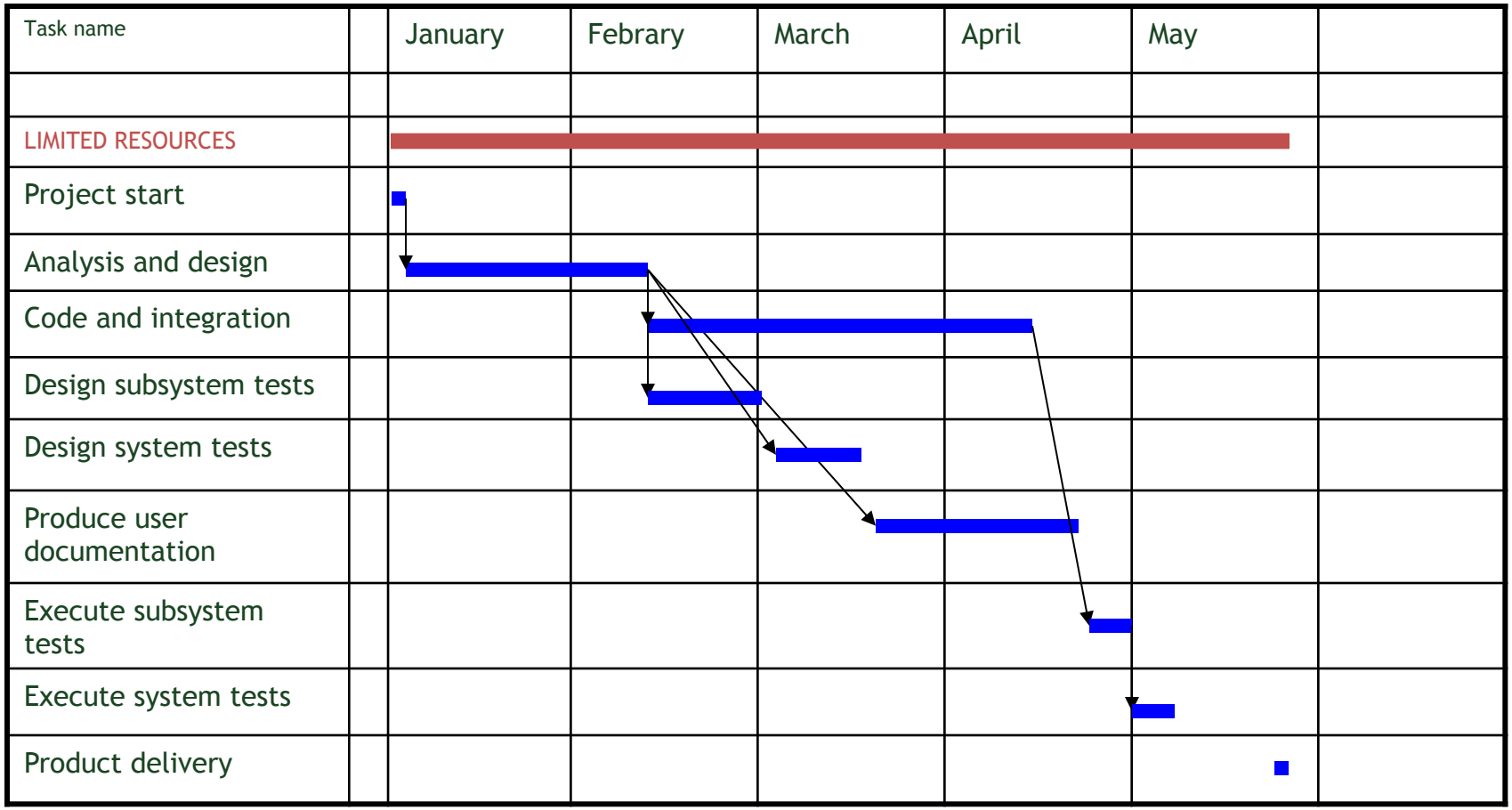
Reducing the Impact of Critical Paths



Reducing the Impact of Critical Paths



Reducing the Impact of Critical Paths



Risk Planning in Building a Plan

- 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 (“the bus”)
- 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

Test Execution

Example Risks

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

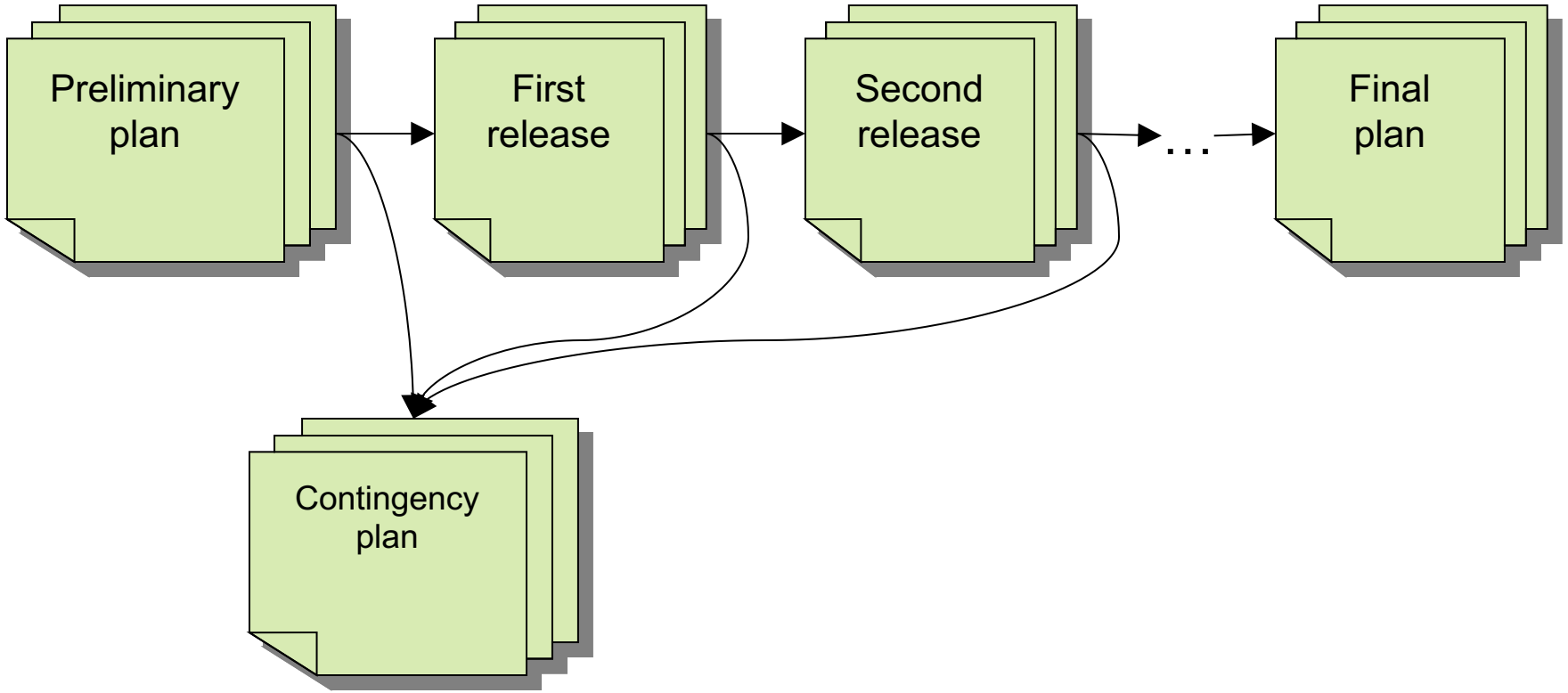
Control Strategies

- Minimize parts that require full system to be executed
- Inspect architecture to assess and improve testability
- Increase intermediate feedback
- Invest in scaffolding (this can be difficult!!!)

Contingency Test Plan

- Part of the initial test plan
  - What could go wrong? How will we know, and how will we recover?
- Evolves with the test plan
- Derives from risk analysis
  - Essential to consider risks explicitly and in detail
- Defines actions in response to bad news (e.g., failed integration test – not unknown!)
  - Plan B at the ready (the sooner, the better)

Evolution of the Plan



Test Plan Types

- System Test Plan
  - Describes when and how testing will occur
  - Information on: software, test objectives and risks, business functions, specific tests
  - A road map: contains specific tests, after execution helps produce the test report
- Unit Test plan
  - Each component / unit should have its own test plan
  - Determine when unit testing is complete

System Test Plan Components

1. *General information*
    1. Summary
    2. Environment and Background
    3. Test Objectives
    4. Expected Defect Rates
    5. References
2. *Plan*
    1. Software Description
    2. Test Team
    3. Milestones
    4. Budgets
    5. Testing checkpoints: schedule, requirements, materials, training, tests to be conducted
    6. Subsequent checkpoints

System Test Plan (continued)

3. *Specifications and Evaluation*
    1. Specifications: business and structural functions, test/function relationships, test progression
    2. Methods and constraints: methodology, tools, extent, data recording, constraints
    3. Specification and evaluation: criteria, data reduction
4. Test Descriptions: control, inputs, outputs, procedures

Unit Testing Plan

1. *Plan*
    1. Unit description
    2. Milestones
    3. Budget
    4. Test approach
    5. Functions not tested
    6. Test constraints
2. *Business and structural function testing*
    1. Business functions
    2. Structural functions
    3. Test descriptions
    4. Expected results
    5. Conditions to stop
    6. Test number cross-reference

Unit Testing Plan (continued)

3. *Interface Test description*
    1. Interface
    2. Test description
    3. Expected results
    4. Test number cross-reference
4. *Test progression*

Improving Current and Next Processes

- Identifying weak aspects of a test 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
      - 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 PDF format. PDF 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 components
      - identifying potentially faulty modules can improve the cost effectiveness of fault detection
    - Some classes of faults dominate
      - 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 test plan must assign roles and responsibilities to people
- assignment of responsibility occurs at
  - strategic level
    - test strategy
    - structure of the organization
    - external requirements (e.g., certification agency)
  - tactical level
    - test 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

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

Summary

- Planning is necessary to
  - order, provision, and coordinate quality activities
    - coordinate quality process with overall development
    - includes allocation of roles and responsibilities
  - provide unambiguous milestones for judging progress
- Process *visibility* is key
  - ability to monitor quality and schedule at each step
    - intermediate verification steps: because cost grows with time between error and repair
  - monitor risks explicitly, with contingency plan ready
- Monitoring feeds process improvement
  - of a single project, and across projects