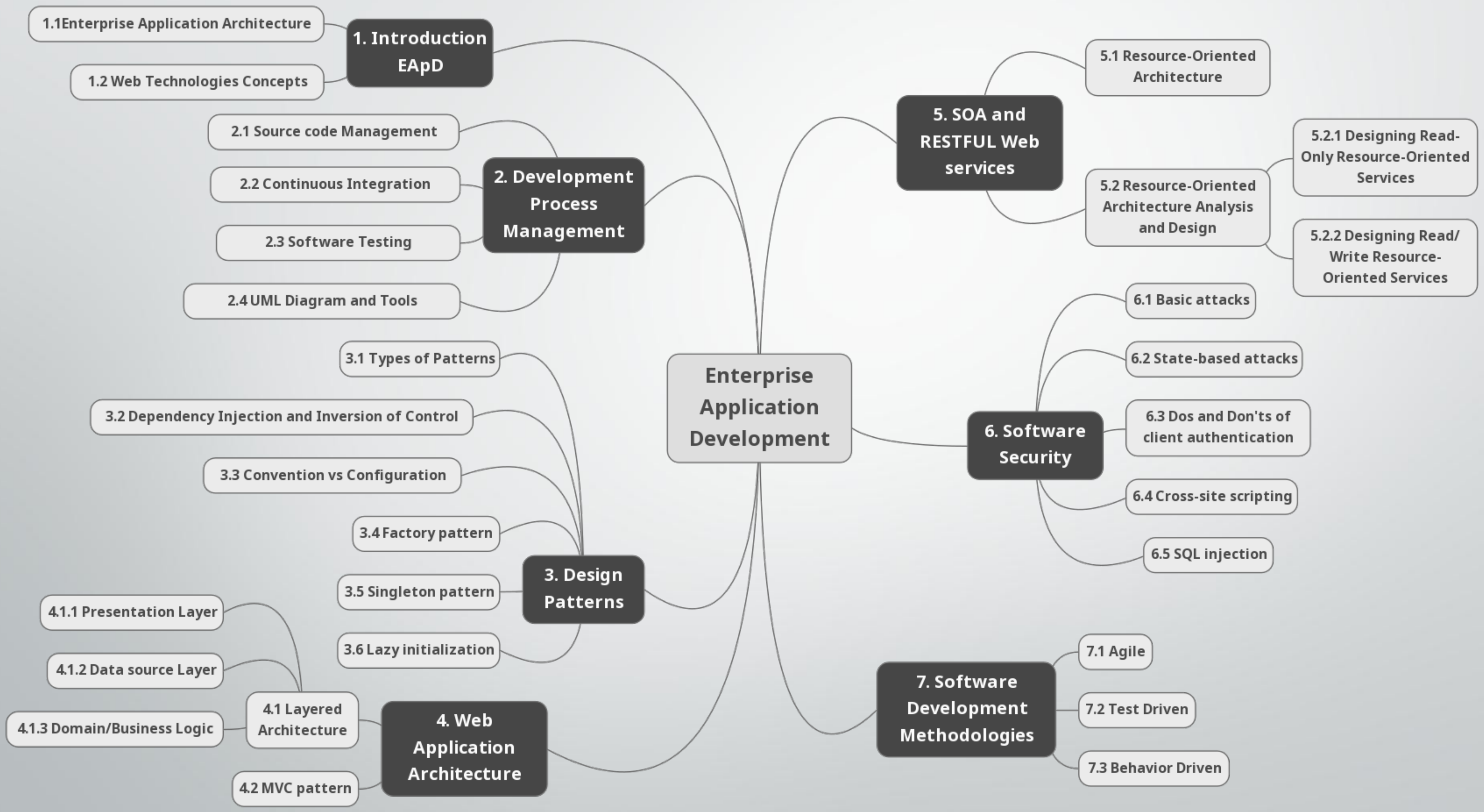
Enterprise Application Development

[BE SE-7th Semester]

Nepal College of Information Technology

POKHARA UNIVERSITY

Enterprise Application Development


Enterprise Application Development
1. Introduction EApD
1.1Enterprise Application Architecture
1.2 Web Technologies Concepts
2. Development Process Management
2.1 Source code Management
2.2 Continuous Integration
2.3 Software Testing
2.4 UML Diagram and Tools
3. Design Patterns
3.1 Types of Patterns
3.2 Dependency Injection and Inversion of Control
3.3 Convention vs Configuration
3.4 Factory pattern
3.5 Singleton pattern
3.6 Lazy initialization
4. Web Application Architecture
4.1 Layered Architecture
4.1.1 Presentation Layer
4.1.2 Data source Layer
4.1.3 Domain/Business Logic
4.2 MVC pattern
5. SOA and RESTFUL Web services
5.1 Resource-Oriented Architecture
5.2 Resource-Oriented Architecture Analysis and Design
5.2.1 Designing Read-Only Resource-Oriented Services
5.2.2 Designing Read/ Write Resource-Oriented Services
6. Software Security
6.1 Basic attacks
6.2 State-based attacks
6.3 Dos and Don'ts of client authentication
6.4 Cross-site scripting
6.5 SQL injection
7. Software Development Methodologies
7.1 Agile
7.2 Test Driven
7.3 Behavior Driven

Source Code Management

- Also known as Configuration Management
- Source Code Managers are tools that:
  – Archive your development files
  – Serve as a single point of entry/exit when adding or updating development files

Why You Want
A Source Control System ?

- Supports concurrent development
- Manage diverging source code bases
- Records file/release versions
- Easy access to all previous revisions
- Can record *why* a revision was made
- Optimal disk space usage
- You’ll end up doing something equivalent anyway so it may as well be automated

Source Code Management Tools Are Not

A substitute for project management

A replacement for developer communication

How They Work

- Central database of source code, documentation, build tools
- Each file stored only once - all other versions are diffs of that one copy
- To Make a Change
  - Check out the latest version of a file
  - Make the changes
  - Update the database

What should be in the database

- Source Code
- Documentation
- Build Tools
  - Often need old versions of the tools to build old versions of the software
  - Ensures software is rebuilt exactly as the customer received it
- Test Suites
- Anything else you might want later

Version Control

- Companies ship several products from the same source base (ie Win NT and Windows 2000 versions of MS Office)
- When tracking down bugs you want to examine the code as it was when the product shipped

Code Sharing

- Multiple people can work on the same source base without colliding
  - (1) Locks individual files so only one person at a time can modify it *OR*
  - (2) Allows multiple people to modify a source file and the system will automatically merge the changes (usually)

Locking

- Only one person can work on a file at once
- Works fairly well if developers work on different areas of the project and don’t conflict often
- Problem 1: People forget to unlock files when they are done
- Problem 2: People work around locking by editing a private copy and checking in when the file is finally unlocked - easy to goof and lose changes

Merging

- Several people can work on a file at once
- Before committing changes, each user merges their copy with the latest copy in the database

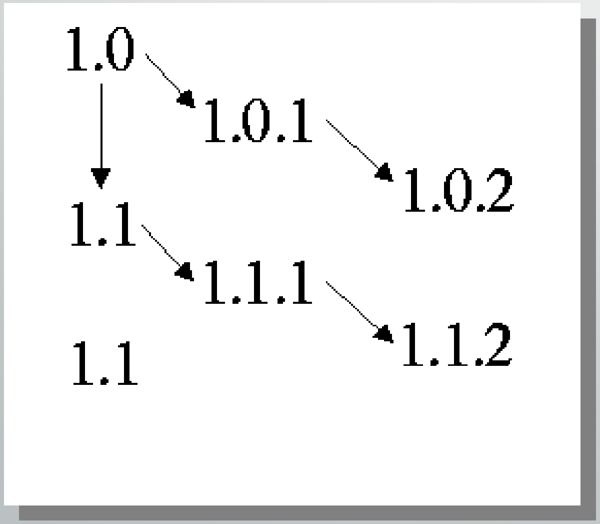
– This is normally done automatically by the system and usually works, but you should not blindly accept the result of the merge

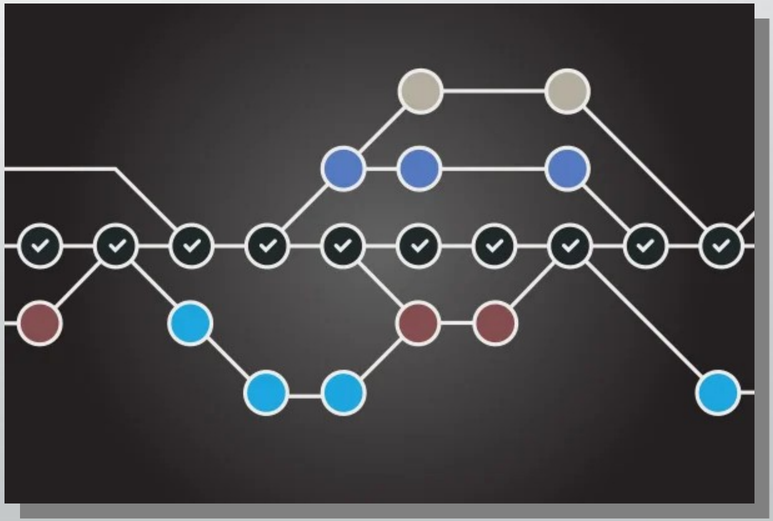
Labelling

- Label all the files in the source base that make up a product at each milestone
- Just before and just after a major change
(eg. changing several interfaces)
- When a new version ships

Version Trees

- Each file in the database has a version tree
- Can branch off the version tree to allow separate development paths
- Typically a main path (trunk) for the next major version and branches off of shipped versions for maintenance





Branching

- When a new version ships, typically create a branch in the version tree for maintenance
- Double update: fix a defect in the latest version and then merge the changes (often by hand) into the maintenance version
- Also create personal versions so you can make a change against a stable source base and then merge in the latest version later

Examples

- **Revision Control System (RCS)**

  → Solaris: man rcsintro

- **Concurrent Versions System (CVS)**

  →Solaris: man cvs

  → www.cyclic.com/cvs/info.html

- **Visual SourceSafe**

  → msdn.microsoft.com/SSAFE

- **ClearCase** → www.rational.com

- **Git** →www.git-scm.com

Revision Control System (RCS)

- File management only
- Transaction model
  - check out and lock
  - edit
  - check in and unlock
- Little support for binaries

Concurrent Versions System (CVS)

- Built on top of RCS
  - Therefore little support for binaries
- Database can be remote
- No locking: merge before commit
- Fast
- Integrates with emacs

SourceSafe

- Microsoft’s entry into the field
- Project-based
- Checkout-edit-checkin model
- Built-in web site creation tools
- Integrates with MSDEV

Clearcase

- Clearcase is configuration management on steroids
- You create a *view* of the database with a *config spec*, which describes how to select files from the database.
- When you set a view, Clearcase creates a virtual filesystem containing only those versions of the files selected by the config spec

Clearcase Features

- Distributed System
  - Several groups at different locations can work on the same database
- Can install triggers
  - Example: e-mail the author of a file when some one makes a change to it
- Uses merging model like CVS, but can also lock files

More Clearcase Features

- Integrates with MSDEV
- Build Management
  – Knows to rebuild out-of-date files even if your makefile doesn’t
- Slow and a bit buggy

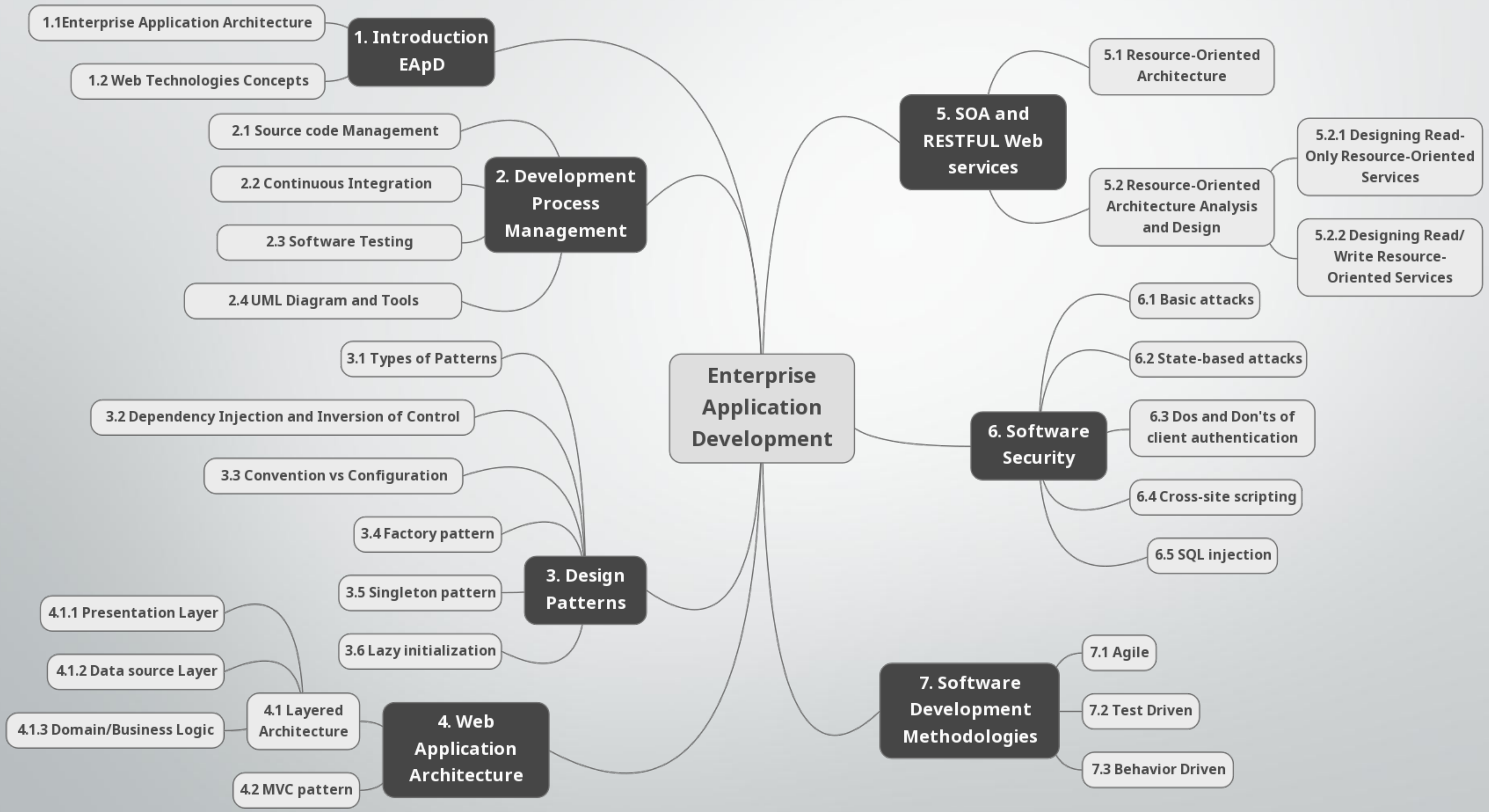
Helpful Rules for Version Control Bliss

- Archived Files Should Always Compile
- Code Review Files Before Check-in
- Compile and run latest archived files *as a set* before Check-in
- No Cheating (even “simple bug fixes” need to undergo this process)

Version Control →Best Practices

- Commit logical changesets (atomic commits)
- Commit Early, Commit Often
- Write Reasonable Commit Messages
- Don’t Commit Generated Sources
- Don’t Commit Half-Done Work
- Test Before You Commit
- Use Branches
- Agree on a Workflow

Enterprise Application Development



Enterprise Application Development
1. Introduction EApD
1.1Enterprise Application Architecture
1.2 Web Technologies Concepts
2. Development Process Management
2.1 Source code Management
2.2 Continuous Integration
2.3 Software Testing
2.4 UML Diagram and Tools
3. Design Patterns
3.1 Types of Patterns
3.2 Dependency Injection and Inversion of Control
3.3 Convention vs Configuration
3.4 Factory pattern
3.5 Singleton pattern
3.6 Lazy initialization
4. Web Application Architecture
4.1 Layered Architecture
4.1.1 Presentation Layer
4.1.2 Data source Layer
4.1.3 Domain/Business Logic
4.2 MVC pattern
5. SOA and RESTFUL Web services
5.1 Resource-Oriented Architecture
5.2 Resource-Oriented Architecture Analysis and Design
5.2.1 Designing Read-Only Resource-Oriented Services
5.2.2 Designing Read/ Write Resource-Oriented Services
6. Software Security
6.1 Basic attacks
6.2 State-based attacks
6.3 Dos and Don'ts of client authentication
6.4 Cross-site scripting
6.5 SQL injection
7. Software Development Methodologies
7.1 Agile
7.2 Test Driven
7.3 Behavior Driven