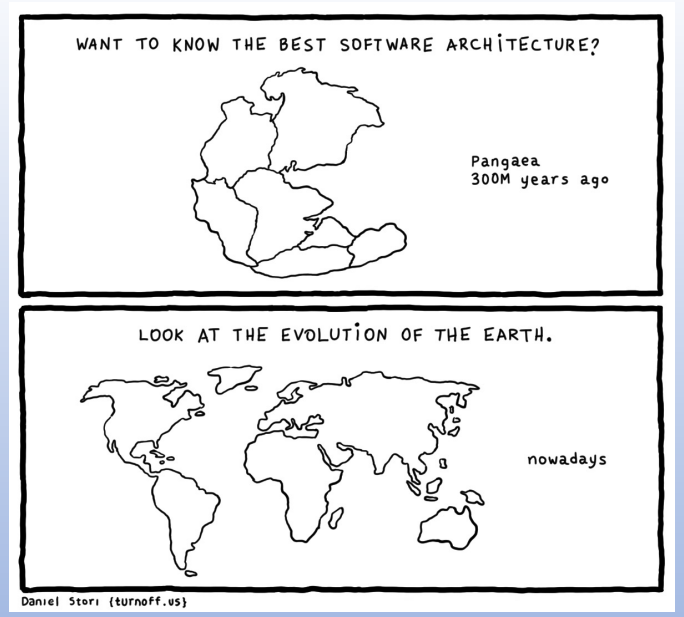
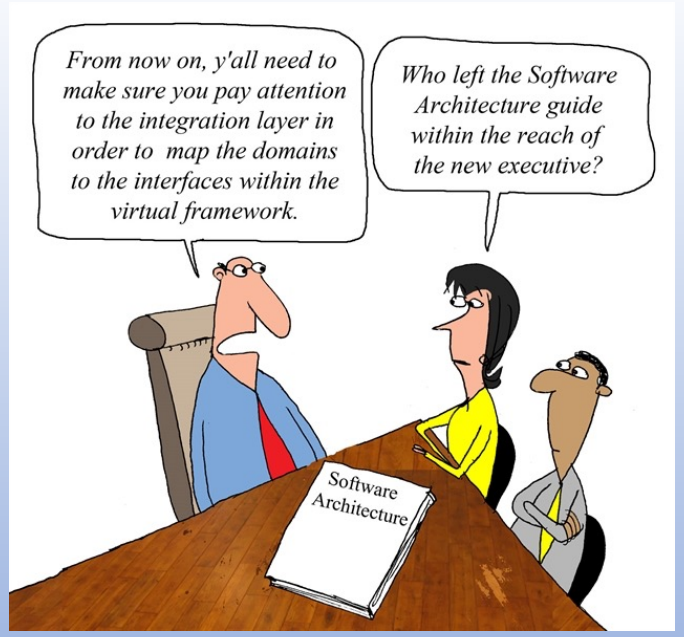
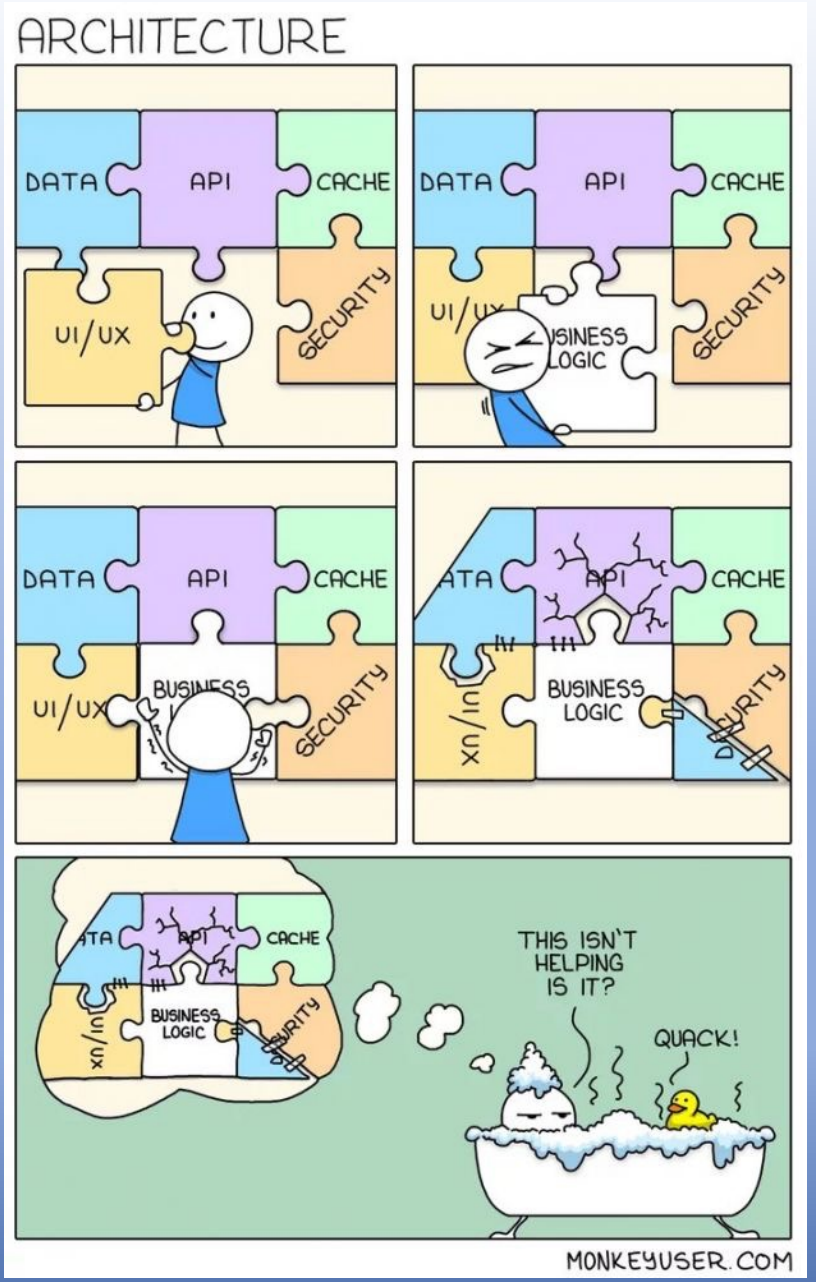






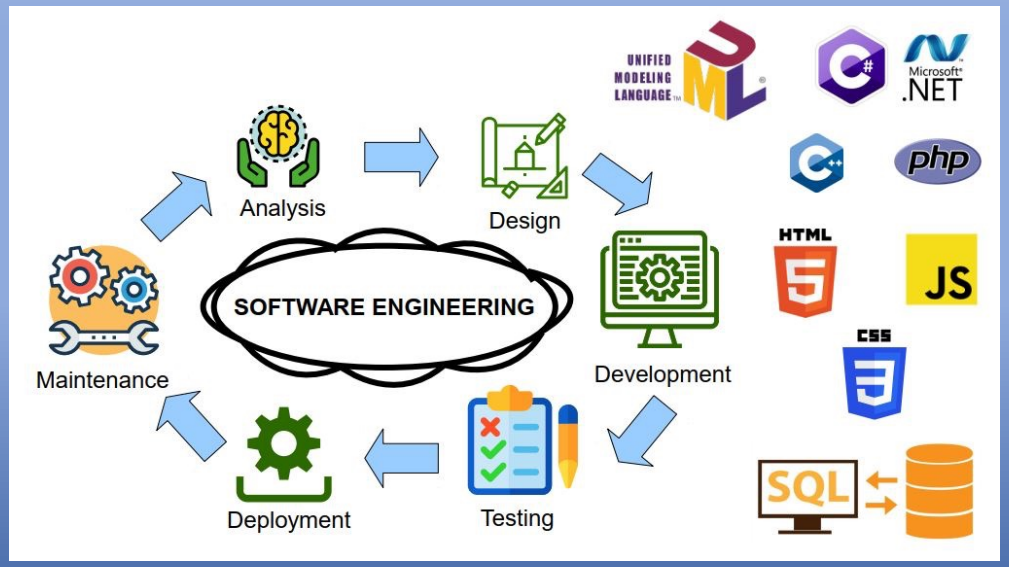
Software Architectures

CSCI2340: (Graduate) Software Engineering

Steven P. Reiss

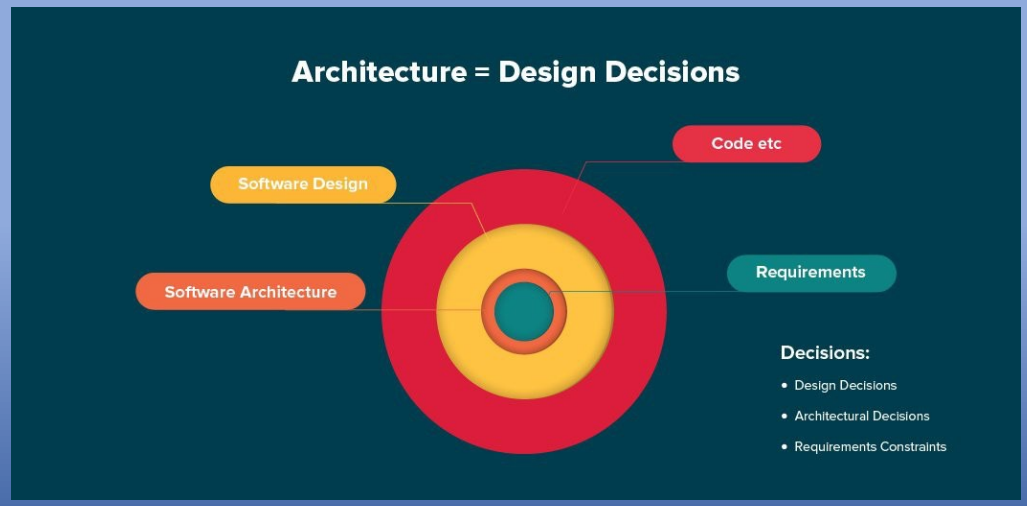
Design

- Requirements tell us what users NEED built
- Specifications tells us WHAT to build
- Next, we need to decide HOW to build it
  - This is DESIGN



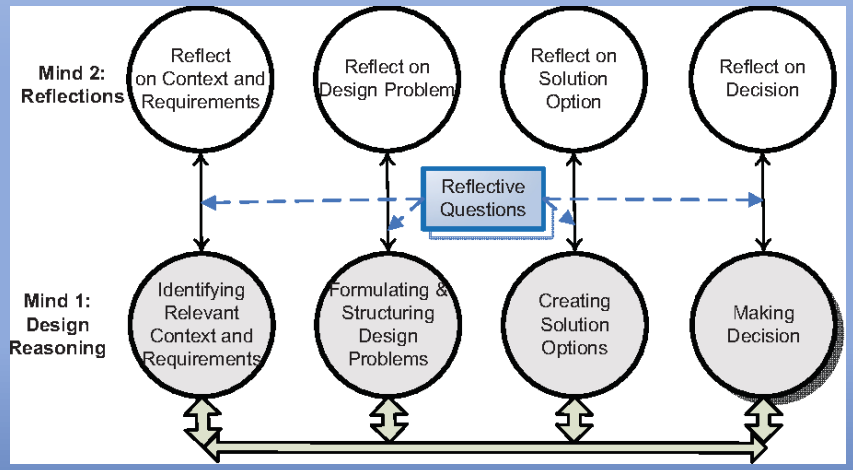
Design Decisions

- Design involves lots of decisions
  - Some are big (overall structure); Some are small (hash or tree map)
- Most do not matter
  - Anything can be made to work; anything can be redone
- But each contributes to making the code better
  - More maintainable
  - Smaller and simpler
  - Less subject to risk
  - Easier to work on as a team
  - Easier to understand



Making Design Decisions

- What is the effect on the system
  - Understand the consequences, implications, …
  - How they fit with the solution, now and in the future
  - Ease of implementation
  - Simplicity of the result
    - Size of the code
    - Complexity of the code
    - Complexity of interaction
  - Ease of understanding
  - Consistency
  - Performance
  - Risk Management

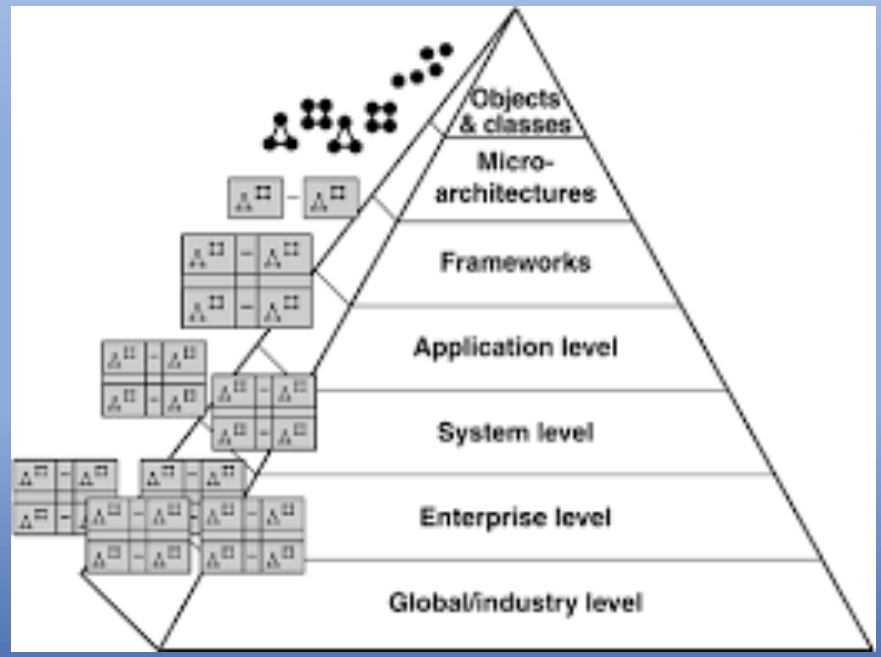


What Makes a Good Designer

- Experience
  - Read lots of designs, understand what others have done
  - As you use a system, think about it might be designed
  - Understand what works and what doesn’t
  - Remember your past decisions (and their effects)
- Understand Design Patterns
  - Design patterns are a collection of standard working ways
  - Need to understand when they should be used
  - Need to understand when they should NOT be used

Decisions are Made at All Levels

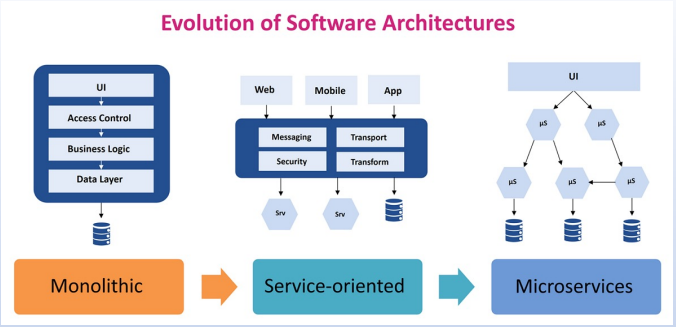
- What construct to use (for or while)
- What data structure to use (HashMap or TreeMap)
- How to split into methods
- How to split into classes
- How to split into packages
- How to create and use APIs
- The overall architecture of the system
  - Software Architecture



Software Architecture

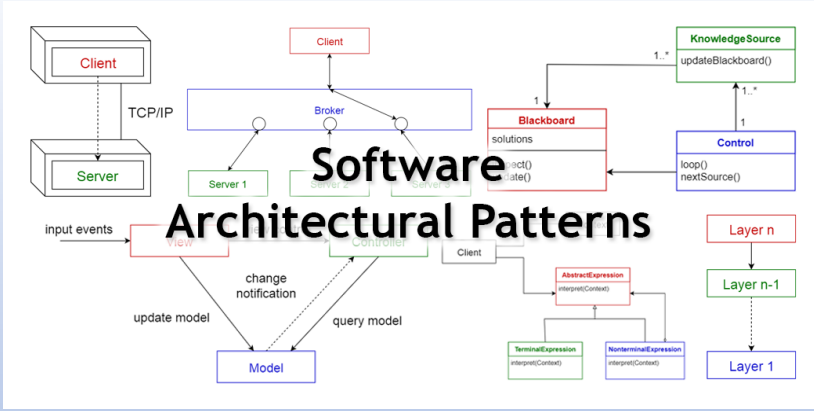
- *Software architecture refers to the fundamental structures of a software system and the discipline of creating such structures and systems. Each structure comprises software elements, relations among them, and properties of both elements and relations.*
- *Architecture is the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution. -IEEE 1471*

Software Architecture



- *An architecture is the set of significant decisions about the organization of a software system, the selection of structural elements and their interfaces by which the system is composed, together with their behavior as specified in the collaborations among those elements, the composition of these elements into progressively larger subsystems, and the architectural style that guides this organization -- these elements and their interfaces, their collaborations, and their composition. –Kruchten (2003) The Rational Unified Process: An Introduction*

Software Architecture



- The overall view of how the system works
  - When the system is complex enough to warrant
- Typically expressed as boxes and arrows
  - Boxes represent system components
  - Arrows represent communication between components
    - Can be interpreted in various ways
    - Can involve processing as well as pure communication
  - Both are important and central to the architecture

Consider a Web Browser

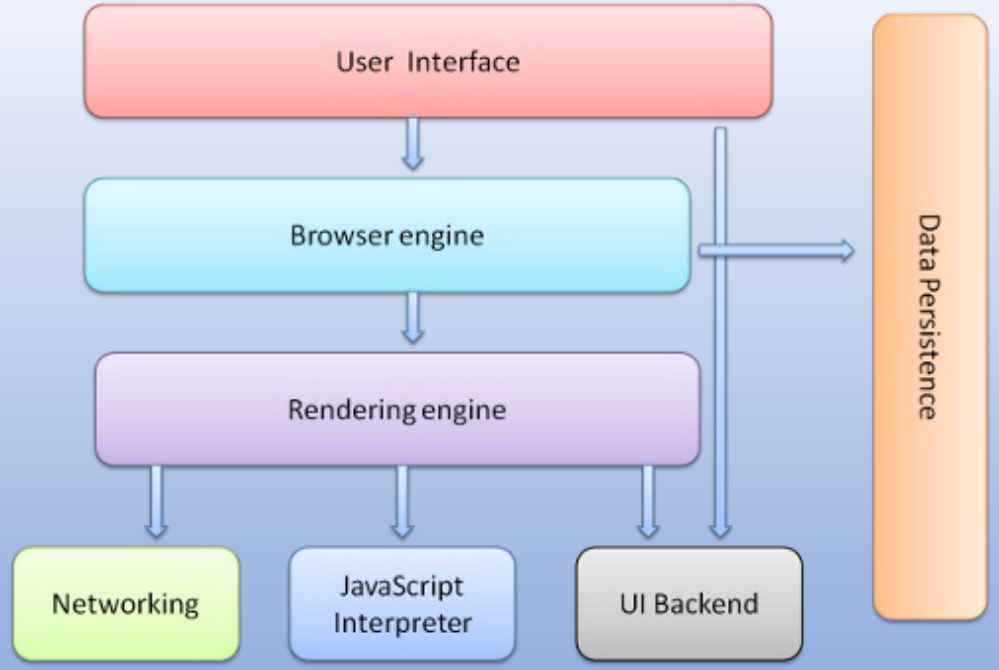
- Specifications
  - Display a web page
    - Download the contents (w/ CSS, JS, images, …)
    - Layout the contents
    - Display the contents
  - Handle interactions (built-in and JS)
  - Manage downloads and bandwidth
  - Manage tabs & multiple browser windows
  - Manage cookies and local storage

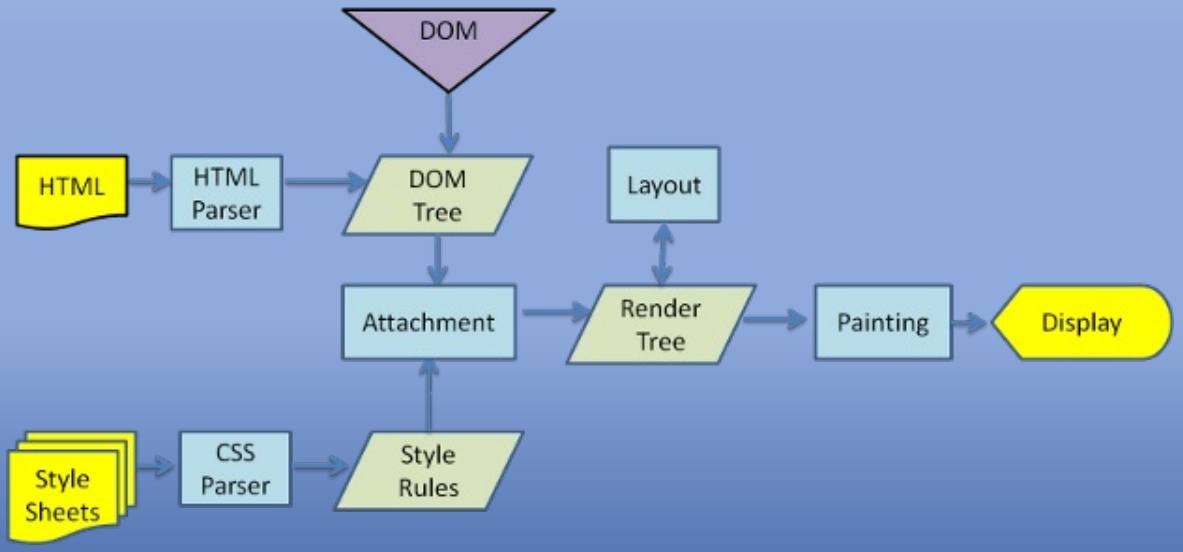
EXERCISE

- Consider your favorite browser
  - How does it work internally?
  - What are the basic components?
  - How are they connected?
  - What are the key technical problems?
  - How to take advantage of multiple cores?
- How might the components be organized?
- How would you build a new web browser?
- Discuss this with your neighbors

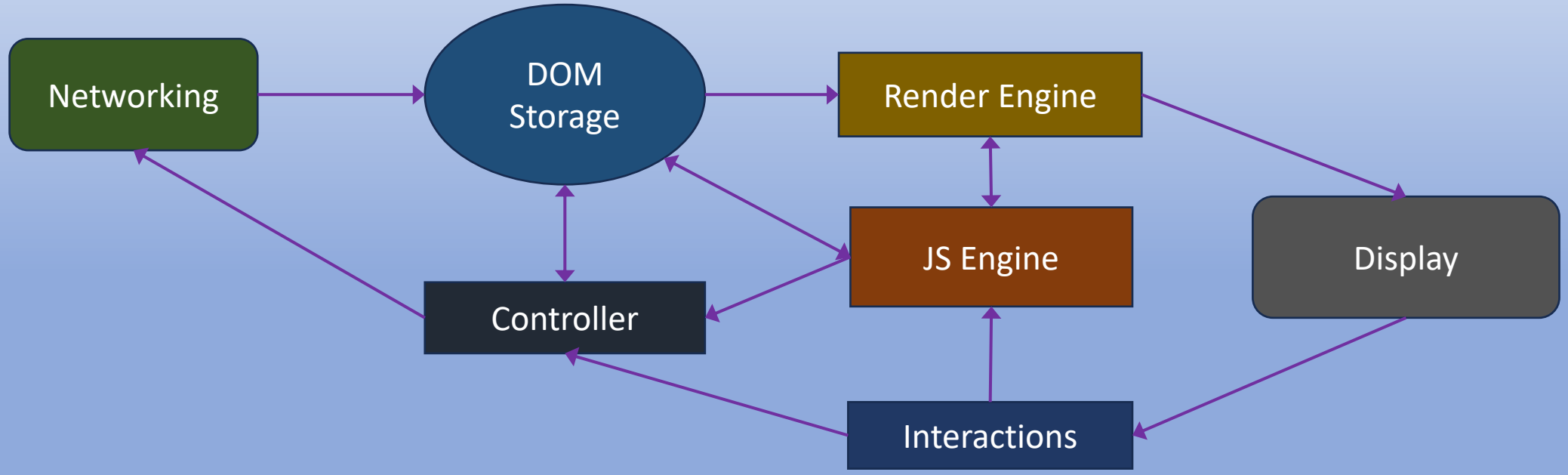
EXERCISE Discussion

- What are the components
- What are the key parts
- What are the difficult parts





Browser Architecture

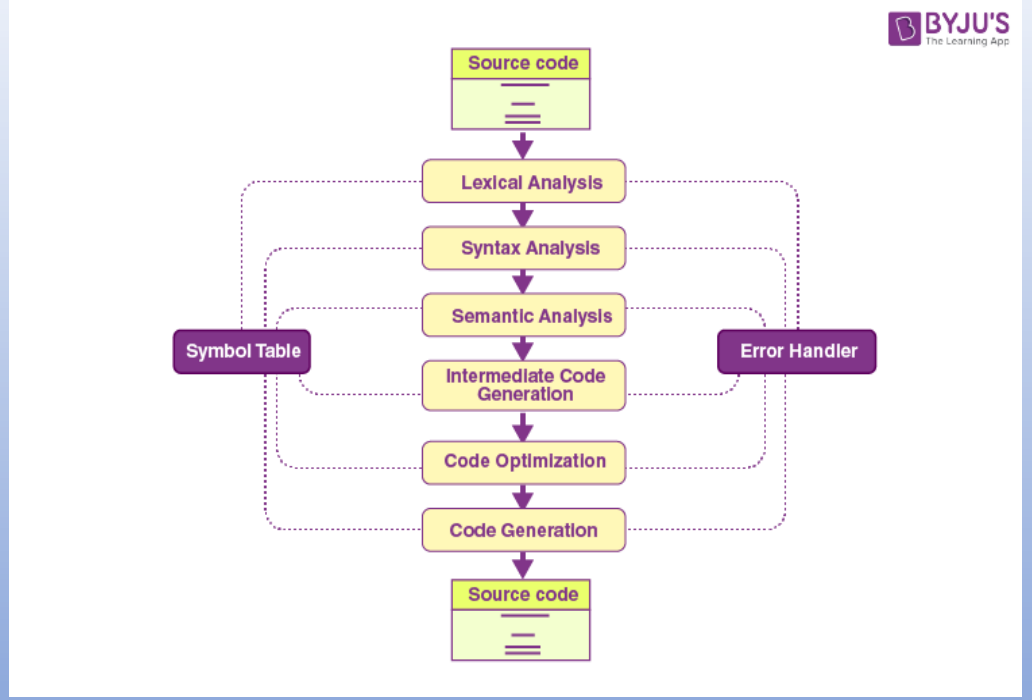


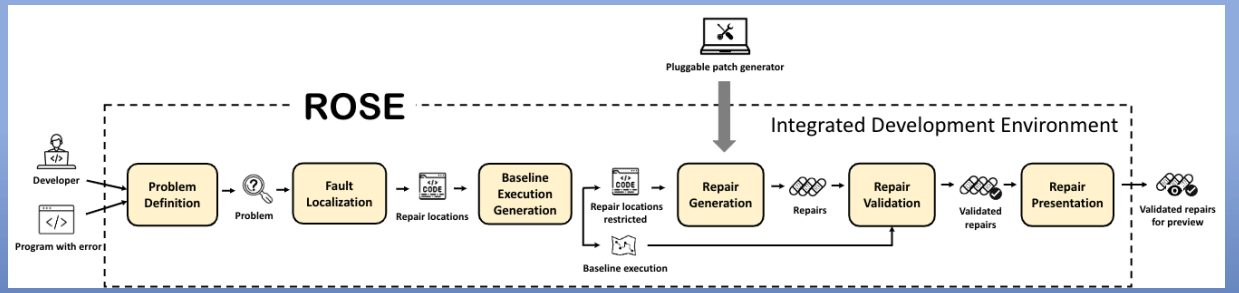
Architectural Patterns

- There is a relatively small set of basic architectures
  - Architectural design patterns
  - Like house architectures (ranch, split-level, cape, colonial, …)
  - Most actual systems are a combination of these
- A good system designer will understand the options
  - What they are
  - How they might be implemented
  - When they are useful
  - What are their strengths and weaknesses

Pipe and Filter

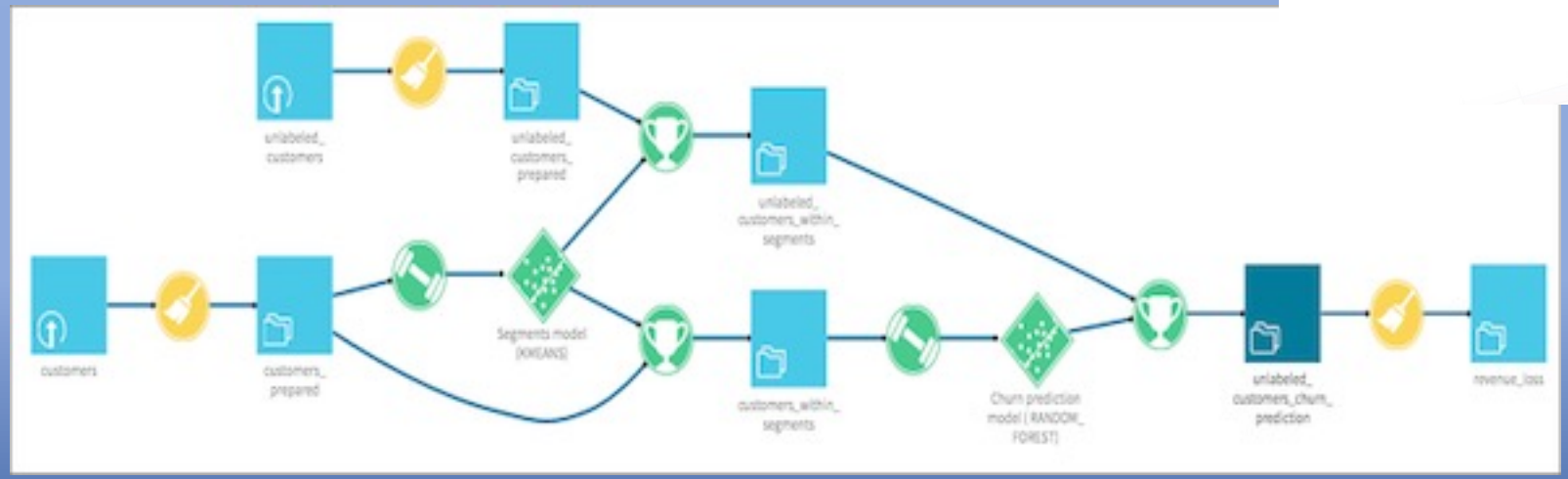
- UNIX shell pipes as a model
  - Filters are processing units
  - Pipes are connections between these
- Classic example
  - Compilers
- Other examples
  - Visualization Engine (BLOOM)
    - Trace collection and processing
    - Filtering and merge data sources
    - Visualization
  - ROSE (automatic program repair)





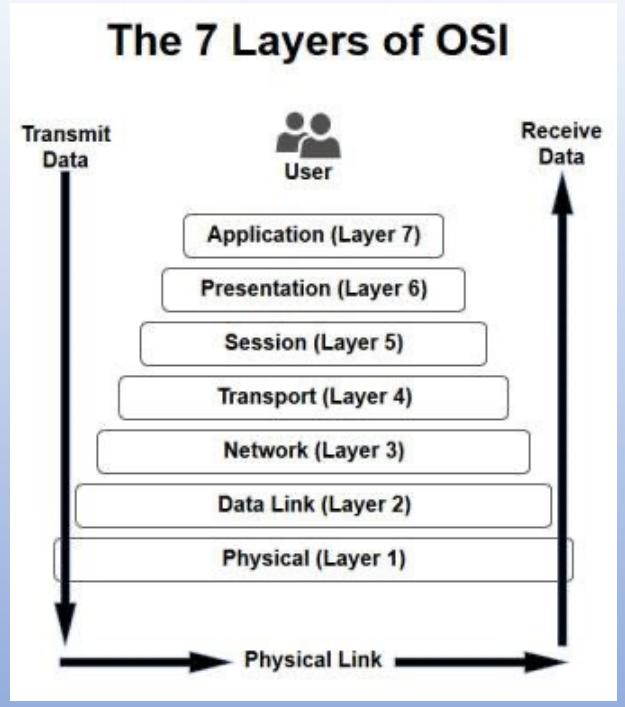
Generalized P&F: Streaming Architectures

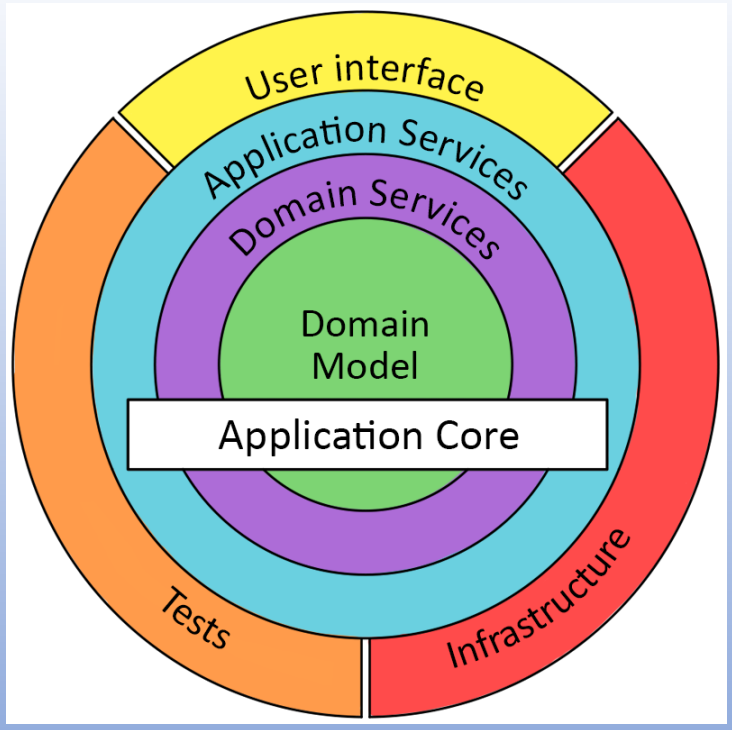
- Stream-based databases
- Stellar Computer (graphics)
- Data Visualization

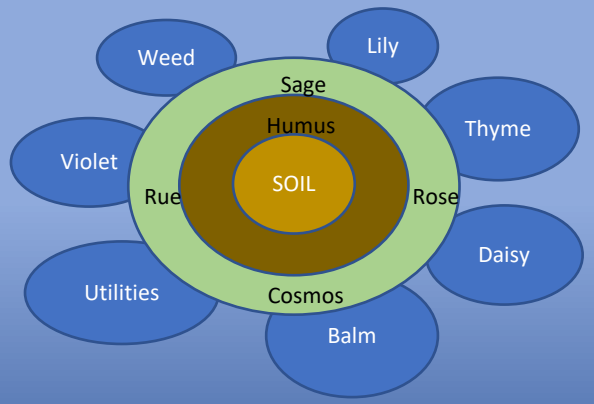


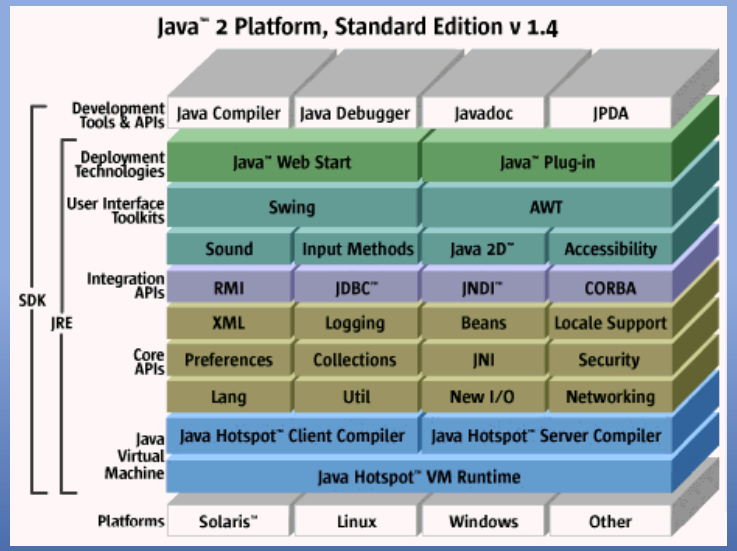
Layered System

- Onion Model
- Hub and Spokes Model
- Classic Example
  - Networking Protocols
- Other Examples
  - Garden
  - Java
  - Traditional Apps

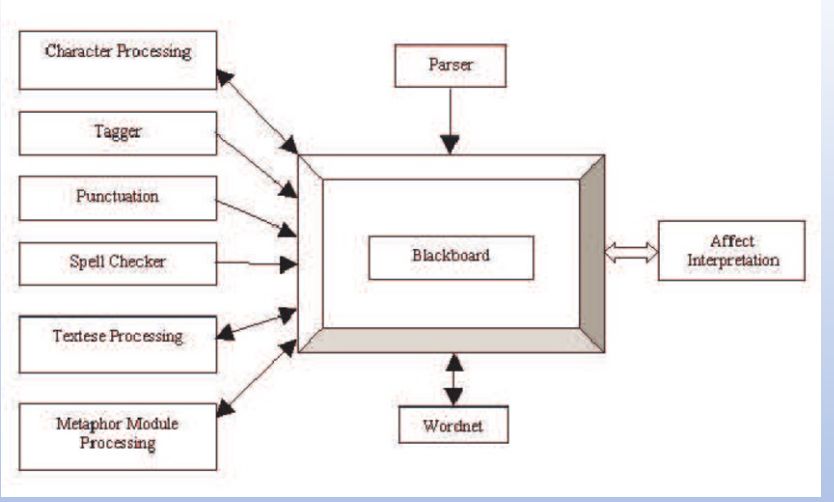




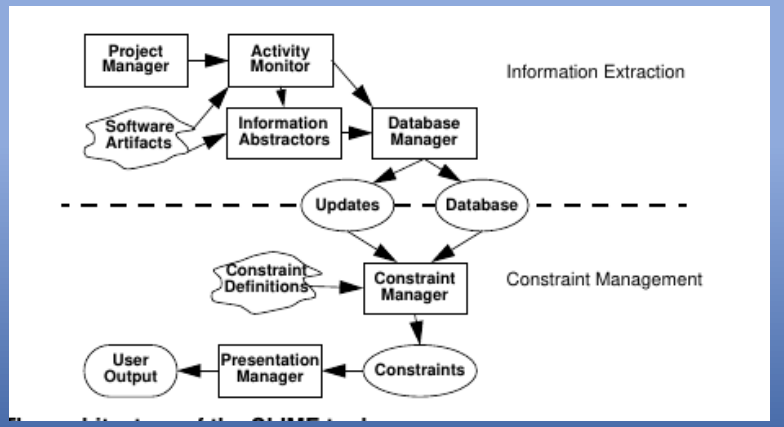




Repository Architectures



- Shared data structure
  - Blackboard, data store, database
  - Various components access and update that structure
  - Often with notifications on change
    - Communication done via the central structure
- Classic Example
  - Programming Environment – commonality is the source code
    - PECAN – AST-based
    - Eclipse and Idea are similar if not expanded
  - AI blackboard systems
- Other Examples
  - Clime with central database
  - MVC-based web applications

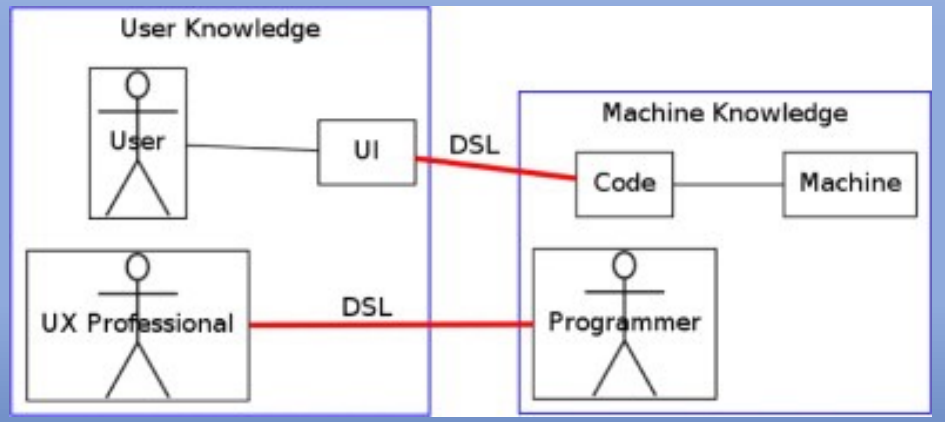


Interpreters

- Domain-Specific Languages
  - Define a language for writing the implementation
  - Write the actual system in that language
  - Interpret (compile) to run the system
  - Language often is just a library
- Classic Examples
  - Scientific Computing Libraries and DSLs
  - Systems for processing big data
    - Python-based scripting
    - Also computer vision, machine learning
  - Rule-based systems
- Other Examples
  - UPOD: smartsign, smarthab, smartthings, smarthome

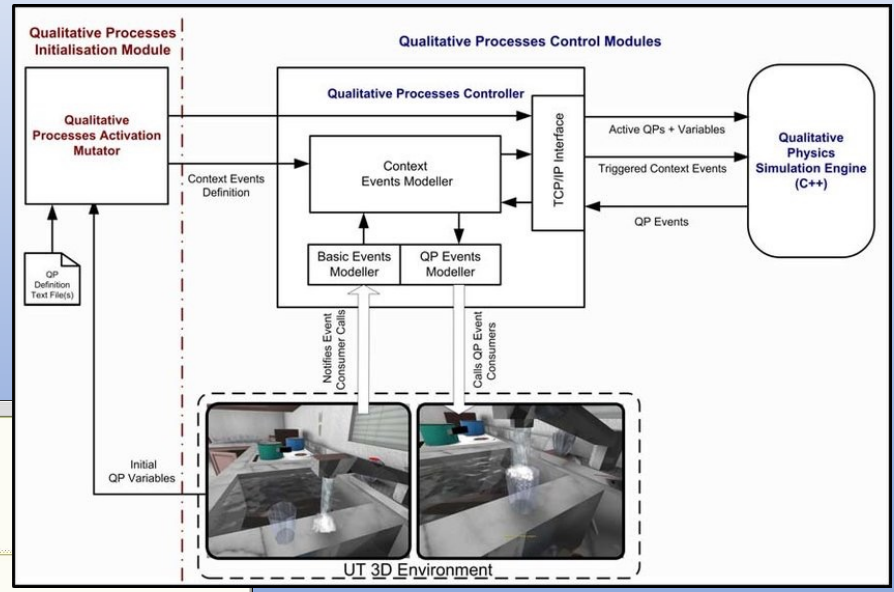
DSL Examples

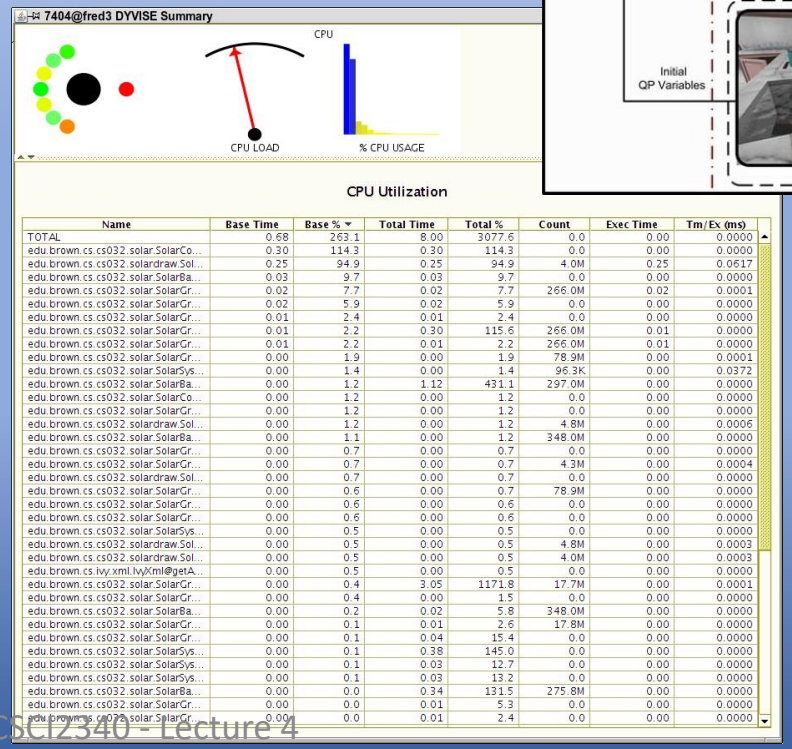
DSL	Domain
SQL	Database Manipulation
Postscript	Publishing
Hibernate	Object Relational Mapping
Regex	Pattern Matching
BNL	Business Natural Language
Adhersion	Telecom



Process Control Architectures

- Mostly used in IoT (RT/Embedded)
- Control loop with feedback
- Classic Example
  - Thermostat
- Other Examples
  - Self-driving cars
  - Many real-time systems
  - DYVISE tuning

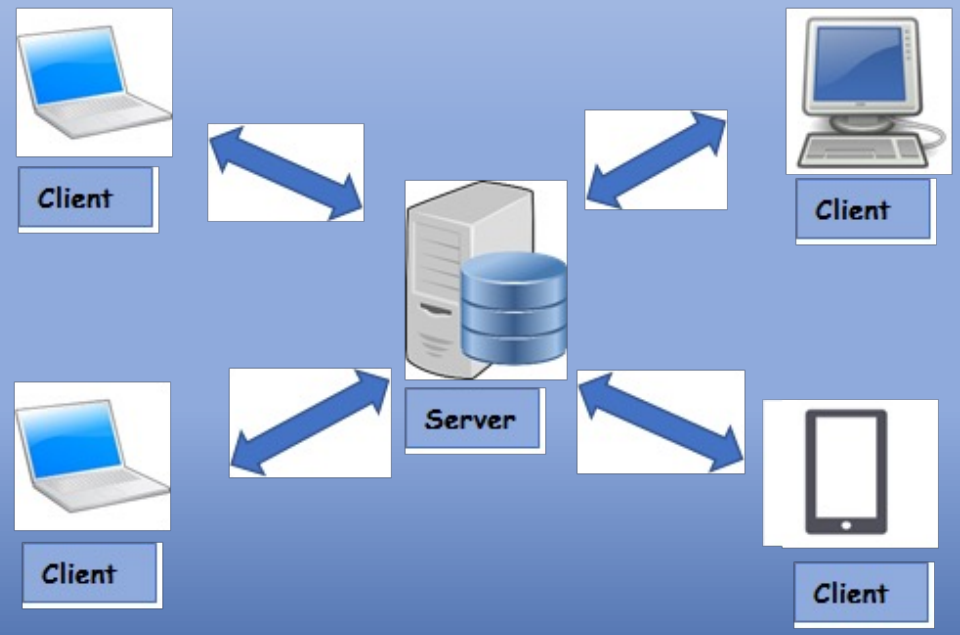




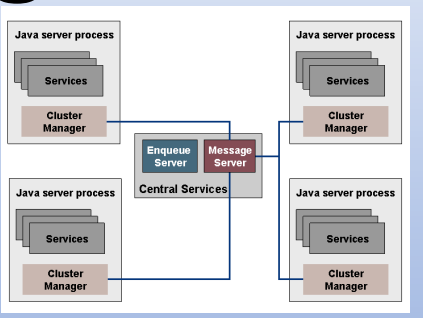
Name	Base Time	Base %	Total Time	Total %	Count	Exec Time	Tm/Ex (ms)
TOTAL	0.68	263.1	8.00	3077.6	0.0	0.00	0.0000
edu.brown.cs.cs032.solar.SolarCo...	0.30	114.3	0.30	114.3	0.0	0.00	0.0000
edu.brown.cs.cs032.solardraw.Sol...	0.25	94.9	0.25	94.9	4.0M	0.25	0.0617
edu.brown.cs.cs032.solar.SolarBa...	0.03	9.7	0.03	9.7	0.0	0.00	0.0000
edu.brown.cs.cs032.solar.SolarGr...	0.02	7.7	0.02	7.7	266.0M	0.02	0.0001
edu.brown.cs.cs032.solar.SolarGr...	0.02	5.9	0.02	5.9	0.0	0.00	0.0000
edu.brown.cs.cs032.solar.SolarGr...	0.01	2.4	0.01	2.4	0.0	0.00	0.0000
edu.brown.cs.cs032.solar.SolarGr...	0.01	2.2	0.30	115.6	266.0M	0.01	0.0000
edu.brown.cs.cs032.solar.SolarGr...	0.01	2.2	0.01	2.2	266.0M	0.01	0.0000
edu.brown.cs.cs032.solar.SolarGr...	0.00	1.9	0.00	1.9	78.9M	0.00	0.0001
edu.brown.cs.cs032.solar.SolarSys...	0.00	1.4	0.00	1.4	96.3K	0.00	0.0372
edu.brown.cs.cs032.solar.SolarBa...	0.00	1.2	1.12	431.1	297.0M	0.00	0.0000
edu.brown.cs.cs032.solar.SolarCo...	0.00	1.2	0.00	1.2	0.0	0.00	0.0000
edu.brown.cs.cs032.solar.SolarGr...	0.00	1.2	0.00	1.2	0.0	0.00	0.0000
edu.brown.cs.cs032.solardraw.Sol...	0.00	1.2	0.00	1.2	4.8M	0.00	0.0006
edu.brown.cs.cs032.solar.SolarBa...	0.00	1.1	0.00	1.2	348.0M	0.00	0.0000
edu.brown.cs.cs032.solar.SolarGr...	0.00	0.7	0.00	0.7	0.0	0.00	0.0000
edu.brown.cs.cs032.solar.SolarGr...	0.00	0.7	0.00	0.7	4.3M	0.00	0.0004
edu.brown.cs.cs032.solardraw.Sol...	0.00	0.7	0.00	0.7	0.0	0.00	0.0000
edu.brown.cs.cs032.solar.SolarGr...	0.00	0.6	0.00	0.7	78.9M	0.00	0.0000
edu.brown.cs.cs032.solar.SolarGr...	0.00	0.6	0.00	0.6	0.0	0.00	0.0000
edu.brown.cs.cs032.solar.SolarGr...	0.00	0.6	0.00	0.6	0.0	0.00	0.0000
edu.brown.cs.cs032.solar.SolarSys...	0.00	0.5	0.00	0.5	0.0	0.00	0.0000
edu.brown.cs.cs032.solardraw.Sol...	0.00	0.5	0.00	0.5	4.8M	0.00	0.0003
edu.brown.cs.cs032.solardraw.Sol...	0.00	0.5	0.00	0.5	4.0M	0.00	0.0003
edu.brown.cs.ivy.xml.IvyXml@getA...	0.00	0.5	0.00	0.5	0.0	0.00	0.0000
edu.brown.cs.cs032.solar.SolarGr...	0.00	0.4	3.05	1171.8	17.7M	0.00	0.0001
edu.brown.cs.cs032.solar.SolarGr...	0.00	0.4	0.00	1.5	0.0	0.00	0.0000
edu.brown.cs.cs032.solar.SolarBa...	0.00	0.2	0.02	5.8	348.0M	0.00	0.0000
edu.brown.cs.cs032.solar.SolarGr...	0.00	0.1	0.01	2.6	17.8M	0.00	0.0000
edu.brown.cs.cs032.solar.SolarGr...	0.00	0.1	0.04	15.4	0.0	0.00	0.0000
edu.brown.cs.cs032.solar.SolarSys...	0.00	0.1	0.38	145.0	0.0	0.00	0.0000
edu.brown.cs.cs032.solar.SolarSys...	0.00	0.1	0.03	12.7	0.0	0.00	0.0000
edu.brown.cs.cs032.solar.SolarSys...	0.00	0.1	0.03	13.2	0.0	0.00	0.0000
edu.brown.cs.cs032.solar.SolarBa...	0.00	0.0	0.34	131.5	275.8M	0.00	0.0000
edu.brown.cs.cs032.solar.SolarGr...	0.00	0.0	0.01	5.3	0.0	0.00	0.0000
edu.brown.cs.cs032.solar.SolarGr...	0.00	0.0	0.01	2.4	0.0	0.00	0.0000

Client-Server Distributed Processing

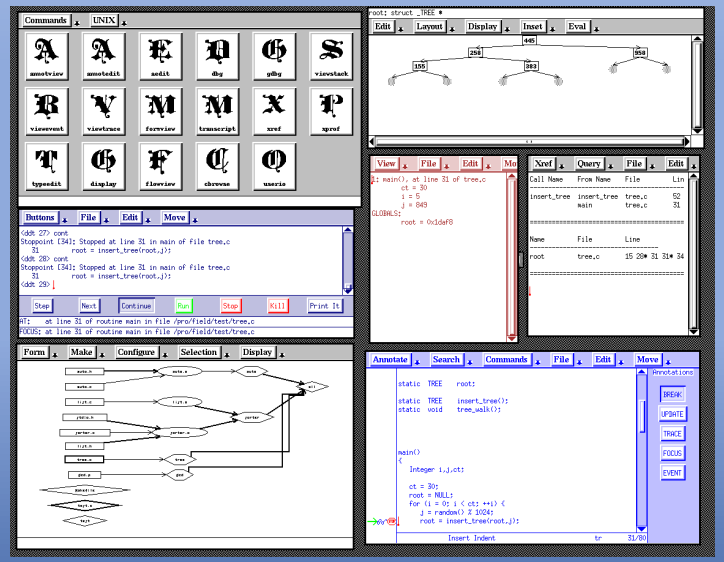
- Server acts as a master
  - Clients talk to the master (not to each other)
  - Message-based communication
  - State can be global or distributed
- Classic Example
  - Computer Games
- Other Examples
  - Web-based applications
- Peer-to-Peer systems
  - TAIGA

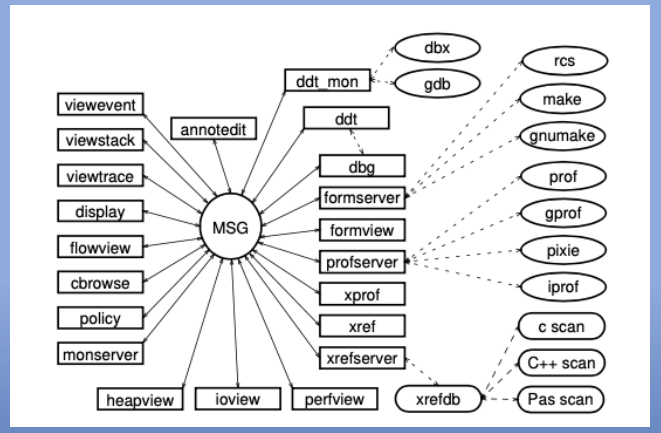


Message-Based Distributed Processing

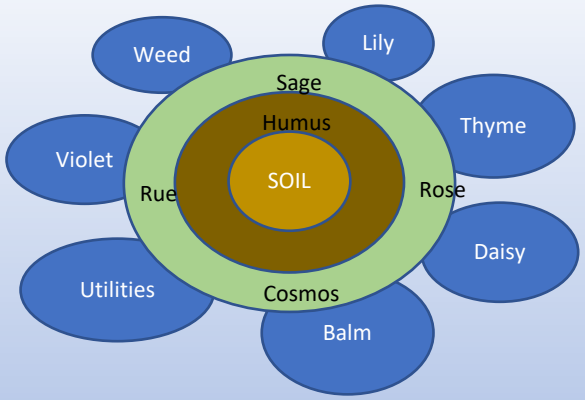


- Set of Processes communicating via messages
  - Messages can be process-to-process
  - More general is a central message server and broadcast messages
- Classic Example
  - FIELD environment
- Other Examples
  - CORBA-based computing
  - Agent-based systems

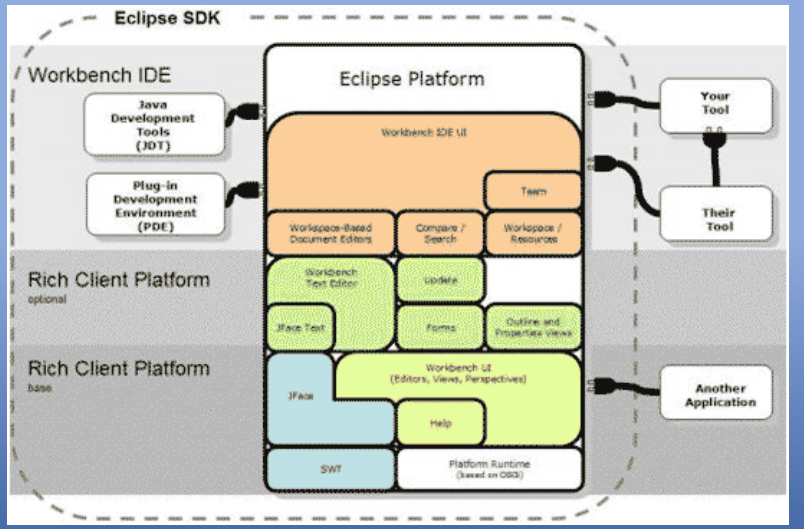




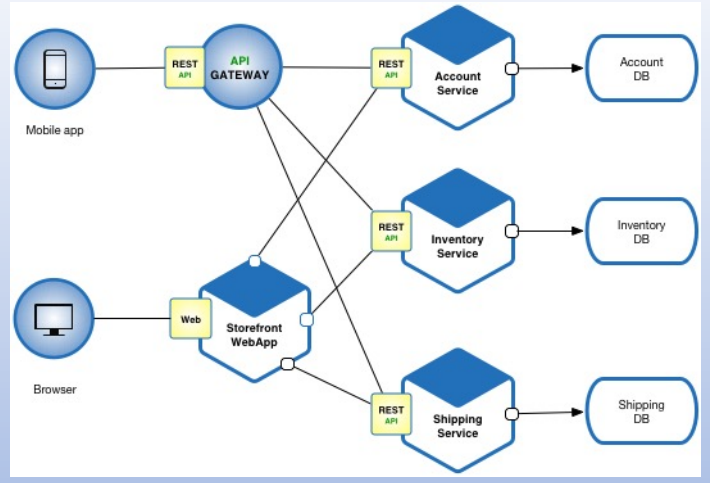
Plug-in Architectures



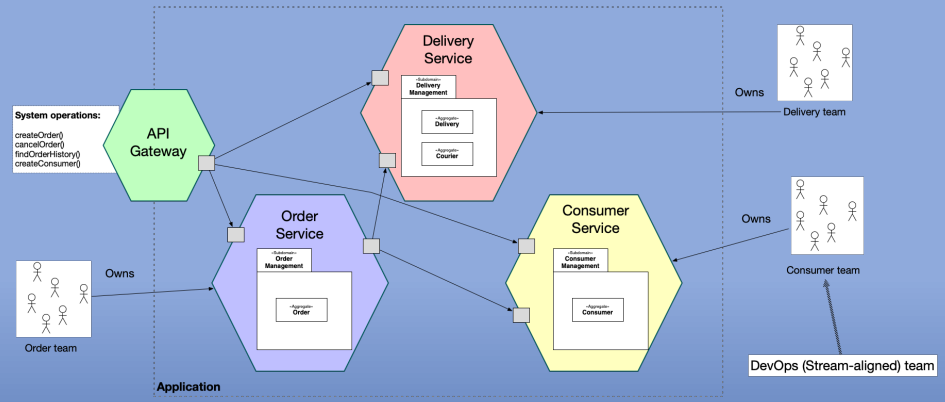
- Core + Extensions with dynamic plug-ins
  - Facilities to dynamically load and integrate plug-in code
    - Done at run time, not compile time
  - Extension points
    - Allow plug-ins to augment the UI, other capabilities of core
    - Allow core to provide notifications to the plug-ins
  - Plug-ins can invoke the core system
    - Register callbacks, do operations
- Examples
  - Eclipse
  - VS Code



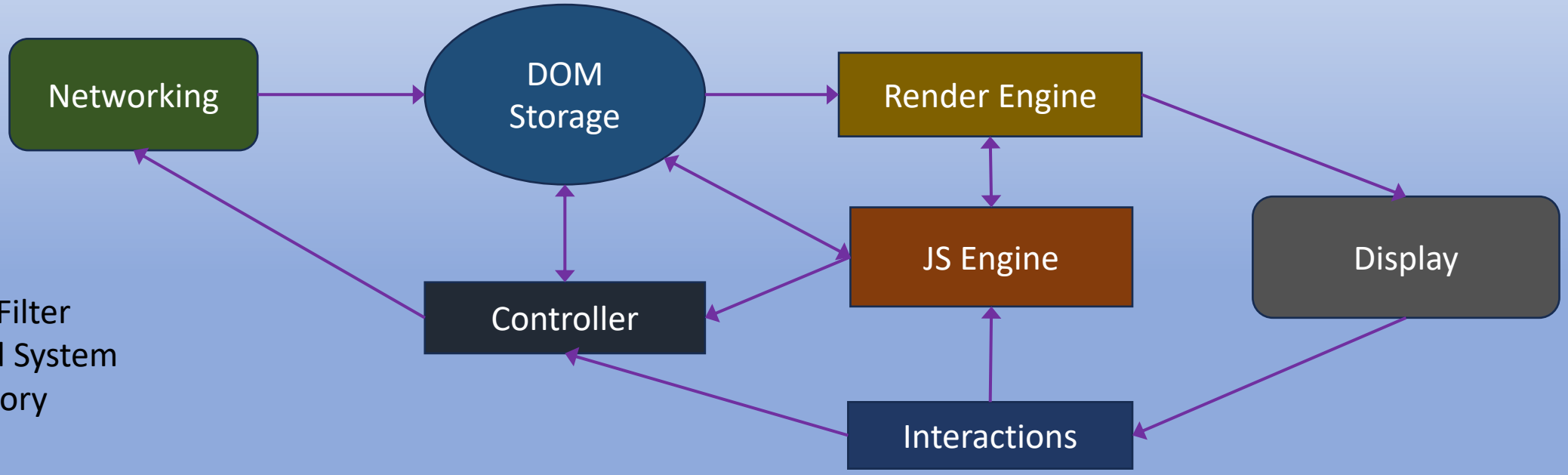
Microservice Architectures



- Application built as a collection of services
  - Services can be implemented independently
  - Services can be reused by multiple applications
  - Services are loosely coupled
  - Database-oriented
- Flexible, scalable, team-oriented
- Examples
  - Modern web applications



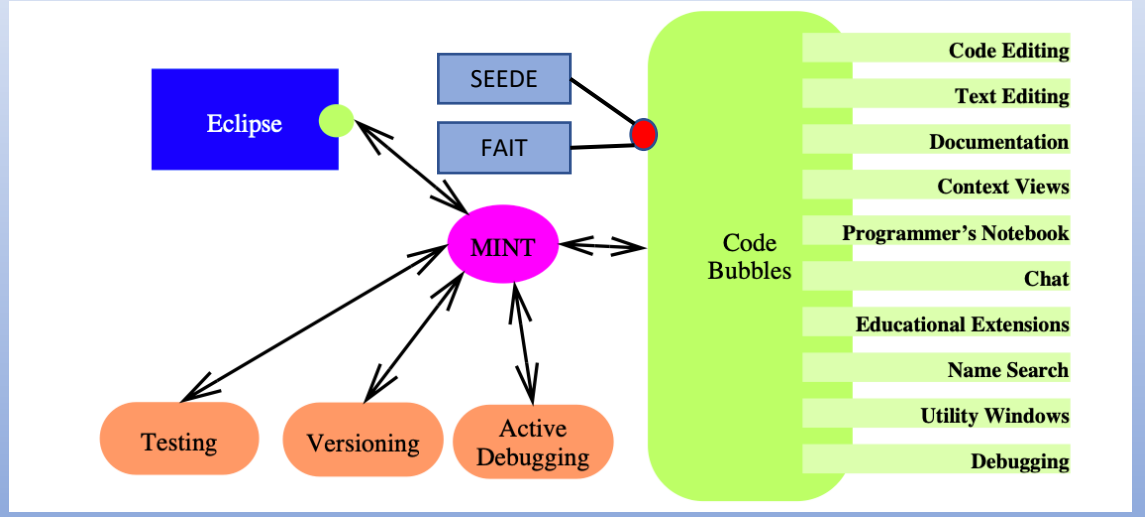
Browser Architecture



Pipe & Filter
Layered System
Repository
Plug-in

Heterogeneous Architectures

- Most complex systems employ multiple architectures
  - Combinations of the above
- Example: Code Bubbles
  - Message-based with communicating processes
  - Layered architecture for main system
  - Plug-in capability for extensions
  - Repository based on source code and file system

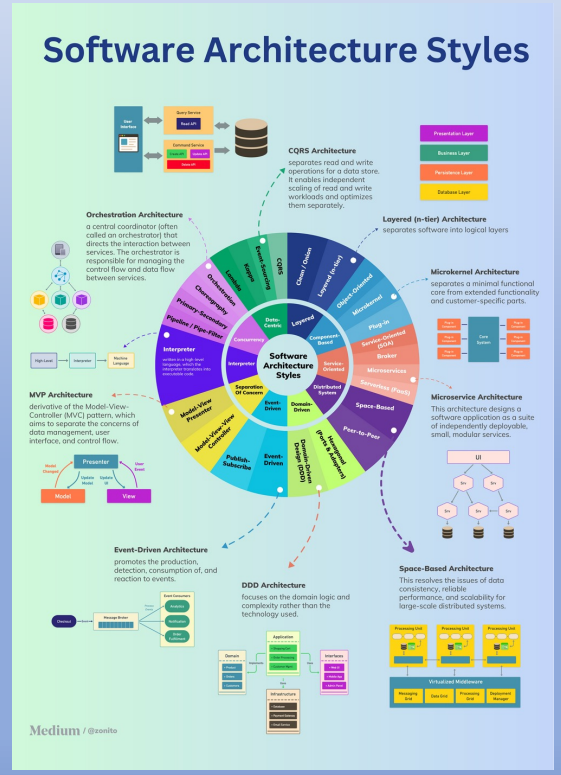


Software Architecture

- Understanding these (and other) high-level models
- Understanding when they are (and are not) appropriate
  - These are essentially high-level design patterns
  - Design patterns always tell WHEN they are appropriate
  - Should also indicate when NOT appropriate
    - When they make the system more complex
    - When they make the system harder to maintain
- Understanding what best applies to your problem
  - What will make your life simpler

Choosing a Software Architecture

- Decide on a set of basic high-level components
  - What is the basic functionality
  - Emphasize data over computation
  - Handling the specifications
- Determine which architectures are appropriate
  - Based on the components & their interactions
- Account for constraints
  - Building a small system first
  - Risks, security, performance, …



Other Things to Consider

- Team-based Designs
  - An architecture that allows an easy split among team members
  - Having people work on individual components is easier
    - Each person should have a well-defined component (or set thereof)
    - Number of component correlates with team size
- Risk-based Designs
  - Understanding what is difficult, unknown, etc.
  - Isolating and addressing risks
- Overall architecture is generally the charge of main designer
  - Having group discussions helps

Research in Software Architectures

- Automatic redesign based on environmental deviations
- Neural-net based architectures (using LLMs)
- Understanding design decisions

HOMEWORK

- If you don’t have a personal GitHub account
  - Create one
- Start thinking about coding the assignment
  - Use Code Bubbles for Java/node/dart if desired
  - Initial implementation due 2/13 (one week)
  - We will discuss coding next time

PROJECT

- Decide on a project name
- Discuss architectures at your next project meeting
  - Develop a proposal
- Continue working on specifications
  - Should have a good high-level idea of what the team will be doing
  - Should have a complete high-level view
  - Details can be filled in later

Further Reading

- Introduction to Software Architecture
  - Classic paper introducing the field
- Textbook – Chapter 5