CS189A
Software Engineering: Concepts and Practices

Tobias Höllerer
Department of Computer Science
UC Santa Barbara

https://capstone.cs.ucsb.edu/189a.html
Today’s Plan

❖ Announcements
  – CS189B approval codes,
  – Section
  – Upcoming deadlines

❖ Software engineering:
  Requirements specification PRDV2.0, design, testing
  – Modularization
  – UML
    o Overview Class (or System) Diagram
    o Sequence Diagrams

❖ Presentations and demos on Dec 5

❖ Team meetings
Overall Plan @Week 9

- Four 2-week sprints
  - Oct 11-25 (PRD v1 – tools, technologies, design, terminology)
  - Oct 25-Nov 8 (use cases/user studies, prototyping, PRD v1, PRD v2)
  - Nov 8-22 (design, prototyping, testing, PRD v2)
  - Nov 22-Dec 3 (prototype demo/pres. prep, prototyping and testing)

- Fall presentations and demos:
  Mon Dec 5, 3:30-5:30 (Broida 1640)

- Specify what the product will do
  - Vision statement
  - Product Requirements Document (PRD) (due Oct 30 and Dec. 2)
  - Design tools, brainstorming, coding (tests and implementation)

- Build and test an initial prototype
  - Typically teams iterate on these activities until they converge to a working prototype!
This Week’s Plan

- Team activities
  - Scrum: Sprint 4, PRDv2, prototyping and testing
- Section: TA meetings – Demo of SW Testing approach

Upcoming deadlines:
- Fri, Dec. 2: PRDv2 due
- Dec 4: Sprint 4 ends
- Monday, Dec 5: Fall presentations and demos
Sprint 4

- Finish PRD v2 (due Friday, Dec. 2), incl. remaining user stories w/ acceptance tests
- Demo prep and presentation practice
  - Demo presentations: last Friday before the finals week
    (8 mins max; check that your laptop works)
  - Final discussion section is for sprint 4 retro and planning
- Break into tasks
  - Estimate/discuss timings (1/2 day – 1 day each)
  - Demo/prototyping plan
  - Total up to have timing per story
- Planning Poker (choose / assign tasks until filled)
PRDv2 in Sprint 4

- Evolve/improve
  - architecture/system diagram
  - in depth writeup: problem, innovation, science, core technical advance; project specifics, team goals/objectives, background, & assumptions
  - Add links (to trello and to github commits) for completed tasks
- More detailed system diagram + detailed design
- 10 additional stories/use cases, 5+ additional implementations/tests
  - Final PRD: 20 stories/use cases, 10+ partially implemented with tests/git commits
- 3+ sequence diagrams
- 3+ UI interaction/sequence diagrams + mockups
- PRDv2 due Dec. 2
PRDv2: Your Living Requirements

Document: A Shared Google Doc

- Authors, Team, Project Title
- Intro: problem, innovation, science, core technical advance (3+ pages)
  - Define project specifics, team goals/objectives, background, and assumptions
- System architecture overview
  - High level diagram (1 page)
  - User interaction and design (1+ pages) – i.e. detailed design
- Requirements (functional and non-functional)
  - User stories or use cases (links) → 20+ for PRDv2 prioritized w/acc. tests
  - Prototyping code, tests, metrics (10+ user stories): github commits/issues
- System models (1+ pages)
  - Contexts, interactions, structural, behavioral (UML)
  - Use cases, sequencing, event response, system state, classes/objects
- Appendices - Technologies employed
Your Project Design: PRDv2

- Architecture (hardware/software)
  - Evolve your overview picture from PRDv1 to provide significantly more detail and any updates or changes

- Detailed design
  - UML diagrams of primary data structures that comprise the system architecture connected via their associations (if any)
    o Ensure that each "class" is balanced in terms of cohesion & coupling
    o Annotate with pre/post conditions when appropriate
  - Sequence diagrams
    o synchronous and asynchronous for key interactions between classes
      ▪ At least 3 different interactions
    o User interactions with the system
      ▪ At least 3 different interactions
      ▪ Can be a human user or a machine user (API) interaction
        ▪ Event response, updated application state
      ▪ If you have a user interface: Provide mockups for primary UIs
PRDv2 User Stories / Use Cases

- Revise spec to add detail to the functional specification to match your design
- Add user stories and break up the stories you have into finer grained stories
  - Provide UML, sequence diagrams, dataflow diagrams
  - Goal: a CS senior should be able to take your doc and implement the project
- For each fine-grained story, provide a description and acceptance test
  - Provide time estimates (1 person-hours) for each story implementation
    o Ensure you can finish the implementation in the time you have (this/next quarter)
  - Prioritize tasks to have a complete prototype by the end of this quarter
    o Focus on the externally facing interfaces, mock out what you cannot get to
  - Follow your testing strategy: write unit tests to implement tasks for your tasks
    o Document these tasks (autogen the documentation/usage)
  - Add trello/pivotal/github PJ task links (titles must match) to PRDv2 for each story
- Prototype designed mandatory tasks; add github commit ID/link to PRDv2
  - Github must have unit tests, documentation (for anything without unit tests), and prototyping implementations for each story in Sprint
- If you have a user interface
  - Provide mockups that are tied to the functionality described in 1+ components
Modularity: Reducing Design Complexity

- Modularity principle suggests dividing a complex system into simpler pieces, called modules
  - Possible: Module = function or Module = functions or Module = functions + data

- When we have a set of modules, we can use separation of concerns and work on each module separately
  - to improve
    - maintainability
    - reusability
    - productivity

- Modularity can also help us to create an abstraction of a module’s environment using interfaces of other modules
Two Modularization Strategies

- Both attempt to generate modules and manage dependencies: **low coupling** and **high cohesion**
  - **Coupling** is a measure of a module’s independence
    - The degree of dependency among modules (lower is better)
    - Minimize and localize change to one module v/s those that depend on it
  - **Cohesion** is a measure of the degree to which all elements of a module are directed toward a single task (how self contained are they?)
    - The internal glue that holds a module together (higher is better)

- Modularization techniques
  - Functional decomposition
  - Parnas’ modularization
    
    “On the Criteria to be Used in Decomposing Systems into Modules”, Parnas, 1972
Functional Decomposition

Functional decomposition – **focus = operations performed on data**

- Divide and conquer approach – modules are steps in the computation
- Use stepwise refinement
  1. Clearly state the intended function
  2. Divide the function to a set of sub-functions and re-express the intended function as an equivalent structure of properly connected sub-functions, each solving part of the problem
  3. Divide each sub-function far enough until the complexity of each sub-function is **manageable**
Functional Decomposition

- One way of achieving functional decomposition: Make each step in the computation a separate module
  - Draw a flowchart showing the steps of the computation and convert steps of the computation to modules
  - **Shortcoming**: Does not specify the granularity of each step

- Another way of achieving functional decomposition is to look at the data flow in the system
  - Represent the system as a set of processes that modify data. Each process takes some data as input and produces some data as output.
  - Each process becomes a module

- **Shortcoming**: Both of these approaches produce a network of modules, not a hierarchy
What About Data Structures?

- **Fred Brooks**: “Show me your code and conceal your data structures, and I shall continue to be mystified. Show me your data structures, and I won’t usually need your code; it’ll be obvious.”
  – Author of The Mythical Man Month and No Silver Bullet (IBMer, Turing Award Winner)

- **Eric Stevens Raymond**: “Smart data structures and dumb code works a lot better than the other way around.”
  – Open source evangelist and author of The Cathedral and the Bazaar and The New Hacker’s Dictionary

- Functional decomposition focuses on operations performed on data

- According to Brooks and Raymond data structures should come first

- Parnas’ modularization approach (from 1972!) focuses on data
Parnas’ Modularization

- Define your set of data structures
- For each data structure
  - Define the set of possible operations on it (as functions)
    - Encapsulate code and data
  - Make public the set of functions that other modules or users employ to interact with the data structure
    - Make everything else (code and data) private
- Make each data structure reusable and extensible (inheritance)
  - And customizable (polymorphism)
The Unified Modeling Language (UML)

- A tool for all phases of software development
  - Requirements specification, architectural design, detailed design & impl

- Many books on UML, some good ones are:
  - “UML Distilled,” Martin Fowler
  - “Using UML,” Perdita Stevens
  - “UML Explained,” Kendall Scott

- The Object Management Group (OMG, a computer industry consortium) defines the UML standard
  - The current UML language specification is available at:
    http://www.uml.org/

- Tools: http://www.visual-paradigm.com/solution/freeumldesigntool/
  - http://yuml.me (online tool)
UML Diagram Types

- Use case diagrams: interactions between a system and its external entities (actors) in terms of use cases
- **Class diagrams**: classes used in a system
- State machine diagrams: ways in which an object changes state; different states affect behaviors
- Activity diagrams: workflow or actions (or sequence of events) during program execution
- **Communication (collaboration) diagrams**: interactions among objects in a system, with an emphasis on what interactions occur
- **Sequence diagrams**: interactions among the objects in a system, but emphasize when interactions occur
UML Class Diagram

- Visual representation of the static structure, interrelationships, and composition of a particular system
- Most used UML diagram type

- Help simplify how objects in a system interact
- Facilitate translating a designed system into code prototypes
## Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Circle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute</td>
<td>itsRadius:double</td>
</tr>
<tr>
<td>operation()</td>
<td>itsCenter:Point</td>
</tr>
<tr>
<td></td>
<td>Area():double</td>
</tr>
<tr>
<td></td>
<td>Circumference():double</td>
</tr>
<tr>
<td></td>
<td>SetCenter(Point)</td>
</tr>
<tr>
<td></td>
<td>SetRadius(double)</td>
</tr>
</tbody>
</table>
## Access Modifiers

**Circle**

<table>
<thead>
<tr>
<th>Access Modifier</th>
<th>Method/Property</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Public</strong></td>
<td>+ itsRadius: Double</td>
</tr>
<tr>
<td></td>
<td>+ itsCenter: Point = (0, 0)</td>
</tr>
<tr>
<td><strong>Protected</strong></td>
<td>- Area(): Double</td>
</tr>
<tr>
<td></td>
<td>- Circumference(): Double</td>
</tr>
<tr>
<td><strong>Private</strong></td>
<td>+ SetCenter(Point)</td>
</tr>
<tr>
<td></td>
<td>+ SetRadius(double)</td>
</tr>
<tr>
<td><strong>Package</strong></td>
<td></td>
</tr>
</tbody>
</table>

**Access Level**

- **Public**: +
- **Protected**: #
- **Private**: -
- **Package**: ~
Class Interrelationships: Logical Connections

- **Association**
  - Passengers
  - Airplane

- **Directed Association**
  - Passengers
  - Airplane

- **Reflexive Association**
  - Airline Staff
  - 0..*

- **Multiplicity**
  - Passengers
  - Airplane

- **Aggregation**
  - Library
  - Books
  - 1..*

- **Composition**
  - Library
  - Books

- **Inheritance**
  - Fixed Account
  - Bank Account

- **Realization**
  - Printer
  - Printer Setup

From: http://creately.com/blog/diagrams/class-diagram-relationships/
Annotations

- For any relationship (edge between classifiers), we can annotate:
  - The name of the relationship (may be directional – indicated with a solid arrowhead in the direction the relationship holds)
  - The role of target instance in the source
  - Cardinality constraints (1:N, N:M, etc.) at either end
  - Possible ordering at either end

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Exactly one</td>
</tr>
<tr>
<td>*</td>
<td>Many (any number)</td>
</tr>
<tr>
<td>0..1</td>
<td>Optional (zero or one)</td>
</tr>
<tr>
<td>m..n</td>
<td>Specified range</td>
</tr>
<tr>
<td>{ordered}*</td>
<td>Ordered</td>
</tr>
</tbody>
</table>
Class Diagram for ATM System

UC Santa Barbara

- **ATM**
- **CashDispenser**
- **DepositSlot**
- **Screen**
- **Keypad**

**BankDatabase**

**Account**

- **executes**
- **authenticates user against**
- **contains**
- **accesses/modifies an account balance through**

- **Withdraw**
Abstract Class

- Defines one or more abstract methods
- Intended to serve as a base class
  - Class name is **italicized**
- Either it contains data or it contains at least one non-abstract method (otherwise it is an Interface)
- Inheritance denoted with an open arrowhead

![Diagram of Abstract Class](image)
Interfaces

- Defines abstract methods
- Inheritance denoted with an open arrowhead

Interface (all methods are abstract)

```
<<interface>>

DrawingContext

SetPoint(int, int, bool)
ClearScreen()
GetVerticalSize(): int
GetHorizontalSize(): int
```

Abstract Class
Some methods are abstract and some are implemented

```
Shape
  Draw()
  Erase()

Circle
Square
```
UML Diagrams

- UML provides several types of diagrams to model collaboration behavior of a system

- Communication diagram:
  which objects participate in collaborations

- Sequence diagram:
  when messages are sent between objects over time
UML Communication Diagrams

- **Objects**: rectangles containing names in the form object Name : ClassName
  - Only one object of each type: Omit object name, i.e., : ClassName

- **Communication**: solid lines message name appears next to the line, usually operation name (i.e., method name in Java) of the receiving object (direction indicated)

- **Message**: solid filled arrow ——
  - Synchronous call in the UML, method call in Java: Control passed from the sender to the receiver

- **Asynchronous** messages: ——>
Examples

- Single message:

```
:ATM execute() :BalanceInquiry
```

- Also has sequence and nested:

```
:Screen

3: displayMessage( message )

:BalanceInquiry

1: getAvailableBalance( accountNumber )
2: getTotalBalance( accountNumber )

:BankDatabase

1.1: getAvailableBalance()
2.1: getTotalBalance()

: Account
```
Sequence Diagrams

- Model the timing of collaborations
- Dotted line extending down: object’s lifeline, with progression of time
- Solid arrow: a message between two objects
  - Activation on the receiving object’s lifeline
- Thin vertical rectangle: an object is executing
- Return message: the activated object returning control to the original object
Example Sequence Diagram

: Withdraw

: Keypad

: Screen

: BankDatabase

: Account

: CashDispenser

displayMessage(message)

getInput()

getAvailableBalance(accountNumber)

isSufficientCashAvailable(amount)

debit(accountNumber, amount)

dispenseCash(amount)

displayMessage(message)
More on Sequence Diagrams

: Class1

new()

activationSynchronousMessage()

activationAsynchronousMessage()

inquiry()

confirmation()

delete()

: Class2
PRDv2: Your Living Requirements Document: A Shared Google Doc

- Authors, Team, Project Title
- Intro: problem, innovation, science, core technical advance (3+ pages)
  - Define project specifics, team goals/objectives, background, and assumptions
- System architecture overview
  - High level diagram (1 page)
  - User interaction and design (1+ pages) – ie detailed design
- Requirements (functional and non-functional)
  - User stories or use cases (links) → 20+ for PRDv2 prioritized w/acc. tests
  - Prototyping code, tests, metrics (10+ user stories): github commits/issues
- System models (1+ pages)
  - Contexts, interactions, structural, behavioral (UML)
  - Use cases, sequencing, event response, system state, classes/objects
- Appendices - Technologies employed
Your Project Design: PRDv2

- **Architecture (hardware/software)**
  - Evolve your overview picture from PRDv1 to provide significantly more detail and any updates or changes

- **Detailed design**
  - UML diagrams of primary data structures that comprise the system architecture connected via their associations (if any)
    - Ensure that each "class" is balanced in terms of cohesion & coupling
    - Annotate with pre/post conditions when appropriate
  - Sequence diagrams
    - Synchronous and asynchronous for key interactions between classes
      - At least 3 different interactions
    - User interactions with the system
      - At least 3 different interactions
      - Can be a human user or a machine user (API) interaction
        » Event response, updated application state
    - If you have a user interface: Provide mockups for primary UIs
PRDv2 User Stories / Use Cases

- Revise spec to add detail to the functional specification to match your design
- Add user stories and break up the stories you have into finer grained stories
  - Provide UML, sequence diagrams, dataflow diagrams
  - Goal: a CS senior should be able to take your doc and implement the project
- For each fine-grained story, provide a description and acceptance test
  - Provide time estimates (1 person-hours) for each story implementation
    - Ensure you can finish the implementation in the time you have (this/next quarter)
  - Prioritize tasks to have a complete prototype by the end of this quarter
    - Focus on the externally facing interfaces, mock out what you cannot get to
  - Write unit tests to implement tasks for mandatory tasks
    - Document these tasks (autogen the documentation/usage)
  - Add trello/pivotal task links (titles must match) to PRDv2 for each story
- Prototype designed mandatory tasks; add github commit ID/link to PRDv2
  - Github must have unit tests, documentation (for anything without unit tests), and prototyping implementations for each story in Sprint
- If you have a user interface
  - Provide mockups that are tied to the functionality described in 1+ components