CS189A - Capstone

Christopher Kruegel
Department of Computer Science
UC Santa Barbara
http://www.cs.ucsb.edu/~chris/

Software Crisis

- Software's chronic crisis: Development of large software systems is a challenging task
  - Large software systems often: Do not provide the desired functionality; Take too long to build; Cost too much to build Require too much resources (time, space) to run; Cannot evolve to meet changing needs
- Software engineering focuses on addressing challenges that arise in development of large software systems using a systematic, disciplined, quantifiable approach
- There are essential difficulties in software development which makes it a hard task:
  - Complexity; Conformity; Changeability; Invisibility

Software Process Models

- Software life-cycle:
  - Requirements analysis and specification, design, implementation, testing and integration, maintenance

- Software process models
  - Waterfall: sequential, document driven
  - Evolutionary approaches: iterative and incremental software development
    - Spiral model
    - Sync-and-Stabilize
    - Scrum
    - Extreme programming
    - Agile software development

Software Requirements

IEEE Recommended Practice for Software Requirements Specifications

"Getting started: Using use cases to capture requirements," James Rumbaugh

- Desirable properties of requirements:
  - correct, unambiguous, complete, consistent, verifiable, modifiable, traceable
- Formal vs. informal specification
- Use cases, use case scenarios

Software Specification and Modeling

Unified Modeling Language (UML)

- Use case diagrams
- Class diagrams
- Sequence diagrams
- Collaborations diagrams
- Activity diagrams
- Statecharts and state diagrams

Principles of Software Engineering

The fundamental principles in software engineering are especially important during software design:

- Separation of Concerns
- Iterative (Stepwise) Refinement
- Abstraction
- Modularity
- Anticipation of Change

Modularization

"On the criteria to be used in decomposing systems into modules," Parnas
"Designing software for ease of extension and contraction," D.L. Parnas

- Basic principle for modularity: **Information hiding**
- Modularization with uses hierarchy

Design by Contract

"Applying Design by Contract," B. Meyer.

- Pre-conditions, post-conditions, class invariants
- Establishing pre-condition is the responsibility of the caller
- Establishing the post-condition is the responsibility of the callee
- Runtime contract monitoring

Design Patterns

- Design patterns provide a mechanism for expressing common object-oriented design structures
- Design patterns identify, name and abstract common themes in object-oriented design
- Design patterns can be considered micro architectures that contribute to overall system architecture
- Design patterns are helpful
  - In developing a design
  - In communicating the design
  - In understanding a design
- Patterns we discussed: Composite, Strategy, Decorator, Abstract Factory, Bridge, Iterator, Observer

Validation, Verification and Testing

- Reviews, walkthroughs, inspections
- Software testing:
  - black-box vs. white-box; functional vs. structural
  - random testing, exhaustive testing
  - domain testing, boundary conditions
  - coverage criteria: statement, branch & path coverage, condition coverage, multiple condition coverage
  - unit testing, stubs, drivers
  - integration& testing: top-down vs. bottom-up integration and testing
  - regression testing