Logical Architecture, Package Design

Curt Clifton

Rose-Hulman Institute of Technology

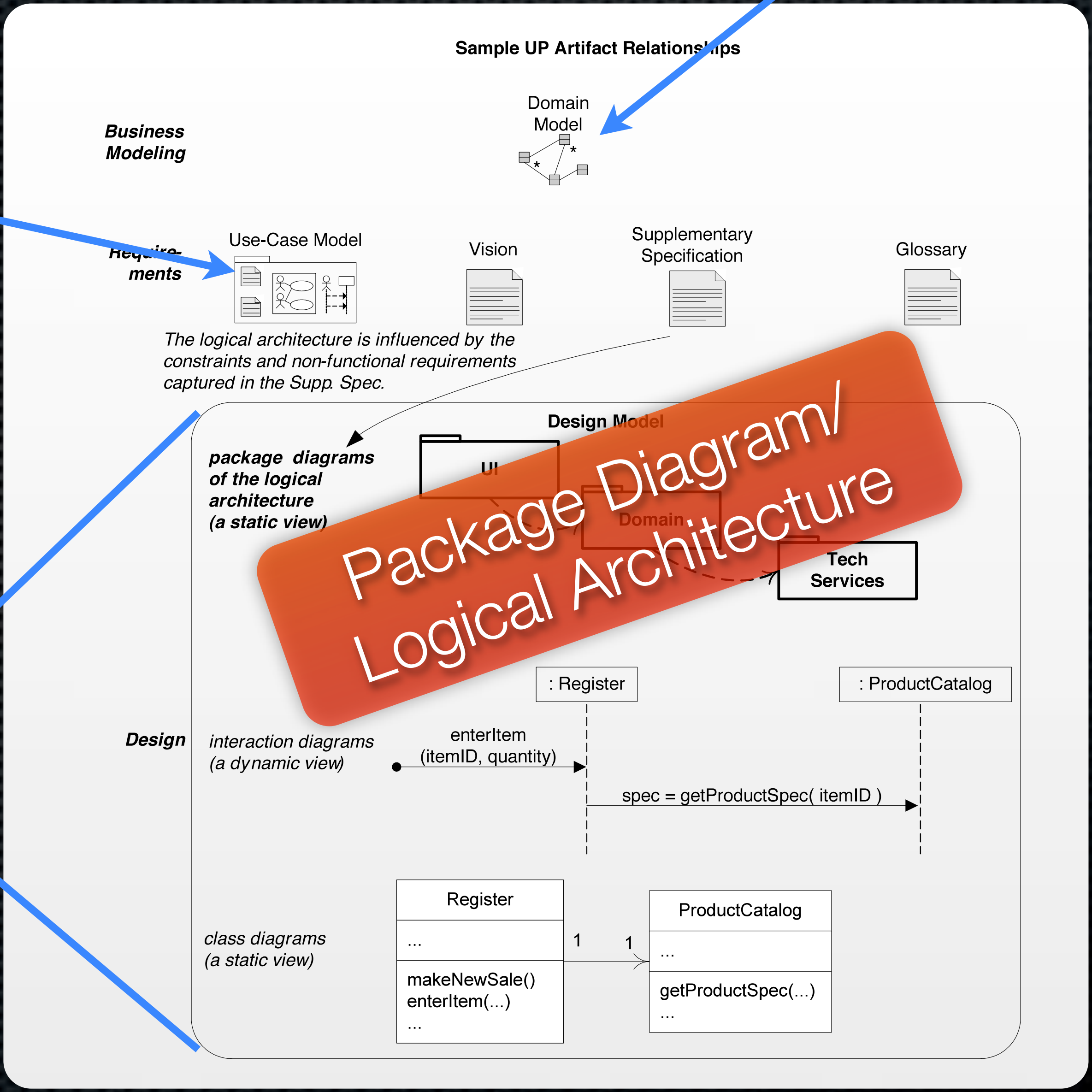
Q1

Where Are We?

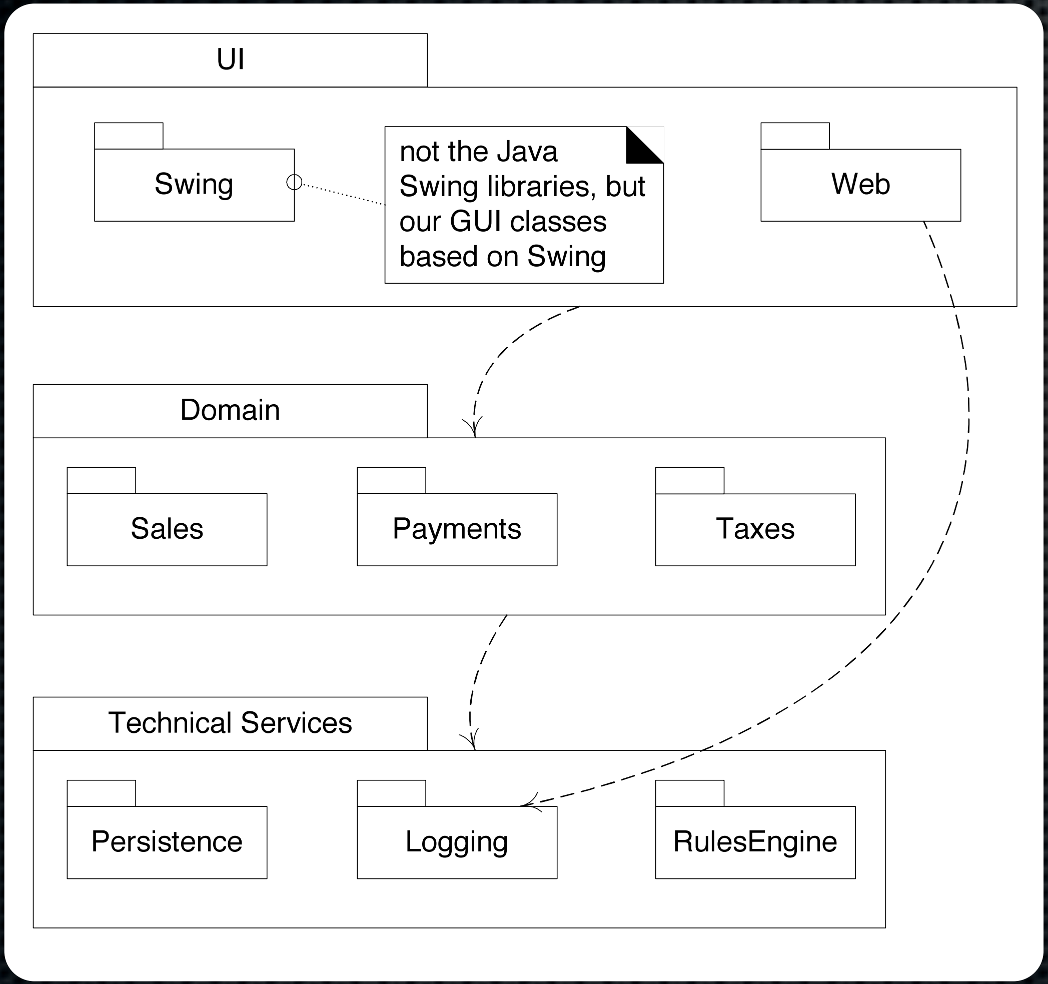
Use Case Model including System Sequence Diagrams and Operation Contracts

Domain Model

Design Model


Sample UP Artifact Relationships
Business Modeling
Domain Model
Require-ments
Use-Case Model
Vision
Supplementary Specification
Glossary
The logical architecture is influenced by the constraints and non-functional requirements captured in the Supp. Spec.
Design Model
package diagrams of the logical architecture (a static view)
UI
Domain
Tech Services
Package Diagram/ Logical Architecture
Design
interaction diagrams (a dynamic view)
: Register
: ProductCatalog
enterItem (itemID, quantity)
spec = getProductSpec(itemID)
class diagrams (a static view)
Register
makeNewSale()
enterItem(...)
ProductCatalog
getProductSpec(...)
1
1

Layered Architectures



- **Coarse-grained grouping** of components based on **shared responsibility** for major aspects of system
- Typically **higher layers call lower ones**, but not vice-versa

Software architecture: the large-scale motivations, constraints, organization, patterns, responsibilities, and connections of a system

Structure and connections

Components, connectors, and topology

Why Worry about Architecture?

- Analyze the effectiveness of a design
- Consider alternatives before significant investment
- Reduce risk
- Provide abstractions for reasoning about design
- Plan for implementation

UML Package Diagrams

- Describes grouping of elements
- Can group **anything**:
  - Classes
  - Other packages
- **More general** than Java packages or C# namespaces

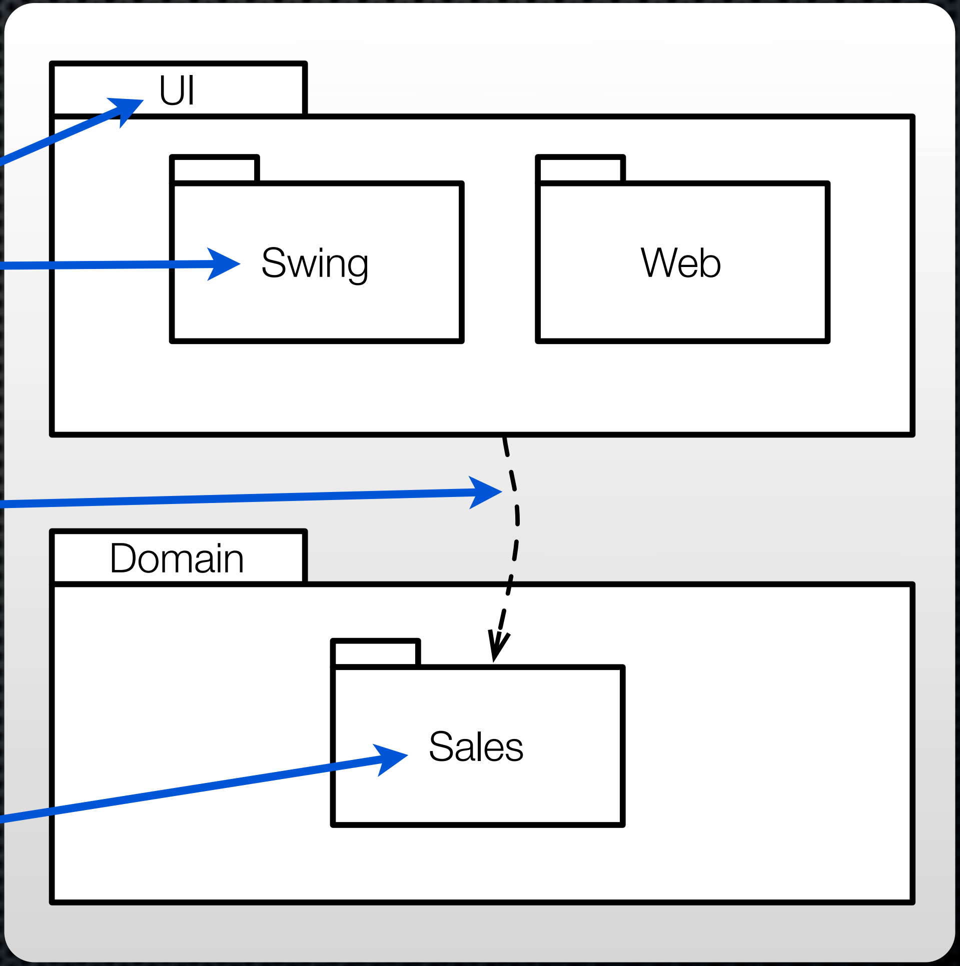
Q2

UML Package Diagrams

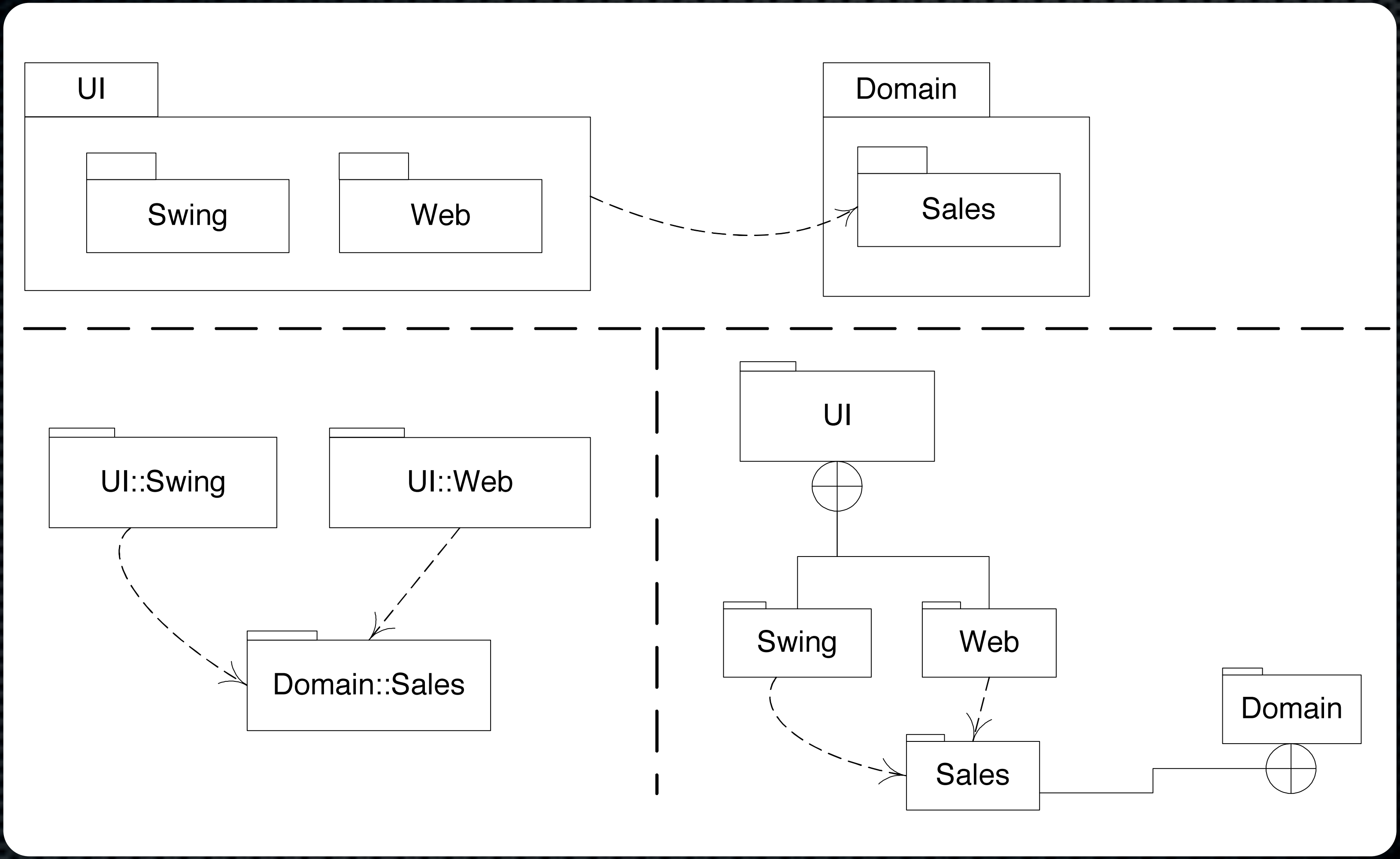
Package Names

Dependency Line

Fully qualified name is: *Domain::Sales*



Alternative Nesting Notations



Designing with Layers Solves Problems

- Rippling source code changes
- Intertwining of application and UI logic
- Intertwining of application logic and technical services
- Difficult division of labor

Q4

Layers of Benefits

- Separation of concerns
  - Reduces coupling and dependencies; improves cohesion; increases reuse potential and clarity
- Essential complexity is encapsulated
- Can replace some layers with new implementations
- Can distribute some layers
- Can divide development within/across teams

Common Layers in More Detail

- UI
- Application
- Domain
- Business Infrastructure
- Technical Services
- Foundation

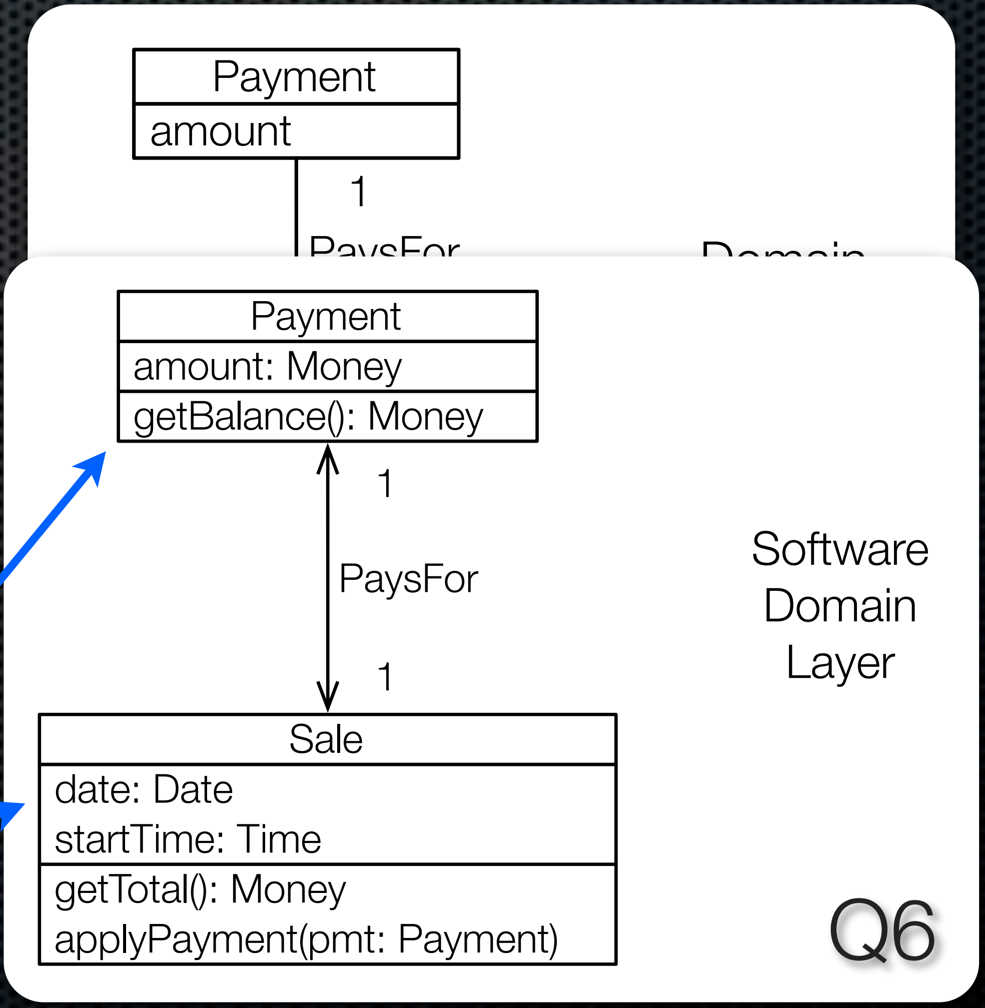
Systems will have many, but not necessarily all, of these

Q5

Designing the Domain Layer

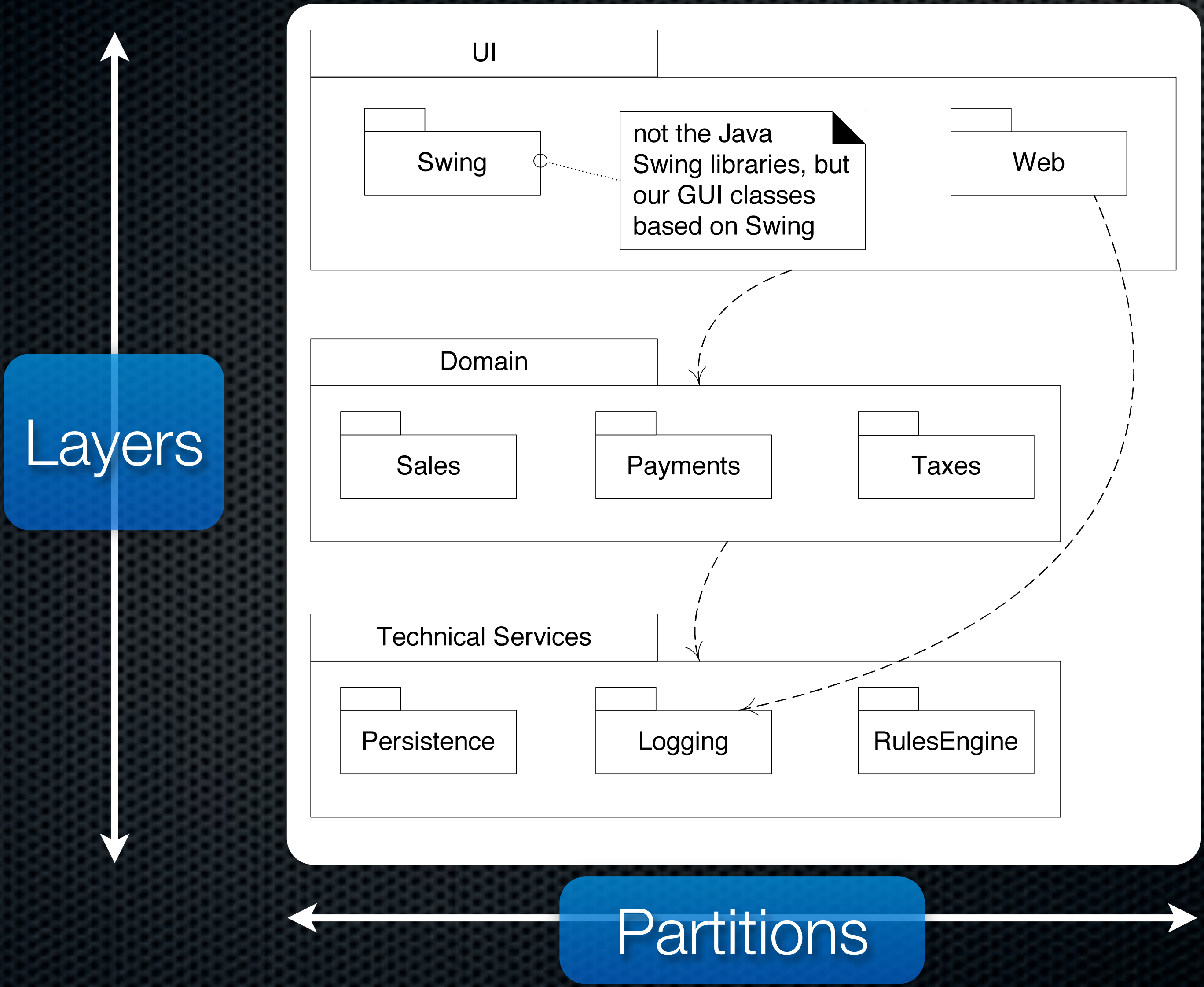
- Create software objects with names and information similar to the real-world domain
- Assign application logic responsibilities

“Domain Objects”



Q6

Terminology: Layers vs. Partitions

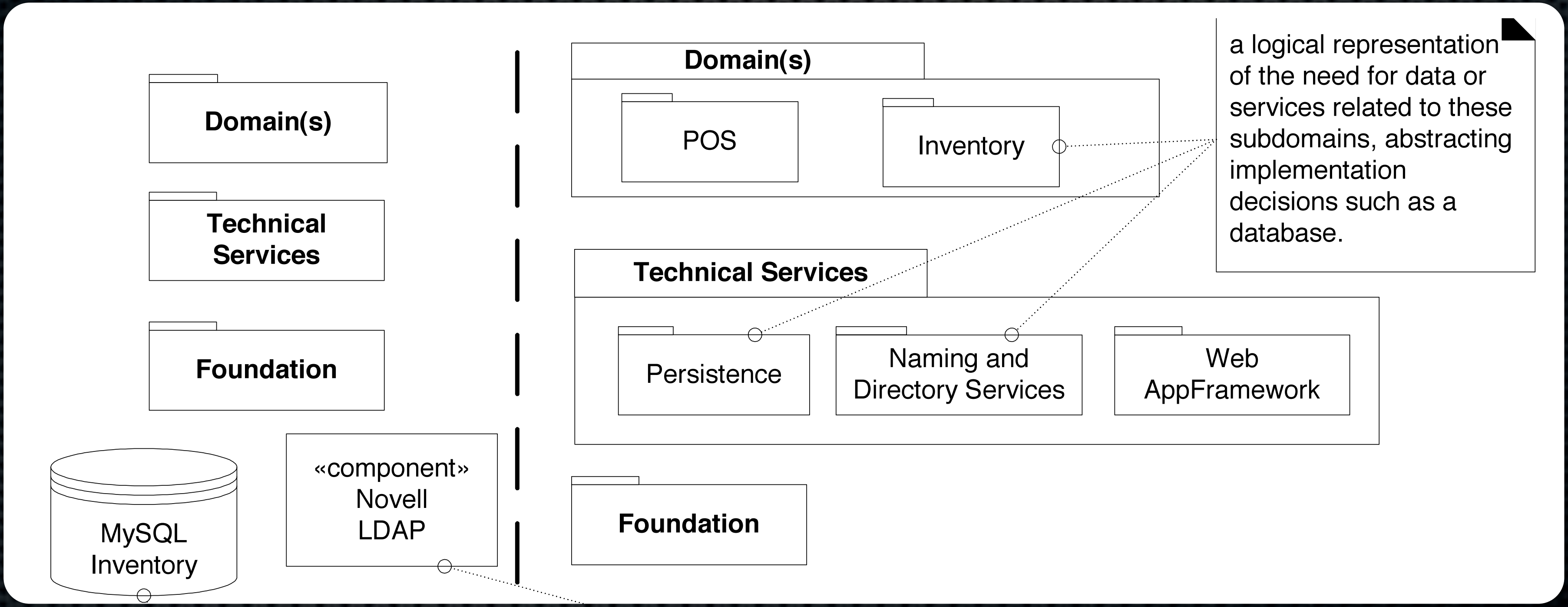


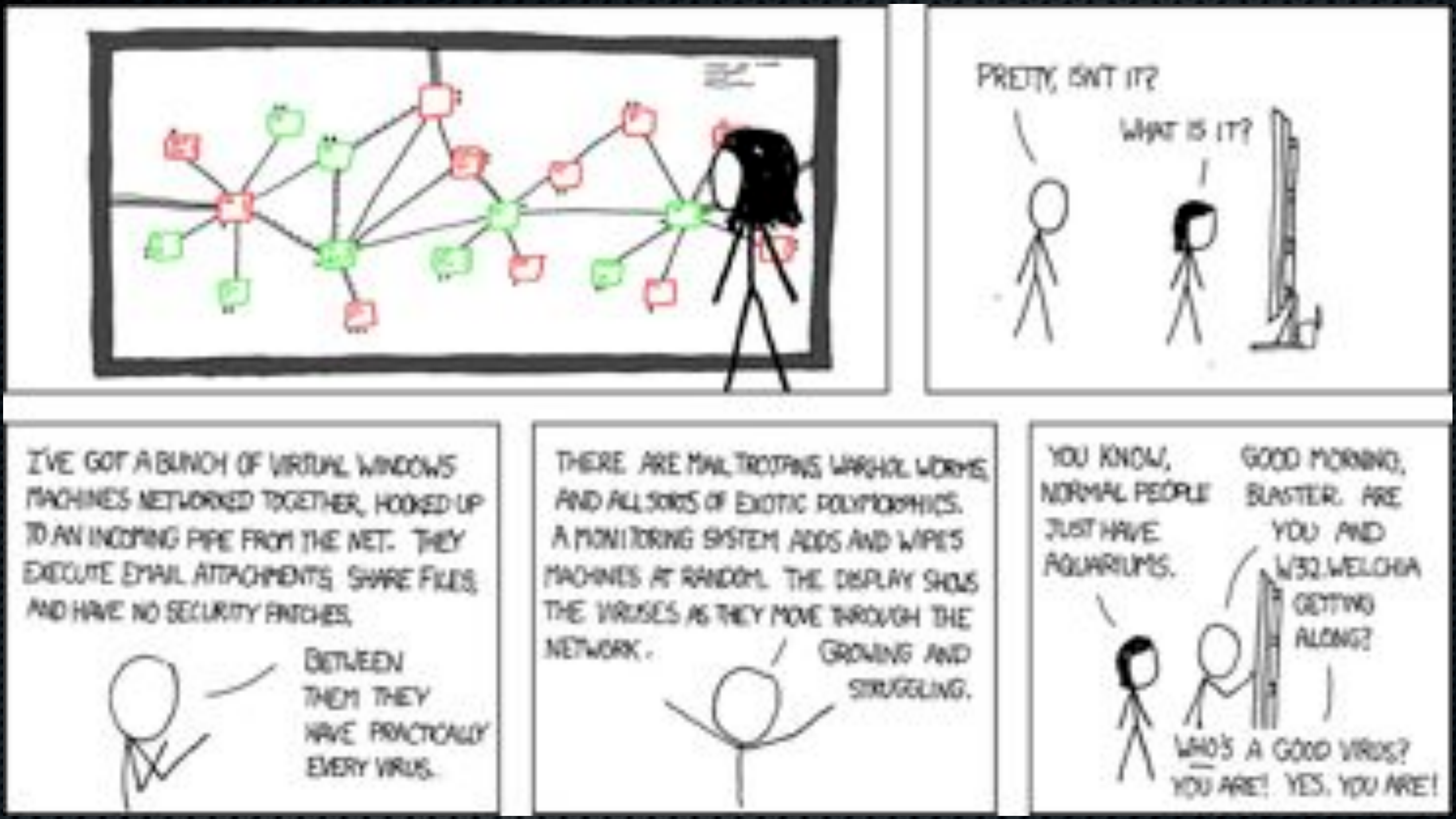
Q7

Common Mistake: Showing External Resources

Worse

Better


Domain(s)
Technical Services
Foundation
MySQL Inventory
«component» Novell LDAP
Domain(s)
POS
Inventory
Technical Services
Persistence
Naming and Directory Services
Web AppFramework
Foundation
a logical representation of the need for data or services related to these subdomains, abstracting implementation decisions such as a database.


PRETTY, ISN'T IT?
WHAT IS IT?
I'VE GOT A BUNCH OF VIRTUAL WINDOWS MACHINES NETWORKED TOGETHER, HOOKED UP TO AN INCOMING PIPE FROM THE NET. THEY EXECUTE EMAIL ATTACHMENTS, SHARE FILES, AND HAVE NO SECURITY PATCHES.
BETWEEN THEM THEY HAVE PRACTICALLY EVERY VIRUS.
THERE ARE MAIL TROJANS, WARHOL WORMS, AND ALL SORTS OF EXOTIC POLYMORPHICS. A MONITORING SYSTEM ADDS AND WIPES MACHINES AT RANDOM. THE DISPLAY SHOWS THE VIRUSES AS THEY MOVE THROUGH THE NETWORK,
GROWING AND STRUGGLING.
YOU KNOW, NORMAL PEOPLE JUST HAVE AQUARIUMS.
GOOD MORNING, BLASTER. ARE YOU AND W32.WELCHIA GETTING ALONG?
WHO'S A GOOD VIRUS? YOU ARE! YES, YOU ARE!

http://xkcd.com/350/

Viruses so far have been really disappointing on the 'disable the internet' front, and time is running out. When Linux/Mac win in a decade or so the game will be over.

Model-View Separation Principle

Easiest way to recognize an OO amateur!

- Do not connect non-UI objects directly to UI objects
  - A Sale object shouldn't have a reference to a JFrame
- Do not put application logic in UI object methods
  - A UI event handler should just delegate to the domain layer
- Model == domain layer, View == UI layer

Q8

Benefits of Model-View Separation

- Provides cohesive model definitions
- Enables separate development
- Localizes changes to interface requirements
- Can add new views
- Allows simultaneous views
- Allows execution of model without UI

From SSDs to Layers

- System operations on the SSDs will become the messages sent from the UI layer to the domain layer

Q9

What’s Next?

Techniques for Object Design

Common Object Design Techniques

- **Just code it**: design while coding, heavy emphasis on refactoring and powerful IDEs
- **Draw, then code**: sketch some UML, then code it
- **Just draw it**: generate code from diagrams

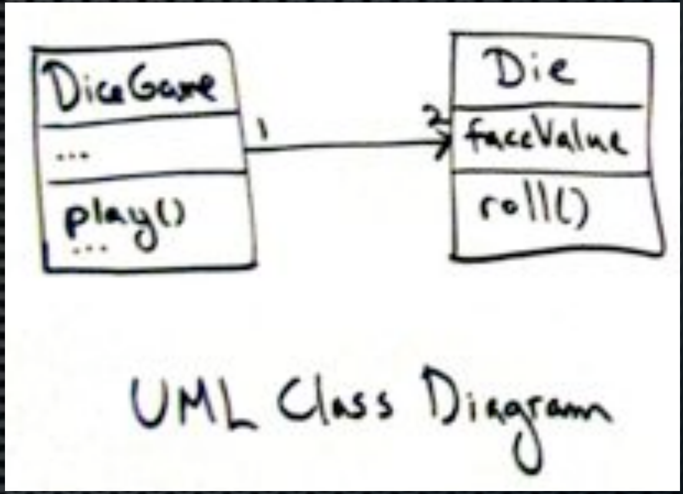
http://www.virginmedia.com/movies/galleries/previews/indiana-jones-idols.php?ssid=7

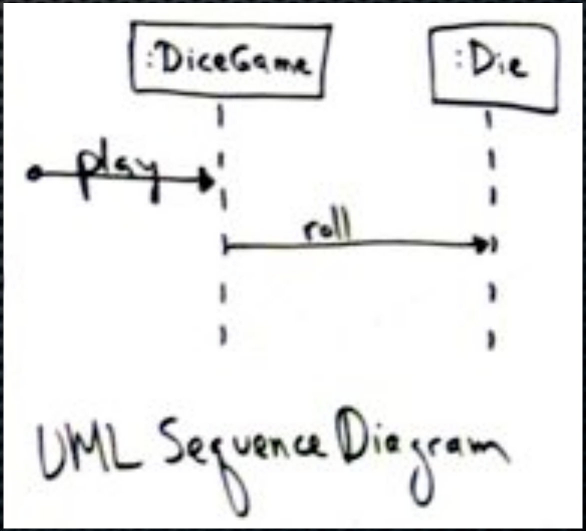
Static vs. Dynamic Modeling

- Static models
  - Class diagrams
- Dynamic models
  - Sequence diagrams
  - Communication diagrams

Interaction diagrams

Spend time on interaction diagrams, not just class diagrams

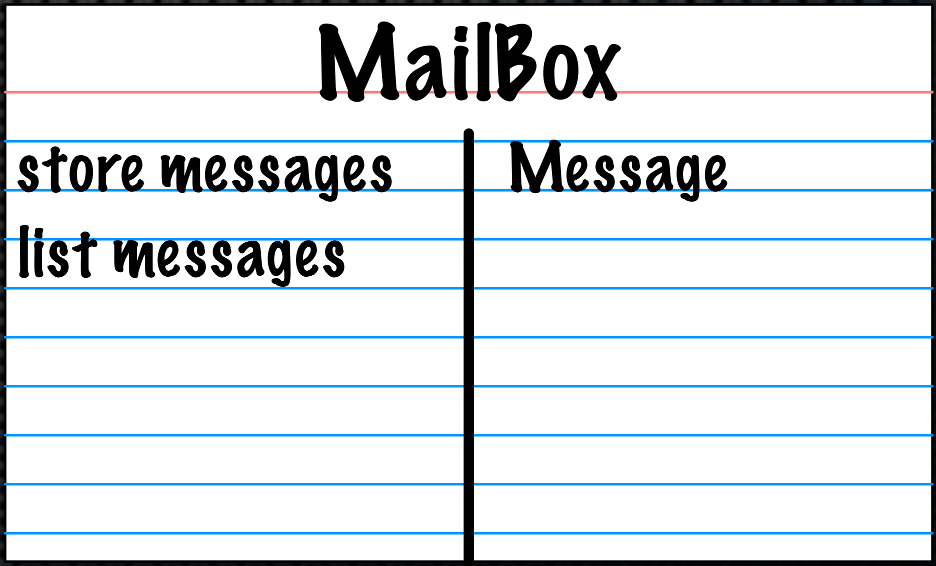




Q10

CRC Cards: A text-based technique

- **C**lass
- **R**esponsibilities
- **C**ollaborators



MailBox

store messages	Message
list messages	

What Matters Most?

- Principles of assigning responsibilities to objects
- Design patterns