



ILLINOIS INSTITUTE
OF TECHNOLOGY

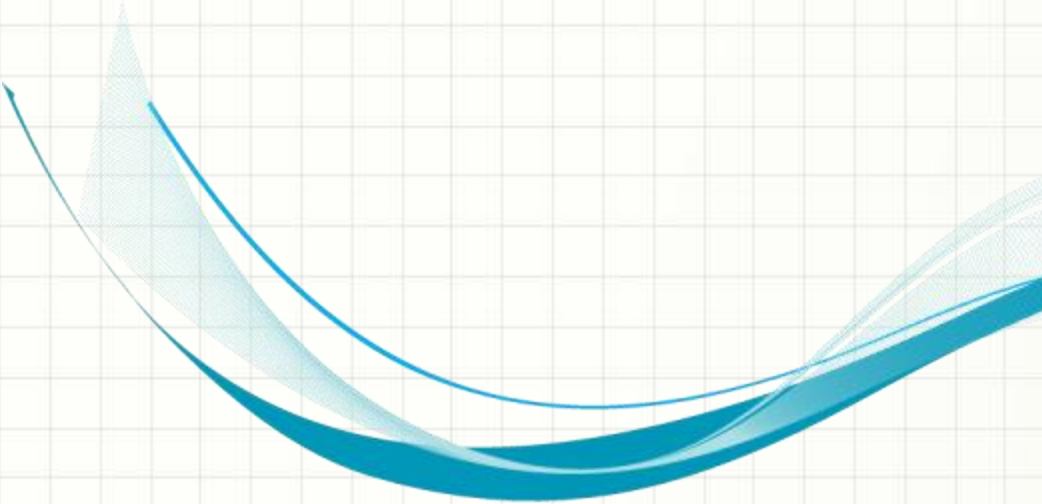
Transforming Lives. Inventing the Future.

www.iit.edu

SOFTWARE ENGINEERING

CS 487

Dennis Hood
Computer Science



Week 4

Proactive and Reactive Quality Assurance

Lesson Overview

- Proactive and Reactive Quality Assurance
- Reading
 - Ch. 8 – Software Testing
 - Ch. 24 – Quality Management
- Objectives
 - Analyze testing as a critical life-cycle phase and quality as a critical goal of software engineering
 - Explore both the reactive (testing) and proactive (quality management) approaches for maximizing quality and likelihood of success

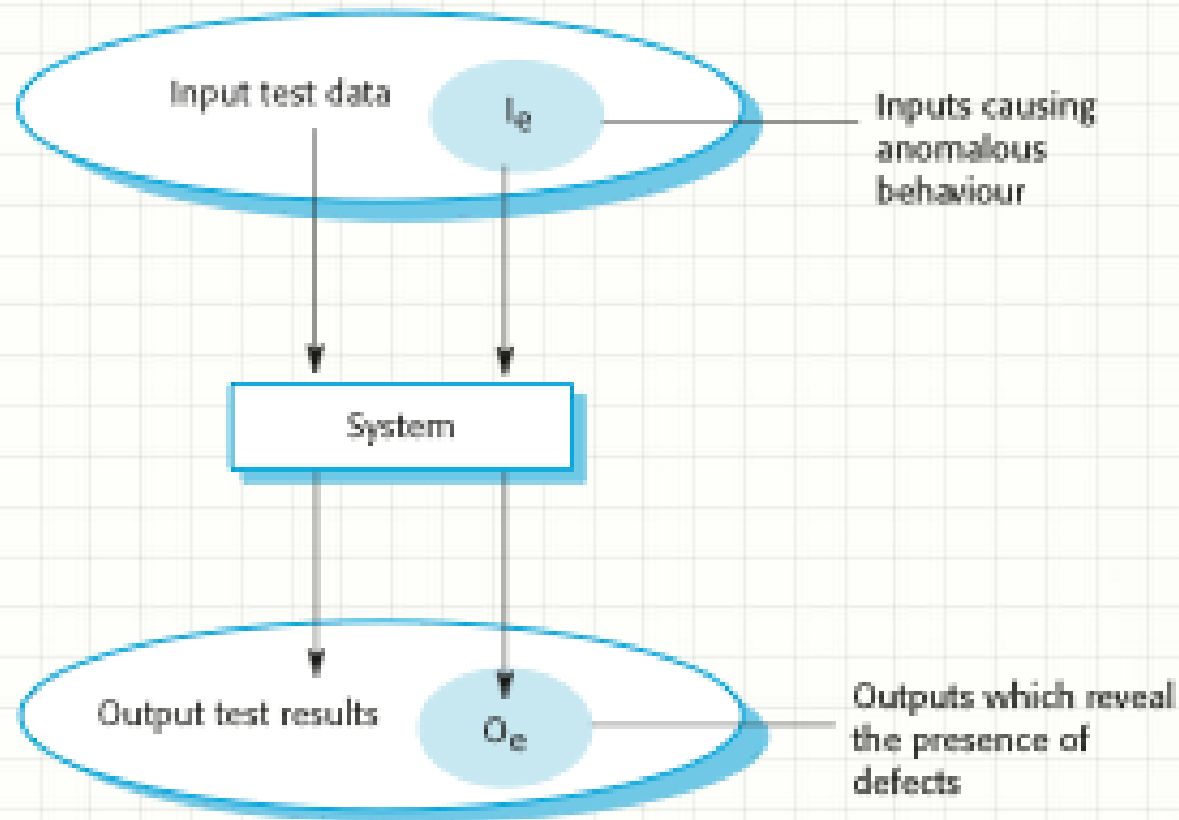
Topics for Discussion

- Why would perfect proactive QA make reactive testing unnecessary?, and why don't organizations simply do that?
- How much reactive QA is necessary to prove that the proactive QA was perfect?
- Describe the proactive and reactive elements of building security and compare them to SW Eng QA.
- How could an organization quantify the costs and benefits of QA (both proactive and reactive)?
- Use risk exposure to describe an approach for assessing 1) more training; 2) more testing.

The Role of Testing

- Goals
 - Demonstrate correctness, completeness, etc.
 - At least one test per requirement
 - Tests to fully exercise features
 - Discover any defects
 - Or at least gain confidence that all have been discovered
- V&V
 - Validate that we are building the right product
 - Verify that we are building it right

Testing Model



Fit for Purpose

- The system is good enough for its intended use
- Software purpose – the more critical the system, the more important that it is reliable
- User expectations
 - Tolerance for defects
 - Sense of value in the system's capabilities
- Marketing environment
 - Competition can both drive organization's to strive for greater quality OR motivate them to release systems without fully testing
 - User expectations of quality are related to price

Inspections

- Static review of “readable” representations of the system
 - “Executability” is not a pre-condition
 - Source code, design documents, etc.
 - Inspected for adherence to specification and standards
- Inspections vs. Testing
 - In testing, defects can “hide beneath” other defects
 - Testing requires a certain degree of completeness
 - Inspections can look at other important factors beyond correctness, such as portability, maintainability, efficiency, etc.

Inspection Checks

- Variable management
- Flow control management
- Input/output management
- Interface management
- Memory management
- Exception handling

Development Testing

- Testing done local to the development effort
 - Often done informally by the developer
 - Primarily for defect detection – “debugging”
- Levels of granularity
 - Unit testing – exercise the functionality of logical units of the system
 - Component testing – verify proper operation of interacting entities such as objects
 - System testing – exercise the system as a whole for proper operation, exception handling, tolerance of load, etc.

Test-driven Development

- Interleave testing and code development
 - Develop a portion of code and its associated test(s)
 - Move onto the next increment of code only when current increment passes testing
- TDD approach
 - Forces partitioning of system into portions
 - Ensures “clean” code (in portions)
 - Utilize automated testing
 - Facilitates deeper understanding of the system
 - Provides a level of documentation
 - Needs to be supported by occasional “big picture” assessment

Release Testing

- Establish a “fit for use” version of the system
 - Requires independent verification
 - Focus is on validation more than defect discovery
 - Usually “black-box” in nature (ins and outs)
- Requirements testing
 - Requirements should be testable
 - Demonstrate proper implementation of system requirements
- Scenario testing
 - Verify “realistic” operation
- Performance or load testing

User Testing

- User acceptance is the ultimate goal of systems development
- User involvement is critical to successful development
- Users should be involved in test planning and execution
- Approaches
 - Alpha – within development process
 - Beta – “field” testing of a preliminary release
 - Acceptance – users determine “fitness”

Test Cases

- Effectiveness
 - Efficiently discover defects
 - Credibly show proper operation
- Efficiency
 - Repeatable
 - Self-documenting
 - Easy to develop and maintain
- Strategies
 - Test normal AND abnormal
 - Use realistic data
 - Test boundaries

Interface Testing

- Exercise the interface
 - Parameter passing
 - Return values and types
 - Synchronization
 - State management
- Box testing
 - Predictable output for given input
 - “Correctly incorrect” output for given improper input

Test Planning

- Testing process description
- Requirements traceability
- Items to be tested
- Schedule
- Results recording procedures
- Required hardware and software
- Constraints

Cleanroom Software Development

- Target is zero-defect software
- Keep the development environment “ultra clean”
- Approach
 - Formally specify the system showing system response to stimuli (state transitions)
 - Utilize incremental development with significant user involvement
 - Rely on structured programming and limit the use of control and data abstraction
 - Rigorous software inspections
 - Statistical testing to determine reliability

Quality Defined

- Quality
 - The degree to which the project fulfills requirements
 - A degree of excellence
 - A critical yet understated requirement
- Quality Management
 - Creating policies and procedures
 - Enforcing them to ensure compliance with project requirements

QA vs. QC

- Quality Assurance
 - Prevent defects
 - Improve the level of quality through an efficient set of activities performed throughout the life cycle
- Quality Control
 - Eliminate defective products
 - Improve the rate of acceptable product delivery through an efficient set of defect detection activities, primarily late in the life cycle

Quality Goals

- Prevent, discover and eliminate defects
- Deliver customer satisfaction by representing the user in design and development
- Enforce standards and process
- Mind the gate
- Improve processes
- Review, audit, monitor, verify, validate and inspect

The Value of Quality

- Quality increases customer satisfaction
 - Credibility lasts and attracts new business
- Lack of quality leads to rework
 - Unscheduled work means unplanned expense and slipping schedules
 - Work under duress increases the likelihood of more mistakes
- Uptime and performance are largely determined by quality
 - Lack of quality drives the need to change

The QA Environment

- Contractual conditions
 - Scope, time, budget, etc.
- Customer-supplier relationship
 - Change management, acceptance, etc.
- Teamwork
 - Variety of skills, parallel activities, etc.
- Multiple project support
- HCI / usability concerns
- Turnover management
- Maintenance
 - Enhancement and release management, troubleshooting, etc.

Defect Classification

- Incorrect specification of requirements
- Misunderstanding of client's needs
- Deviation from requirements
 - Gold-plating, short-cutting, etc.
- Design errors
- Implementation errors
- Violation of standards
- Poor test coverage
- User interface / usability errors
- Documentation errors

Planning for Quality

- Checkpoints supporting milestones
- Independent verification
- Build-test-fix-retest
- Make it measurable/testable
- Inspire the delivery of high-quality deliverables
 - Establish quality goals
 - Obtain commitment
 - Motivate performance
- Collect the data/information required to improve over time

Perform Quality Control

- Feedback loops
 - Measure the output of a process
 - As compared to expected
 - Understand the results (both good and bad)
 - Use that knowledge to improve the process
- Root-cause analysis
 - Ask yourself what caused a problem to occur,
 - then ask what caused that cause, and so on
- Histograms and Pareto Charts
 - Frequency of problems by problem category
 - 80% of the problems are due to 20% of the causes
 - Invest in eliminating the most problematic causes

Focus on Quality

- Commitment to quality must be part of the organizational culture
- Improvement frameworks
 - Process maturity models
 - Six-Sigma, CMMI
 - The agile approach
 - Rapid delivery of functionality
 - Customer responsiveness
- Assess the quality of all deliverables and reward practices and behaviors that lead to high-quality results
 - Attention to detail
 - Extra effort
 - Defect removal
 - Proactive communication, etc.

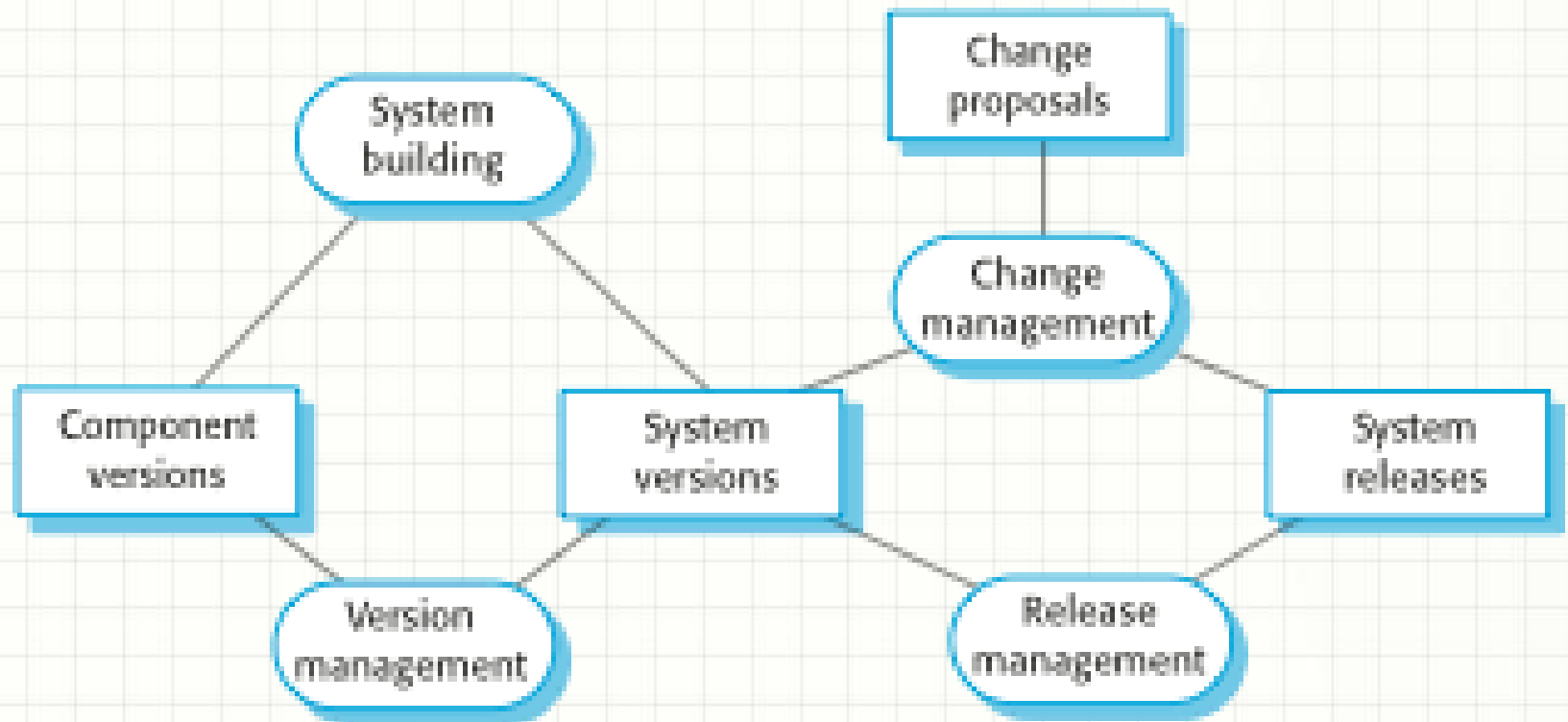
Variability of Process

- Variety
 - The spice of life, but death to a production line
 - Managing variability requires knowledge of
 - The desired output and the degree of tolerance
 - Reducing variability (increasing predictability) takes effort (\$)
 - Side benefits can be significant
- Focus on improvement
 - A focal point is essential to achieving significant, sustainable improvement
- Feedback provides the mechanism for assessment

Configuration Management

- Configuration Management (CM) supports the evolution of software systems through a progression of versions
- Policies, processes and tools
- CM activities
 - Change management
 - Version management
 - System building
 - Release management

CM Activities



Change Management

- Change happens, mainly for good
- Change requests
 - Formality reduces risk (and dampens change)
 - Impact (time, effort, cost, risk, value, etc.)
- Decision making
 - Change control board (CCB)
- Implementation
 - Project planning and documentation
 - Timing
 - Cutover and rollback

Version Management (VM)

- Inevitable evolution often results of multiple “supported” versions
- VM responsibilities
 - Version and release identification
 - Storage management
 - Tracking change history
 - Check out – modify – check in
 - Shared component modification management
- Branching and merging
 - Branching allows for parallel development
 - Merging brings changes together

Baselines

Codeline (A)



Codeline (B)



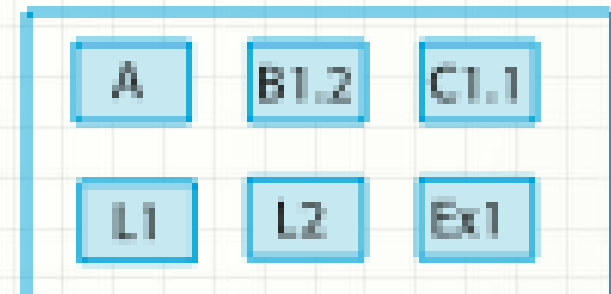
Codeline (C)



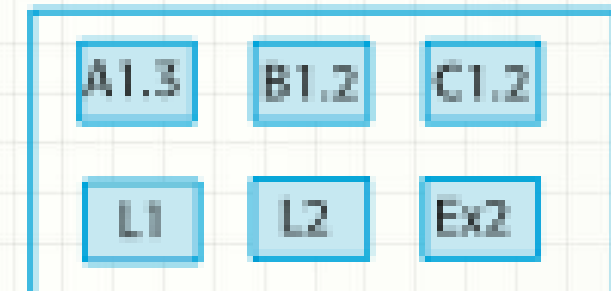
Libraries and external components



Baseline - V1



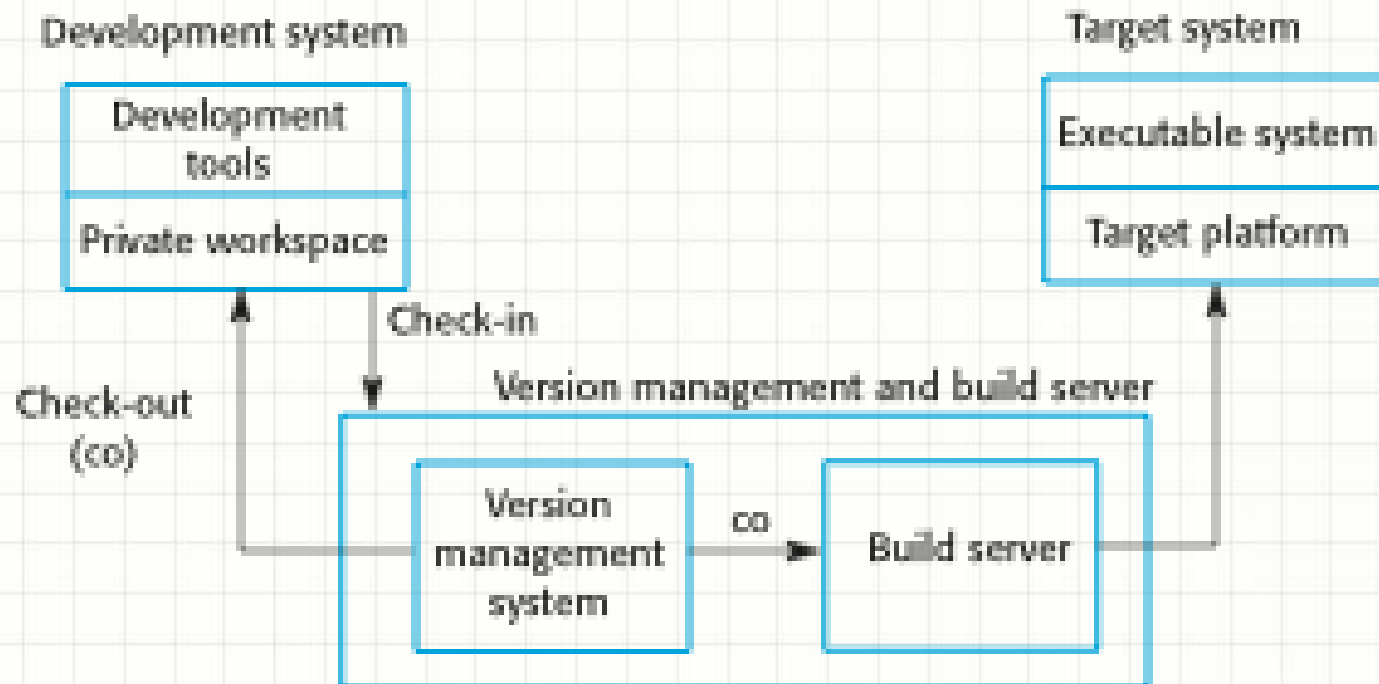
Baseline - V2



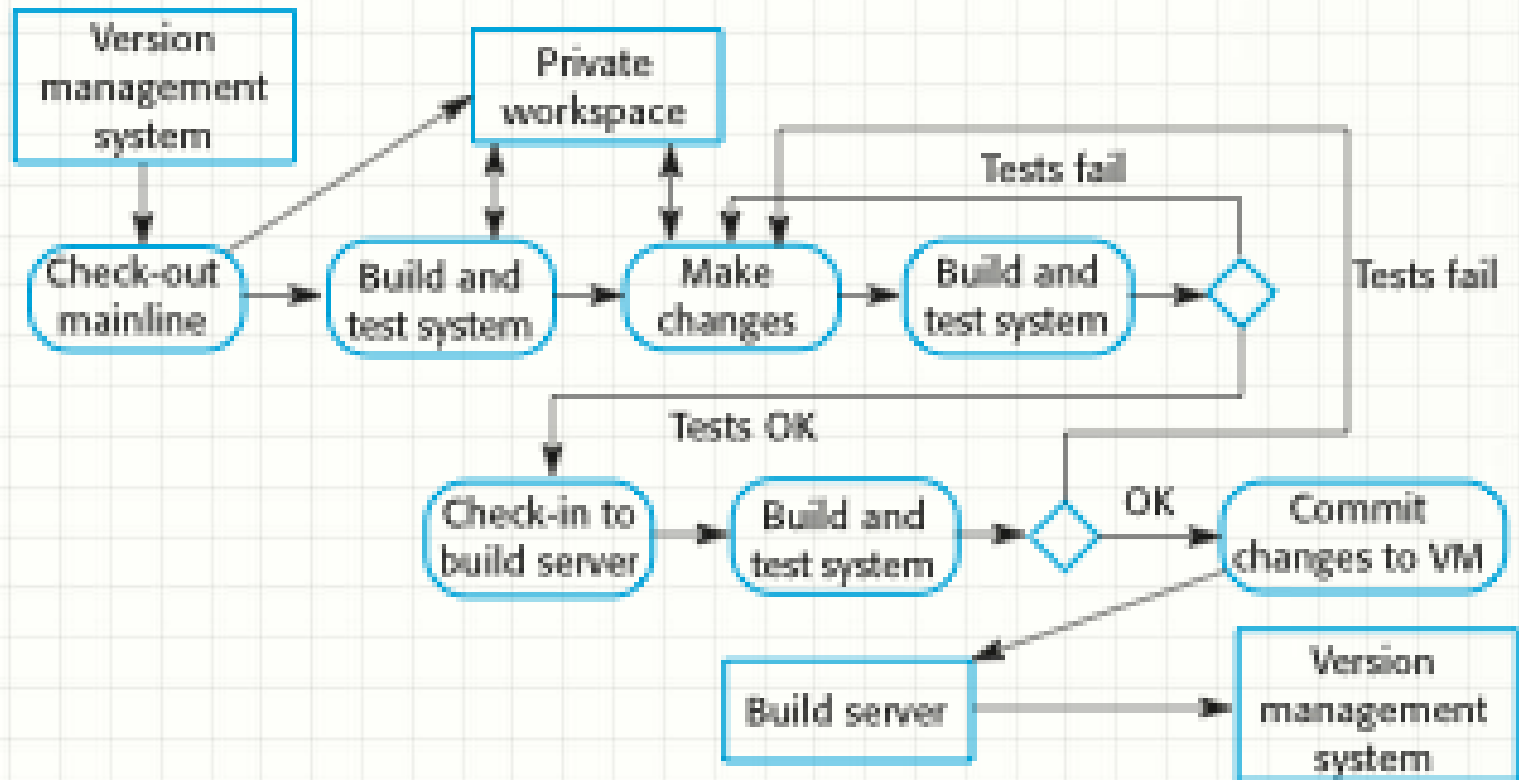
Mainline

System Building

- Compiling and linking the latest checked-in components into a version of the system
- Tool support
 - Build script generation
 - VM system integration
 - Minimal recompilation
 - Executable system creation
 - Test automation
 - Reporting
 - Documentation generation



Continuous Integration



Release Management

- A package for distribution
 - Major – significant new functionality
 - Minor – patches / fixes
- Release planning
 - Current system quality
 - Platform changes
 - New-feature release followed by bug-fix release
 - Competition
 - Marketing commitments
 - Custom changes