Scrum & Software Design

CS 461

Scrum: INVEST

Bill Wake

I Independent

The user story should be self-contained, in a way that there is no inherent dependency on another user story.

N Negotiable

User stories, up until they are part of an iteration, can always be changed and rewritten.

V Valuable

A user story must deliver value to the end user.

E Estimable

You must always be able to estimate the size of a user story.

S Small

User stories should not be so big as to become impossible to plan/task/prioritize with a certain level of certainty.

T Testable

The user story or its related description must provide the necessary information to make test development possible.

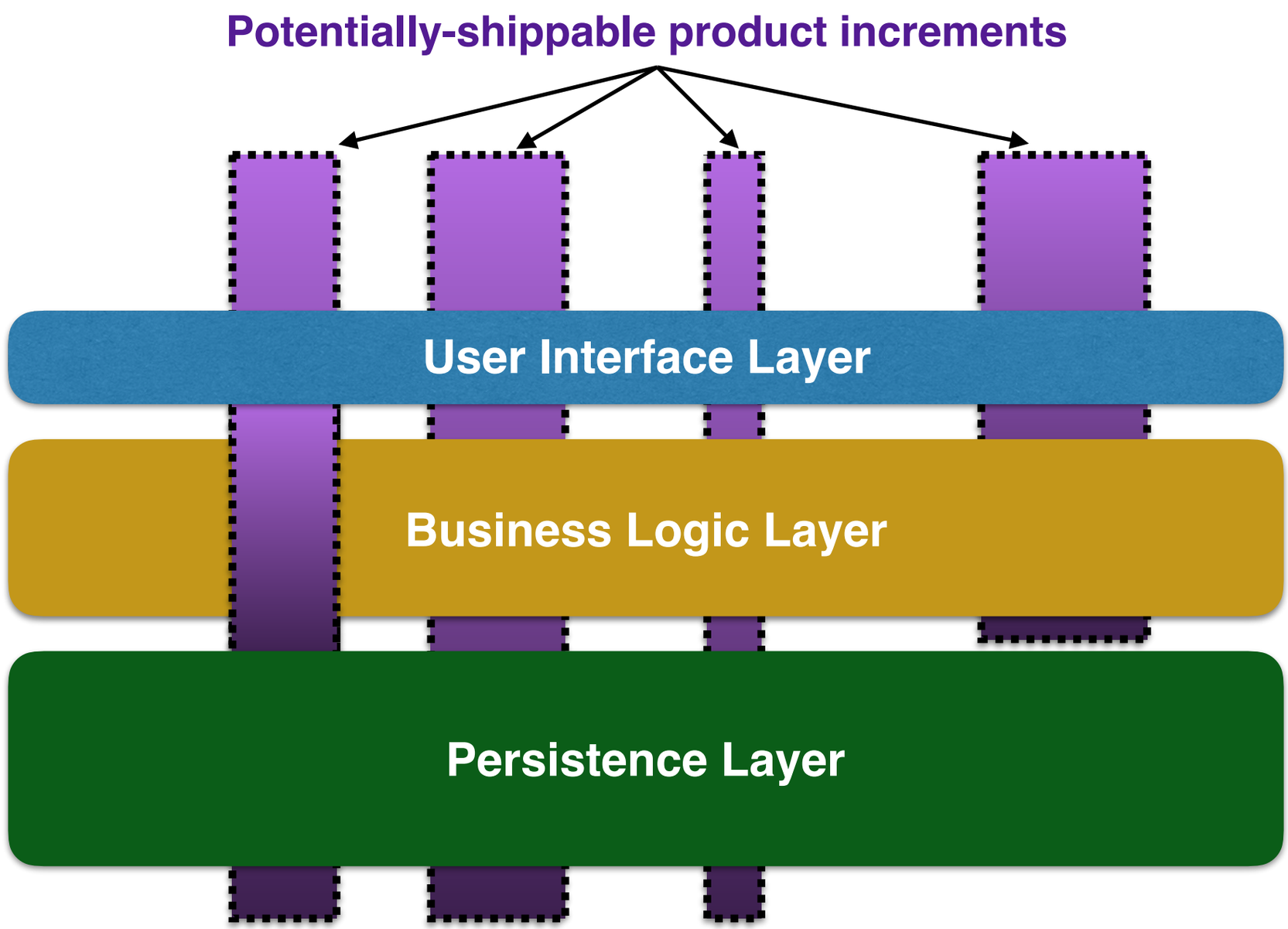
PBI: Product Backlog Item

Specifies the what more than the how of a customer-centric feature

- Often written in User Story form
- Has a product-wide definition of done to prevent technical debt
- May have item-specific acceptance criteria
- Effort is estimated by the team, ideally in relative units
- Effort is roughly 2-3 people 2-3 days, or smaller for advanced teams

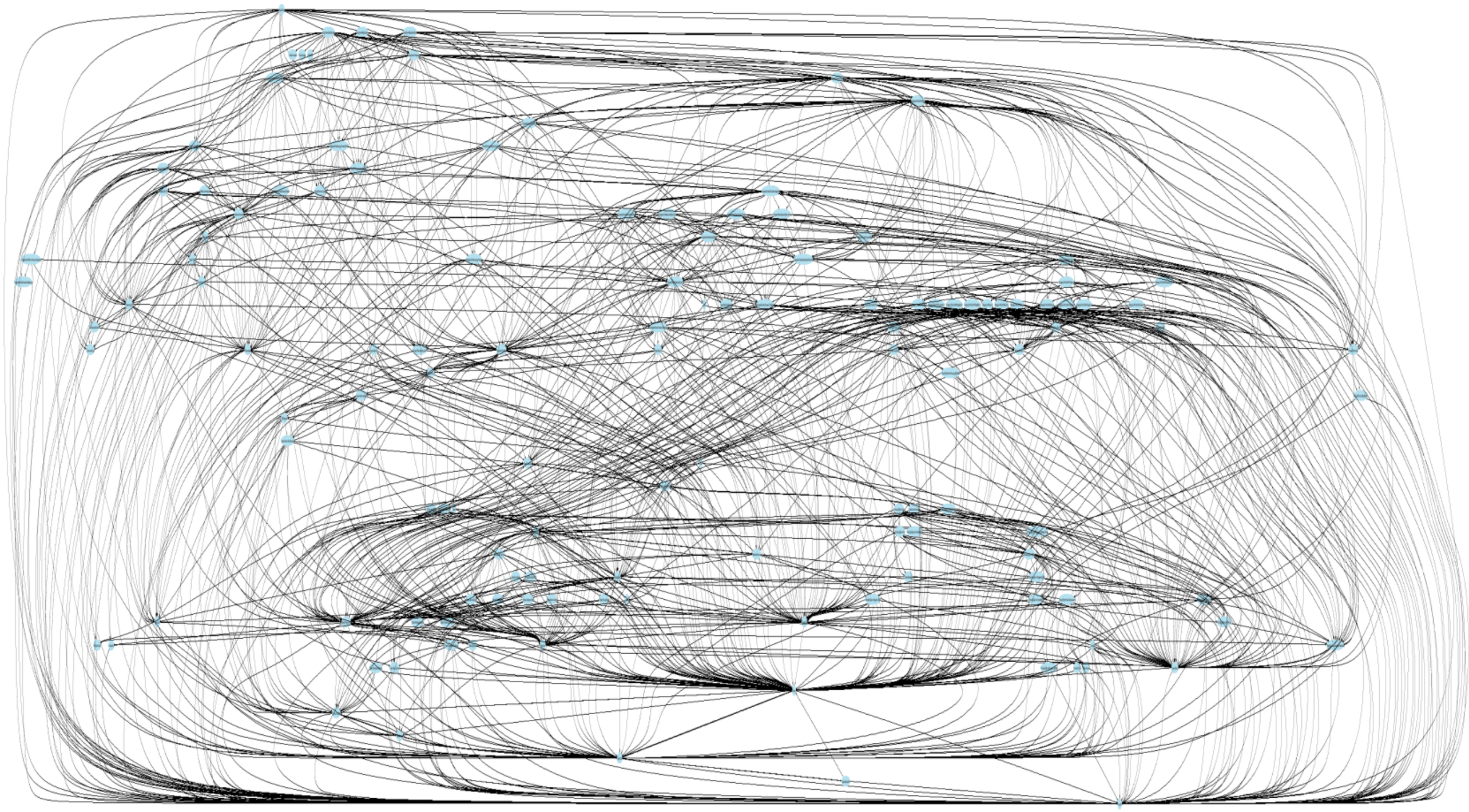
http://scrumreferencecard.com/ScrumReferenceCard.pdf

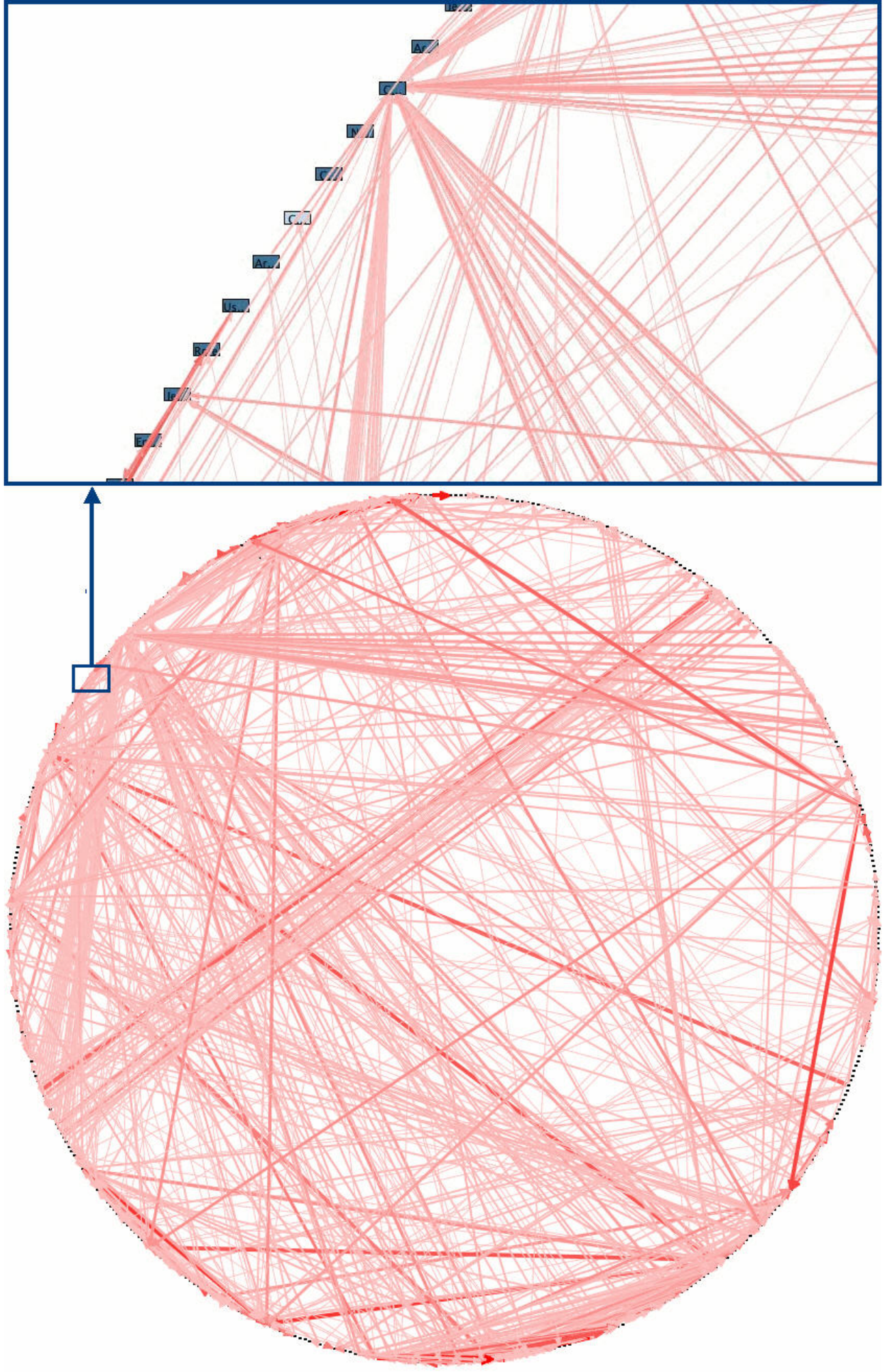
"A skilled ScrumMaster can help the team identify **thin vertical slices** of work that still have business value, while promoting a rigorous definition of *done* that includes proper testing and refactoring."



http://scrumreferencecard.com/ScrumReferenceCard.pdf

The Daily WTF: "The Enterprise Dependency"



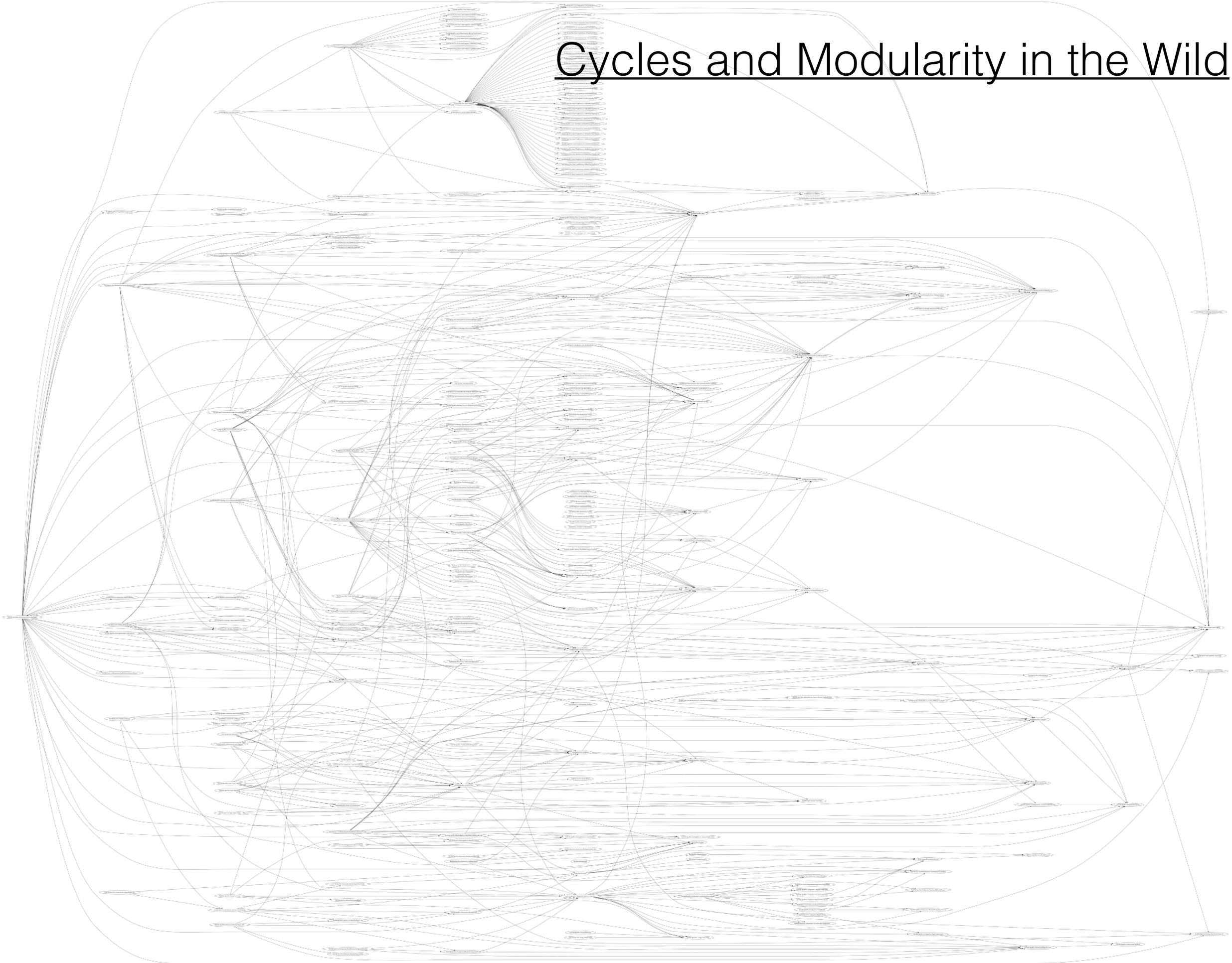


“Enterprise Dependency Big Ball of Yarn”

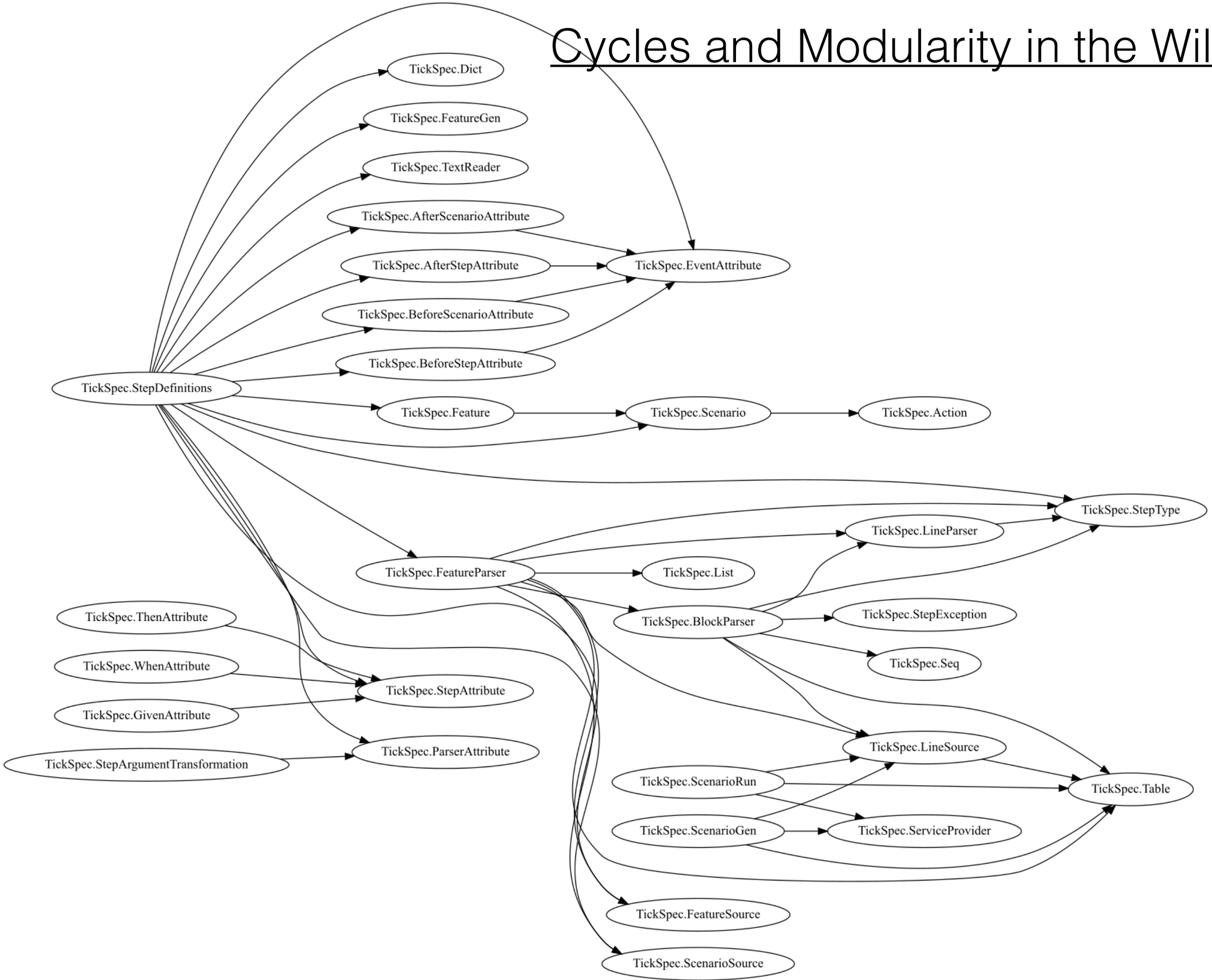
Each box is a class

http://thedailywtf.com/articles/Enterprise-Dependency-Big-Ball-of-Yarn

Cycles and Modularity in the Wild



Cycles and Modularity in the Wild



Some are Easier to Slice than Others

Compare:

Old static webpages

+ CSS

+ Javascript

+ database

+ server-side scripting

Complexity

Desktop application

+ multi-window/panes

+ user configuration

+ installed libraries

+ client-server integration

Design to Manage Complexity

- Don’t just let it happen
- Suitable architecture
- Purposeful design and modeling
- Use proven strategies

Reduce dependencies, decouple components

Divide and simplify

Utilize **design patterns**

Use good OOP: information hiding, inheritance, abstraction

Use Agile methods

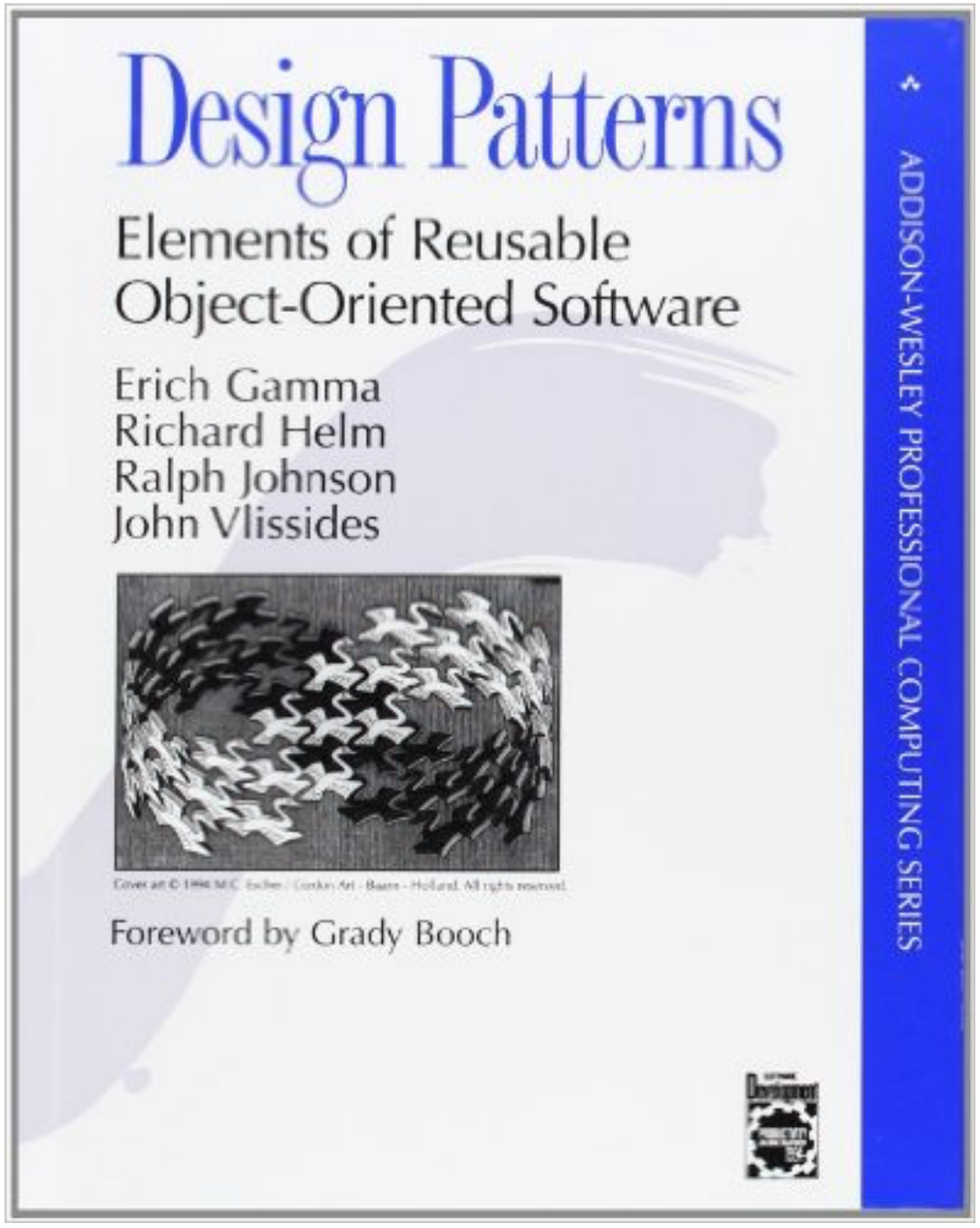
Ubiquitous testing

Use liberally: packages, namespaces

Minimize Side Effects :-)

Design Patterns

- Good solution to commonly encountered problem
- Gives different abstractions common names
- Good for languages like Java, C++, C#

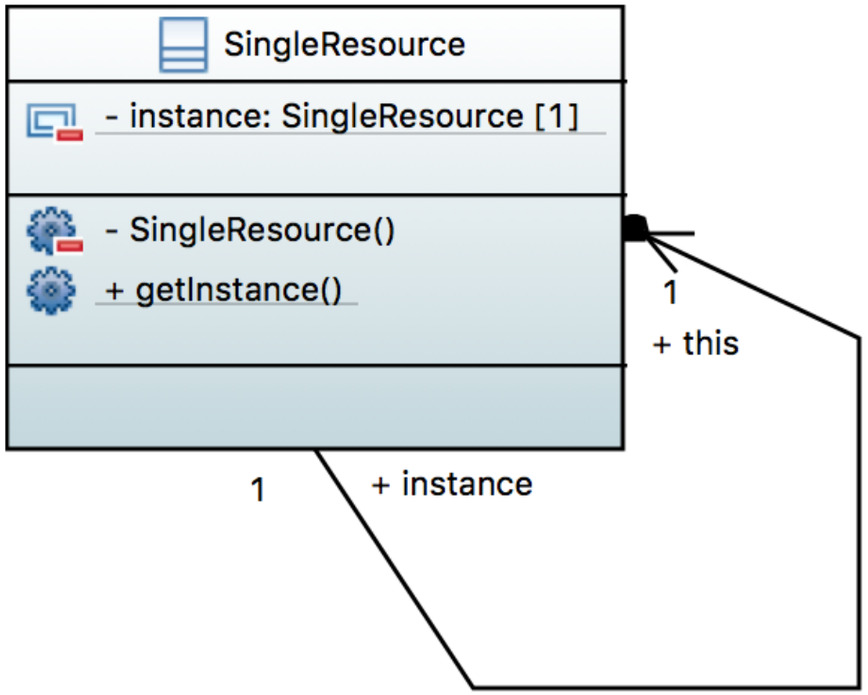


Popular Patterns

- Singleton
- Iterator
- Observer
- Visitor
- Adaptor
- Command
- Delegate
- Factory
- Decorator
- Model-View-Controller (MVC)

more of a paradigm
or architecture

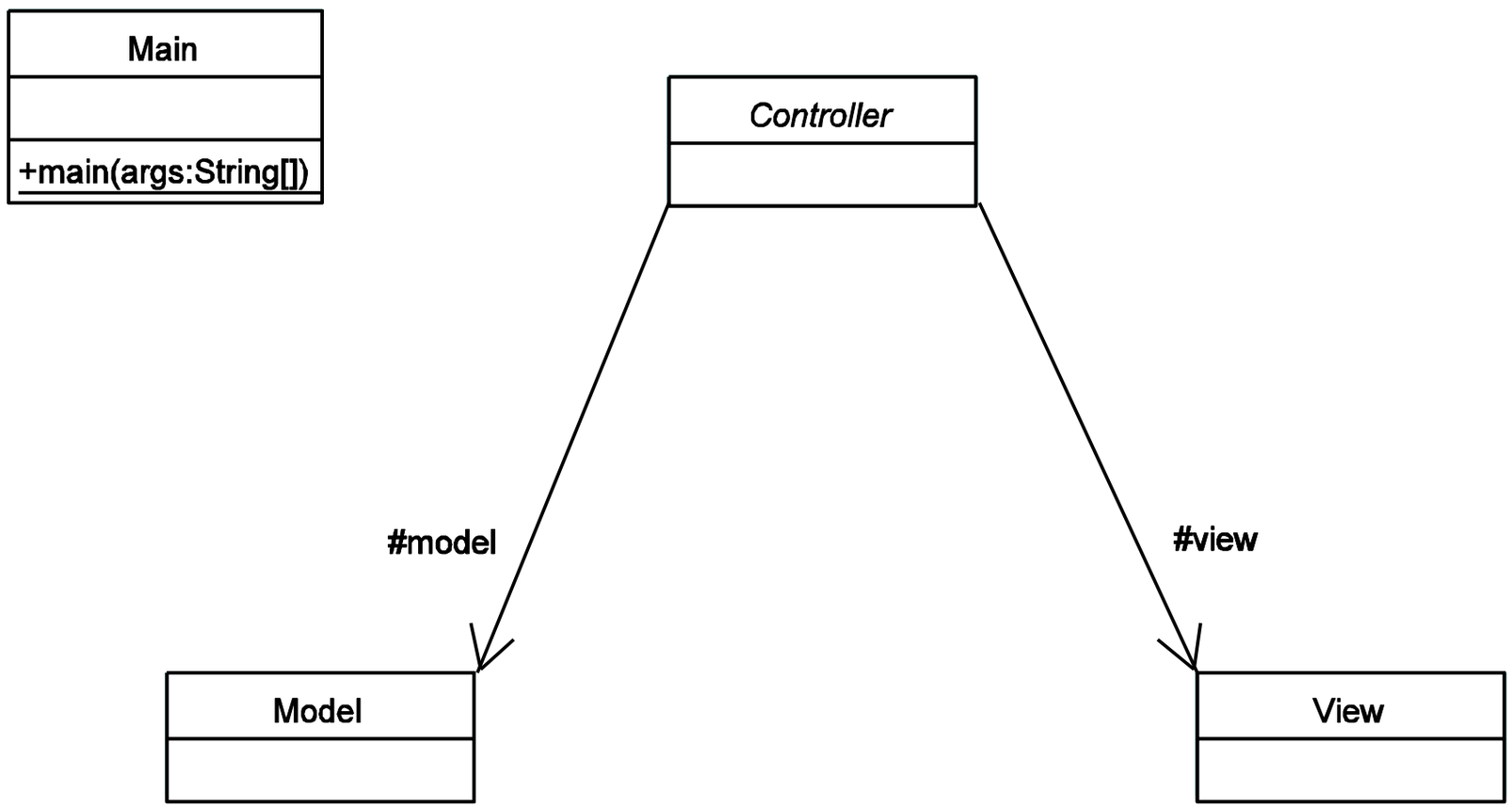
Singleton




```java
// Singleton class
public class SingleResource
{
    private static SingleResource instance = new SingleResource();

    // note: private constructor
    private SingleResource(){}

    public static SingleResource getInstance()
    {
        return instance;
    }
}



public class Main
{
    public static void main(String[] args)
    {
        SingleResource resource = SingleResource.getInstance();
    }
}
```

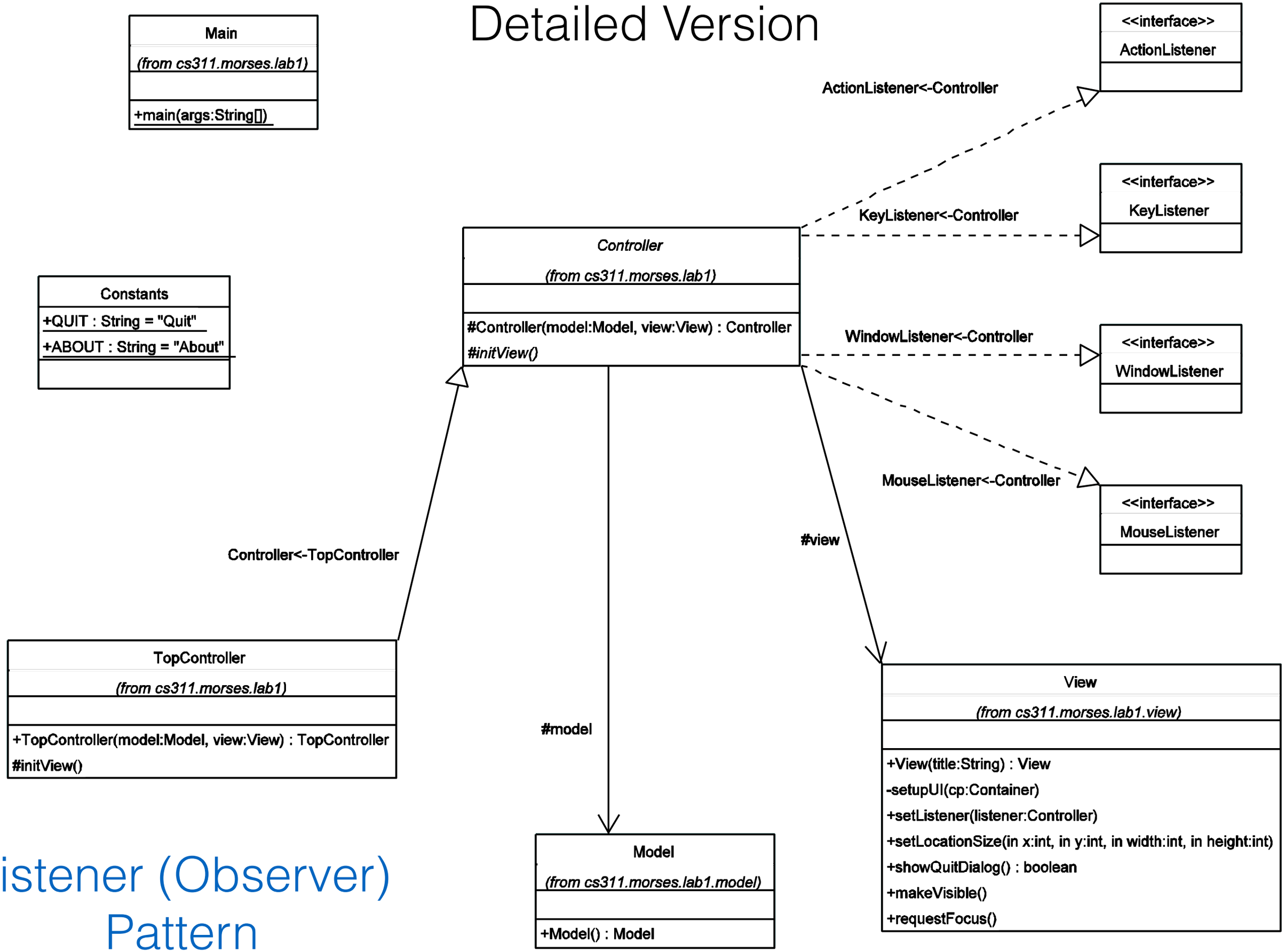
Model View Controller (MVC)

Java Application Example



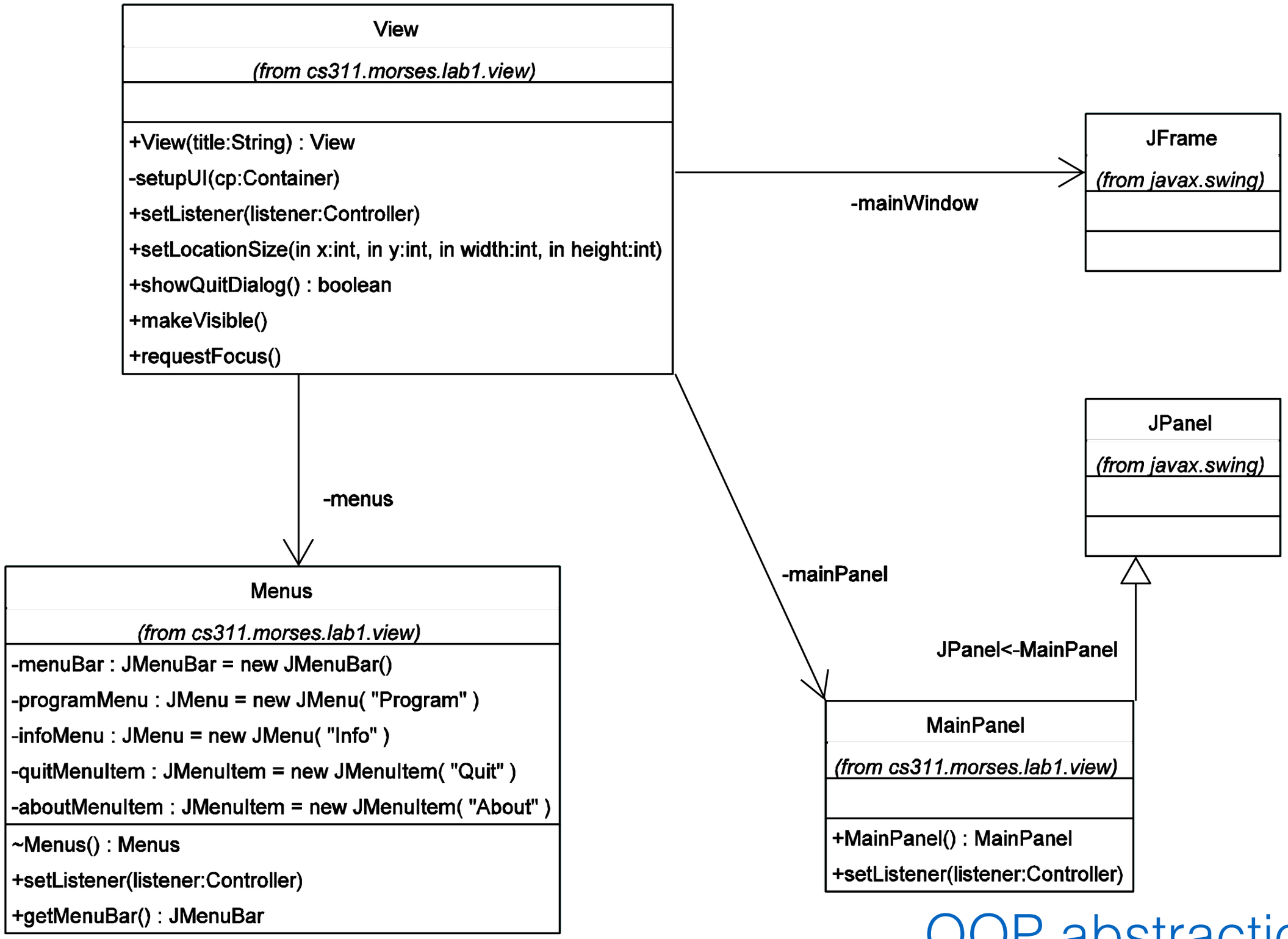
S.F. Morse and C. L. Anderson, "Introducting Application Design and Software Engineering Principles in Introductory CS Courses: Model-View-Controller Java Application Framework," *The Journal of Computing Sciences in Colleges* **20**(2), 190 – 201 (2004).

Detailed Version



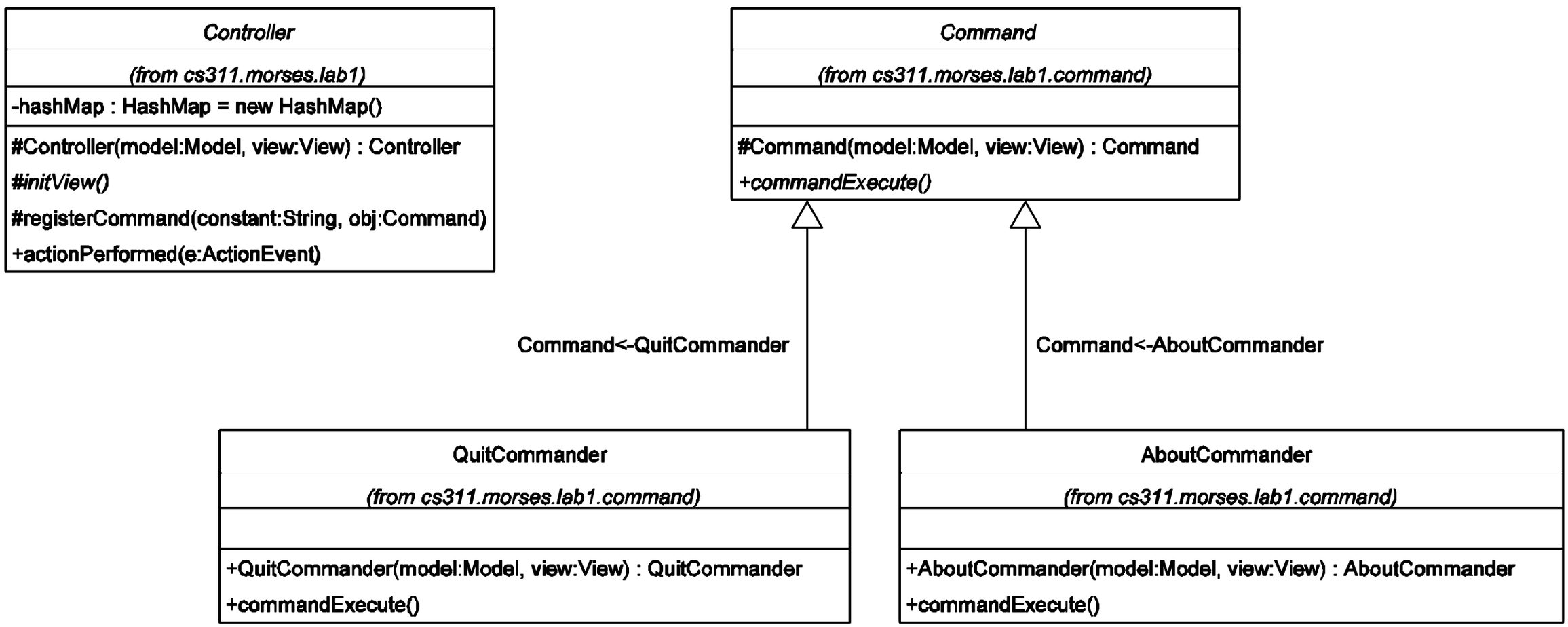
Listener (Observer) Pattern

View subsystem



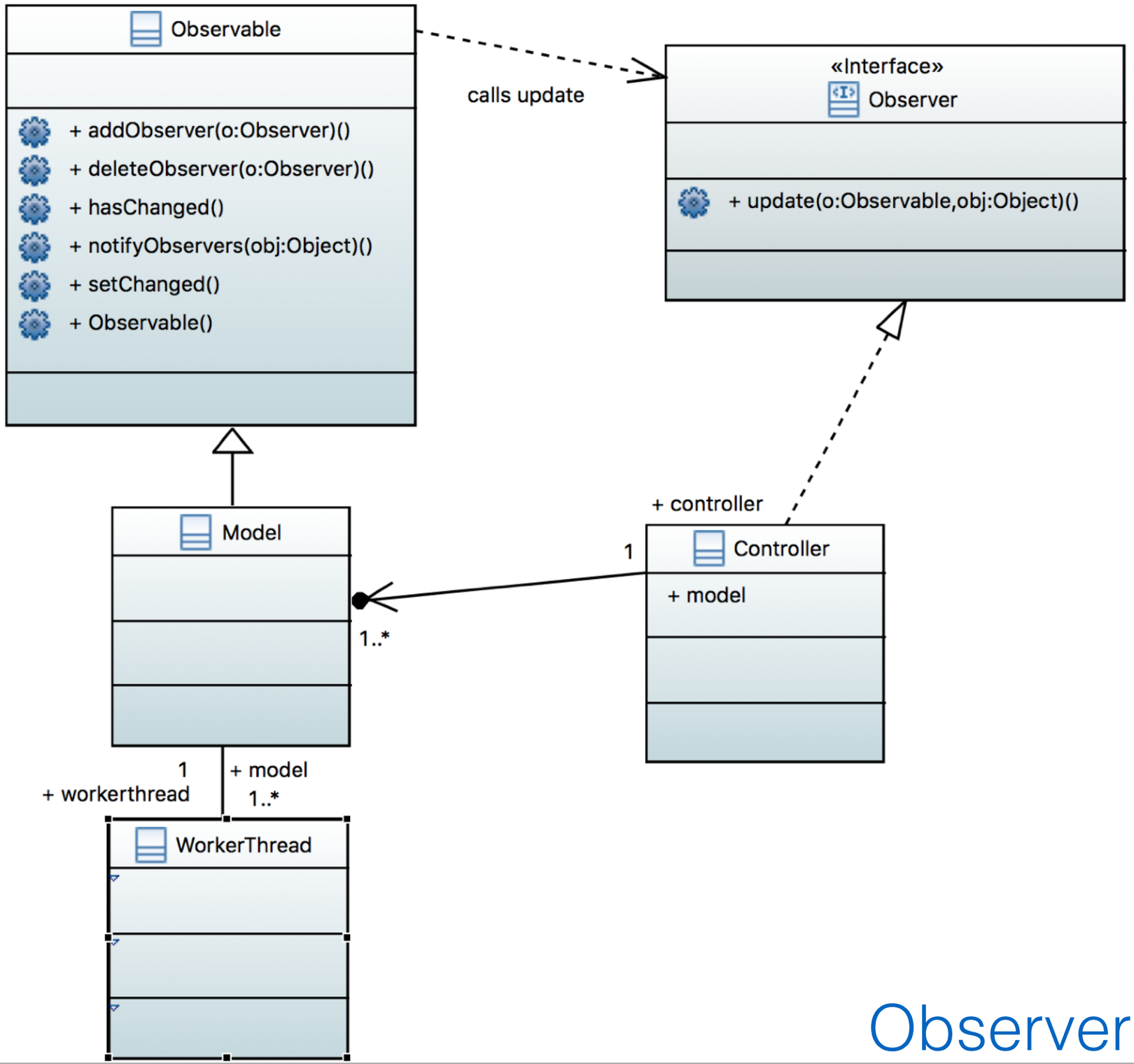
OOP abstractions

Controller — Command subsystem

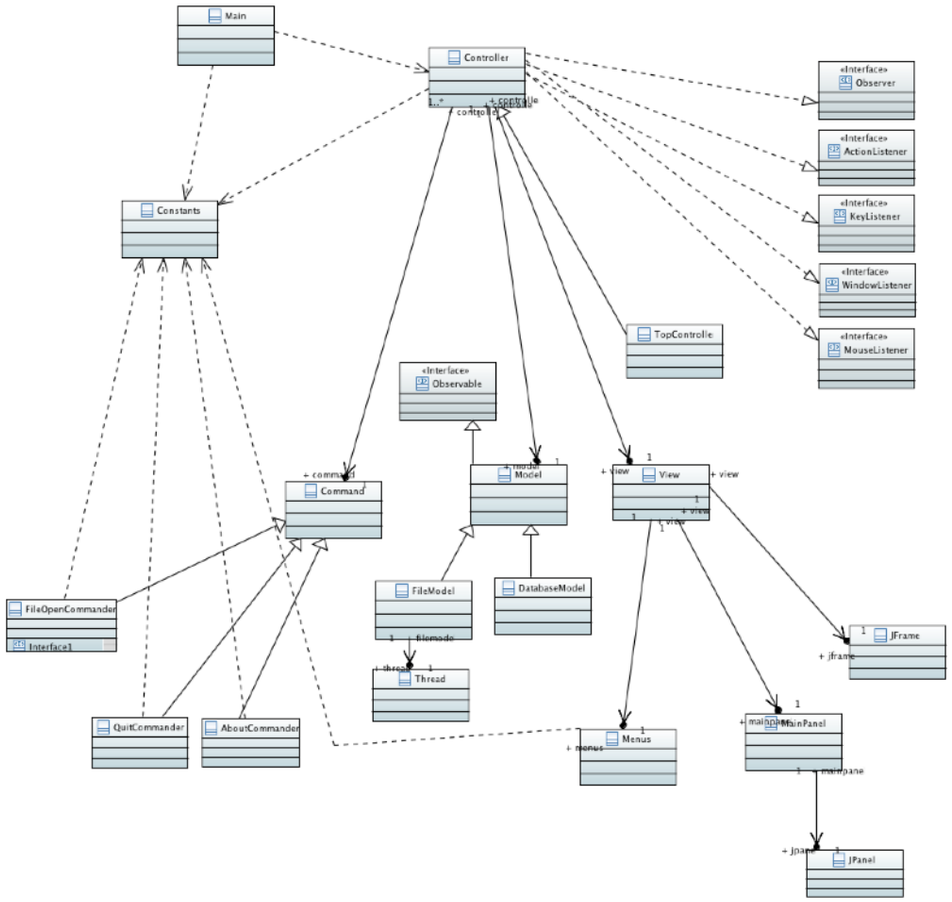


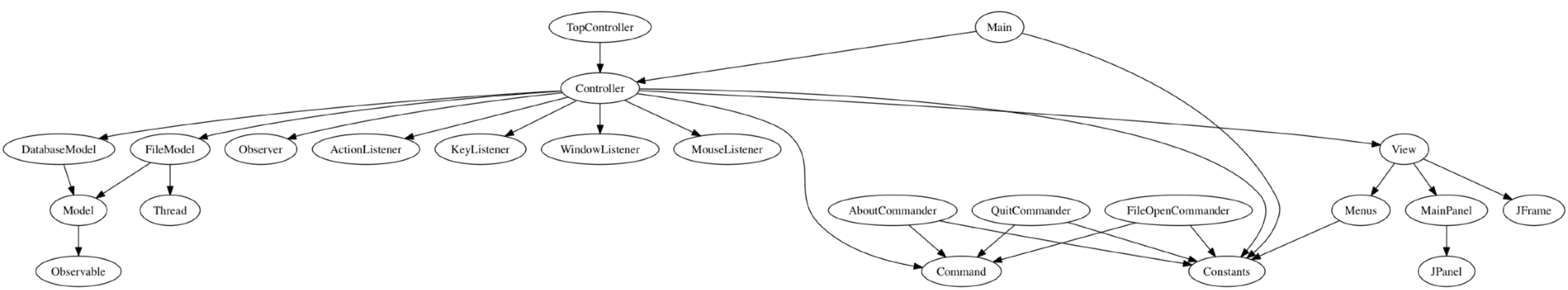
Command Pattern

Model — with Observer



All together





Models, Views, and Controllers

What does MVC look like?