UNIVERSITY OF LEICESTER

Writing Maintainable Code and Managing Change

CO3095, CO7095, CO7508

José Miguel Rojas – *material provided by Neil Walkinshaw*

Coding Conventions and Design / Architecture Patterns

Reasoning in Abstract Terms

Focus on the essentials, remove irrelevant detail.

Easily share solutions that have proven to be successful.

Coding Guidelines

- Design
- Often

es.

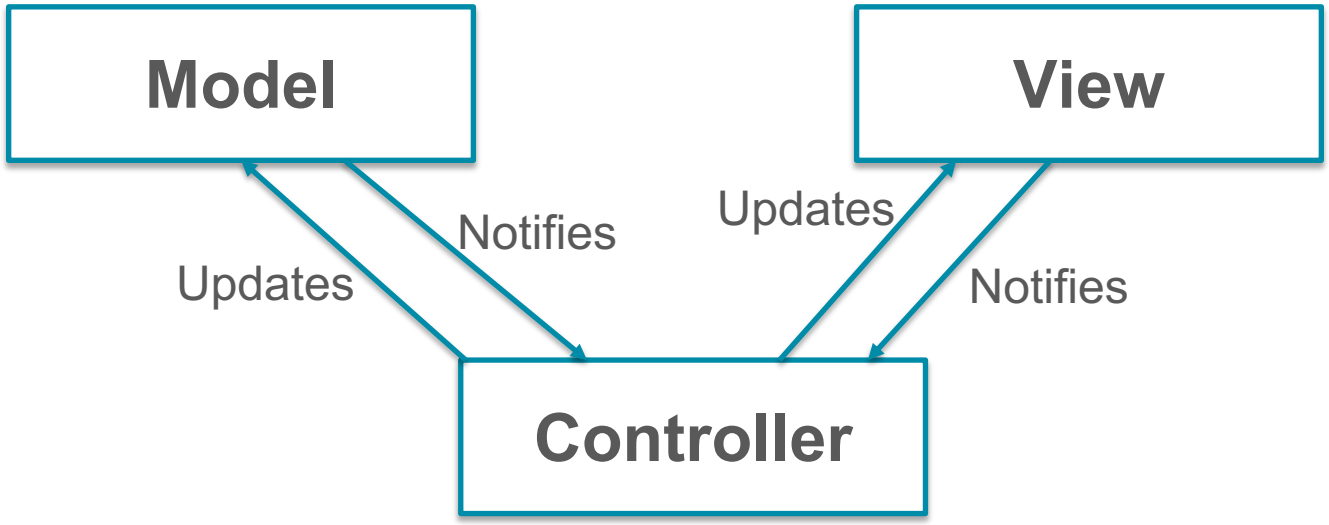
andards.

- **Source files**
  - Line terminators and the ASCII 'space' characters are the only white-space characters allowed in a file. Tabs are not allowed.
  - A source file consists of: (1) License information (if present), (2) Package statement, (3) import statements, (4) exactly one top-level class. Exactly one blank line separates each of these sections.
  - No wild-card imports.
  - No line-wrapping.
- **Formatting**
  - Braces are always used where optional.
  - Use of braces should follow the "egyptian style" - the opening brace should be in-line with the preceding declaration or predicate, and the closing brace trails the block on its own line.
  - Blocks should be indented with two spaces.
  - Empty blocks may be concise - i.e. {}
  - Only one statement per line.
  - Multiple consecutive blank lines are permitted, but never encouraged.
  - Optional grouping parameters are encouraged.
- **Naming**
  - Class names should be written in UpperCamelCase.
  - Method names should be written in lowerCamelCase.
  - Constant names should be written in CONSTANT_CASE.
  - Non-constants, parameter names, local variable names, should be written in lowerCamelCase.
- **Programming Practices**
  - Caught exceptions should never be ignored.
  - Do not use object finalizers.
- **JavaDocs**
  - At the minimum, Javadoc is present for every public class, and every public or protected member of such a class, unless the method is truly self explanatory (e.g. is a getter method), or overrides a method.

Google Coding Style: http://google.github.io/styleguide/

The Model View Controller Pattern

"How do I link my control logic, visual interface, and data?"



- An architectural pattern
- Organise the code into three packages to separate concerns
- Basis for Ruby on Rails, Spring, …

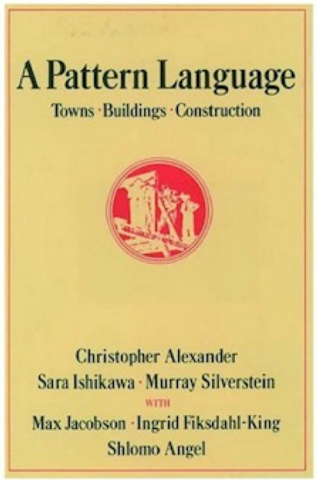
Patterns

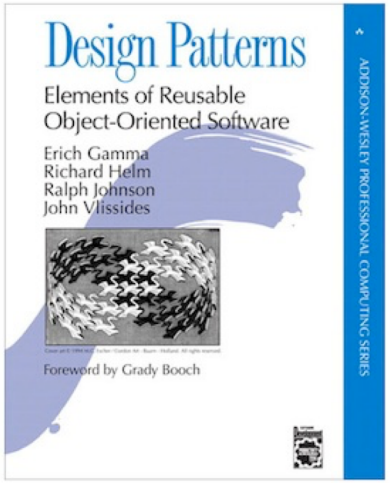
"How do I link my control logic, visual interface, and data?"

"How do I notify one piece of code every time the state of a particular variable changes?"

"How can I manage interactions with a complex set of libraries?"

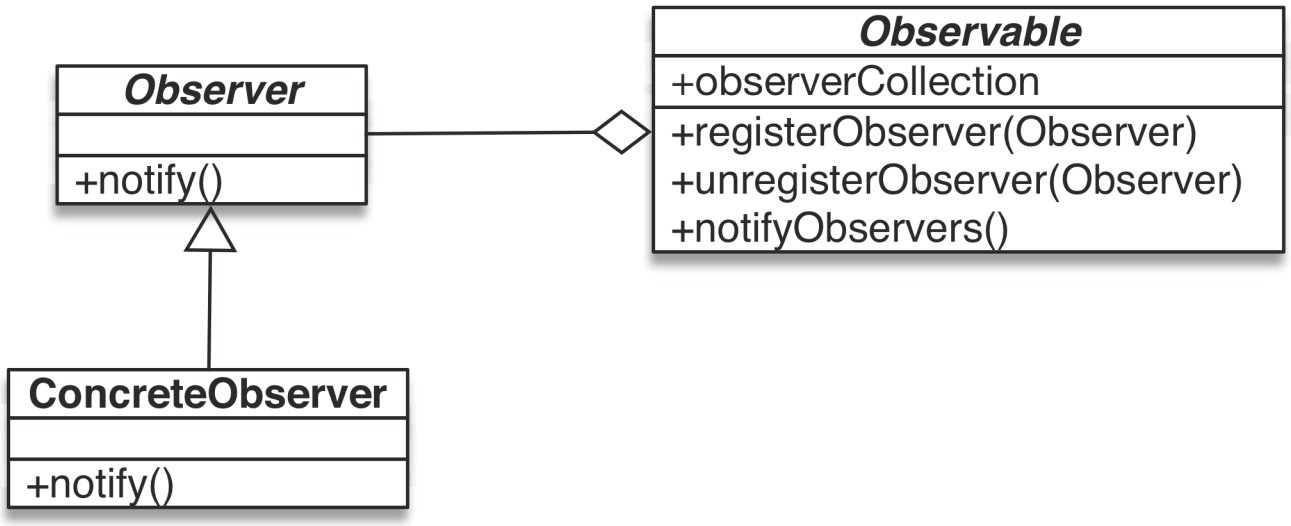
Repeatable solutions to common problems





The Observer Pattern

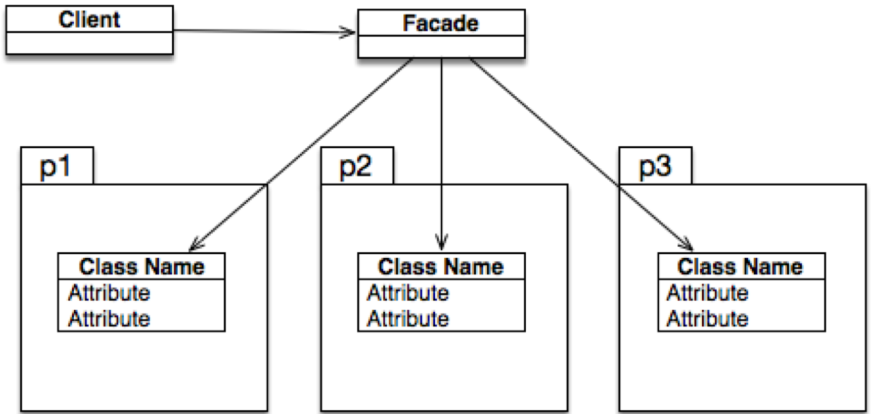
"How do I notify one piece of code every time the state of a particular variable changes?"



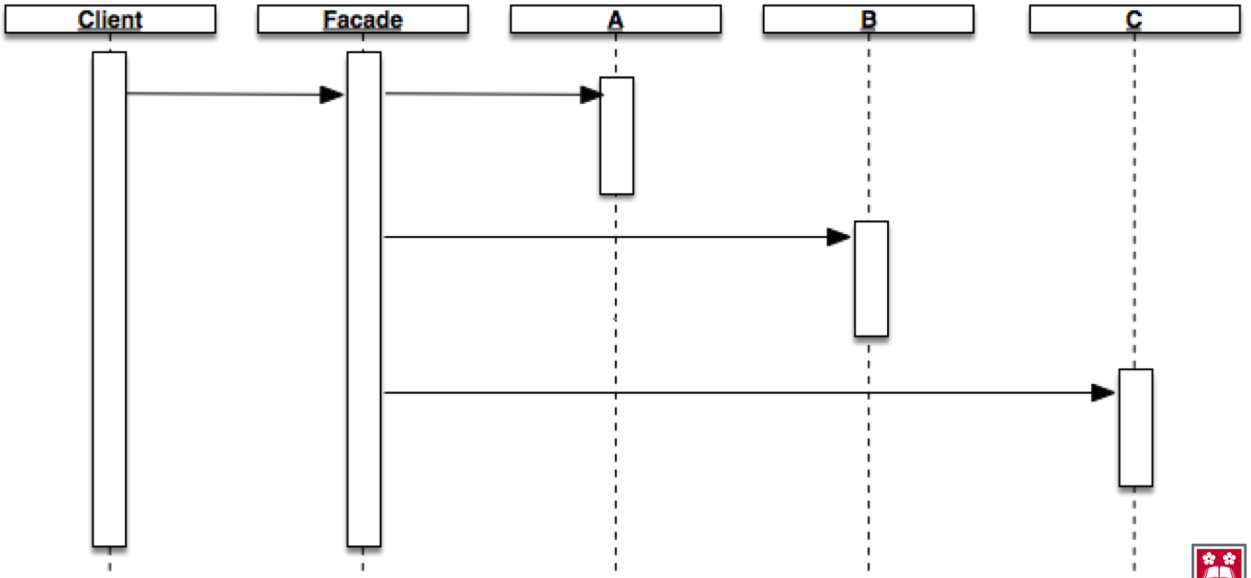
A Design Pattern

Built into many development frameworks (e.g. Java SDK)

Facade Pattern



Wrap access to complex sub-systems behind a single “facade” class.



Fostering Understandability

- Patterns and coding rules foster understandability
- Developers can understand systems in abstract terms
- Immediately apparent that:
  - classes in package X are controller-related
  - class X is an “Observable”
  - class Y is extended by an “Adaptor”…
- **Modularise** key concerns
- Easier to debug, extend, and maintain

Collaborative Software Development

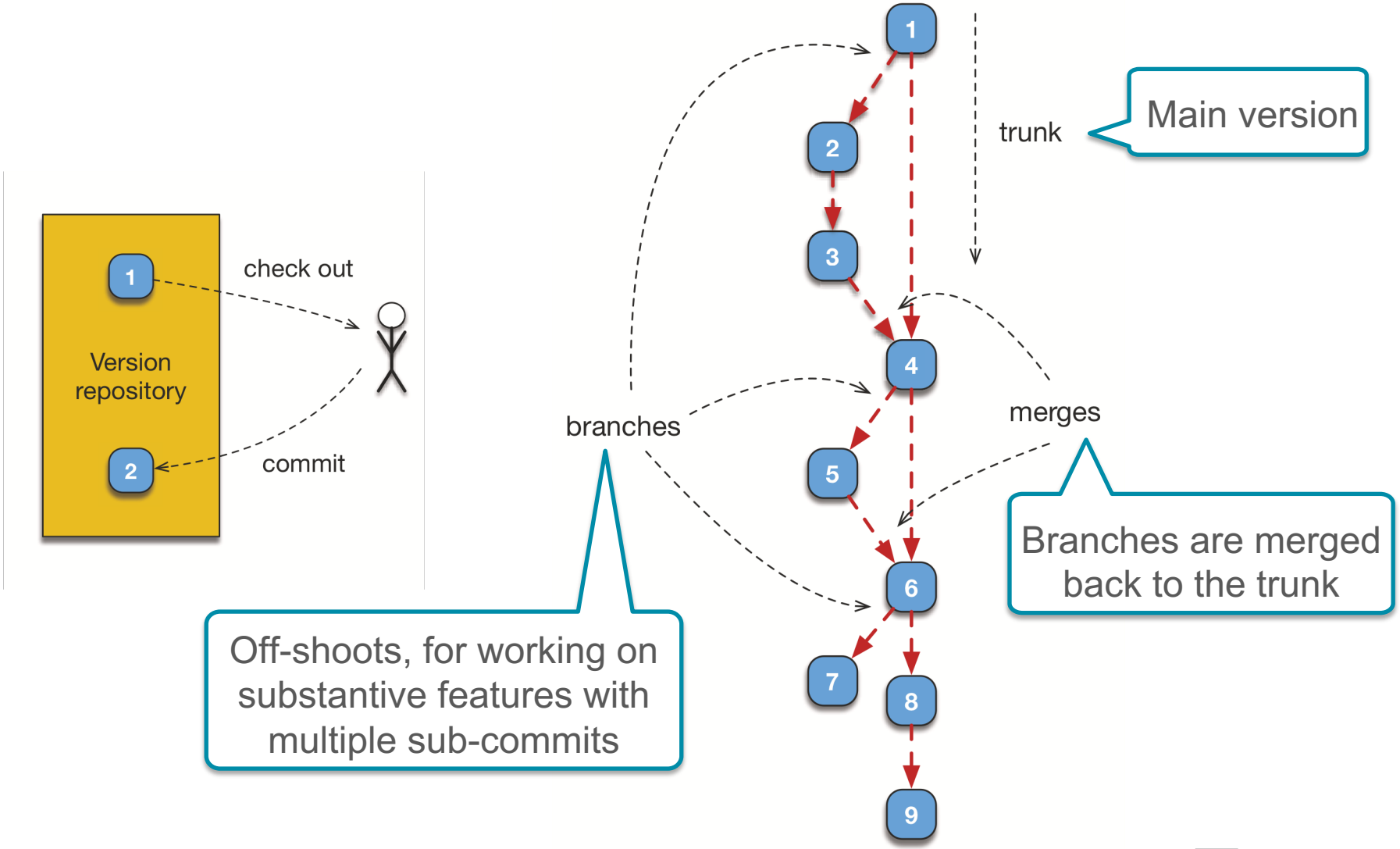
Software Continually Changes

“The node.js source code as visualized by Gource”,
“Linux Kernel Development, 1991-2015” on Youtube

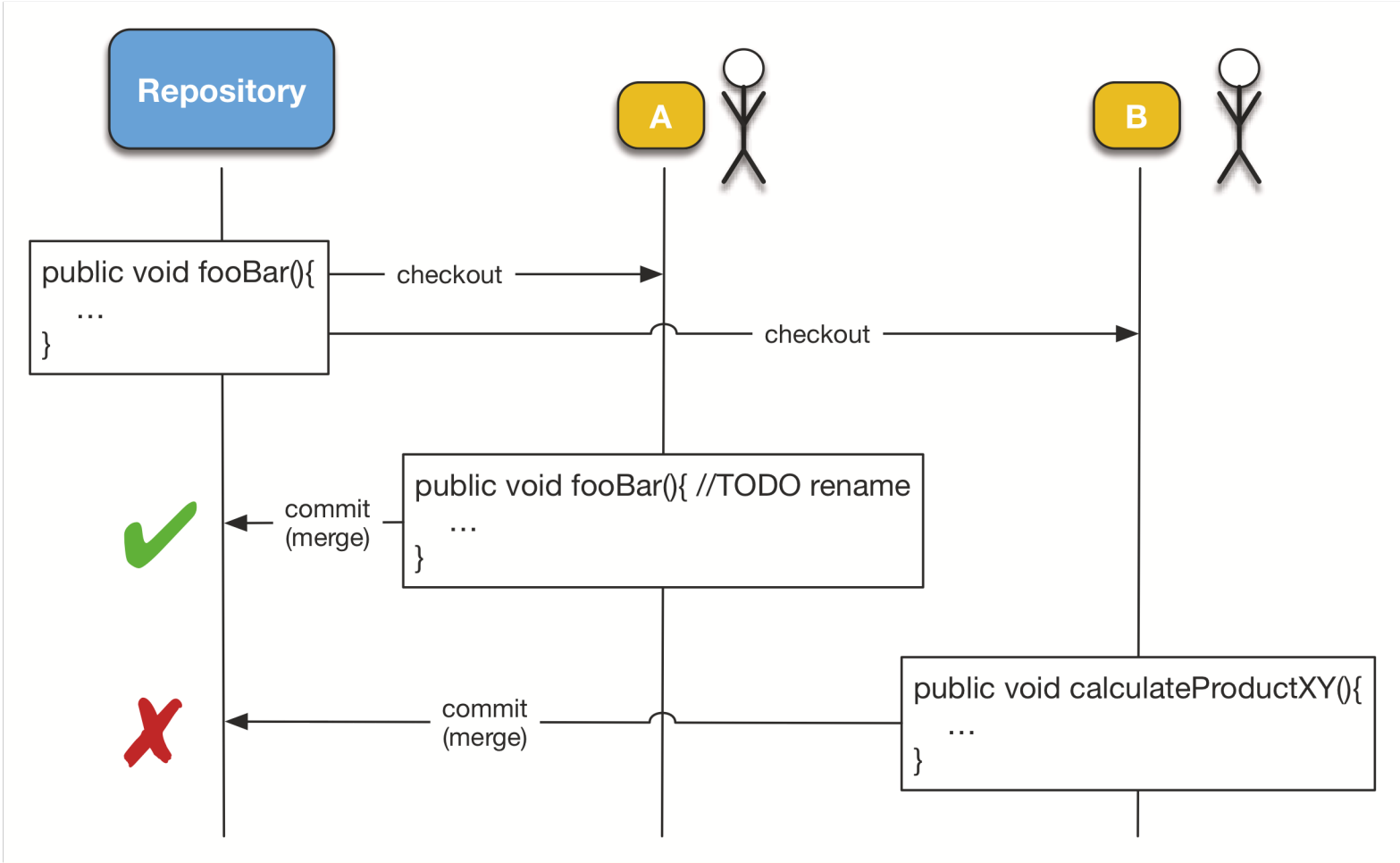
What are the challenges?

- Source code is **dense and interconnected**.
- Changes can have **wide impact** and may affect many files
  - Lead to changes in test cases
  - Require changes in documentation
  - May have to be reflected in requirements…
- Changes may have **unintended consequences**
  - How to undo a change if it breaks functionality?

Tracking Changes with Version Control Systems



Merge Conflicts

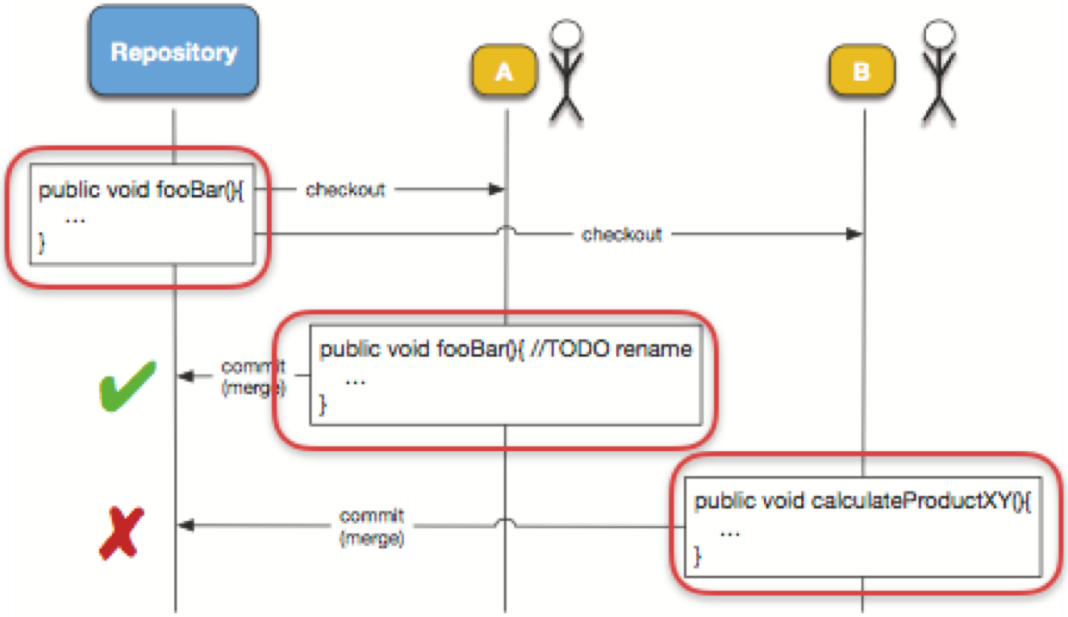


Three-way Merging

Developer resolves conflict by reconciling three versions of code:

1. The version that they checked out
2. The version as updated (by others) since the check-out
3. The conflicting version they want to commit

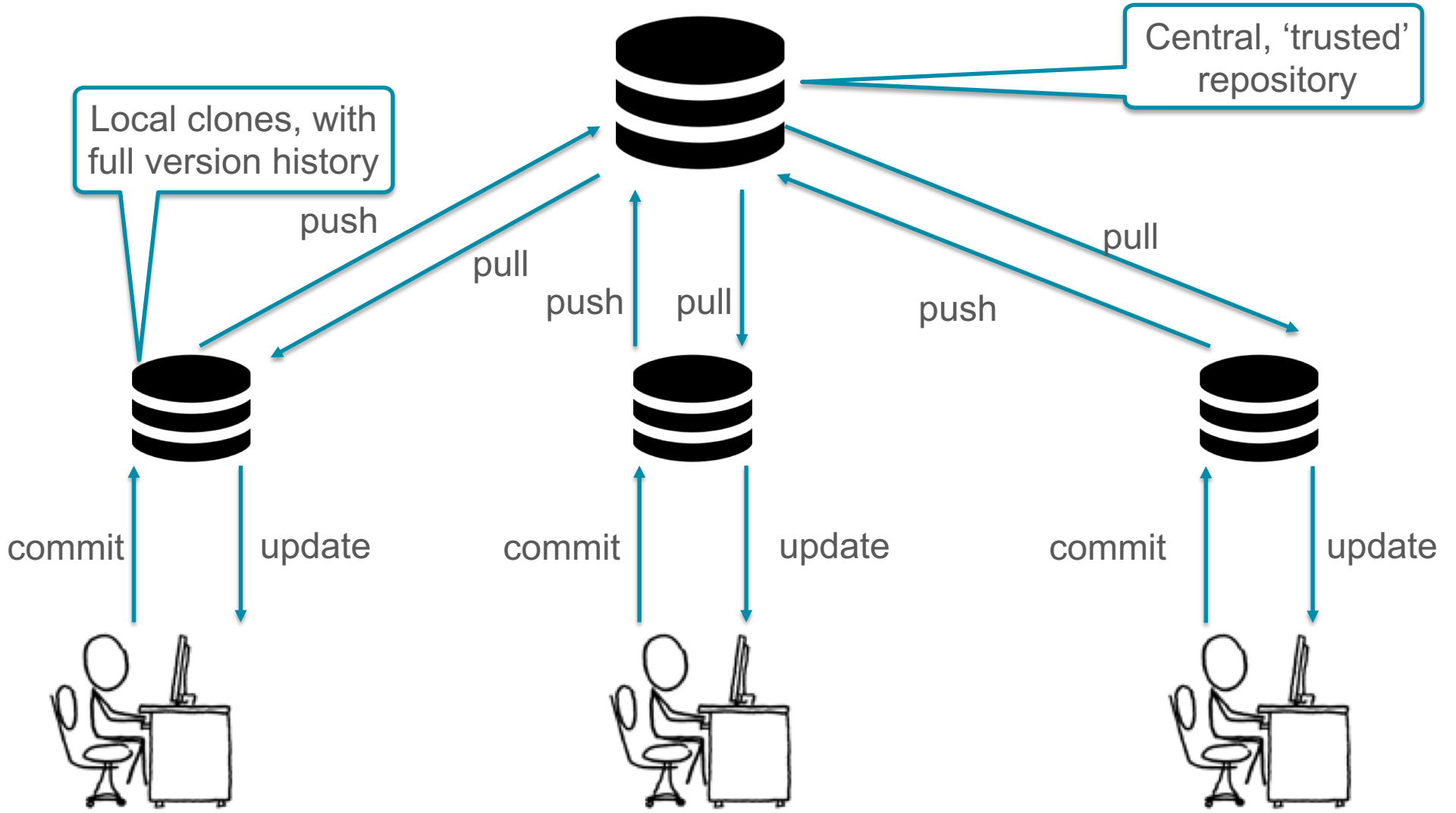
Many editors support this, such as GNU DiffUtils



Avoiding Merge Conflicts

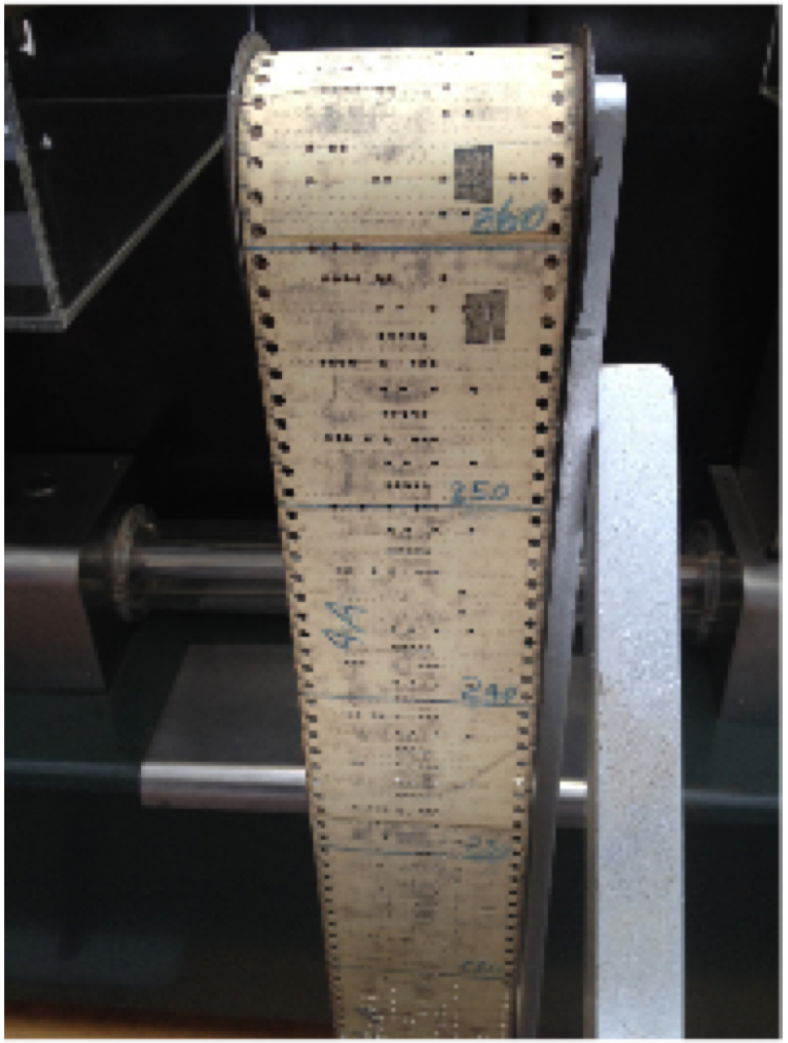
- Merge conflicts are problematic
  - Resolving them can be **time-consuming**
  - Can also **introduce bugs**
  - Symptomatic of people **duplicating** each other's **effort**
- Can be avoided / minimised by:
  - **Communication** amongst developers
- Daily SCRUM, Kanban boards, etc
  - Restriction to **small "atomic" commits**

Decentralised Version Control Systems



From “versions” to “patches”

- Each commit and update is a “patch”.
- A patch is a “diff”
  - The edit required to change one version into another
- Easier to communicate and understand changes…
  - …as opposed to entire versions



Semantic Versioning

How to assign version numbers?

1, 2, 3, 4, 5, 6, 7, …?

3.1, 95, ME, 98, 2000, XP, 2007, 2010?

Major.Minor.Patch

Start from 0. Increment every time you make compatibility-breaking API changes

Increment every time you add functionality in a backward-compatible manner

Increment every time a minor change is made / bug is fixed (in a backwards compatible way)

Summary

Patterns and style guides ensure understandability by encouraging developers to adopt recognisable solutions.

Code is subject to continuous change, which poses a management challenge.

Version repositories support collaborative development.

Distributed version repositories enable collaboration without direct access to a server.

Semantic versioning makes it easier to understand the nature of a change.