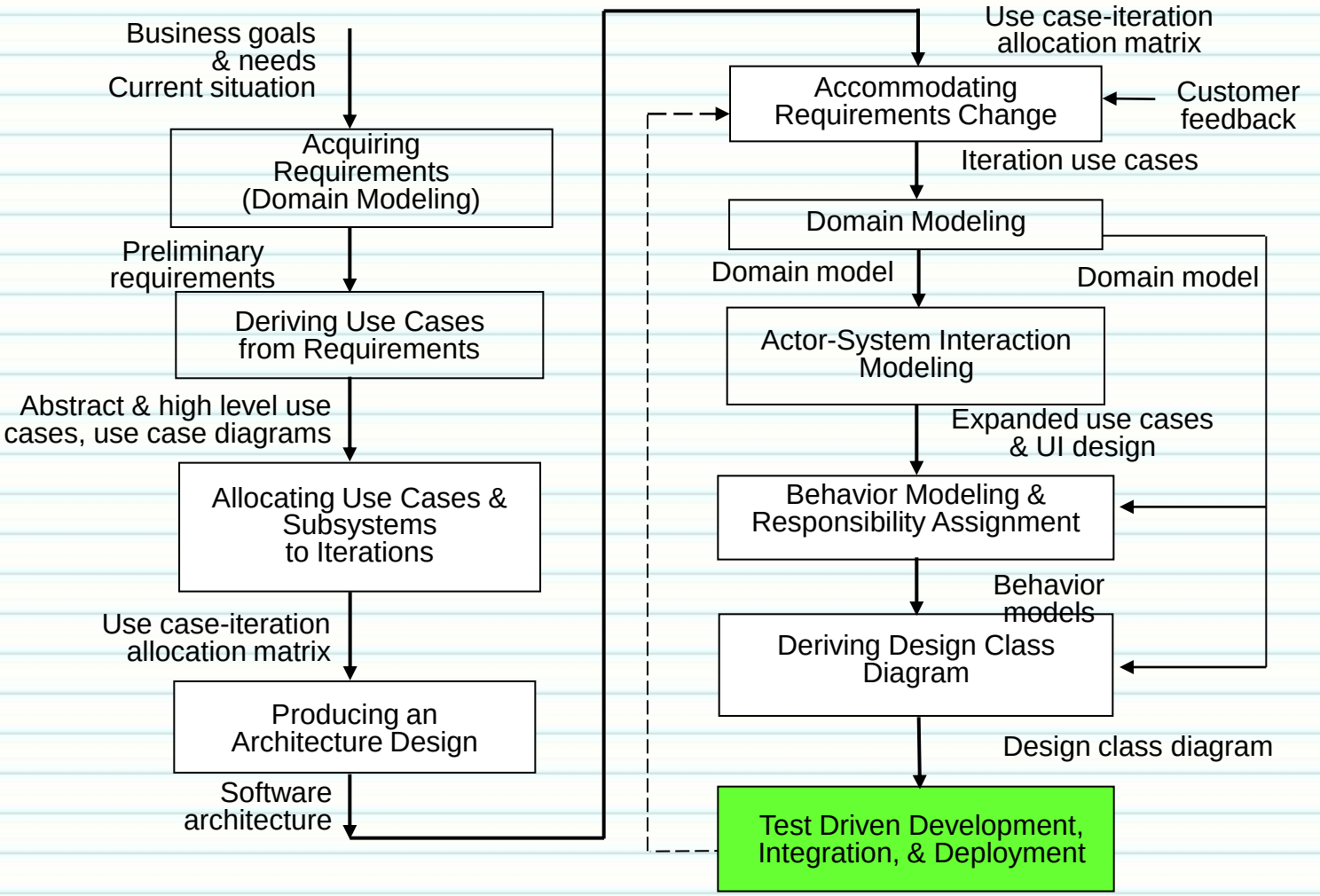
Chapter 18 – Implementation Considerations

Dr John H Robb, PMP
UT Arlington
Computer Science and Engineering

Key Takeaway Points

- Everyone in the team should follow the same coding standards.
- Test-driven development, pair programming, and code review improve the quality of the code.
- Classes should be implemented according to their dependencies to reduce the need for test stubs.

Implementation in the Methodology Context



Coding Standards

- Define the required and optional items.
- Define the format and language.
- Define the coding requirements and conventions.
- Define the rules and responsibilities to create and implement the coding standards, as well as review and improve the practice.

Components of Coding Standards?

- *File header* file location, version number, author, project, update history.
- *Description of classes* – a functional description for each class including
  - purpose
  - description of methods
  - description of fields
  - in-code comments
- Conventions

Programming Standard Considerations

- Test is the most effective and efficient when certain test issues are considered during the design of the software
  - Limit (or restrict) the use of **user defined** exception handling.
    - Important because it can be over-used when proper guidance is not provided. **User defined e**xception handling should not be used to address normal case conditions (e.g., a decision statement). Its use should be very sparse or not all at.
    - Generation of exception conditions during test can range from difficult to impossible.
  - Limit and prune the use of deep inheritance structures
    - Deep inheritance structures can suffer from the yo-yo problem
    - Deep inheritance structures can make test very complicated – sometimes requiring the entire class hierarchy to be utilized just to perform a simple unit test
  - Be careful when utilizing polymorphism
    - Static vs. dynamic binding (generics)
    - Dynamic binding can be very difficult to test
  - Encapsulation/information hiding is a great design technique
    - But it may be worth considering adding extra methods exclusively for testing just to provide additional visibility into the internals of a class

Design Considerations for Test (cont.)

  - Consideration should be given for creating limits on cyclomatic complexity (to reduce test size)
  - Consideration should be given on essential complexity (restricting unstructured code)
  - Consideration should be given on restricting use of bit wise operations in logical expressions (use short-circuiting instead)
- These items should be addressed in a coding standards and automatically checked by static code analyzers as an entry criteria for a technical review (e.g., peer review, etc.)

Coding Conventions

- *Naming conventions* specify rules for naming packages, modules, paths, files, classes, attributes, functions, constants, and the like.
- *Formatting conventions* specify formatting rules for line breaking, indentation, alignment, and spacing.
- *In-code comment conventions* define the requirements and guidelines for writing in-code documentation.

Good Guidelines for Programming Standards documents

- Define barely enough coding standards.
- The coding standards should be easy to understand and practice.
- The coding standards should be documented and include examples.
- Training on how to use the coding standards is helpful.
- Automate as much as possible

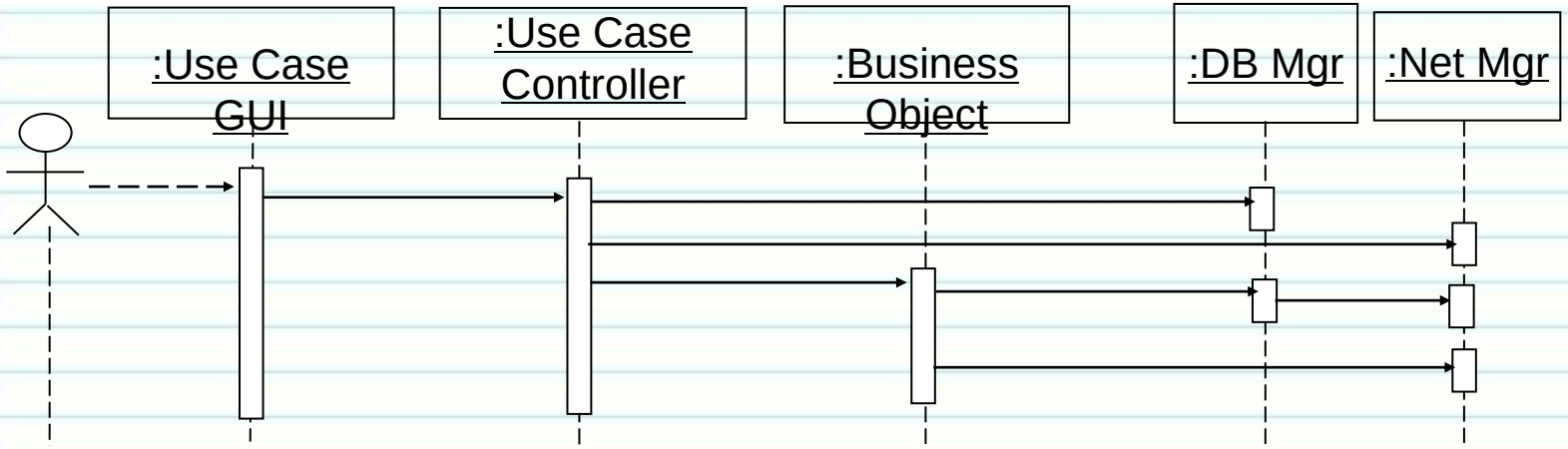
Guidelines for Practicing Coding Standards

- Define **barely enough** coding standards.
- The coding standards should be easy to understand and practice.
- The coding standards should be documented and include examples.
- Training on how to use the coding standards is helpful.
- The coding standards, once defined and published, should be practiced, enforced, and checked for compliance regularly.
- It is important to assign the responsibilities to individuals and make sure that all involved know the assignment.
- The practice of coding standards should involve stakeholders.

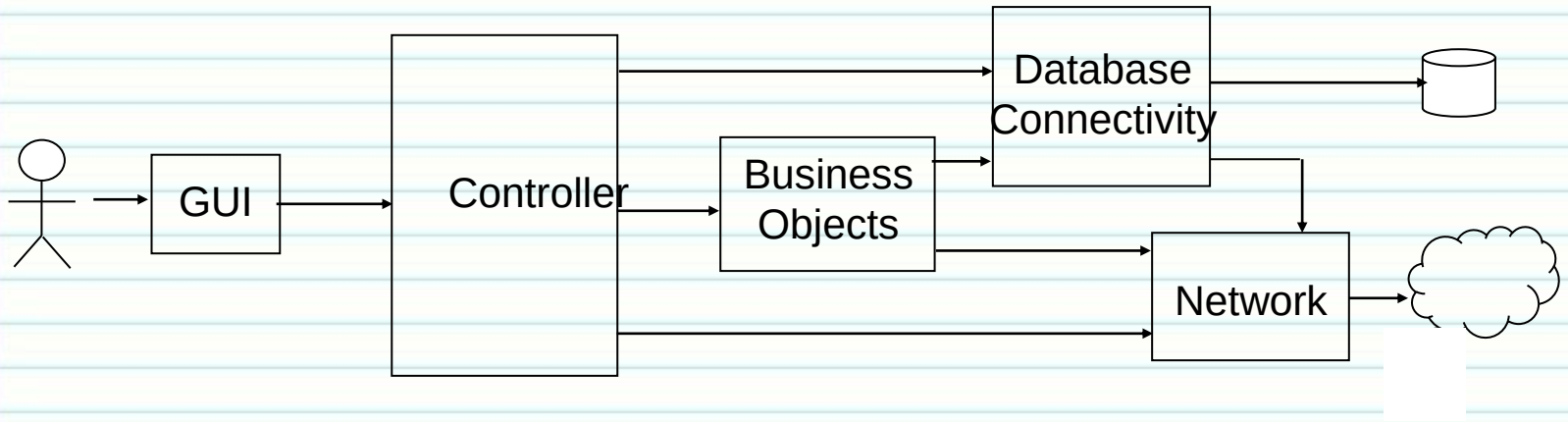
Organizing the Implementation Artifacts

- *Architectural-style organization:* Classes are organized according to an architectural style.
- *Functional subsystem organization:* Classes are organized according to the functional subsystems of the software system.
- *Hybrid organizations:*
  - architectural-style functional subsystem organization
  - functional subsystem architectural-style organization

N-Tier Architecture

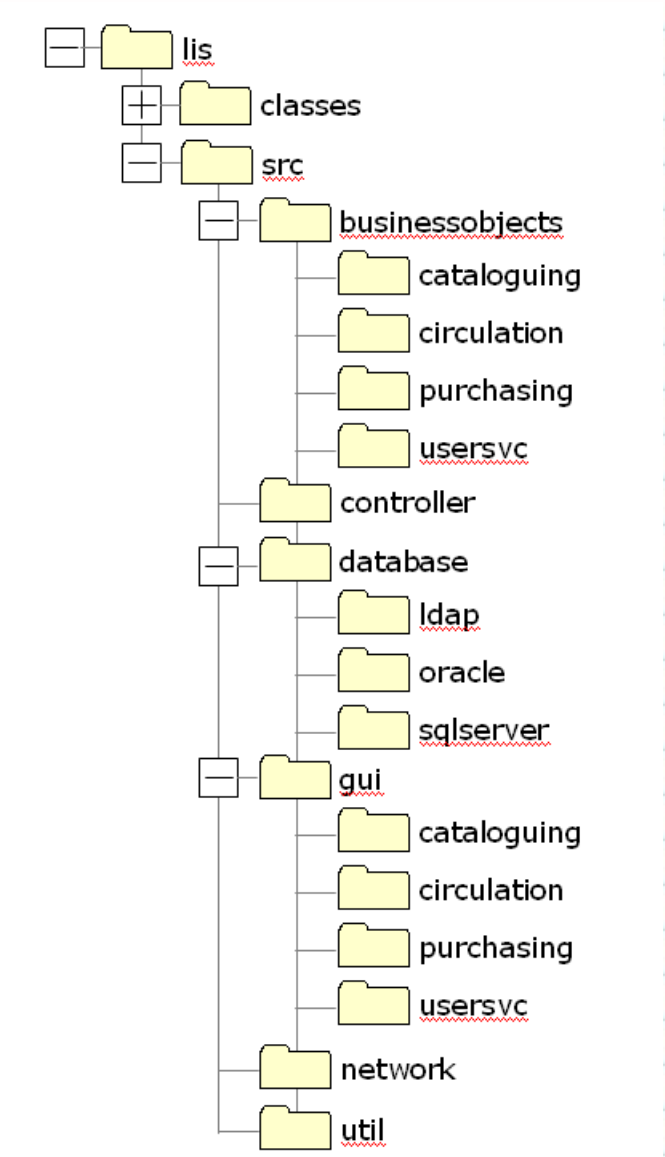


(a) Sequence diagram showing layers of a system



(b) Corresponding N-Tier architecture

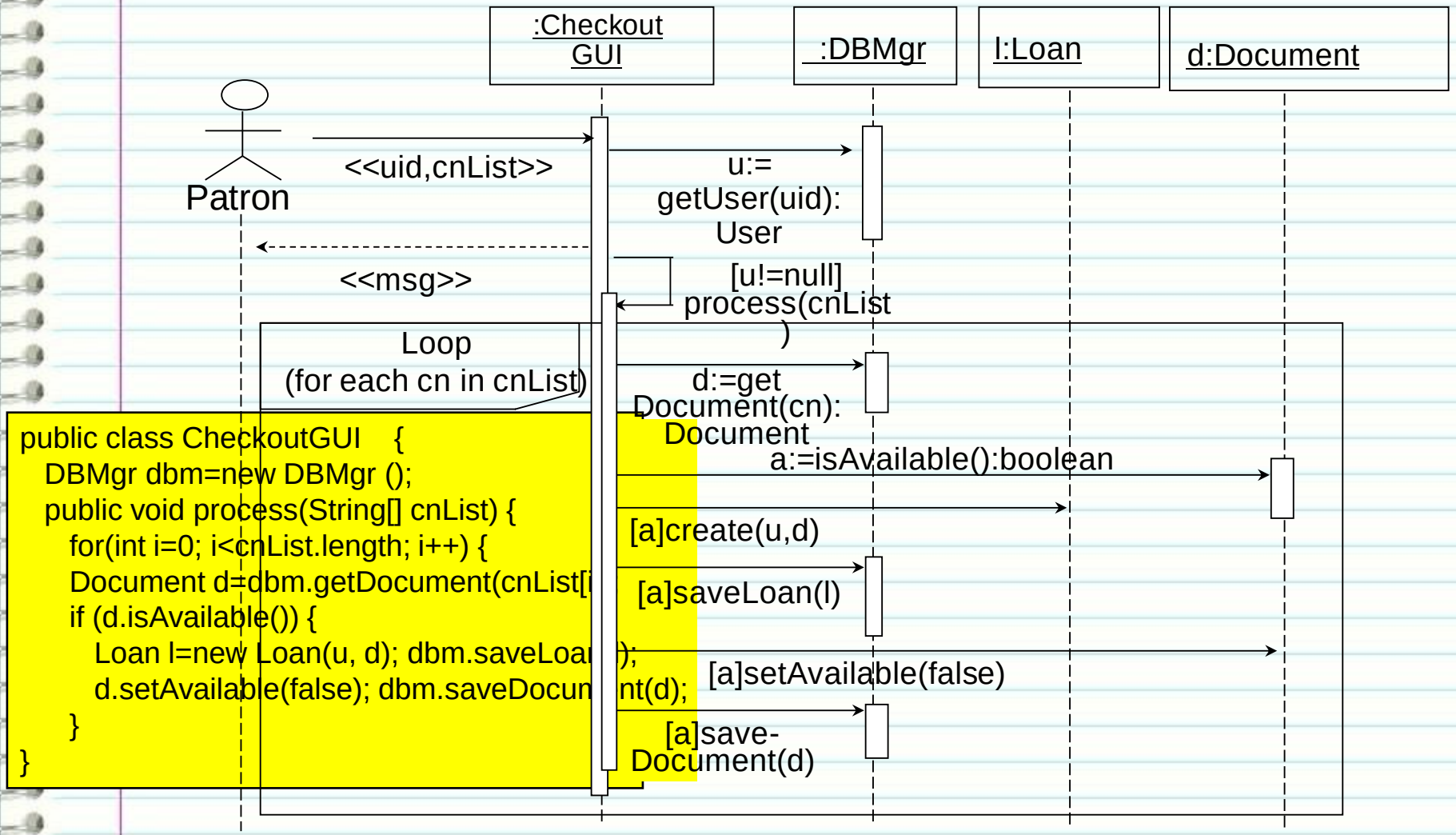
Corresponding Directory Structure



Implementation

- Assign classes to team members according to
  - dependencies between classes, and
  - abilities of the team members
  - to reduce number of test stubs
- Implement classes
  - test driven development
  - pair programming
  - deriving method skeleton from sequence diagrams
- Implement association relationships

From Sequence Diagram to Implementation



Implementing Association Relationships

- A one-to-one association between class A and class B is implemented by
  - A holding a reference to B if A calls a function of B, and/or
  - B holding a reference to A if B calls a function of A.
- A one-to-many association between is implemented by
  - A holding a collection of references to B if A calls functions of B instances, or
  - B holding a reference to A if instances of B call a function of A.
- A many-to-many association is implemented by a collection of references from A to B, and vice versa.

Pair Programming

- Two developers program at one machine simultaneously.
- They discuss how to implement the features.
- The one with the keyboard and mouse implements the functionality, the other reviews the program as it is being typed.
- The developers switch roles periodically, whenever they like, or between programming sessions.
- Each session lasts one to three hours.
- Pairs are not fixed, they switch around all the time.
- All team members are required to work with others. If two people cannot work together, they don't have to pair with each other.

Merits of Pair Programming

- It reduces pressure and brings fun to programming because there is always someone with whom to share the stress and joy of success.
- It enhances team communication because the partners exchange ideas during pair programming.
- Pair-switching helps team members gain insight into the components of the system. This improves productivity and quality.
- It enhances mutual understanding and collaboration because pair programming lets the team members understand each other and learn from each other in various ways and aspects.
- It tends to produce simpler and more efficient solutions faster because discussions stimulate creative thinking and help in problem solving.

Pair Programming Limitations

- It is not for everyone—there are programmers who prefer to work alone.
- Discussions between the partners can take a lot of time. Therefore, it should focus on solving the problem and getting
- the work done.
- It could be slow to start due to the need to adapt to the differences of the partners.
- Other limitations:
  - the partners have to be at the same location
  - it may be difficult for the partners to find a meeting time due to conflicting schedules, and
  - it might not be as effective as systematic review methods in detecting defects.