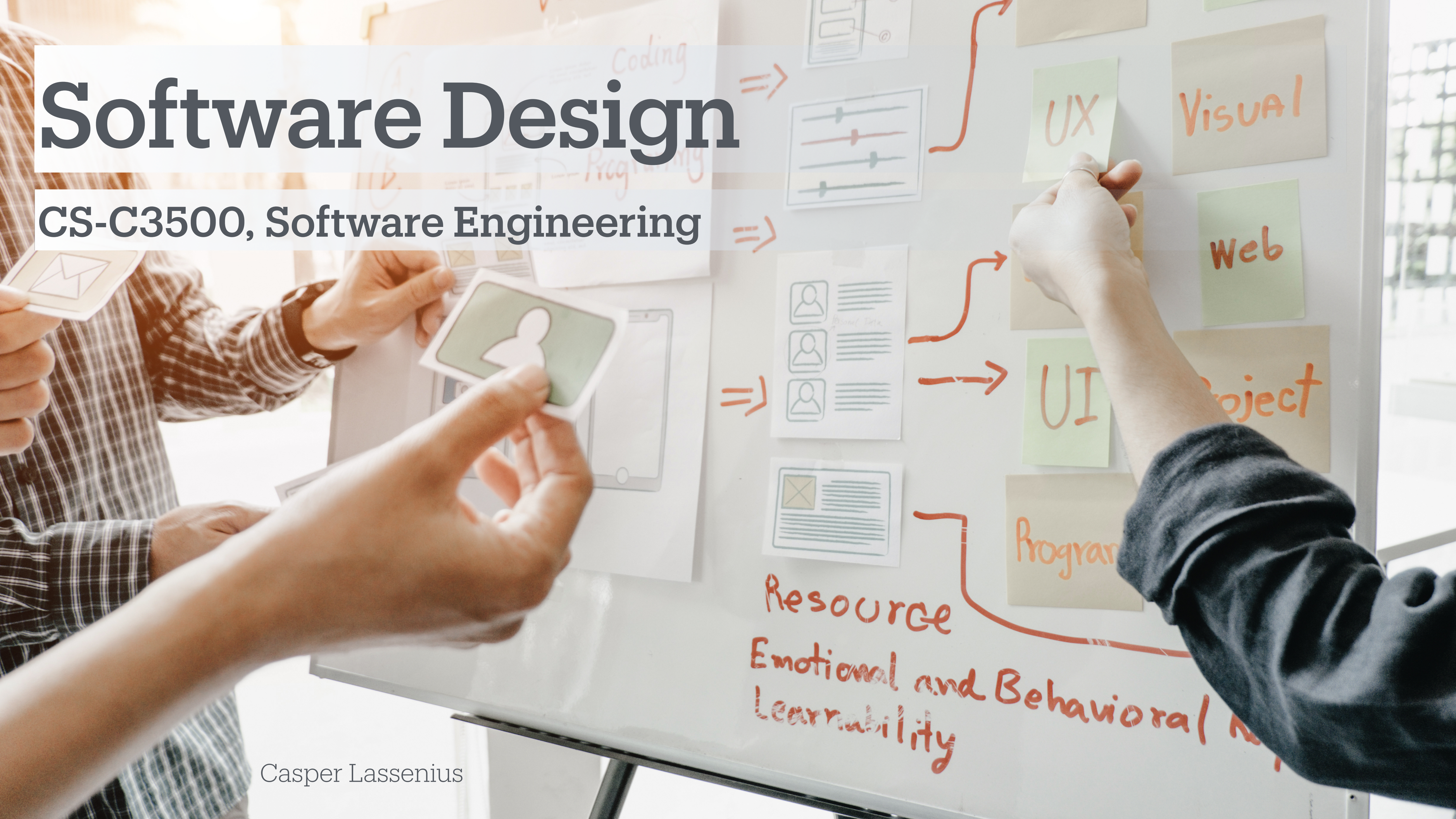


Software Design

CS-C3500, Software Engineering



Agenda

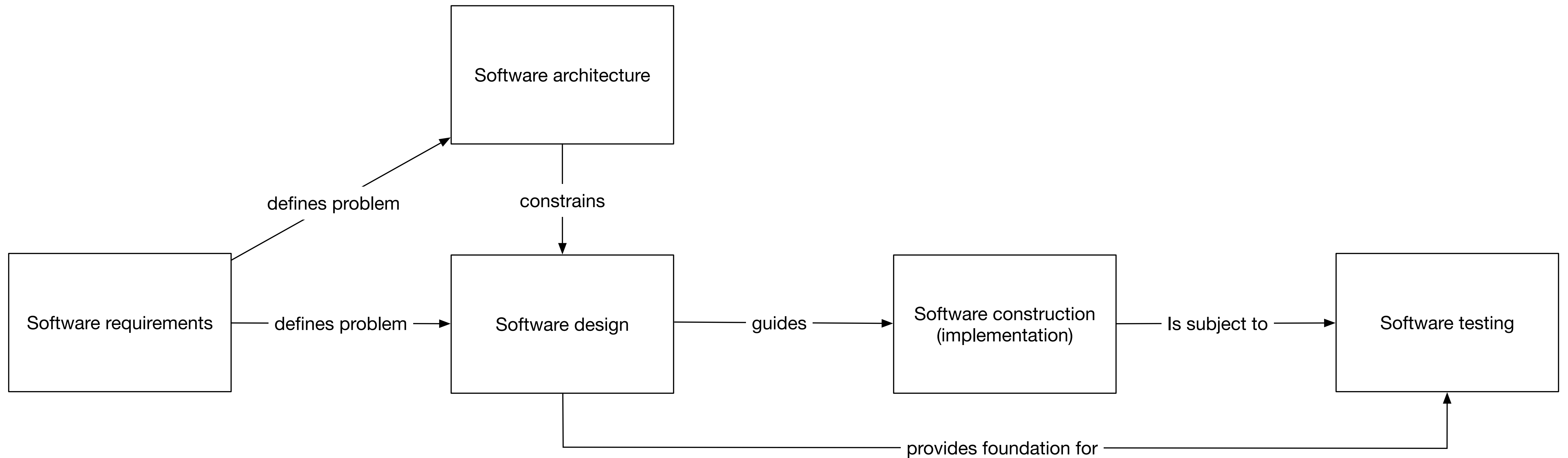
- Key issues
- Principles
- Stages
- Outputs and Notations
- Patterns
- Strategies
- Evaluation

AGENDA



“The application of software engineering discipline in which software requirements are analyzed to define the software’s external characteristics and internal structure as the basis for software construction”

Software Design Context



Key Issues

- Basic concerns
 - How to define, refine, organize, interconnect, and package software components
- Quality concerns
 - Performance
 - Security
 - Reliability
 - Usability
 - Maintainability
- Other issues, “aspects”, i.e., cross-cutting concerns, such as logging and persistence



Principles (1/2)

- Abstraction
- Separation of Concerns
- Modularization (refinement, decomposition)
- Encapsulation
- Separation of interface and implementation



Principles (2/2)

- Coupling
- Cohesion
- Uniformity
- Completeness (sufficiency)
- Verifiability
- Ethically aligned design



Design Levels

- Architectural design
 - Fundamentals of the system as a whole
- High-level (external facing) design of the system and its components
 - Top-level structure and organization
 - Components and their interactions
- Detailed (internal facing) design
 - Internals of each component to facilitate implementation



Software Design Outputs

- Include
 - Aspects of the problems to be solved
 - Solution vocabulary
 - Major design decisions
 - Rationale
- Result: “design description”, “design specification”
 - Texts, diagrams, models, prototypes
 - Design can evolve during development, in particular when using agile methods
 - Many notations exist to represent design artifacts, several are typically used, e.g., structural and behavioral descriptions



Examples of Notations

- **Structural**

- *Class diagrams*
- Component diagrams
- Deployment diagrams
- Entity relationship diagrams (ERD)
- Interface description languages
- Structure charts

- **Behavioral**

- Activity Diagrams
- *Interaction diagrams*
- Data flow diagrams (DFDs)
- Flowcharts
- State diagrams, *state charts*
- Formal specification languages
- Pseudocode

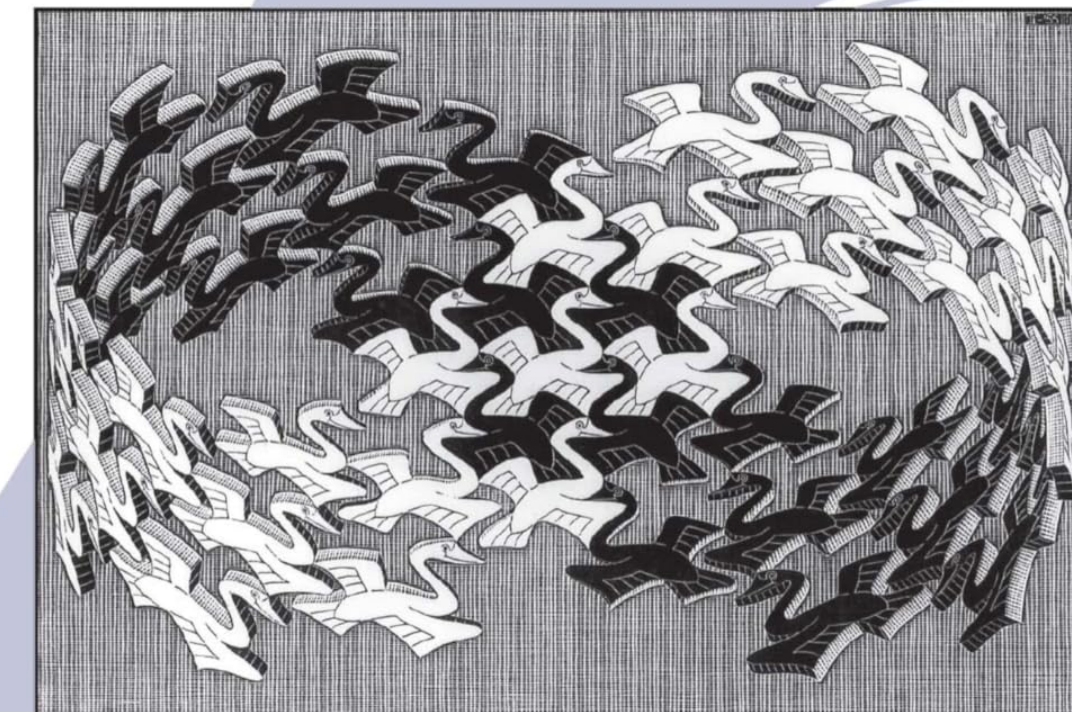
Design Pattern

- “A common solution to a common problem in a given context”
- Examples
 - Model-View-Controller
 - Factory
 - Decorator, Facade, ...
 - Publish-Subscribe

Design Patterns

Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides



Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.

Foreword by Grady Booch

Design Strategies

- General
 - Top-down, bottom-up
- Function-oriented (structured)
- Data-centered
- Object-oriented
- User-centered
- Component-based
- Aspect-oriented



Evaluating the Design

- Quality attributes, “ilities” and “nesses”
 - Modularity, maintainability, portability, testability, usability
 - Correctness, robustness
- Techniques
 - Reviews
 - Static analysis
 - Simulation and prototyping



Uses of Software Design

- Communication
- Enforcement
- Validation
- Work breakdown / Project planning and control



Challenges

- Balance design value with costs
- Keeping the models up-to-date
- Learnability / Understandability
- Integrating design with modern life-cycle approaches
- Tool support





“Use the Source, Luke”

Yoda, senior software engineer

10%

Of coders report using UML systematically and consistently

Summary

- Software design is the bridge between requirements and the implementation
- Design supports implementation, work breakdown, and validation
- Design is concerned with key issues and
- There are many different notations, and the domain, specifics of the software, and the use of the model determines what notation is most suitable
- Typical design stages include architectural, high-level, and low-level design