Architectural Blueprints – Views of Software Architecture and You

Presented by Chris Mennie

University of Waterloo

Goals

- Clarify different architectural views
  - 4+1, Hofmeister, etc
- Some pointers on how to do architectural document

About Kruchten

- Doctorate degree in computer science from the French Institute of Telecommunications
- Director of Process Development at Rational Software
- Led the development of the Rational Unified Process

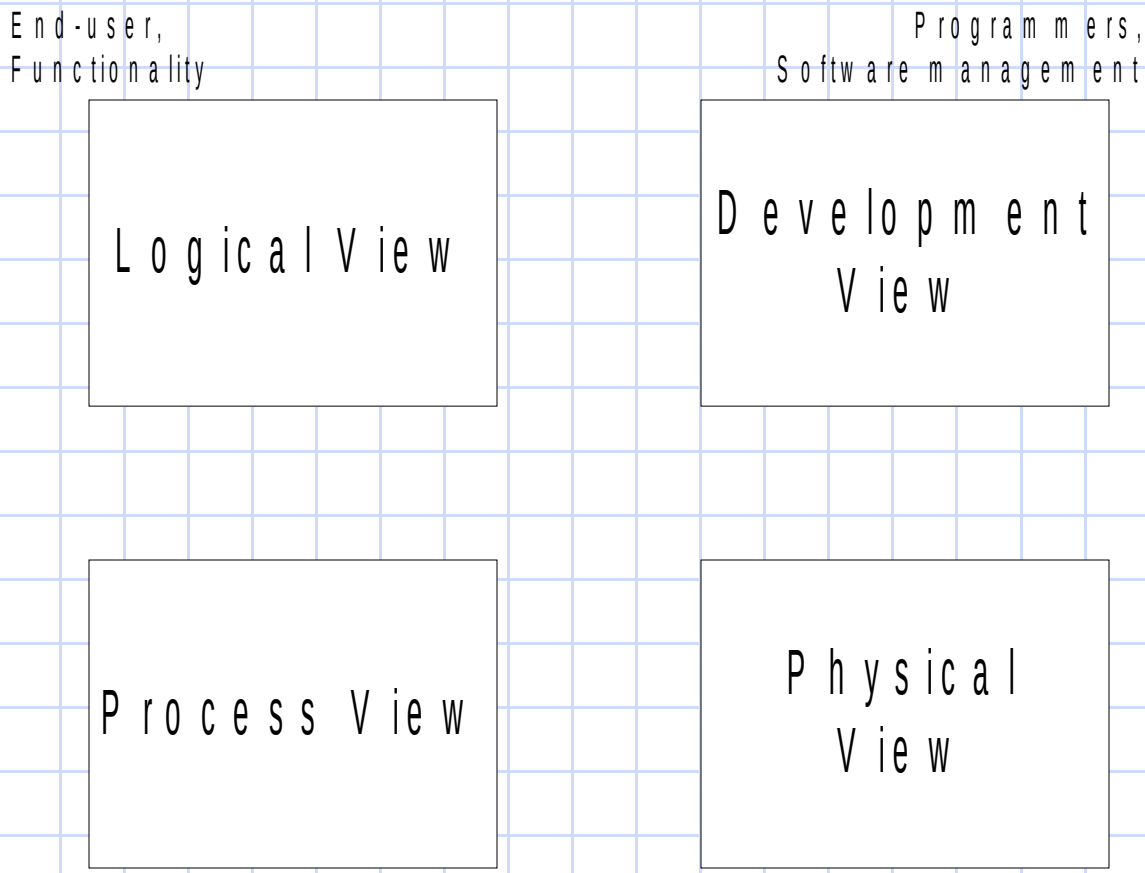
Not Enough Clarity

- Issue:
  - One diagram hard to capture everything
  - Different stakeholders care about different issues
  - What to do?
- Solution:
  - Look at architecture in different ways

Views

- A view is a different perspective of the whole system:
  1. Logical View - (Object) model of the system
  2. Development View - Static organization of the software in development
  3. Process View – Dynamic view of system (concurrency and synchronization)
  4. Physical View – Mapping software to hardware

Views


End-user,
Functionality
Programmers,
Software management
Logical View
Development View
Process View
Physical View
Integrators,
Performance,
Scalability
System engineers,
Topology,
Communications

Logical View

- Focuses on functional requirements
- Mainly of interest to end user
- System decomposed into key abstractions (in form of objects/classes)
- Class diagram shows logical relationships between classes
- May also include internal class behaviour descriptions

Logical View from Example

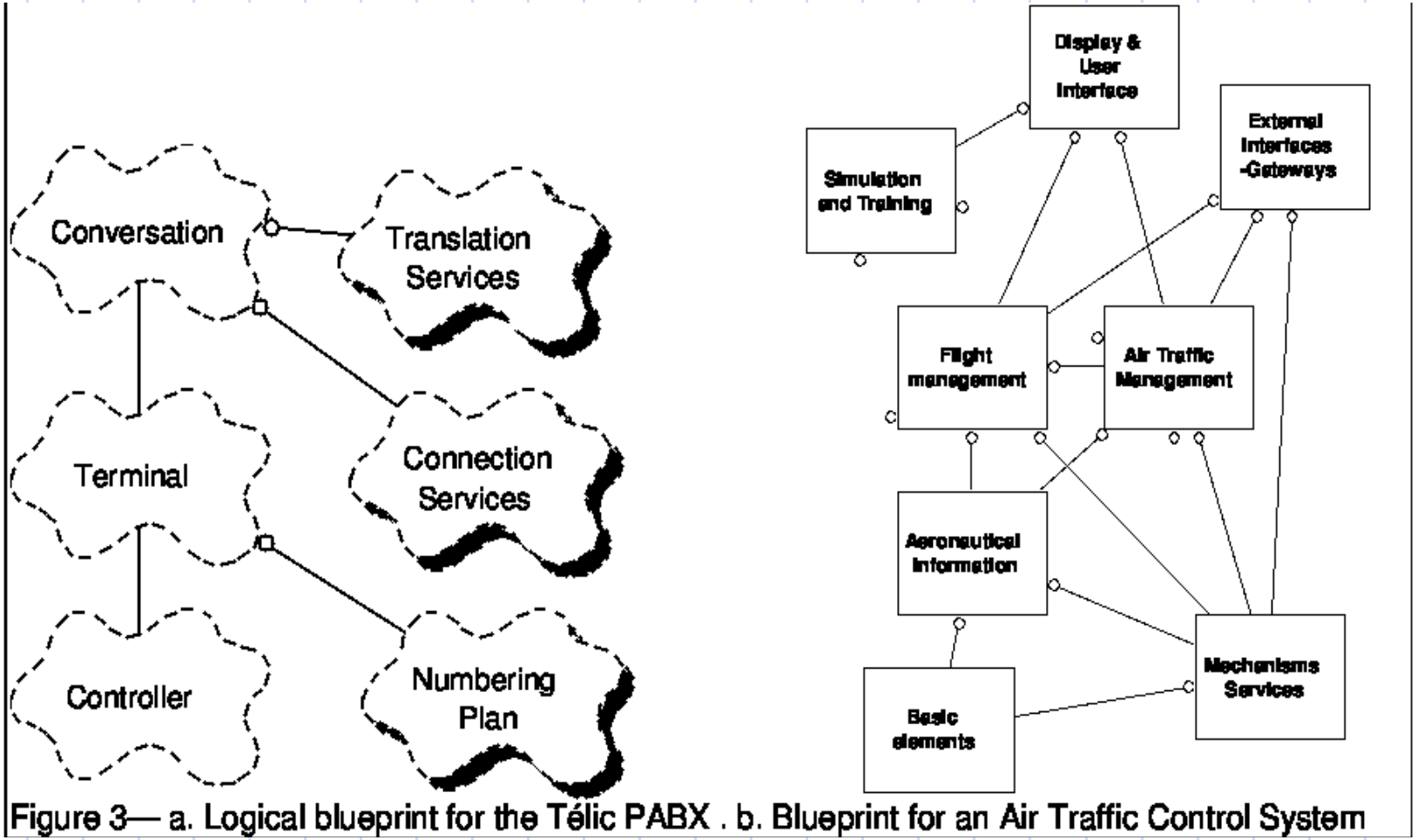
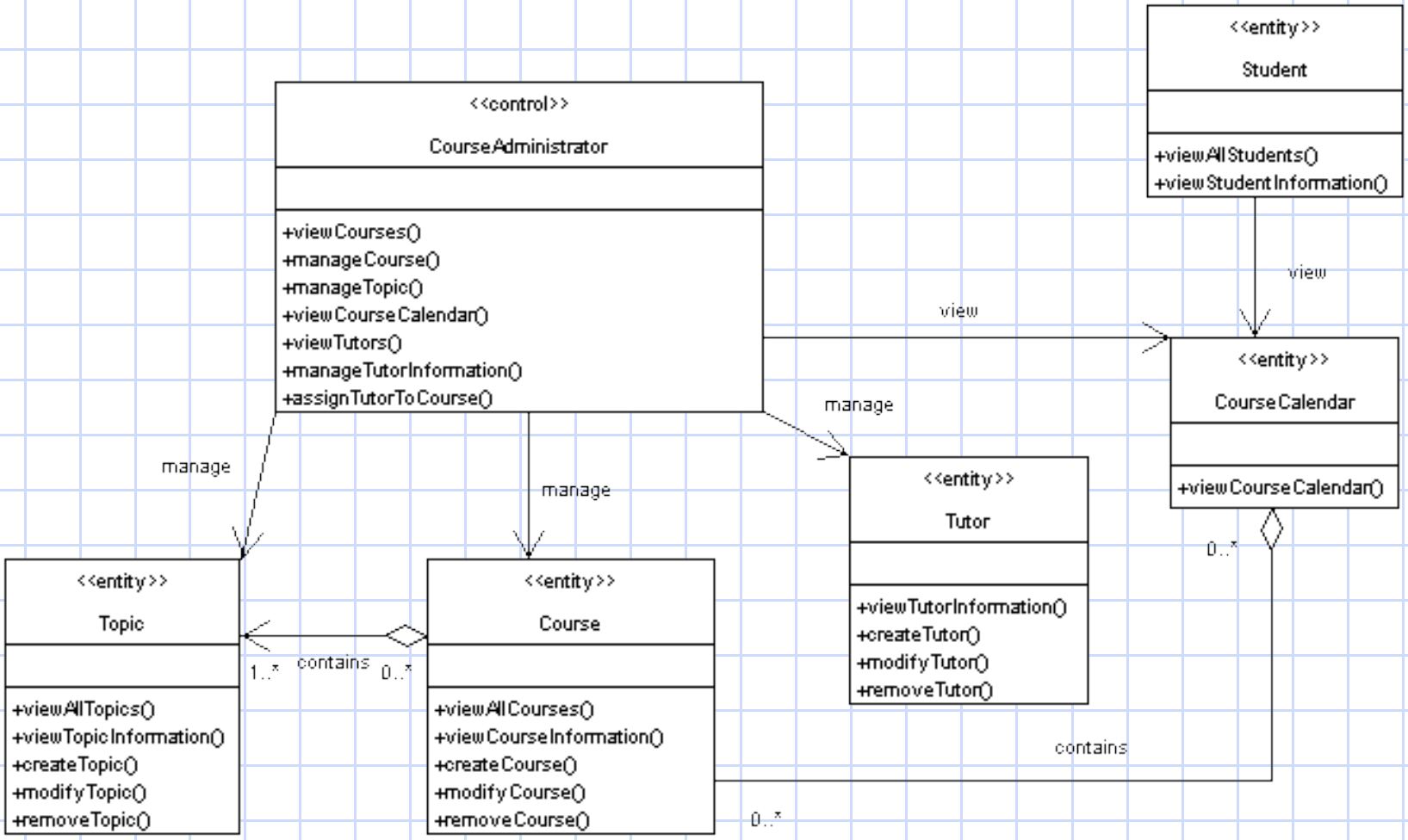


Figure 3— a. Logical blueprint for the Télic PABX . b. Blueprint for an Air Traffic Control System

Logical View (as in SRS)



Logical View

- Model function described with state or activity diagrams (ie: internal behaviour)
- Verification/validation and description done with sequence/collaboration diagrams
- Not meant for your architecture diagrams in this class

Development View

- Focuses on actual software organization
- Mainly of interest to coders
- System is decomposed into components and subcomponents
- All interfaces described
- Serves as basis for resource estimation and management
- Close to logical view and usually deduced from it
- Can be abstract or concrete or both

Development View

- Will have many things in common with Logical View
- More detailed though
  - Things like workers, locks, protocol translators

Development View from Paper

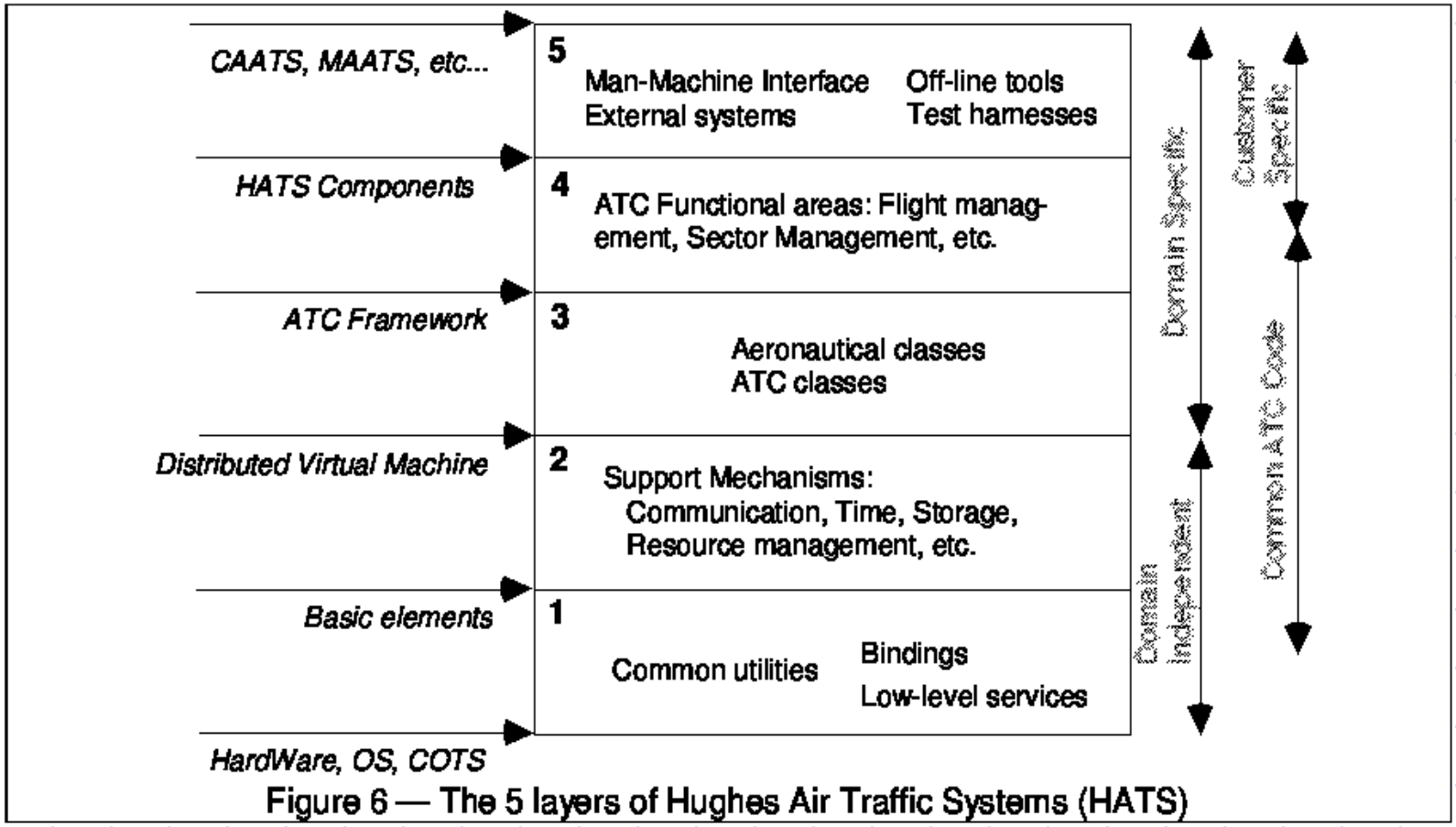
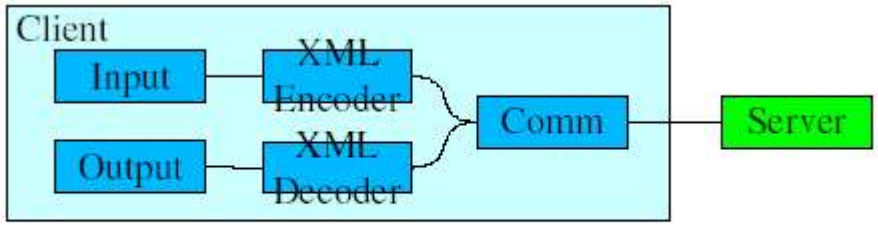


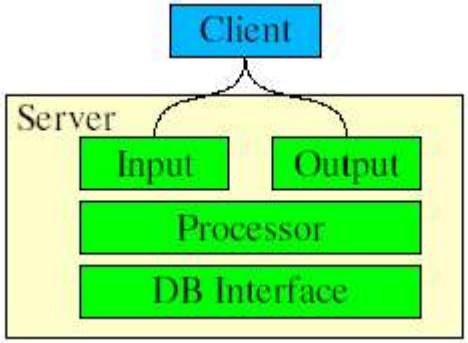
Figure 6 — The 5 layers of Hughes Air Traffic Systems (HATS)

Development View Example

High Level:

Lower Level:





Development View Example

- Start with a reasonable amount of detail
- Give diagram and descriptions
- Describe the modules/components and how they relate to each other
- Give enough, but not too much detail (tricky)

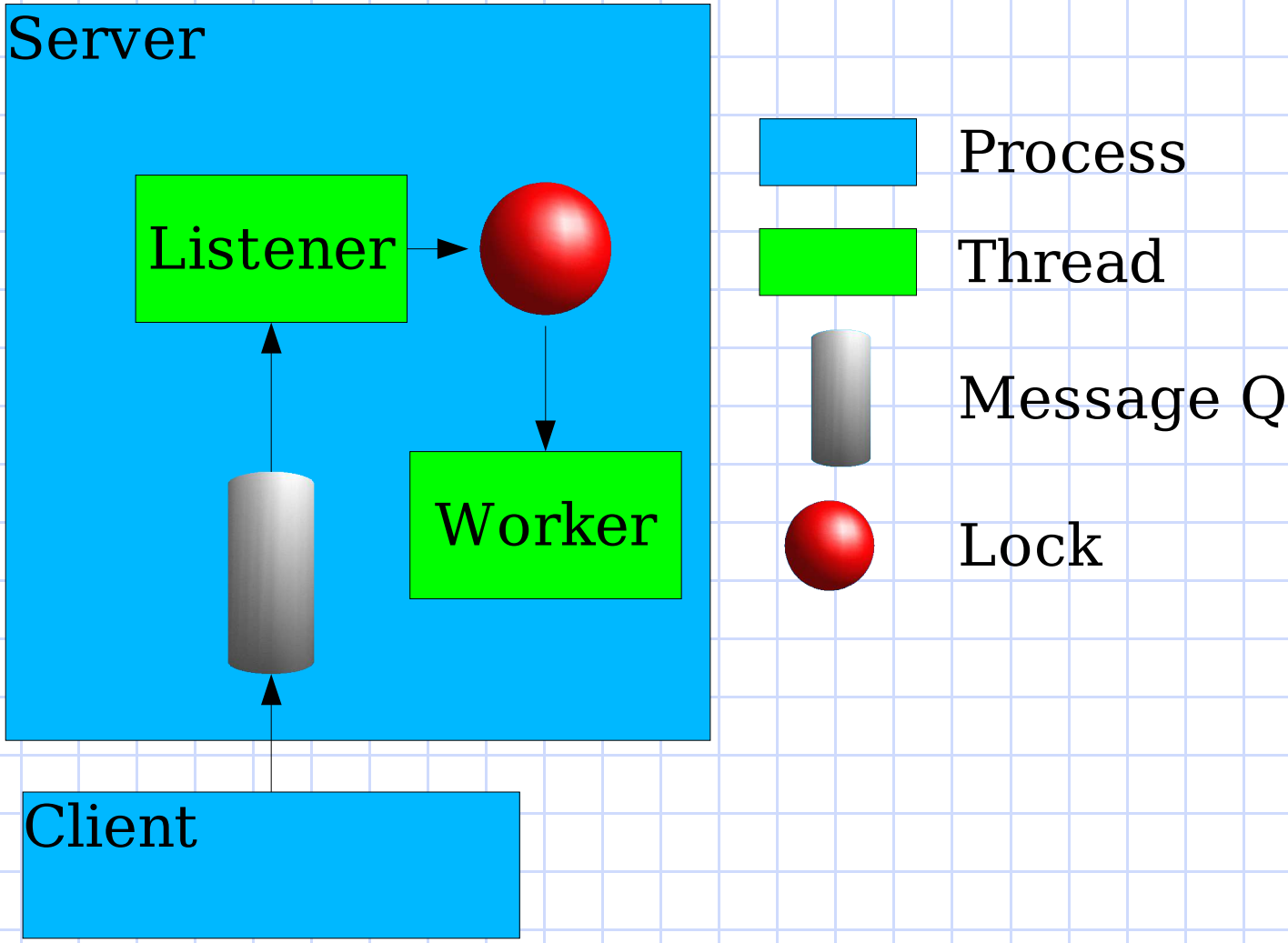
Development View Example

- "Expand" components to show more details
- Keep doing this "recursively"
- Feel free to use hybrid styles
  - May mix styles depending on level of component
- Feel free to invent notation
  - Describe in detail meanings of notation no matter how obvious

Process View

- Considers non-functional requirements
- Mainly of interest to designer and integrator
- Addresses concurrency and distribution
- Describes dynamic view of system (processes, threads, etc) and mapping to them
- Message flow and process loads can be estimated from this view
- Covers processes, threads, message queues, locks, etc

Process View Example



Development and Process Views

- Need to refer process view back to development view
- Try to keep labels consistent between these views
  - I won't assume something with the same name in both views is the same
  - Be explicit in tying views together
- Decompose diagrams if necessary, like with development view

Interfaces

- Every interface needs to be described
  - What constitutes an interface? Every line, to a point
- Both the delivery mechanism and protocols used
  - ie: RPC, method calls, XML over TCP, etc
- Give the schema, API, etc
- For protocols, give examples of their typical use

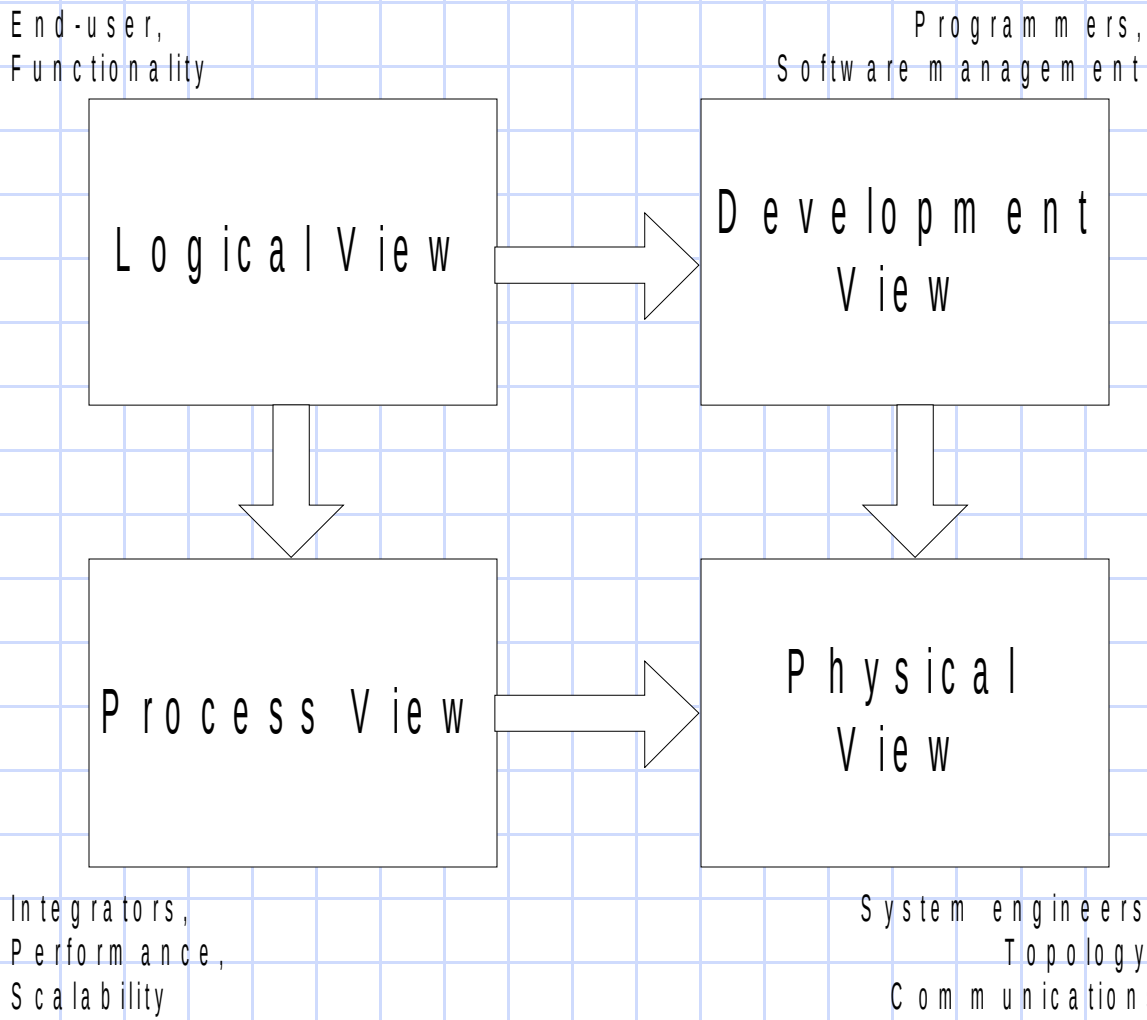
Physical View

- Considers requirements such as reliability, performance, scalability
- Mainly of interest to system designer
- Details how processes, threads, objects, etc are mapped to physical resources
- Closely related and influenced by process view

Physical View

- Process view needs to map to physical view
- Only include relevant components
  - It is irrelevant that the “Admin Console” has a keyboard, the architecture isn’t affected
  - Conversely, a word processor from the 70’s might care about the keyboard connected
- Your physical views won’t be terribly interesting, more for process view map

Views


End-user,
Functionality
Programmers,
Software management
Logical View
Development View
Process View
Physical View
Integrators,
Performance,
Scalability
System engineers,
Topology,
Communications

+1: Scenarios

- Unifies all views by providing intent
- Similar to a cross between UML use cases and scenarios
- In a sense an abstraction of the most important requirements
- Drives discovery of architectural elements
- Serves as a validation of architecture

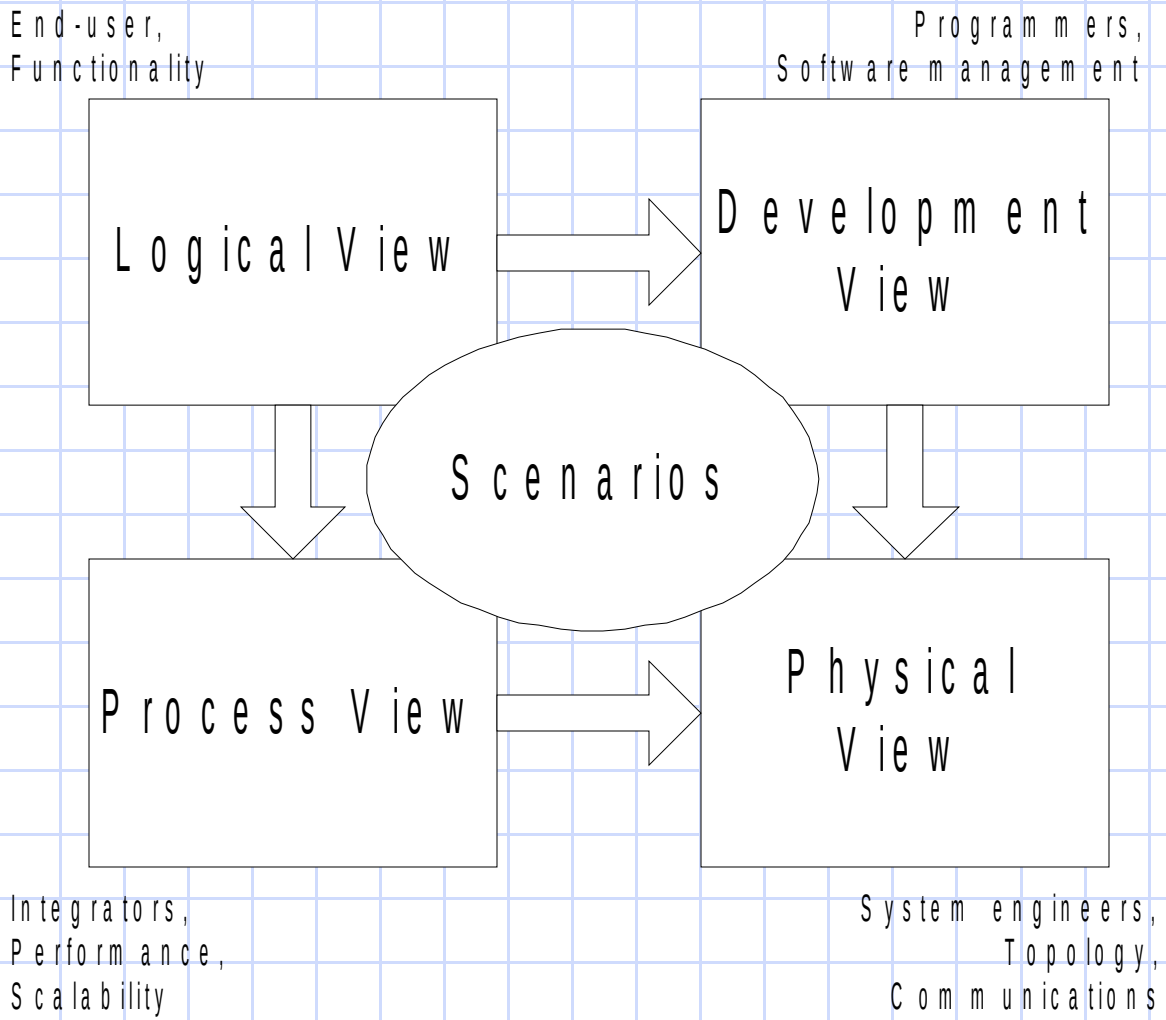
Scenarios

- Scenarios for your logical architecture are in your SRS
  - Use cases and sequence diagrams
- Provide scenarios for development view and possibly process view
- Don’t provide new information, just validate and clarify previous views
- Scenarios and interfaces are important to pulling the overall arch together

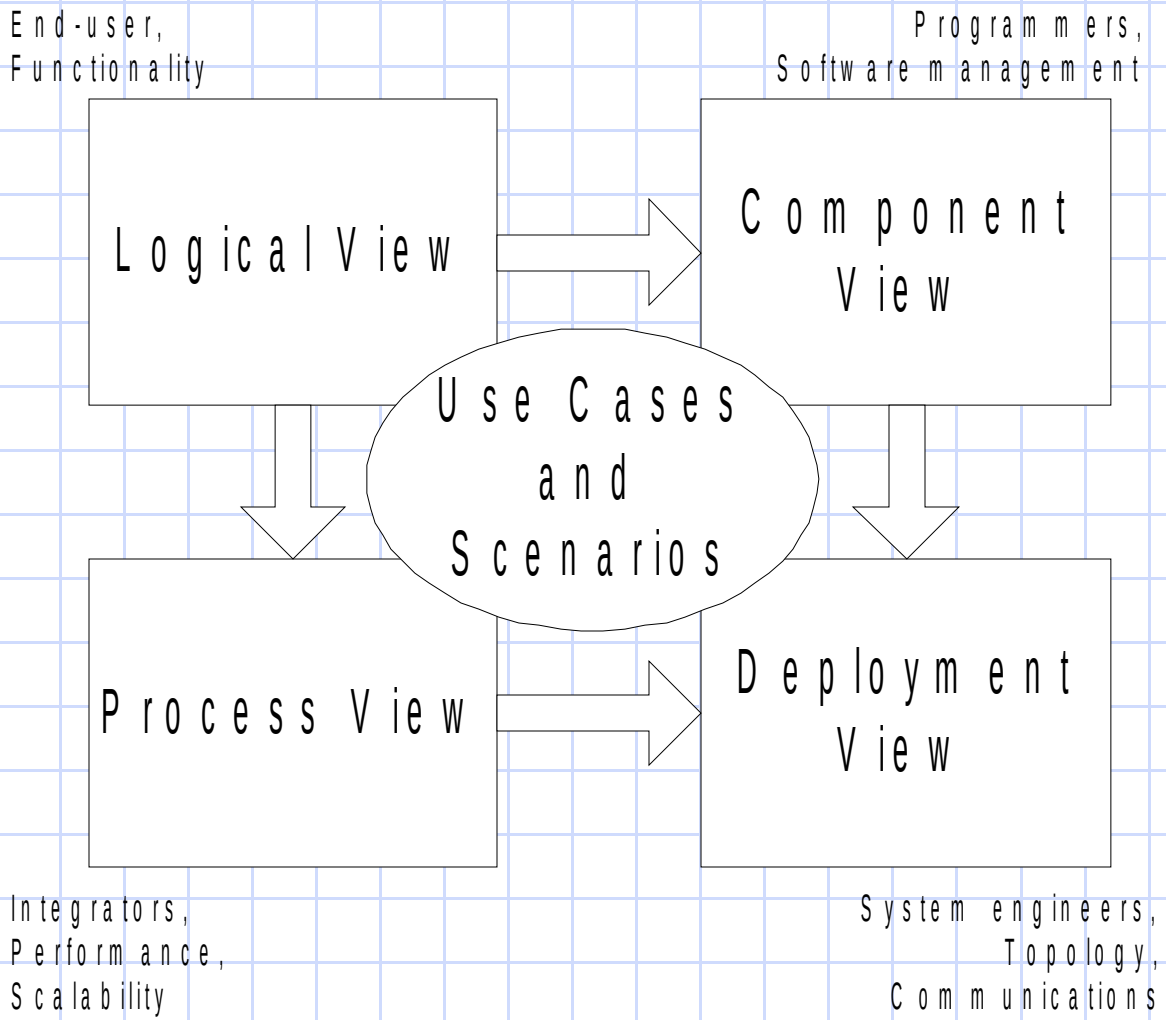
Scenarios

- Scenarios could be in the form of sequence diagrams or collaboration diagrams

Views


End-user,
Functionality
Programmers,
Software management
Logical View
Development View
Scenarios
Process View
Physical View
Integrators,
Performance,
Scalability
System engineers,
Topology,
Communications

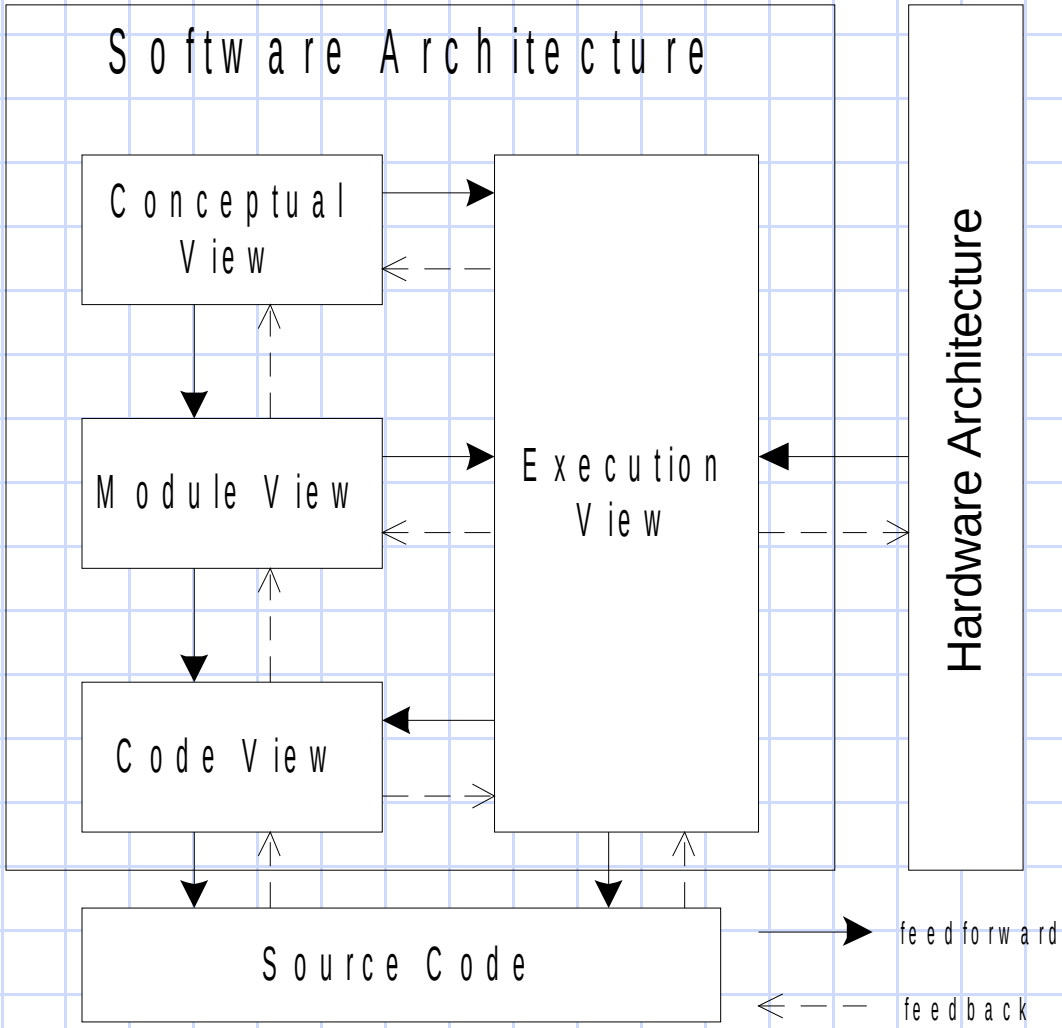
UML / Rational's Views



Hofmeister *et al.* Views

- Four views from study of large complex industrial systems
  1. Code View – Organisation of code into sources, directories, versioning, etc
  2. Module View – Decomposition of the system into modules and layers
  3. Execution View – Decompose system into processes and resource utilisation
  4. Conceptual View – Logical view of system; design elements and relationships among them

Relations Amongst Views



Kruchten vs. Hofmeister *et al*.

- Conceptual and Logical views very similar
- Module and Development views very similar
- Execution view similar to Physical view, and some of Process view
- Code view is new

Code View

- Describe the structure of the code
- File / directory hierarchies
- Class inheritance diagrams
  - Not detailed class information (members)
- Relate the code view to development and process views
  - What class represents what component, or vice-versa
  - What directory represents a process, etc?

Code View

- Use diagrams, tables, paragraphs, whatever works
- Be clear in your mappings

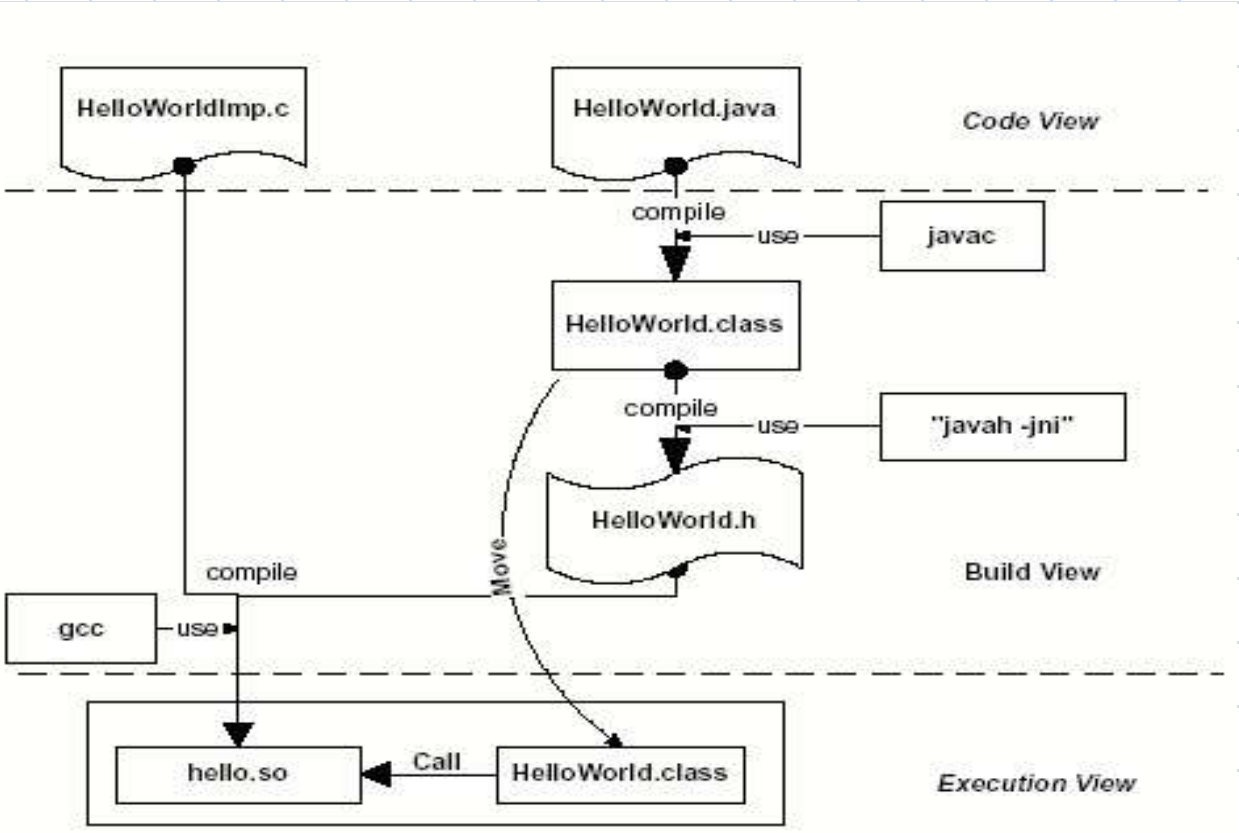
Going about it

1. Write development view, then process view (and physical)
2. Detail all interfaces and do estimates
3. Write your code
   - Don't integrate at the last minute!
4. Write code view, relating it to process and development views
5. Ensure consistency amongst views (very important)

Build-Time Architecture View

- Some systems have interesting build-time behaviour
- Different from other views
- Captures dynamic behaviour
  - Build scripts
  - Configuration choices
- Model compilation dependencies or automatic code generation

The Build-Time View of JNI



Conclusions

- More than one way to look at software architecture
- Different kinds of architecture
- Views proposed are reasonable and used in practice
- Many kinds of views exist and one shouldn’t hesitate to be creative in striving for clarity

Conclusions

- A guide to architecture documents written for previous term
  - May be helpful for ideas
- An example architecture document is given
  - For ideas only! Don’t follow too closely as it was for a different class, assignment
  - It wouldn’t do that well for this deliverable