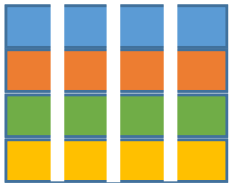
# Evaluating the Architecture and the Viewpoints of Architecture

IT Architecture and User Driven Software Design (BUITA)

25. September 2018

Keld Bødker



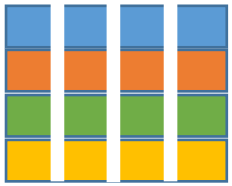
Evaluating the architecture

The viewpoints of architecture

Dependencies between views
-how to ensure consistency?

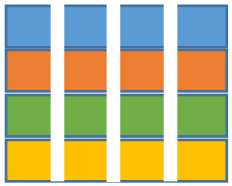
# Software Systems Architecture

- Part I: fundamentals (ch. 2-5)
- Part II: the process of software architecture (ch. 6-14)
- Part III: the viewpoint catalog (ch. 15-23)
- Part IV: the perspective catalog (ch. 24-29)
- Part V: putting it all together (ch. 30)



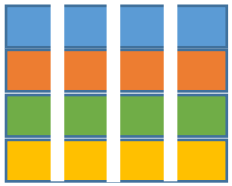
# Evaluating the architecture

- Chapter 14, Rozansky & Woods
- Why evaluate?
  - Validating abstractions
  - Checking technical correctness
  - Selling the architecture
  - Explaining the architecture
  - Validating assumptions
  - Providing management decision points
  - Offering a basis for formal agreement
  - Ensuring technical integrity



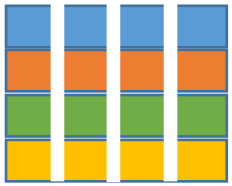
# How to evaluate?

- Presentations
- Formal reviews and walkthroughs:
  - Moderator
  - Presenter
  - Reviewers
- Scenario-based evaluation (SAAM + ATAM)
  - Architecture-oriented
  - Stakeholder-oriented
- Building design prototypes
  - Throw away prototypes for specific test aspects and proof-of-concept
  - Skeleton system for evolutionary prototyping



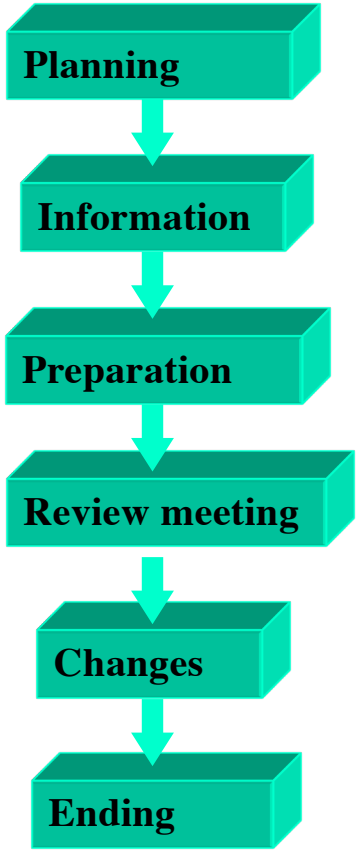
# Review

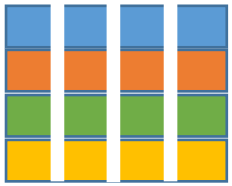
- Meeting to review the quality of a product
- Purpose
  - To identify improvements
  - To ensure consistent quality
  - To acknowledge the product
  - To train
- The reviewers' task is to identify problems, not solve them



# Review

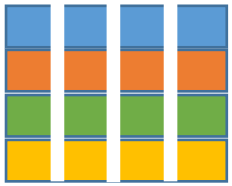
- Well defined process
  - Phases
  - Process descriptions
- Well defined roles
  - Moderator, reviewer, presenter, secretary
- Well defined goals
- Well defined measurements



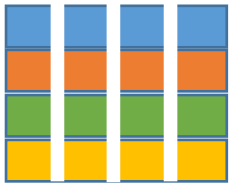
# Review experiences

- Use ”grading” scale
  - Critical, serious, moderat, minor
- Søren T. Lyngsø/ DENMARK
  - One hour per page (in total, i.e. including preparation and planning)
  - Review 7-9 pages/hour
  - Review meetings in the morning are most effective



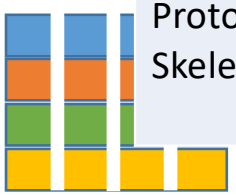
# Scenario-based evaluation (SAAM + ATAM)

- SAAM (original) - Software Architecture Assessment Method
- ATAM (more sophisticated) - Architecture Tradeoff Analysis Method
  - Understand requirements
  - Understand proposed architecture
  - Identify prioritized scenarios
  - Analyze the architecture
  - Draw conclusions
- Based on system usage scenarios identified/prioritized by stakeholders



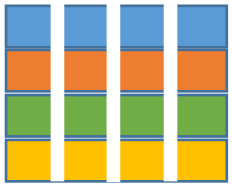
# Evaluation techniques

| Technique | Advantages | Limitations |
|---|---|---|
| Presentation | Quick to create; easy to tailor<br>Cheap and easy to do<br>Immediate feedback | Shallow level of analysis<br>Heavily dependent of engagement and commitment of participants |
| Formal reviews and walkthroughs | Involves participants more deeply | Requires a substantial amount of preparation |
| Scenarios | Can provide a deep and sophisticated analysis<br>More explicit understanding of tradeoffs | More complex and expensive<br>Training and significant preparation is needed for instructors |
| Prototypes + proof-of-concept<br>Skeleton system | Concrete evaluation<br>Provide useful demonstrations to stakeholders | Quite expensive or time-consuming to develop<br>Risk end up being hijacked (P+PoP) |



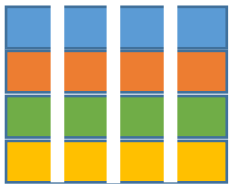
# Evaluation techniques - exercise

- Explain figure 14-3  (p. 231) to your neighbour



# Software Systems Architecture

- Part I: fundamentals (ch. 2-5)
- Part II: the process of software architecture (ch. 6-14)
- Part III: the viewpoint catalog (ch. 15-23)
- Part IV: the perspective catalog (ch. 24-29)
- Part V: putting it all together (ch. 30)



# Viewpoint catalog

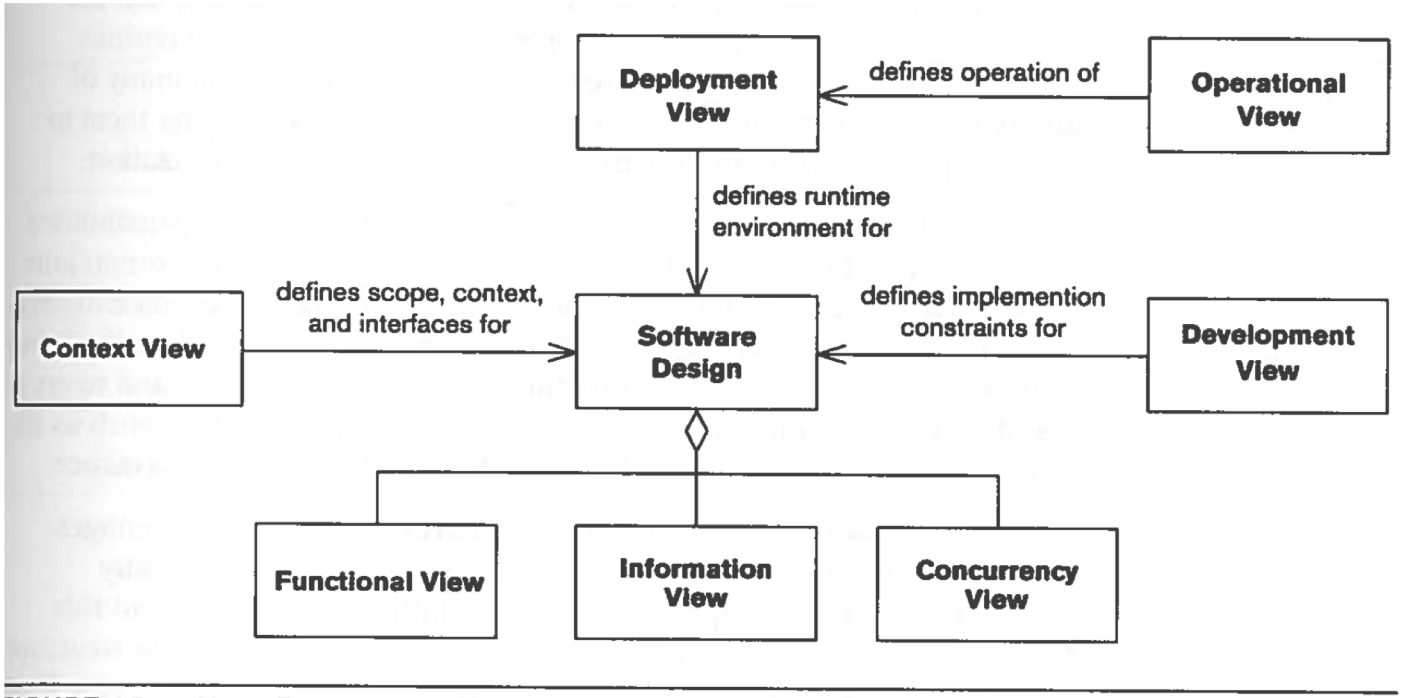
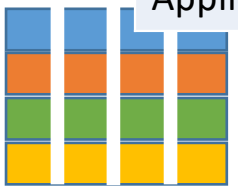



FIGURE 15–1 VIEW RELATIONSHIPS

# Context viewpoint

defines what the system does/does not do; boundaries between system and the outside world; how the system interacts with other systems, organizations, and people

| **Definition** | **Describes the relationships, dependencies and interactions between the system and its environment (the people, systems and external entities that it interacts with)** |
|---|---|
| Concerns | System scope and responsibilities, system quality objectives, identity of external entities and services and data used, nature and characteristics of external entities, identity and responsibilities of external interfaces, nature and characteristics of external interfaces, other external interdependencies, impact of the system on its environment, overall completeness consistency and coherence |
| Models | Context model, scope definition, interaction scenarios |
| Problems and pitfalls | Missing or incorrect context model elements, uneven focus, inappropriate level of detail, scope creep, implicit or assumed scope or requirements, missing implicit dependencies, loose or inaccurate interface descriptions, overcomplicated interactions, overuse of jargon |
| Applicability | All systems |



# Context diagram

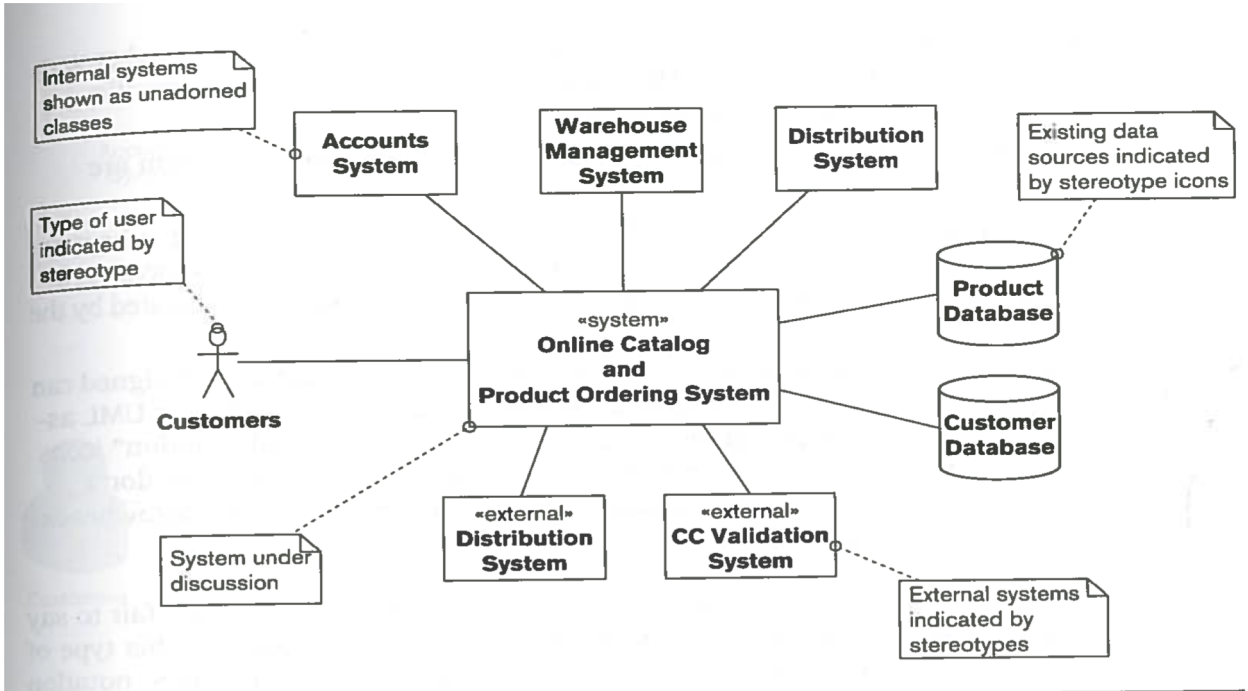
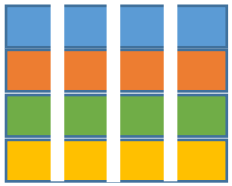



FIGURE 16–2 UML CONTEXT DIAGRAM

# Functional viewpoint

documents the system's functional structure—including the key functional elements, their responsibilities, the interfaces they expose, and the interactions between them

| Definition | Describes the system's runtime functional elements and their responsibilities, interfaces, and primary interactions |
|---|---|
| Concerns | Functional capabilities, external interfaces, internal structure, and design philosophy |
| Models | Functional structure model (e.g. component diagram) |
| Problems and pitfalls | Poorly defined interfaces, poorly understood responsibilities, infrastructure modeled as functional elements, overloaded view, diagrams without element definitions, difficulty in reconciling the needs of multiple stakeholders, wrong level of detail, "God elements," and too many dependencies |
| Applicability | All systems |



# Functional structure model/ UML component diagram

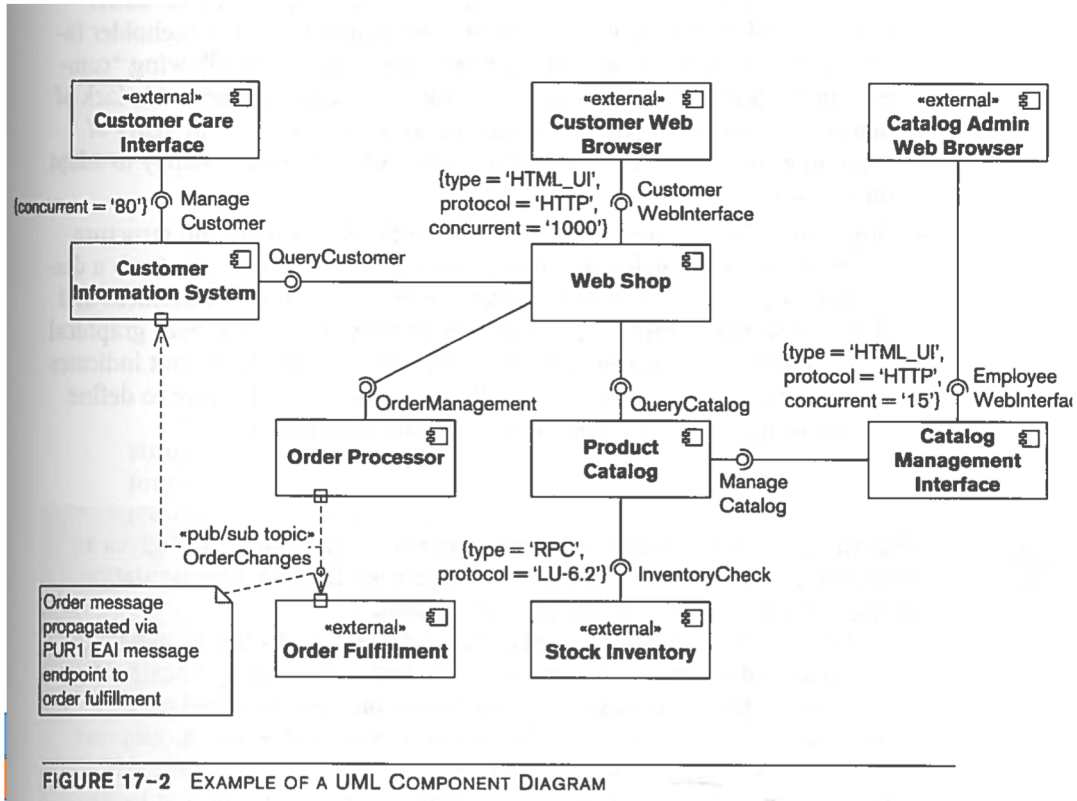



FIGURE 17–2 EXAMPLE OF A UML COMPONENT DIAGRAM

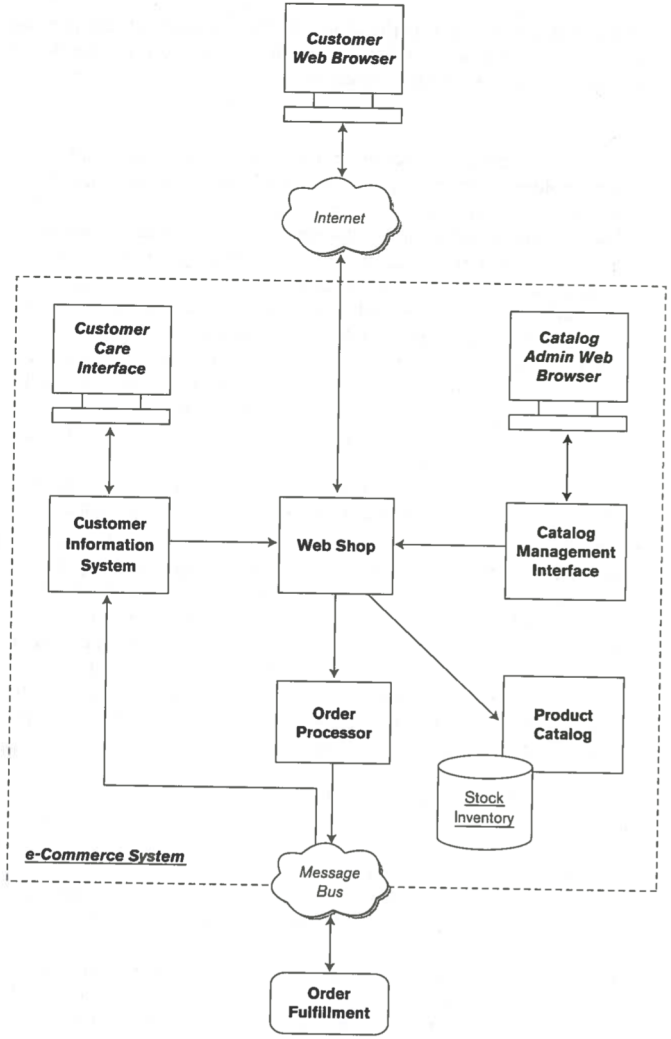



FIGURE 17–3 EXAMPLE OF A BOXES-AND-LINES DIAGRAM

# Information viewpoint

…..to answer questions about how your system will store, manipulate, manage, and distribute information

| Definition | Describes the way that the architecture stores, manipulates, manages, and distributes information |
|---|---|
| Concerns | Information structure and content; information purpose and usage; information ownership; enterprise-owned information; identifiers and mappings; transaction management and recovery; volatility of data semantics; information storage models; information flow; information consistency; information quality; timeliness, latency, and age; and archiving and information retention |
| Models | Static information structure models, information flow models, information lifecycle models, information ownership models, information quality analysis, metadata models, and volumetric models |
| Problems and pitfalls | Representation incompatibilities, unavoidable multiple updaters, key-matching deficiencies, interface complexity, overloaded central database, inconsistent distributed databases, poor information quality, excessive information latency, and inadequate volumetrics |
| Applicability | Any system that has more than trivial information management needs |

# ER diagram/UML class model

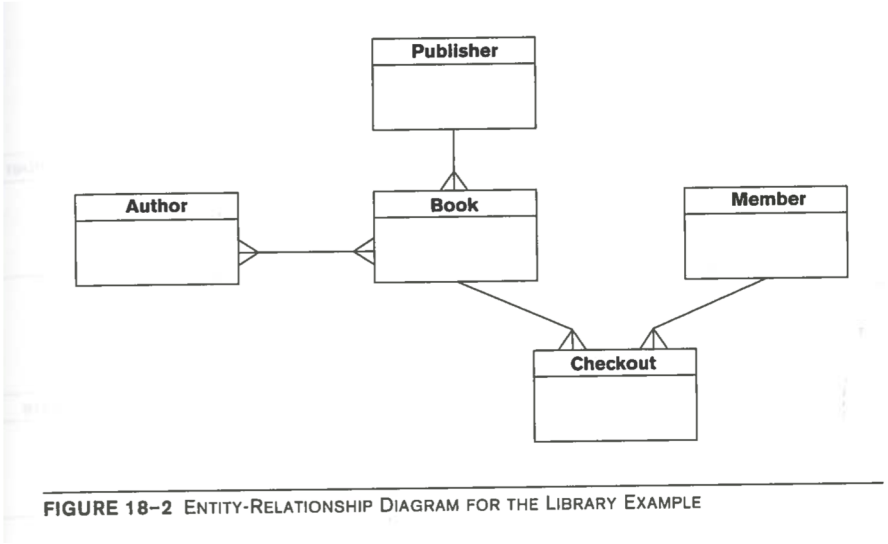



FIGURE 18–2 ENTITY-RELATIONSHIP DIAGRAM FOR THE LIBRARY EXAMPLE

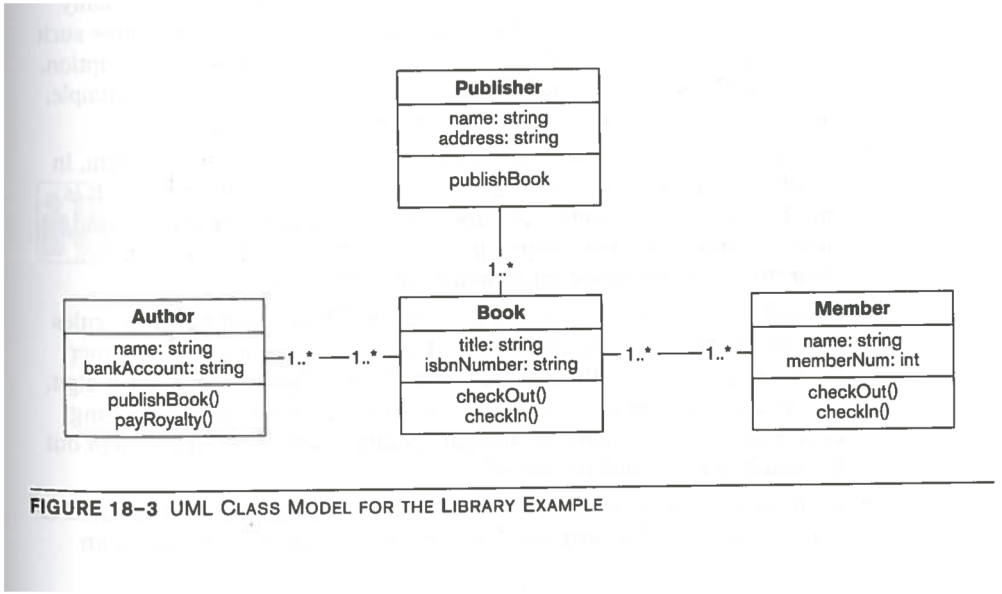
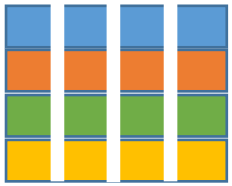



FIGURE 18–3 UML CLASS MODEL FOR THE LIBRARY EXAMPLE



# Life cycle model

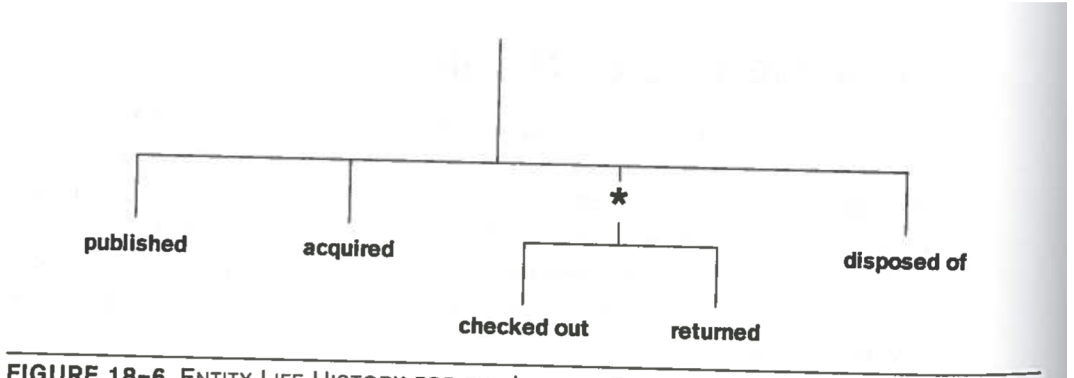



FIGURE 18–6 ENTITY LIFE HISTORY FOR THE LIBRARY EXAMPLE

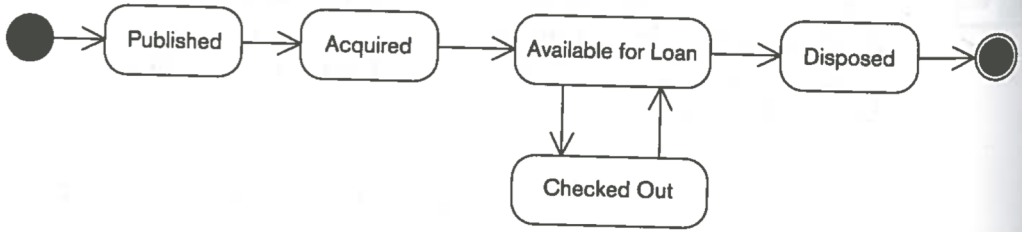
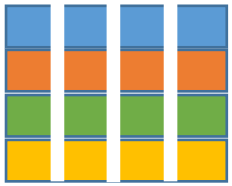


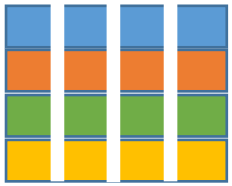

FIGURE 18–7 UML STATE DIAGRAM FOR A BOOK IN THE LIBRARY EXAMPLE



# Concurrency viewpoint

defining the parts of the system that can run at the same time and how this is to be controlled

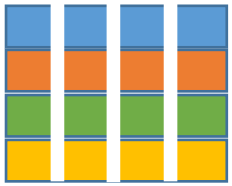
| **Definition** | **Describes the concurrency structure of the system, mapping functional elements to concurrency units to clearly identify the parts of the system that can execute concurrently, and shows how this is coordinated and controlled** |
|---|---|
| Concerns | Task structure, mapping of functional elements to tasks, interprocess communication, state management, synchronization and integrity, startup and shutdown, task failure, and reentrancy |
| Models | System-level concurrency models and state models |
| Problems and pitfalls | Modeling of the wrong concurrency, excessive complexity, resource contention, deadlock, and race conditions |
| Applicability | All information systems with a number of concurrent threads of execution |



# Development viewpoint

….code structure and dependencies, build and configuration management of deliverables, system-wide design constraints, and system-wide standards to ensure technical integrity

| Definition | Describes the architecture that supports the software development process |
|---|---|
| Concerns | Module organization, common processing, standardization of design, standardization of testing, instrumentation, and codeline organization |
| Models | Module structure models, common design models, and codeline models |
| Problems and pitfalls | Too much detail, overburdening the AD, uneven focus, lack of developer focus, lack of precision, and problems with the specified environment |
| Applicability | All systems with significant software development involved in their creation |



# Module structure model

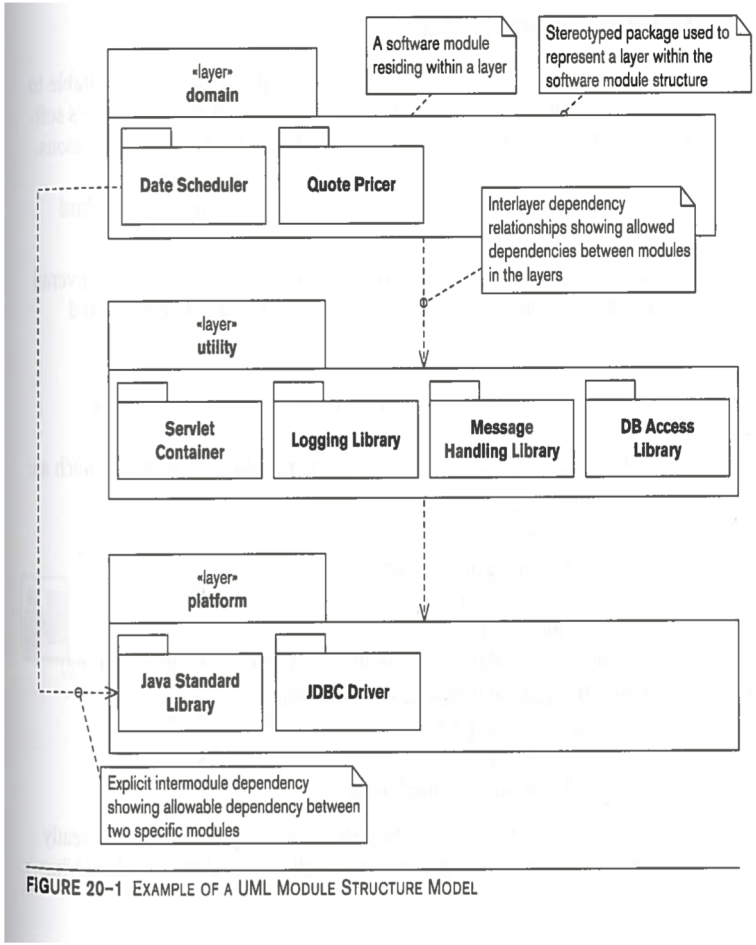
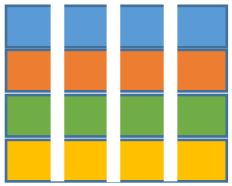



FIGURE 20–1 EXAMPLE OF A UML MODULE STRUCTURE MODEL

# Deployment viewpoint

….defines the physical environment in which the system is intended to run, including the hardware environment your system needs

| Definition | Describes the environment into which the system will be deployed, including the dependencies the system has on its runtime environment |
|---|---|
| Concerns | Types of hardware required, specification and quantity of hardware required, third-party software requirements, technology compatibility, network requirements, network capacity required, and physical constraints |
| Models | Runtime platform models, network models, and technology dependency models |
| Problems and pitfalls | Unclear or inaccurate dependencies, unproven technology, lack of specialist technical knowledge, late consideration of the deployment environment, inappropriate headroom and not specifying a disaster recovery environment |
| Applicability | Systems with complex or unfamiliar deployment environments |



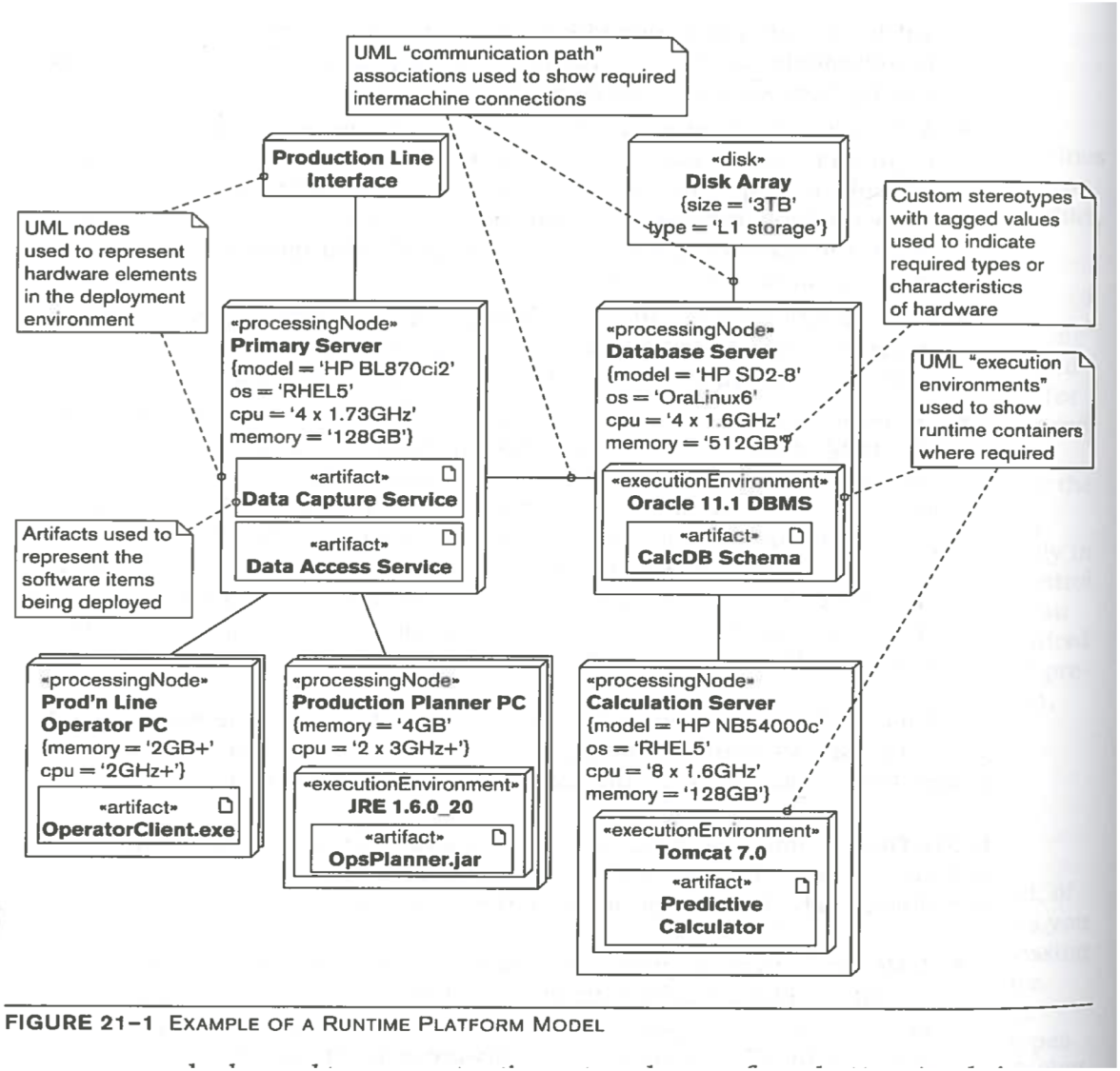
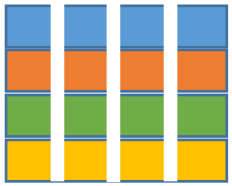



**FIGURE 21–1** EXAMPLE OF A RUNTIME PLATFORM MODEL

# Operational viewpoint

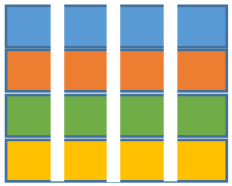
identify a system-wide strategy for addressing the operational concerns of the system's stakeholders and to identify solutions that address these

| Definition | Describes how the system will be operated, administered, and supported when it is running in its production environment |
|---|---|
| Concerns | Installation and upgrade, functional migration, data migration, operational monitoring and control, alerting, configuration management, performance monitoring, support, and backup and restore |
| Models | Installation models, migration models, configuration management models, administration models, and support models |
| Problems and pitfalls | Lack of engagement with the operational staff, lack of backout planning, lack of migration planning, insufficient migration window, missing management tools, lack of integration into the production environment, inadequate backup models, and |
| Applicability | Any system being deployed into a complex or critical operational environment |



# Explain Table 3-2 (p. 42) to you neighbour

| Views | OLTP | Calculation service | DSS/MIS | High-volume web site | Enterprise package |
|---|---|---|---|---|---|
| context | +++ | + | +++ | ++ | ++ |
| functional | +++ | +++ | + | +++ | +++ |
| information | ++ | + | +++ | ++ | ++ |
| concurrency | + | +++ | + | ++ | (+++) |
| development | +++ | +++ | + | ++ | (+++) |
| deployment | +++ | +++ | +++ | +++ | +++ |
| operational | (+++) | + | ++ | ++ | +++ |



# Dependencies

….how do we ensure consistency?

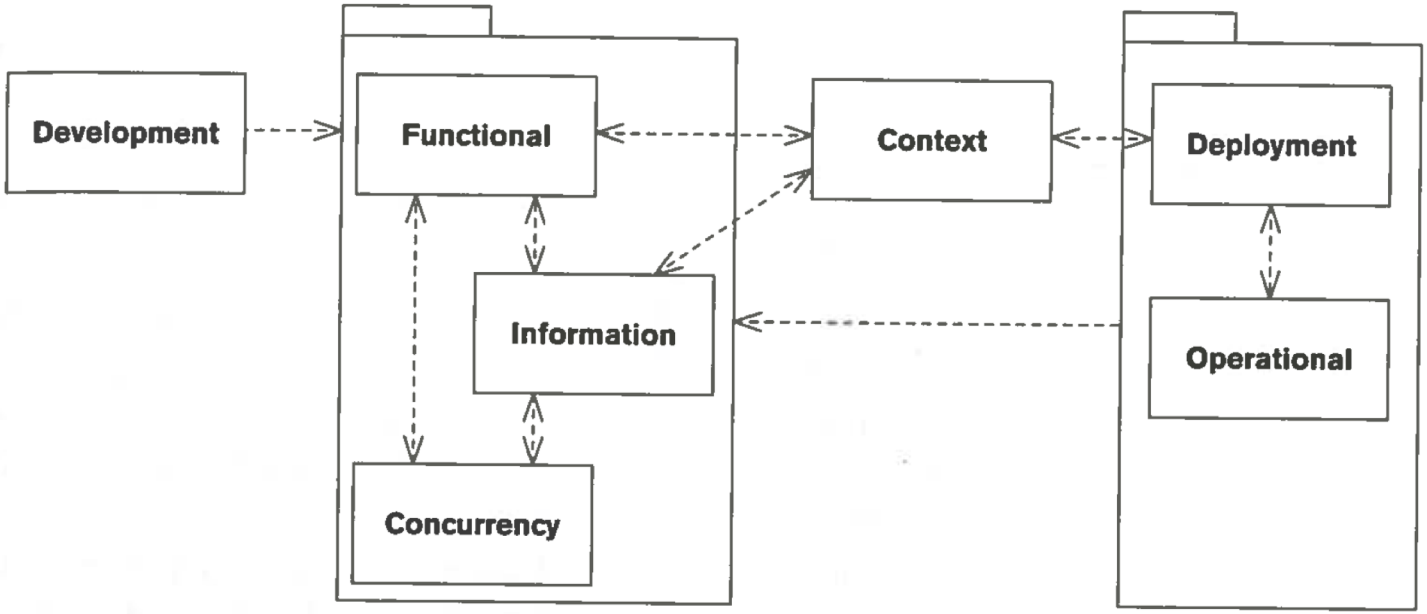
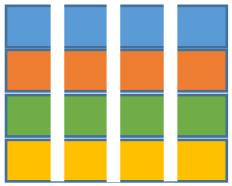



FIGURE 23–1 DEPENDENCIES BETWEEN VIEWS

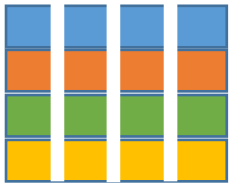
# Viewpoints – exercise (1)

- Using the CUTNMOVE context diagram as your outset
- Identify the entities in relation to ”booking” and ”paying”
- Establish the relations between the entities



# Viewpoints – exercise (2)

- Using the CUTNMOVE context diagram as your outset
- Develop a component diagram in relation to ”paying”



Evaluating the architecture

The viewpoints of architecture

Dependencies between views
-how to ensure consistency?