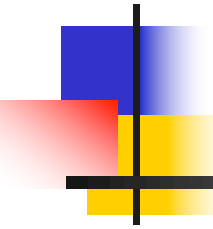
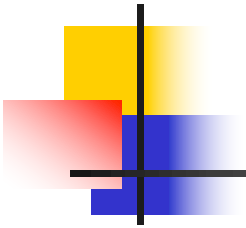
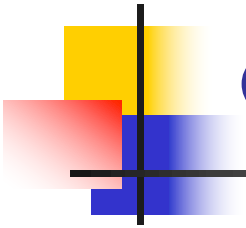


Software Project Management

Software Configuration Management

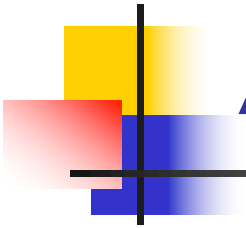


- Getting started with Software Configuration Management
- Susan Dart, Babich, Bersoff



Objectives

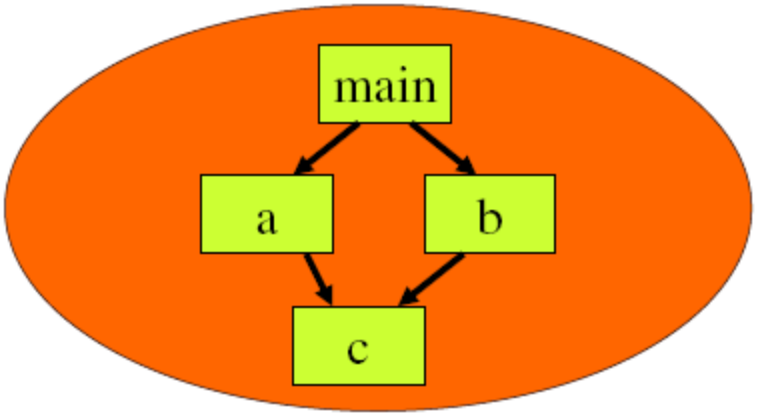
- Learn the basic functions of Software Configuration Management
- Learn the history of SCM from a historical Point of View



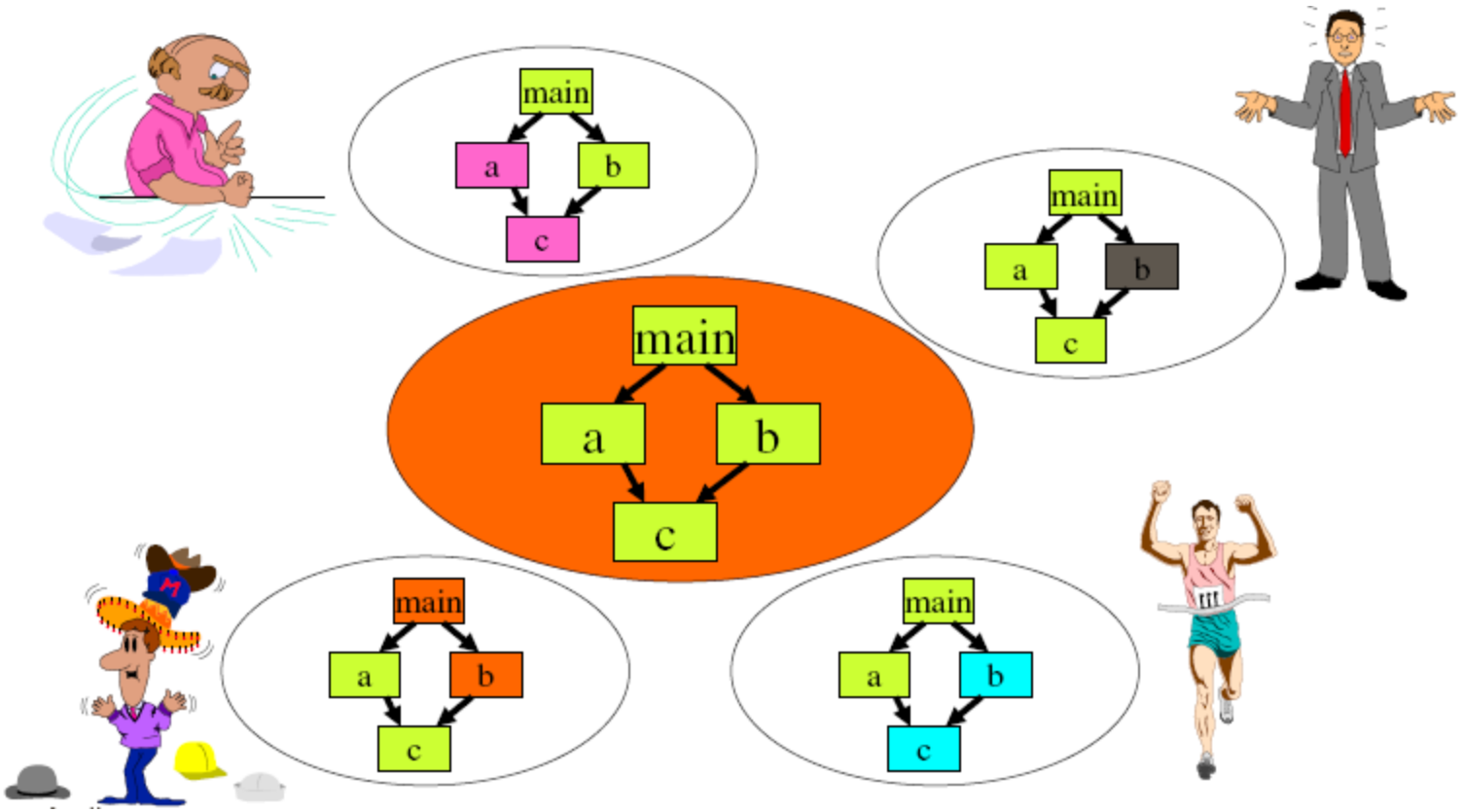
A Definition of SCM

- Software Configuration Management is a discipline for controlling the evolution of software systems.

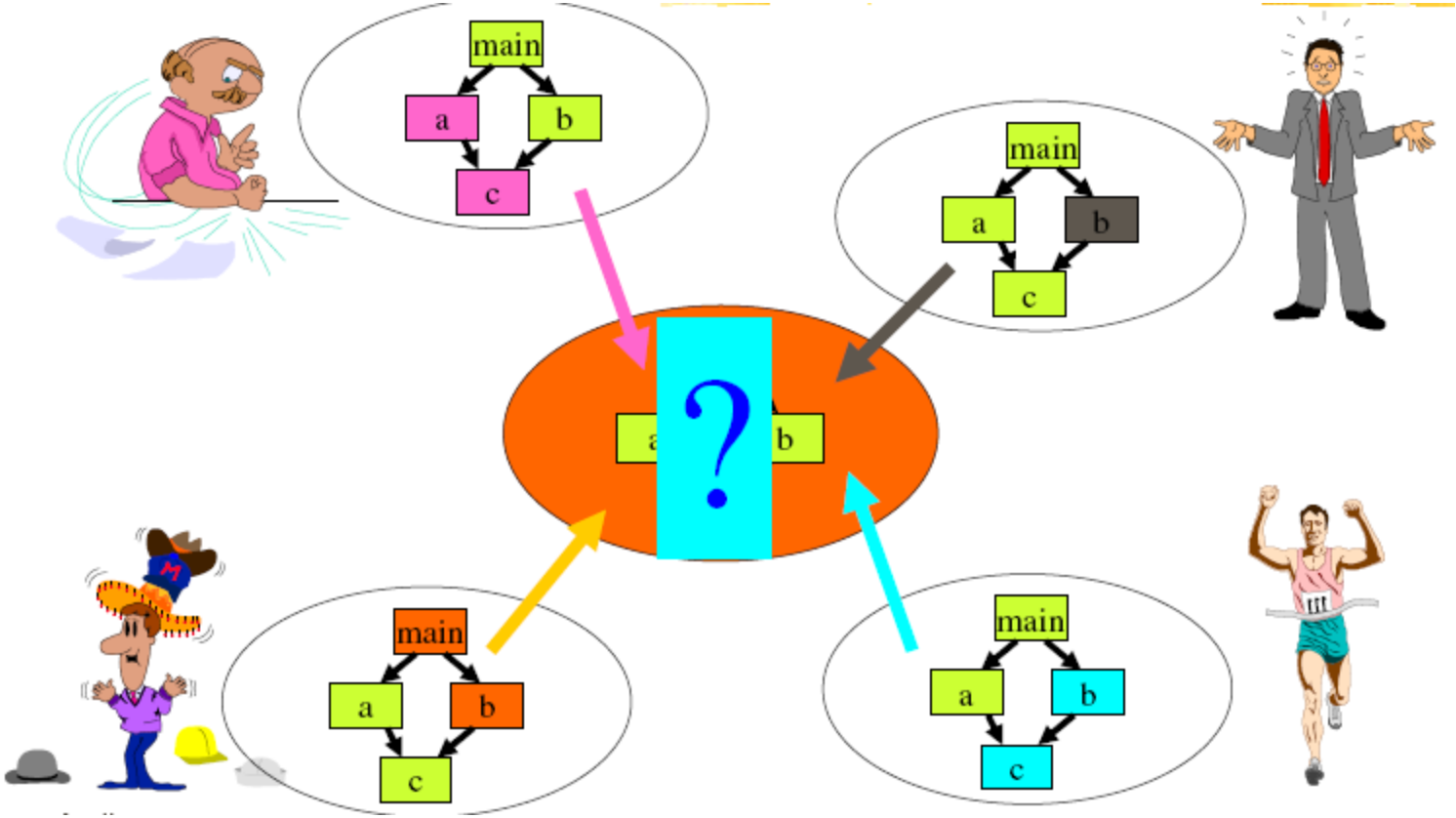
Sharing Data in a Team



The Maintenance Problem -1-

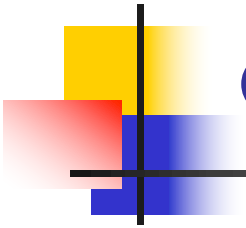


The Maintenance Problem -2-



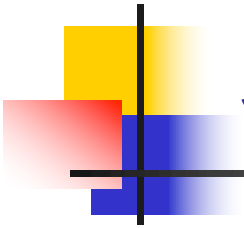
SCM Definition IEEE 1983

- Highlight the following operational aspect of CM:
  - Identification:
    - An identification scheme reflects the structure of the product, identifies component and their type, making them unique and accessible in some form
  - Control
    - Controlling the release of a product and changes to it throughout the lifecycle by having controls in place that ensure consistent software via the creation of a baseline product



Cont…

- Status Accounting
  - Recording and reporting the status of components and change requests, and gathering vital statistics about components in the product
- Audit and Review
  - Validating the completeness of a product and maintaining consistency among the components by ensuring that the product is a well-defined collection of components

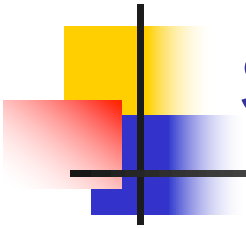


SCM Definition IEEE 1983

- Configuration Management is the process of identifying and defining the items in the system, controlling the changes to these items throughout their life cycle, recording and reporting the status of items and change requests, and verifying the completeness and correctness of items.

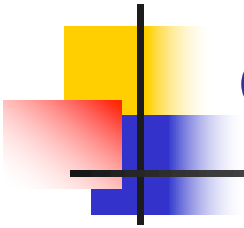
SCM Definition (DoD)

- Configuration Management (CM) is a discipline that applies technical and administrative direction and surveillance over the lifecycle of items to:
  - Identify (and document) configuration items
  - Control changes to configuration items
  - Record and report information needed to manage configuration items (status)
  - Audit configuration items to verify conformance to specifications



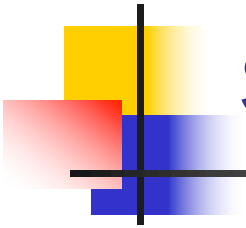
SCM Definition

- Some writer also broaden the definition of CM by adding the following functionalities
  - Manufacture:
    - Managing the construction and building of the product in an optimal manner
  - Process management
    - Ensuring the carrying out of the organization's procedures, polices and lifecycle model
  - Team management
    - Controlling the work and interaction between multiple users on a product



Configuration Item

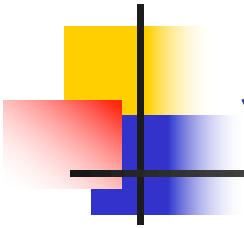
- Source code (C, C++, Java, C#,…)
- Documentation
  - Requirements specification
  - Design specification
  - User Documentation
- Build Files
  - Make files
  - Scripts
- Etc…



SCM components

- Identification
- Control
- Status Accounting
- Auditing
- (+ Storage)

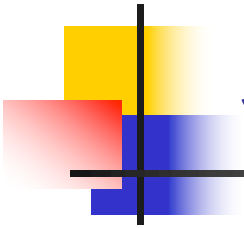
  of configuration Items



Software Configuration Management

- Product Integrity
  - Trace-ability within Product Life Cycle
  - User Functionality
  - Non Functionality
  - Cost
  - Time

Elements of SCM, E.H. Bersoff, 1984

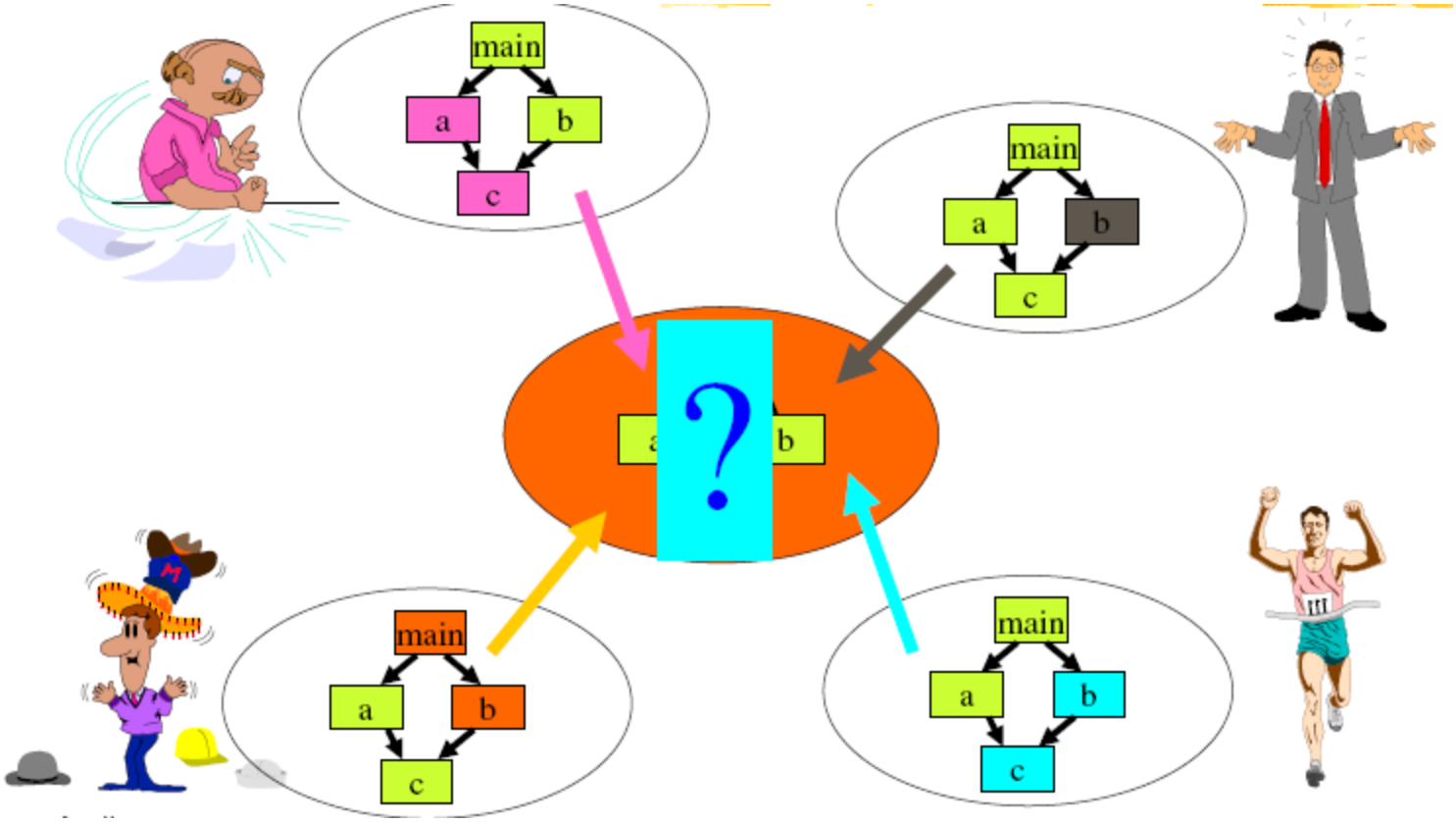


Software Configuration Management

- Team Productivity
  - Minimize Confusion in Team
  - Art of identifying, organizing and controlling modifications to the software being built by a team
  - Maximize Productivity by Minimizing Mistakes

    SCM coordination for Team Productivity, W. Babich, 1986

Confusing Situation



SCM Functionality Area

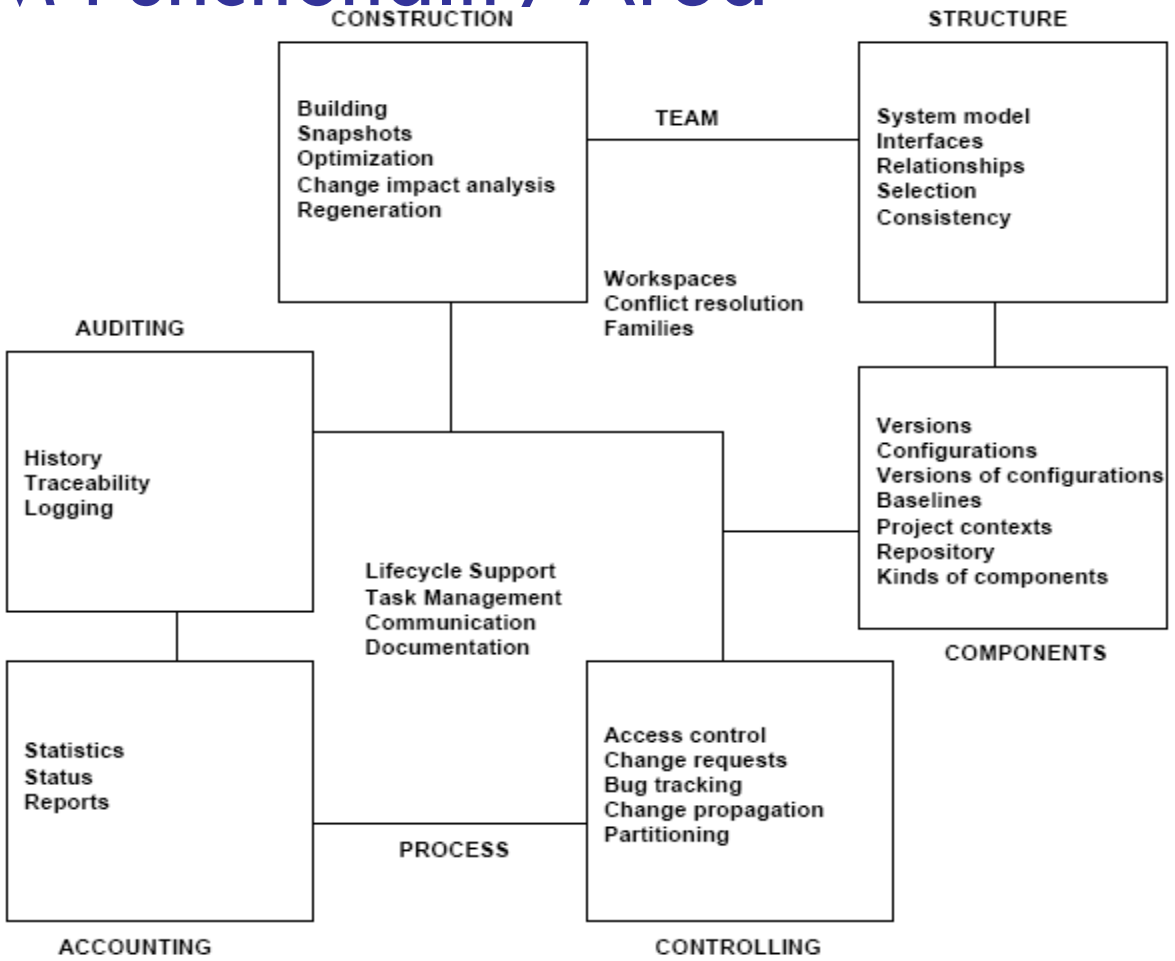
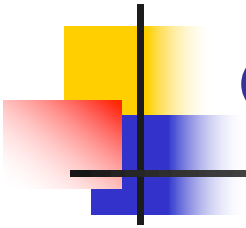


Figure 1: CM Functionality Requirements

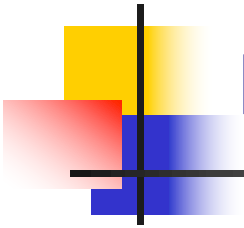
SCM Functionality Area

- **Components:**
  - **identifies, classifies, stores and accesses the components that make up the product.**
- **Structure:**
  - **represents the architecture of the product**
- **Construction:**
  - **supports the construction of the product and its artefacts**
- **Auditing:**
  - **keeps an audit trail of the product and its process**
- **Accounting:**
  - **gathers statistics about the product and the process**
- **Controlling :**
  - **controls how and when changes are made**
- **Process:**
  - **supports the management of how the product evolves**
- **Team:**
  - **enables a project team to develop and maintain a family of products**



Capability Maturity Model

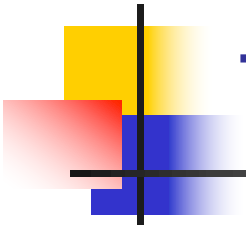
- Software Configuration Management
- Key Process Area in CMM Level 2
- Goals
  - Software configuration management activities are planned.
  - Selected software work products are identified, controlled and available
  - Changes to identified software work products are controlled
  - Affected groups and individuals are informed of the status and content of software baselines



Evolution of SCM

- ’60’s : No SCM
- ’70’s –’80’s : Complex Software
  - Programming in the large
- ’90’s : Complex software with Teams
  - Programming in the many
- ’00’s : Geographical Distributed
  - Programming in the wide

Ivica crnkovic et al. Chapter 3, Historical Overview, 2003

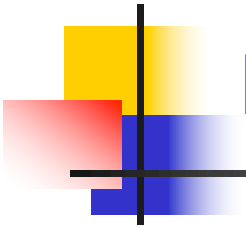


Typical Questions

- This worked Yesterday! What happened?
- I can not recreate your bug in this copy?
- What happened to my fix of last week?
- Program execution (debugger) does not match source code?
- Was that bug fixed in this copy?
- …

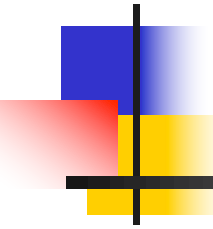
Functions in SCM

- Version Management
  - Versions and code-lines
- Configuration Selection
  - Labelling
- Concurrent Development
  - Branching and Merging
- Distributed Development
  - Europe and India
- Build Management
  - Compile, Link
- Release Management
  - Packaging
- Workspace Management
  - Control and Merging
- Change Management
  - Change and Defects Process
- Status Accounting
  - PR list and Document list
- Status Auditing
  - User Req. Satisfaction
- PDM integration
  - Hardware Development



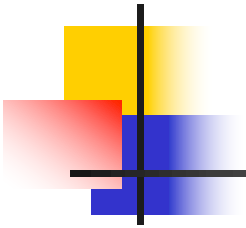
Recap

- SCM is one of the most successful stories in software engineering
- SCM provides supporting Functions for
  - Project Management
  - Development Process
  - Team Development
- Benefit of SCM is hard to measure

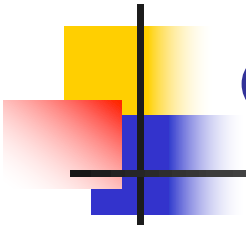


Software Project Management

Version Management



- Getting started with Version Management
- Crnkovic and Berczuk

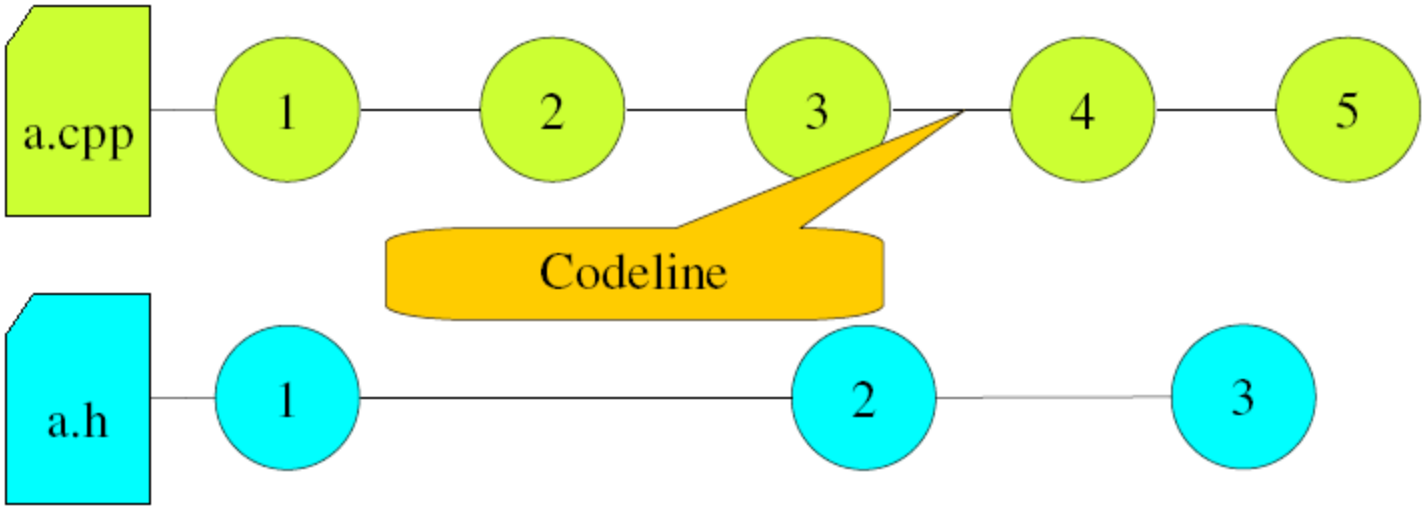


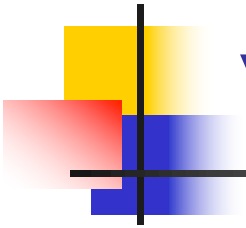
Objectives

- Learn the basics of Version Management
  - Version, Codeline, Check-out, Commit
  - Change Set, Branch
- Learn How to apply these basics in software development

Version Management

- Control of CI versions



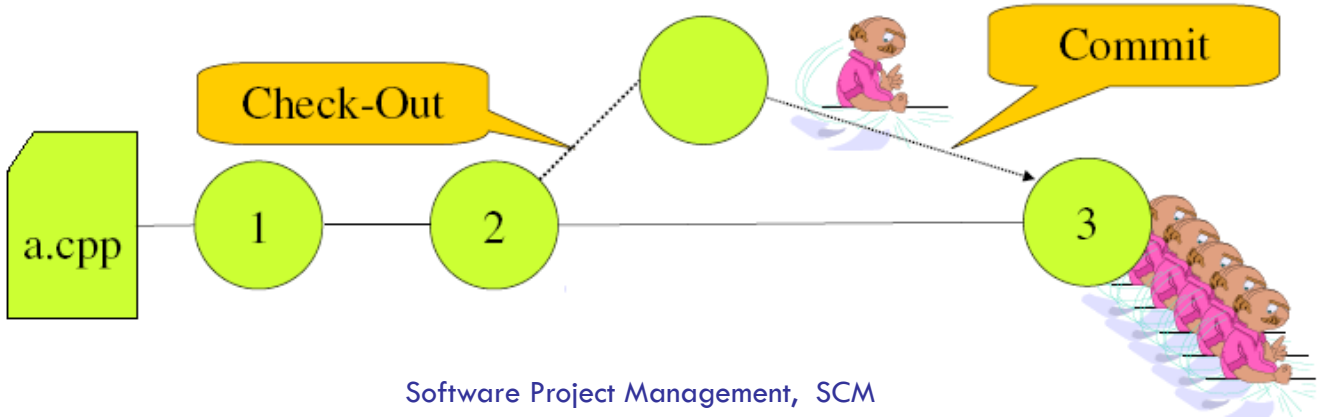


Why do we need Versions

- Archiving History of development
  - Roll Back
- Separate “streams” for various customers
  - Need for multiple variants of the product
  - Need for multiple release of the product
- Stable points during development
  - Refer to “older” (stable) versions (for maintenance)

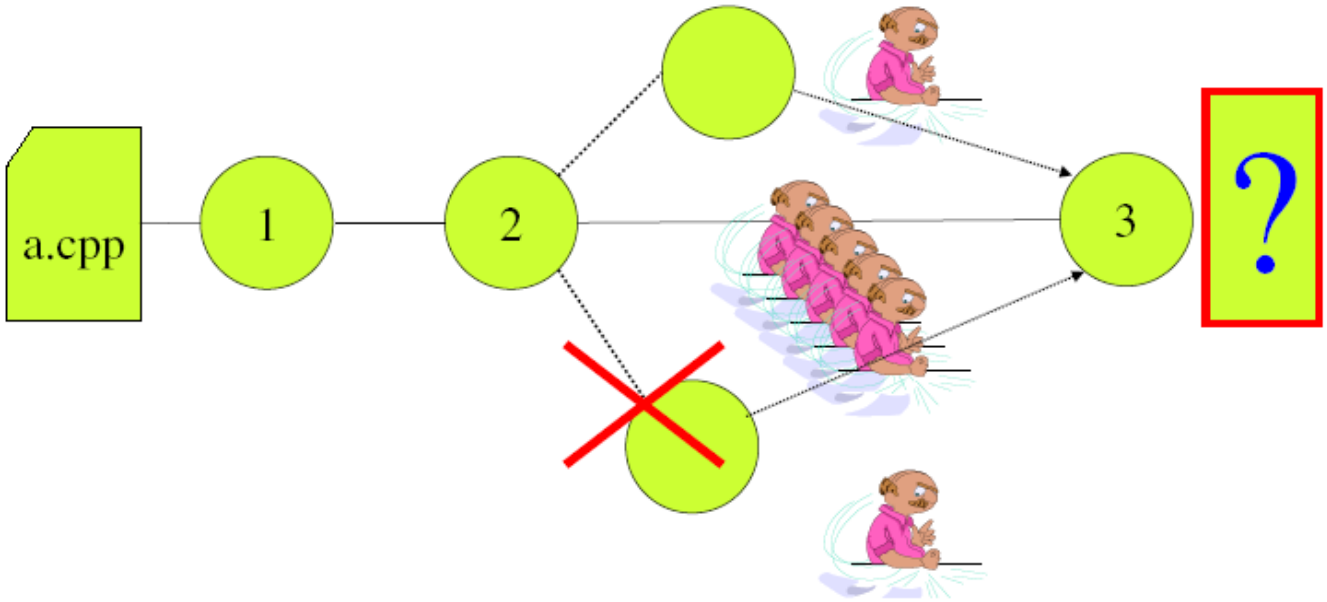
Creating Versions

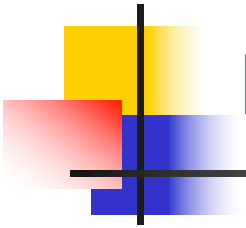
- Check out: Creation of local copy of CI (in order to change it)
- Commit (Check-In): integrate (Private) changes into the repository (to make it available to others)



Commit Conflict

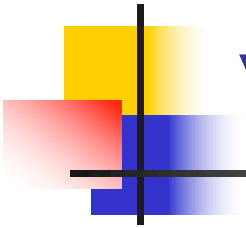
- Lock: exclusive to edit
  - Reserved check-out (ClearCase)





Locked Check-Out Deadlock

- Developer A:
  - Need file A, B and C to make change
  - Files A and B : Check out (locked)
- Developer B:
  - Need file B, C and D to make a change
  - Files C and D : Check out (locked)
- Developer A needs C to Check Out
  - File C is Locked, wait until unlocked
- Developer B needs B to Check Out
  - File B is Locked, wait until unlocked

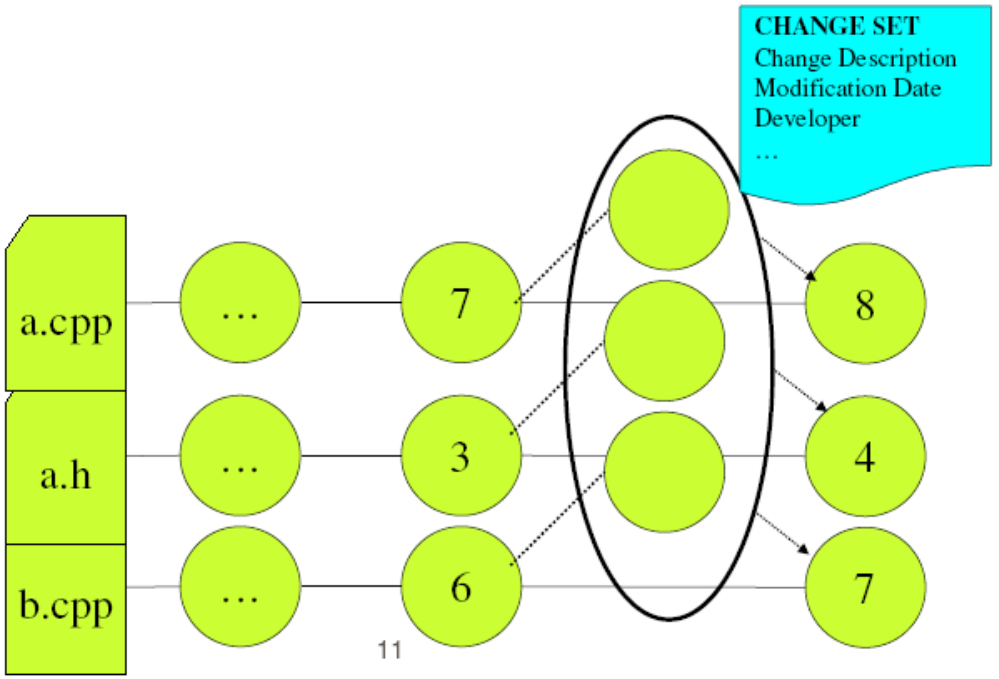


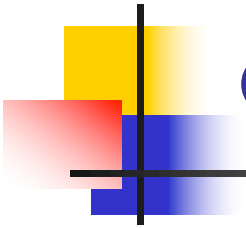
Version Coordination

- Aware of what other is doing
- Provide means to work in parallel
- Coordination of activity in repository
  - Project Leader
  - Architect
- Coordinate “ Merging” of results

Change Set

- Change Set: Coherent Set of CI's belonging to a change



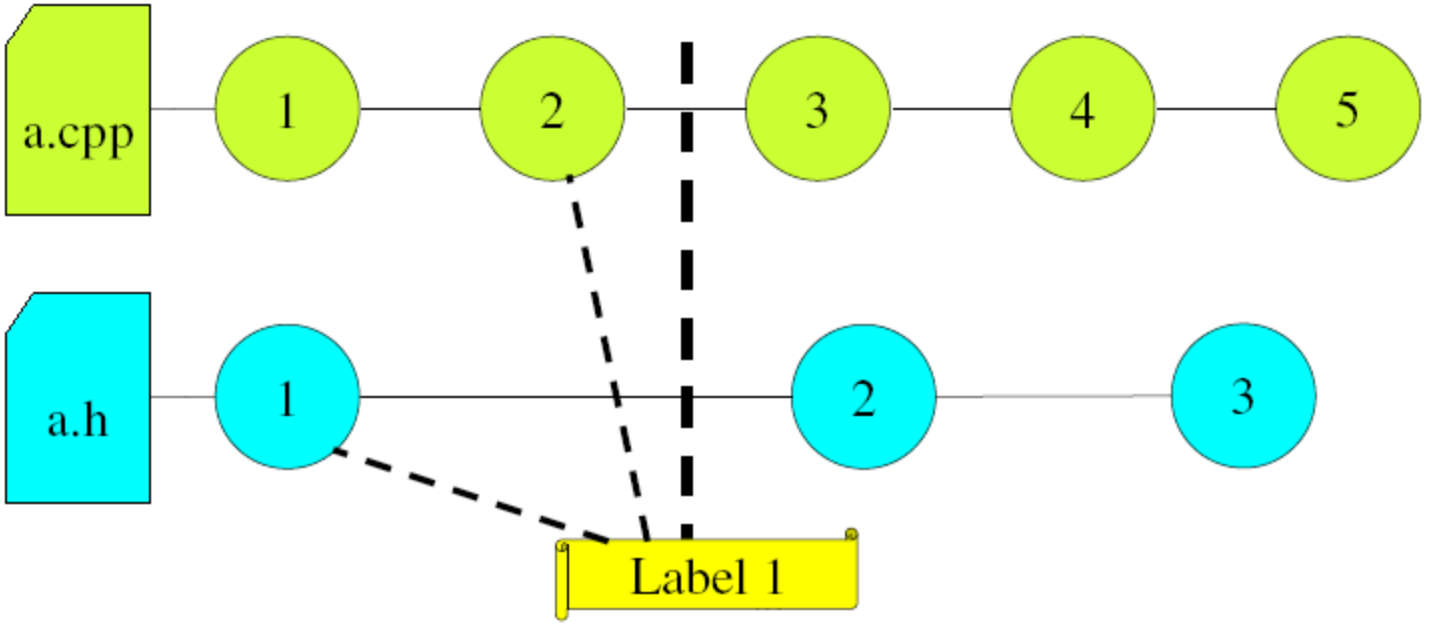


Change Set Coordination

- Know which CI versions belong together
- Know which change sets belong together

Labeling:

- Labelling : setting a tag on a set of CI versions



Configuration Selection

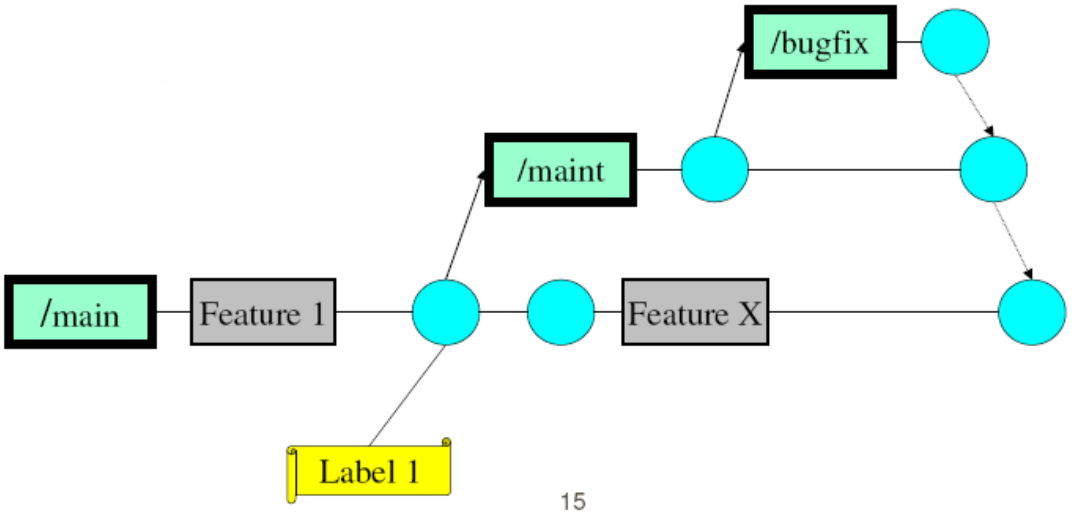
- Configuration Selection :
  - Selection of a set of CI versions
    - Enumerate Files and Versions
    - Refer to Labels
    - Refer to Dates
    - …

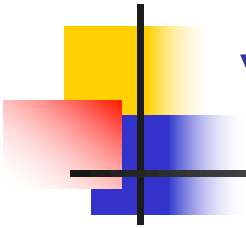
From Crnkovic

Item / Version	Release 2.3	Stable	Latest
Product	2.6.1.1	3.0	3.1
Item A	1.2	1.2	1.3
Item B	1.1	1.1	1.1
Item C	1.1	1.1.1.1	1.2
Item D	4.2	4.3	4.3

Branches

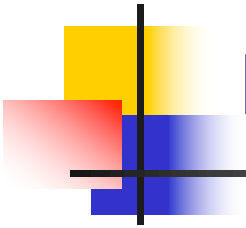
- Branch: Tagged set of CI versions
  - Branch (Create/Merge)
  - Change Reason
  - Version
  - Label





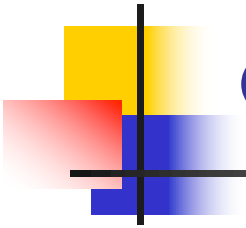
Why Do We need Branching ?

- Developer View
- Team Development
- Project Development
- Multiple Sites
- Variants in a Product Family
- Alpha/Beta-Release (for testing)
- Release
- Bugfix
- Try a Solution
- Maintenance



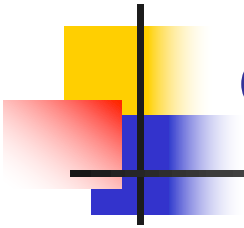
Branching Coordination

- Number of Branching can grow fast
- Branches
  - 1) end in a product and/or
  - 2) merge into another branch
    - Who will merge ?
    - When do we merge?
  - Where to make a change
    - Which branch ?
    - Which branches ?



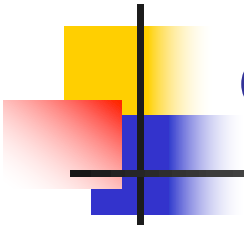
Coordination and SCM Plan

- Version Coordination
- Change set coordination
- Branching Coordination
- Labelling
- …
- Software Configuration Plan



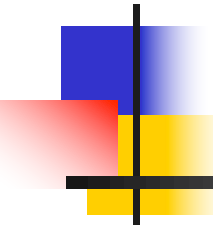
Complexity Explosion example

- Dependencies between CI versions
  - File a.c version 2 complies only with File b.h version 3
- Large Software Project
  - 5 Million Lines of Code
  - 100 K Functions/ Methods
  - 10 K Files (CI)
  - 100 components (CI)
  - 1 product (CI)



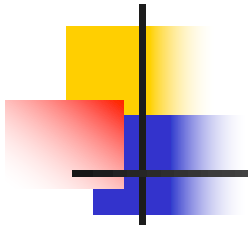
Conclusions

- Number of Versions in a system may explode
- Organization should describe process of how to manage versions example
  - Branching strategies
  - Locking of Versions
- Organizations should describe how to handle multiple parallel projects (strategy)

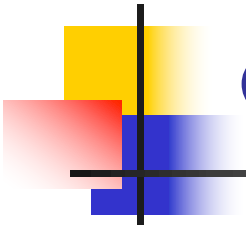


Software Project Management

Build Management

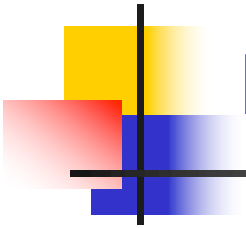


- Getting started with build management



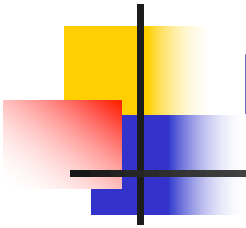
Objectives

- Learn the basics of Build Management
- Being able to understand and analyze Build Management in Industry



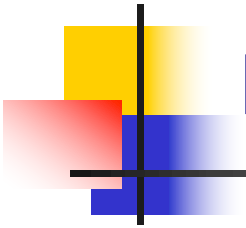
Build Management

- Support building a software system
  - Compile and Link source files
  - Generate User Interfaces (from High Level Specs)
  - Generate Documentation
  - …



Basic Notions of Building

- Build plan : set of build rules
- Build rule : Description of an action to produce derived object(s)
- Build step : Execution of an action (according to some rule)
- Build record : Trace of an actual build

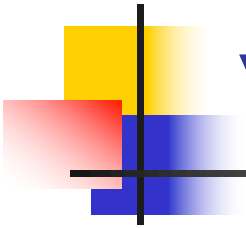


Build Tools

- Make: A program to maintaining Computer Programs (S.I. Feldman)
  - Dependencies between files
  - Based on Changes of Files (Time Stamps)
    - Synchronized time on build engines
    - integration with SCM tools
  - Re-build changed files and files that depend on other changed files
  - Makemake tools exists to create Make scripts
- Visual Studio Project/Solution Files (Microsoft)
- ANT and NANT (Java + .Net extension)

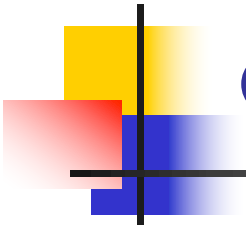
Simple Example of Make

- Main.exe: main.o a.o b.o
  - cc main.o a.o b.o –o main.exe
- Main.o: main.c main.h a.h b.h
  - cc –c main.c
- a.o: a.c a.h
  - cc –c a.c
- b.o: b.c b.h
  - cc –c b.c



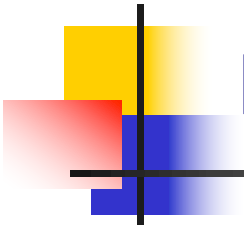
Variations in Build

- Building various “targets” / “configurations”
  - Test version (-Dtest)
  - Debug version (-debug)
  - Machine version (Win7, Win10)
- Being able to build parts of the system
  - Certain Components only
  - Configuration Selection (versions to be build)



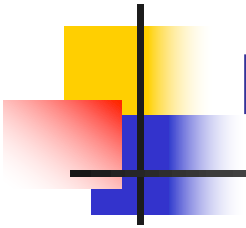
Configuration Selection

- Configuration Selection:
  - selection of a set of CI versions
- Build Management should be aware of configuration selection
  - In selecting versions of source
  - In selecting versions of derived objects



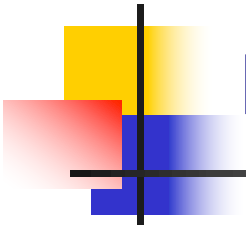
Functionality of BM

- Build and Store derived objects
  - including build record
- Performance of Build
  - Efficient calculation of firing build rules
    - Keep Dependencies in Memory
  - Determine reuse of derived objects (stored in SCM; understand build record)
  - Determine (possible) concurrency in build plan
  - Distributed build steps (use network of build engines)



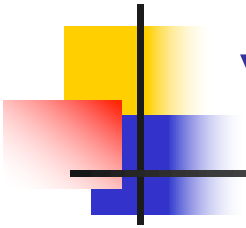
Dependencies in Build Rules

- Dependencies on Source Versions
- Dependencies on Derived Objects
- Dependencies on Libraries
- Dependencies on Tool (versions)
  - Compilers, Linkers, DocGen, …
- Build must rely on correct dependency information



Build in Development Process

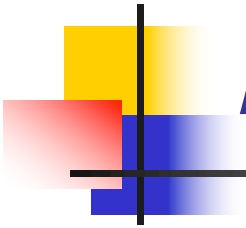
- Daily (System) Build
  - Build Complete System in a controlled way
  - Incorporate “Stable” versions of developers
- Good principles
  - Build repair has highest priority in process
  - Build repairs are resolved by “creator”
  - Build manager owns the build plans



What is build in practice

- Compile
- Link
- Install on target
- Test (on specific target machine)

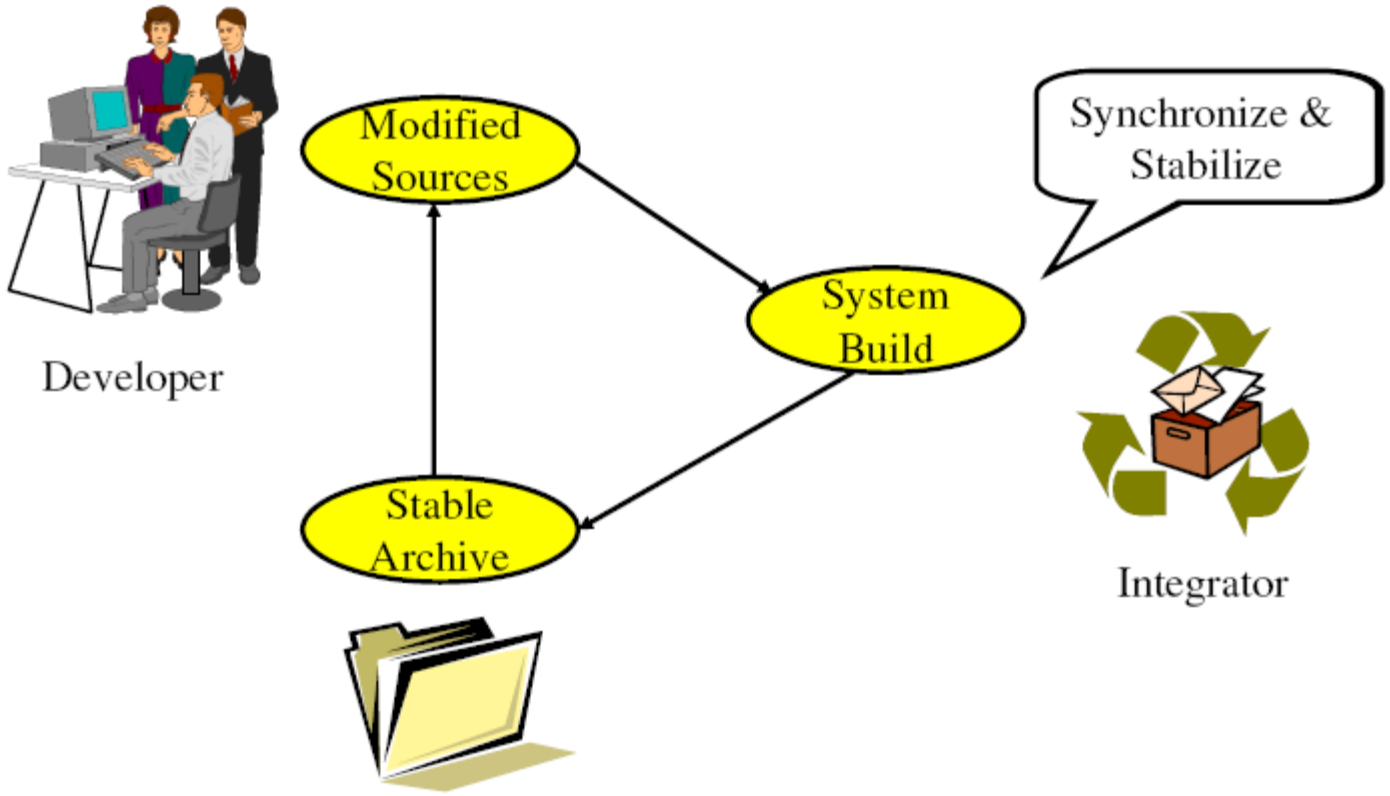
- Anything you want to do when source changes

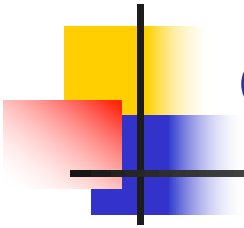


Models

- Daily Build and Smoke Test
  - Synchronize and Stabilize
- Continuous Integration
- Implementation Examples:
  - SoftFab (SkyFab)
  - CruiseControl

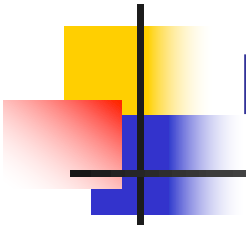
Daily Build and Smoke Test





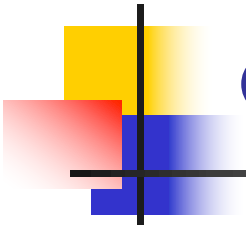
Continuous Integration

- Integrate Often
  - Finding bugs early
  - Resolve (developer interference) conflicts early
- Automated Build Scripts
- Automated Tests



Distributed Build Management

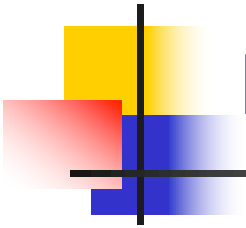
- Tool Support for Distributed Software Engineering
  - SoftFab
  - Int. Conf. Global SW Eng. 2006



CruiseControl

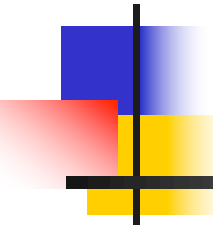
- Framework for continuous build
- Monitor status of many projects

- http://cruisecontrol.sourceforge.net



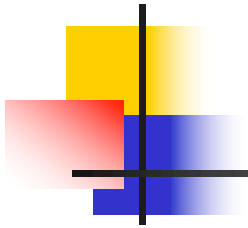
Recap

- Build Management provides build plans for different stakeholders
  - Developers, Maintenance Engineers, System builders, Testers
- Build Management must be aware of Version Control system
- Continuous Integration, DB&ST, ...

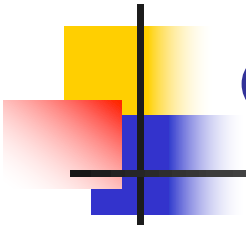


Software Project Management

Release Management

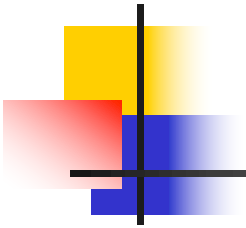


- Getting started with Release Management



Objectives

- Learn the basics of Release Management



Product Release

- Making a product available to its intended customers
  - External Release (Customers)
  - Internal Release (Developers)

- Identification of a Product Release by a "Release Number"

Customer Release Classification -1-

- Alpha Release
  - When? Significant increase of functionality
  - Purpose? Find defects and evaluate
  - Outcome? Bug repairs & New features
- Beta Release
  - What? Find defects and solve
  - Outcome? Bug repairs (no new features)
- Final Release
  - Customer release

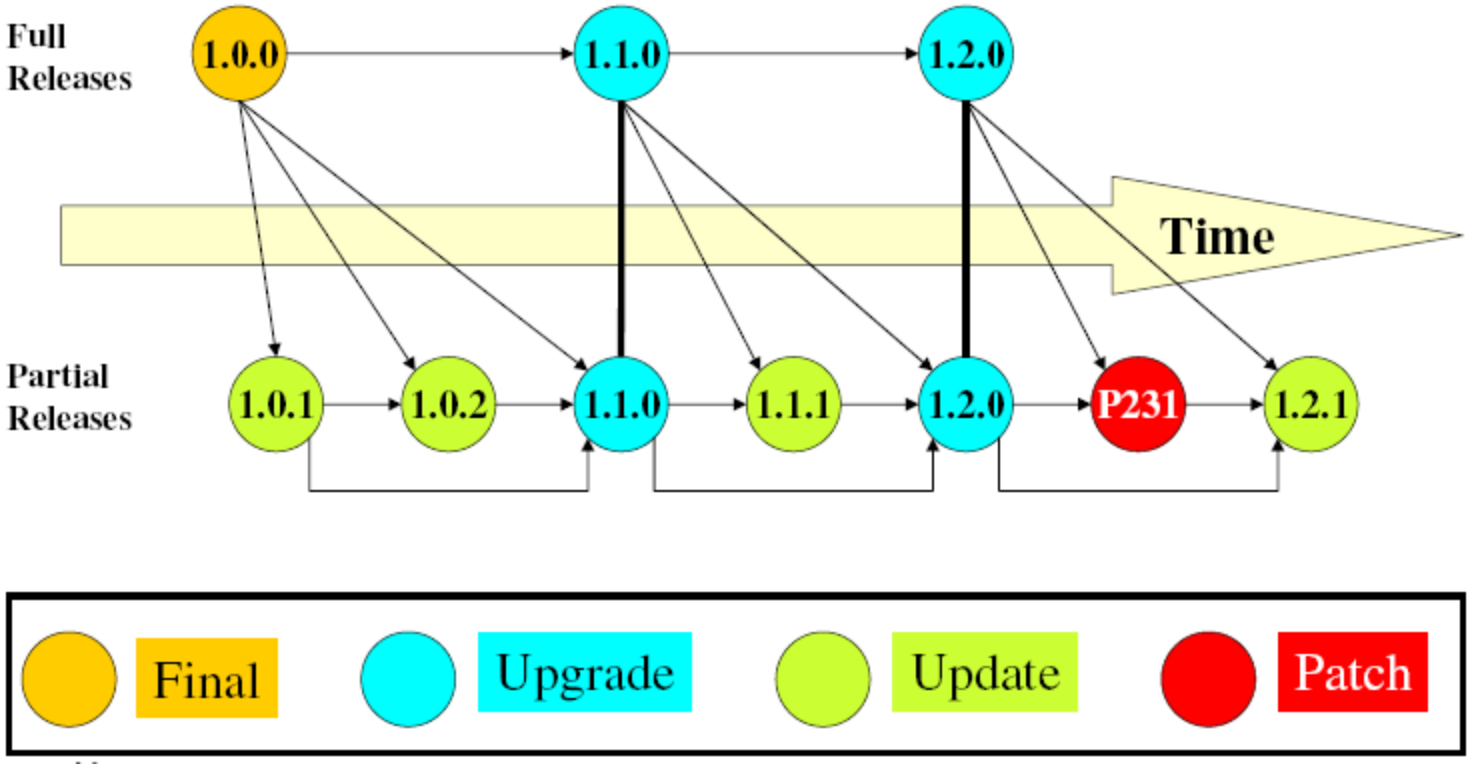
Customer Release Classification -2-

- Upgrade Release
  - Add (small) features (upgrade)
- Update Release
  - Fixing defects (update)
- Patches (Emergency Fixes)
  - Emergency (quick solution; can't wait)
  - Solve customer's difficulties

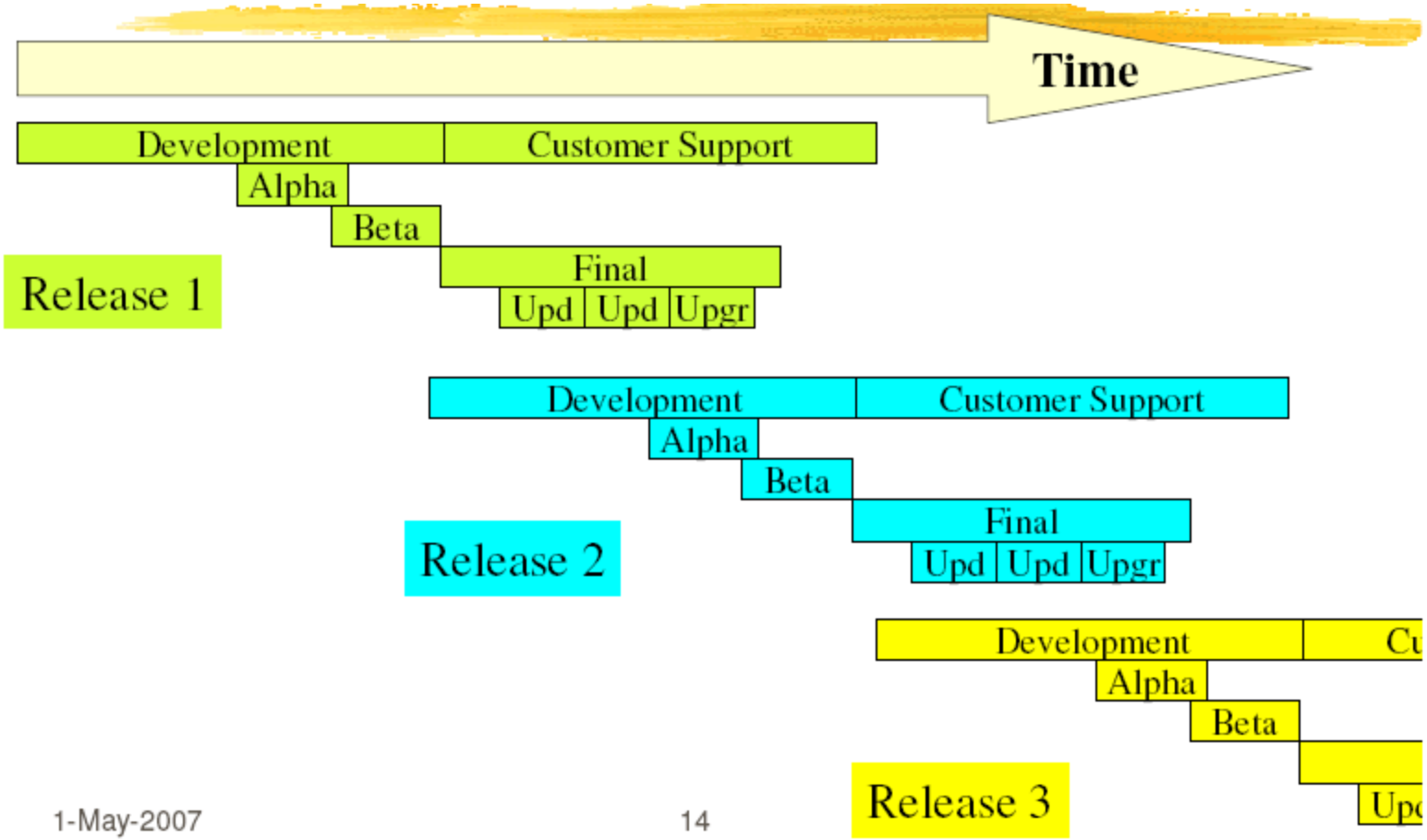
Release Numbering

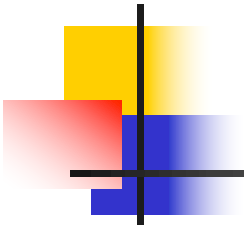
- X.Y.Z[A|Bn]
  - X: Major Release Number
  - Y: Feature Release Number
  - Z: Defect Repair Number
  - A|B: Alpha / Beta Release with a number 'n'
- Full Releases
  - Complete installation
- Partial Releases
  - Only certain components, files, executable, etc
  - Requires a previous Full Release

Full and Partial Releases



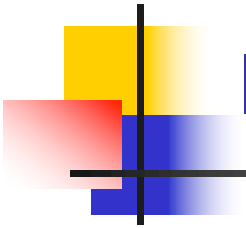
Development and Release





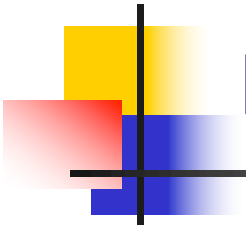
Internal Release

- Deliveries to internal users
  - Other developers (daily build)
  - Component Producer (team delivery)
  - Outsourced / Offshored Software Component
- Upcoming (external) Release Number is unclear
  - (Patch? Update? Upgrade? Final?)



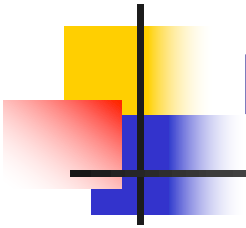
Internal Vs. External Release

- Predict Next Release (after R1.5.2)
  - Internal name is 1.5.3 (lowest possible)
  - Expected next upgrade release 1.6.0
- Special Encoding of Internal Releases
  - 1.5.3.001; 1.5.3.002 (extension)
  - Asterix.5.3 (project name encoding)
  - tj21042004.0929; dj31082005.1501



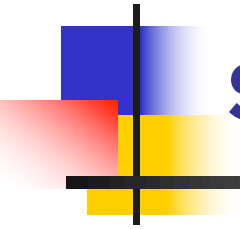
Release Distribution

- Packaging of Software Product
- Medium for distribution
  - CD; DVD; Internet; etc.
- Installation of Software



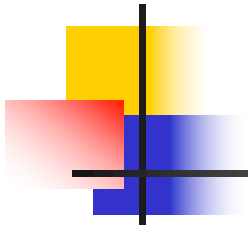
Recap

- Release Management
  - External Releases
    - Full Releases
    - Partial Releases (Upgrade and Patches)
  - Internal Releases
    - E.g. Component development
- Release Distribution
  - Packaging
  - Installation

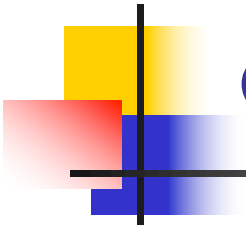


Software Project Management

Change and Defect Management

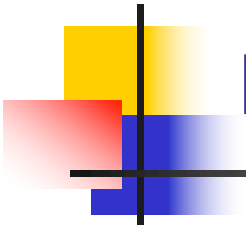


- Getting started with Change and Defect Management



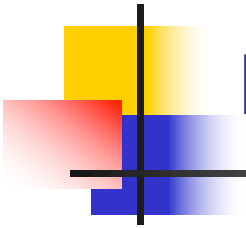
Objectives

- Learn about Defect Management
- Learn about Change Management
- Learn about the implementation of Change/Defect Management



Defect

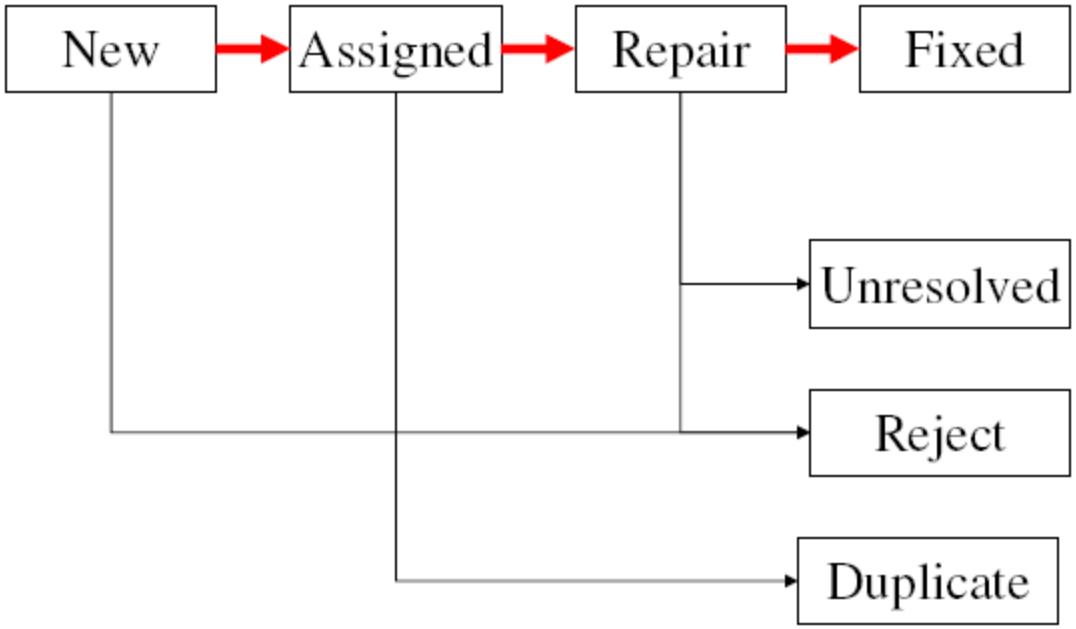
- Any nonconformance of a characteristic with specified requirements
  - Problem Report, PR
- Content? What is needed to describe a Defect?

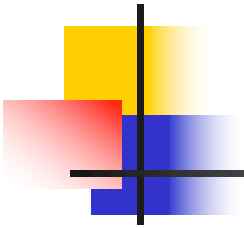


Defect Content

- Description
- Configuration Item
- Release / Baseline / Version
- Date the Defect Raised
- Consequences
- Log files
- Risks
- Severity
- Priority
- State

Defect Flow

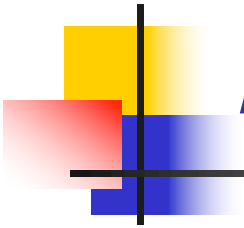




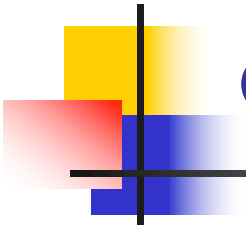
Defect Tracking

- Which defects are resolved in this version?
- Did we fix Defect 12345?
- Is Defect 21345 resolved in platform X?
- Can I merge fix of defect 12345 in my branch?

Integration Change and Version Management

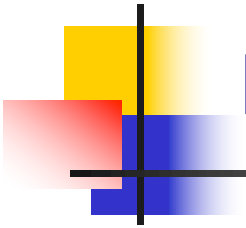


- CR / PR may result some actions to resolve it
- Resolution results in changes of files
- (doc's, source, test scripts, etc.) (by someone)
- New version of these files is committed; some reference to the CR/PR



Change Set

- Set of CI's being modified for resolving PR or implementing CR
- In Theory:
  - Any (random) set of Change Sets is a new release providing the described features and fixed defects (given starting point: baseline)



Recap

- Defects have to be stored and managed
- Change Requests can be handled similar to defects but scope and impact is larger
- Metrics provide insight in project status
- Implementation of Defect Management consists of
  - Clear description of content
  - Flow in which a defect exists
  - Actors make transitions in the flow