Course Overview
Review of Scrum. Introduction to UML.

This lecture is based on two SCRUM presentations:

* Agile Software Development with SCRUM by Shveta Mehtani (http://www.scribd.com/doc/6578688/SCRUMAEG)
* What is Scrum? by Richard Fennell (http://www.slideshare.net/businessquests/black-marble-introduction-to-scrum)

… as adapted by Michael Mateas
The year-long game design studio sequence

- **CS 170**
  - Exposure to a variety of alternative game designs
    - Indie, serious games, political games, art games, etc.
  - Individual concept development
  - Team formation and game design

- **CS 171**
  - The heart of making the game
  - Course is process-based, providing a series of milestones for completing game
  - Some game design work will continue
  - Focus on software engineering issues

- **CS 172**
  - Emergency design revisions (the “oh my god” moment)
  - Build out game content (level design)
  - Final playtesting and tuning
  - Finish game
  - Win awards at indie game competitions
Class mechanics

- Syllabus online at
  - courses.soe.ucsc.edu/courses/cmps171/Winter12/01

- Team feedback tool
  - http://cmps172.soe.ucsc.edu/
Grading Criteria

- A combination of individual and group performance
  - Individual (Sprint) project performance – 51%
    - Pre-sprint planning activities, release planning – 5%
    - Sprint I – 13%
    - Sprint II – 13%
    - Sprint III – 13%
    - Special team role performance – 7%
  - Team project performance – 49%
    - Pre-sprint planning – 5%
    - Sprint I – 8%
    - Sprint II – 8%
    - Sprint III – 8%
    - Release performance – 20%
- Ungraded, but required to pass:
  - Game website
Team firing process

- A team member can be fired from a team for lack of performance, or poor interactions with the rest of the team (“bad apple” behavior)
- Detailed firing process is on course website
- Briefly, two step process:
  - Remediation
    - Identify and try to fix the problem.
    - Letter describes concrete steps and timeline for improved behavior
  - Removal
    - 2/3 vote of CS 171 members of team will fire team member
- Fired team member has two weeks to find a new team, or fails the class
Learning Goals

- **Scrum software development process**
  - Release and Sprint planning
  - Experience performing several Sprints and one Release
  - Project management using Scrum (burndown charts, task boards, daily scrums)
  - Hands-on experience in the Scrum Master role
  - "Bad apple" behavior and how it affects teams
  - Coordination with artists and musicians

- **Software design**
  - Unified modeling language (UML)
  - UML structure and sequence diagrams
  - Experience using UML to represent software designs

- **Game testing**
  - Game playtesting, concept and application
  - Gameplay metrics, including instrumenting of code to collect metrics

- **Software testing**
  - Unit testing, experience using unit testing frameworks
  - Black box and white box testing
  - Different classes of code test coverage (all paths, def-use, etc.)

- **Continuous integration**
  - Role and function of continuous integration
Upcoming deadlines

- **Wednesday (Jan. 11): Release Plan due**
  - A big effort. If you haven’t started already, you’re behind.
  - Due by midnight
  - See website for information and template

- **Thursday (Jan. 12): Sprint 1 Plan due**
  - Due by midnight
  - See website for information and template

- **Friday (Jan. 13): Sprint 1 begins**

- **Friday (Jan. 13): team status reporting**
  - Due by midnight
  - Report on team activities this week

- **Wednesday (Jan. 18): Technical design document due**
  - UML diagrams describing the technical design of your game
  - A big task, need to spin 2-3 people up on this by end of the week
Upcoming events

- Cecil Brown: Games Blacks Love to Play
  - Wednesday, January 18, 2012, 11:00 AM to 12:00 PM
  - The Simularium (Engineering 2, Room 180)

- Rich Hilleman: Game Telemetry and Crowd Sourcing New Game Services: How you take over the game business.
  - Thursday, January 19, 2012, 6:30 PM to 8:00 PM
  - The Dark Lab (Digital Arts Research Center, Room 108)

- Winter Job & Internship Fair
  - Tuesday, January 31, starts 11am
  - Resume workshop: Jan. 24, 12pm
  - Resumania: Jan. 25 (drop-in resume help)
  - See http://careers.ucsc.edu/ for more information
Alternate project

- We are seeking volunteers who wish to work on a novel, directed project for the remainder of 171/172
- You would leave your current team, and form a new team to work on this project.
- Kinect-based project
- Please sign up using sheet at front of class
Sammy Awards

- We are looking for people to volunteer to help organize this year’s Sammy Awards
- Not a big job, fairly light load
- Will work with Robin Crough, Operations Manager, Center for Games and Playable Media
- If interested, sign up on sheet at front of room

Sammy Awards

- Friday, June 15
- Rio Theater (pending final confirmation)
Introducing Chris Lewis

- Ace TA for the class
- Details on how to submit assignments
- Use eCommons, Assignment mechanism
- Need to set up time to have Chris present during one scrum meeting each week
  - Part of Sprint 1 plan document
Artists, Musicians, etc. Independent Study

- Filling out independent study form
Puzzle Defense

- Please come see me at the end of class
- Want to discuss social networking aspects of your project
Photos

- Taking photos of each team, and then putting names to faces.
  - Hello world
  - Micro ventures
  - Sonar
  - Chroma
  - Firewall
  - Puzzle defense
  - Devils bargain
Hello world
Sonar
Chroma
Firewall
Puzzle Defense
Devil’s Bargain
Review of Scrum
Product owner

- Define the features of the product
- Decide on release date and content
- Be responsible for the profitability of the product (ROI)
- Prioritize features according to market value
- Adjust features and priority every iteration, as needed
- Accept or reject work results
Product Owner in CS 171

- The notion of Product Owner is tricky for this project class
  - Each team “owns” their game design. There is no external customer, as in most software projects.
  - But, the Product Owner must be a single person.
    - A team cannot be an effective Product Owner, since the Product Owner must be able to make decisions concerning feature priorities, features to include or cut, etc.
- The Professor/TA is a partial Product Owner
  - Since a grade is being assigned for how well the team does at creating their project, the Professor/TA is a stakeholder.
  - However, the Product Owner must be able to participate in Release Planning meetings, Sprint Planning meetings, and the Professor/TA cannot do this (not scalable).
Approach for this class

Each team appoints one member as the Product Owner

- This is typically the person in the team who “owns” the game design
- They hold the game design vision, or, at the very least, are a person who is entrusted with the authority to make hard tradeoff decisions about the design.
- This person will typically stay in the Product Owner role for the entire quarter
  - But, this can be changed at the start of a Sprint, if someone just doesn’t work out in this role

The Professor/TA retain right to modify Product Owner decisions

- For example, changing feature priorities, feature cut/save decisions
- Same authority as Product Owner, but unlikely to exercise this authority often – authority is delegated with team’s Product Owner
The ScrumMaster

- Represents management to the project
- Responsible for enacting Scrum values and practices
- Removes impediments
- Ensure that the team is fully functional and productive
- Enable close cooperation across all roles and functions
- Shield the team from external interferences
Scrum Master in CS 171

- Each Sprint, one (possibly two) team members are appointed as Scrum Master
  - This role lasts for the entire Sprint
  - Each team member (except the Product Owner) must be a Scrum Master for at least one Sprint
    - On large teams, this role can be shared by two during a sprint

- Scrum Master is responsible for:
  - Maintaining scrum (task) board
  - Maintaining sprint burndown chart
  - Providing detailed feedback each week on activities of team members
  - Ensuring team follows correct Scrum practice

- Performance in this role is part of the individual performance grade for the class
  - Special team role performance – 8%
Release

- A **release** is a major milestone in the development of a software project.
- A release contains a series of product **features**.
- Features are expressed in the form of **user stories**.

The goal of **release planning** is to determine which user stories (features) will be included. This involves:

- Taking the game concept and **decomposing** it into user stories.
- **Estimating** the time required to perform each user story (using story points).
- **Prioritizing** the user stories.
- The release plan forms the input into the Sprint planning process.
User stories

- A product feature is expressed in the form of a user story.
  - This can be viewed as a specific technique for eliciting and writing software requirements.
  - A user story is a software requirement

- User story format
  - As a {user role}, I want {goal} [so that {reason}]
  - Examples:
    - As a player, I need control over a laser pointer so that the cat will follow it.
    - As a player, I need to pick up gameworld objects so that I can collect food and ammunition.
    - As a playtest manager, I need automated collection of gameplay metrics so that levels can be analyzed for areas that are too difficult.

- Class exercise developing a few user stories for your game
INVEST in user stories

- What are the attributes of a good user story?
- **INVEST** conditions
  - Independent
    - Free of implementation dependencies on other stories
  - Negotiable
    - Useful as the basis for discussion between stakeholders and team
  - Valuable
    - Communicates value to player and to team
  - Estimatable
    - Possible to estimate effort to implement user story
  - Sized appropriately
    - Need to be small enough to fit into a Sprint
  - Testable
    - It must be possible to verify that a user story has been implemented.
The relative size (implementation effort) of each user story is estimated using measure known as story points.

**Story points are unitless**
- Are not person-months, meters, hours, etc.
- Key idea is to focus estimating effort on *relative* size
- Use of unitless numbers avoids arguments
  - “That won’t take a week to implement – that’s easily a week and two days”
  - … but the point is trying to determine which tasks are $O(\text{days})$, $O(\text{weeks})$, and $O(\text{months})$ – +/- a few days doesn’t matter!

**Story points are linear**
- A user story requiring 0.5 story points takes half the time to complete as one requiring 1 story point
- Similarly, a user story requiring 3 story points is the same size as one requiring 1 story point and another requiring 2 story points
Using unitless points for estimation

- Exercise using unitless points for estimation
  - Let’s call the distance from Thimann Lecture Hall to the Science & Engineering Library “1 point”
  - What is the distance from Thimann to Engineering 2?
  - What is the distance from Thimann to the base of campus (intersection of Bay and High)?
Prioritizing user stories in a release

- During release planning, user stories must be **prioritized**
  - It’s a cop-out to say “everything is equally important”
  - You will implicitly prioritize things based on the order of implementation even if you don’t explicitly prioritize them up front
  - Better to be explicit about the order of implementation

- What do priorities mean?
  - A user story with highest priority is implemented first
  - A user story with lowest priority is implemented last
  - Lower priority items might **never be implemented**
    - If there is a feature you really want to see in the game, need to ensure it has a high priority

- Product Owner has ultimate authority over setting priorities
Assigning user stories to a Sprint

- During release planning, the team assigns user stories to a particular Sprint in which they can be implemented.
- This requires the team to guess how many story points they can implement during a Sprint:
  - Over time, a team will develop a good sense of their capacity. At the beginning, there will be a lot of uncertainty.
  - Make a good faith guess for now – this will be refined during Sprint planning.
- Important: Sprint goals identified during release planning are a forecast of work that can potentially be done by the team. They are not a commitment.
  - During Sprint planning, each user story will be broken down into tasks:
    - Task estimates are in ideal work hours, and are a commitment.
Output of Release Planning

- At the end of release planning:
  - A **prioritized list** of **user stories**, with implementation time **estimated in story points**, organized into **Sprints**.

<table>
<thead>
<tr>
<th>Plan for Release #1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>User Story</td>
</tr>
<tr>
<td><strong>Sprint 1</strong></td>
<td></td>
</tr>
<tr>
<td>1.</td>
<td>As {role} I ...</td>
</tr>
<tr>
<td>2.</td>
<td>As {role} I ...</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td><strong>Sprint 2</strong></td>
<td></td>
</tr>
<tr>
<td>N.</td>
<td>As {role} I ...</td>
</tr>
<tr>
<td>N+1.</td>
<td>As {role} I ...</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Sprint planning

- Team re-evaluates user stories from the release plan and product backlog they can commit to completing
- Sprint backlog is created
  - User stories are subdivided into tasks
  - Tasks are identified and each is estimated (~8 hours)
  - Collaboratively, not done alone by the ScrumMaster
- High-level design is considered

As a vacation planner, I want to see photos of the hotels so I can have a better idea of facilities
Priority 4 [10 Story Points]

Code the middle tier (8 hours)
Code the user interface (4)
Write test fixtures (4)
Code the foo class (6)
Update performance tests (4)
Task estimation

- Performed as a group, using Planning Poker
- Here, units of estimation are “ideal work hours”
  - The amount of work you can get done under ideal conditions
    - Full knowledge, no interruptions
  - Actual hours elapsed will be greater than ideal hours
- Task estimates are a commitment to accomplish a development task in a certain period of time

How many ideal work hours can each person perform?

- Good question – so far, your group has no track record on this
- For now, pick a conservative figure, such as 10-12 ideal hours/week
- So, each group member can do 30-36 ideal hours of work per Sprint
Sprint planning (3)

- A likely scenario is that you team will find they don’t have enough time to implement all user stories in the release
  - In this case, need to assess user stories
    - Are the priorities all still the same?
    - If so, drop the lowest priority user stories until estimated work agrees with team’s work capacity
    - Can pick these up in later Sprints
  - What if the team finishes too soon (i.e., systemic over-estimation of task length)?
    - Very unlikely to occur – the opposite problem (under-estimation) is far more common
    - If it does happen, the team can add another user story midway through the Sprint
A short statement of what the work will be focused on during the sprint

**Database Application**
Make the application run on SQL Server in addition to Oracle.

**Life Sciences**
Support features necessary for population genetics studies.

**Financial services**
Support more technical indicators than company ABC with real-time, streaming data.
Managing the sprint backlog

- Individuals sign up for work of their own choosing
  - Work is never assigned
- Estimated work remaining is updated daily
- Any team member can add, delete or change the sprint backlog
- Work for the sprint emerges
- If work is unclear, define a sprint backlog item with a larger amount of time and break it down later
- Update work remaining as more becomes known
Output of Sprint planning (for CS 171)

- Task listing (with time estimate), organized by user story (prioritized)
  - User story 1:
    - Task 1 (time estimate)
    - Task 2 (time estimate)
    - ...
  - User story 2:
    - Task 1 (time estimate)
    - Task 2 (time estimate)
    - ...
- Team roles
  - Team member 1: role
  - Team member 2: role
  - ...
- Initial task assignments
  - For each person, what is the first task they are working on?
- Initial task burndown chart
- Initial scrum board set up
- Schedule of Scrum meetings
  - When/where for 3 weekly face-to-face scrum meetings
Introduction to UML
Introduction to UML

- The Unified Modeling Language (UML) consists of a collection of diagrams for describing a software design.

- Creating a UML description forces a team to develop a software design before diving into the nitty-gritty of writing code.
Class diagrams

- A class diagram describes *static* aspects of your object oriented design.
- Classes are drawn as boxes.
- Members are listed inside the box. Fields appear in the top sub-box, methods in the bottom sub-box.
  - Access indicated by + (public), - (private), # (protected) and ~ (package).
- Classes are connected together with lines indicating class relationships.

```
SequentialBehavior
- stepCounter: int
  - child: Step

~getNextStep(): Step
~addChild(s: Step)
~removeChild()
```
Generalization links

- Generalization links indicate subclass relationships
  - Parent/child relationships
- An open arrow points to the parent

**Behavior**

- #parent: GoalStep
- #priority: int
- #signature: String

- ~failBehavior()
- ~succeedBehavior()

**SequentialBehavior**

- stepCounter: int
- child: Step

- ~getNextStep(): Step
- ~addChild(s: Step)
- ~removeChild()
Aggregation links

- Aggregation indicates that instances of one class will contain instances of another class
  - In aggregation, the lifespan of the enclosed instances is independent of the lifespan of the enclosing instance

- Container classes (lists, hashtables, etc.) will always have aggregation links to what they contain, though many classes will contain member instances of other classes

```
SequentialBehavior
  - stepCounter: int
  - child: Step
  ~getNextStep(): Step
  ~addChild(s: Step)
  ~removeChild()

Behavior
  #parent: GoalStep
  #priority: int
  #signature: String
  ~failBehavior()
  ~succeedBehavior()

Step

GoalStep
```
Composition links

- Composition links indicate that one class contains instances of another class, but the contained class is created and destroyed with the instance class
  - The contained instances will be destroyed when the containing instance is destroyed

- In C++, this is the difference between a member variable of type MyClass* and MyClass
Realization

- Realization links relates a class that implements (realizes) a behavior specified by another model element, to the model element that specifies this behavior.

- In Java, classes that implement an interface realize the interface.
- In C++, classes that are children of a pure abstract class realize behavior specified by the pure abstract class.
Dependency links

- Dependency links represent arbitrary relationship between classes, where a change made to one class may require a change to another class
  - The arrow points from the dependent towards the independent class

- You’ll want to use link labels for dependency links

- On the class diagram, only indicate important dependency relationships (ones that help communicate in the team)