CS279 Course Overview

Joe Hoffert

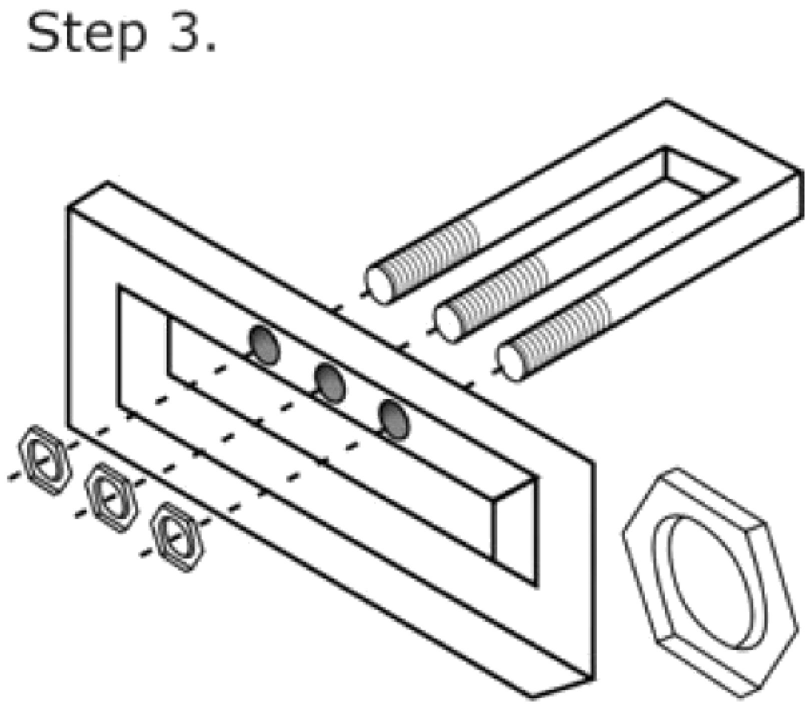
Distributed Real-time Embedded (DRE) Group

Institute for Software Integrated Systems

Vanderbilt University

What is CS279 About?

- This class is about picking an interesting software project and building it using an agile development approach



IKEA

CS 279 Course Information

- CS 279 class web page
  - www.dre.vanderbilt.edu/~jhoffert/cs279/
- My office hours in Featheringill Hall room 226 are
  - Tues/Thurs 2:25pm to 4:25pm

- **Required textbooks**

None!!!

Please send all questions to jhoffert@dre.vanderbilt.edu

I'll send the answers to the class mailing list

CS 279 Ground Rules

- Build cycles must be completed on time
- ~~Work *must* be your own*~~ All group projects!!!
- *Bring your laptops every day (just in case)*
- Your in-class participation is expected (e.g., answering questions)
- You'll get out of this course what you put into it so be prepared to work hard
- Be prepared for occasional guest lectures
- No quizzes, no tests, no exam → instead: weekly demos, code reviews, and a final demo
- Avail yourself of help, e.g., office hours, TAs, mailing list

CS 279 Course Contents

- Focus on developing large-scale software projects in a team setting:
  - Code must be turned in every build cycle
  - Agile software development practices must be followed
  - Bi-weekly demos of code
- Everyone must be a member of a team working on a large scale software project
- The course will completely revolve around producing quality software.
- I will introduce advanced topics in Java/C++, patterns, etc. to aid the projects
- I assume you know Java or C++ fairly well, e.g., you know how to use Eclipse, the classpath, Java/C++ compilers, STL, ACE, etc.
- Feel free to ask me questions via email/class/office hours related to:
  - Eclipse
  - Java, C++
  - Framework XYZ
  - Patterns
  - Development practices
  - Promoting your open source project
  - Etc…

CS 279 Course Contents

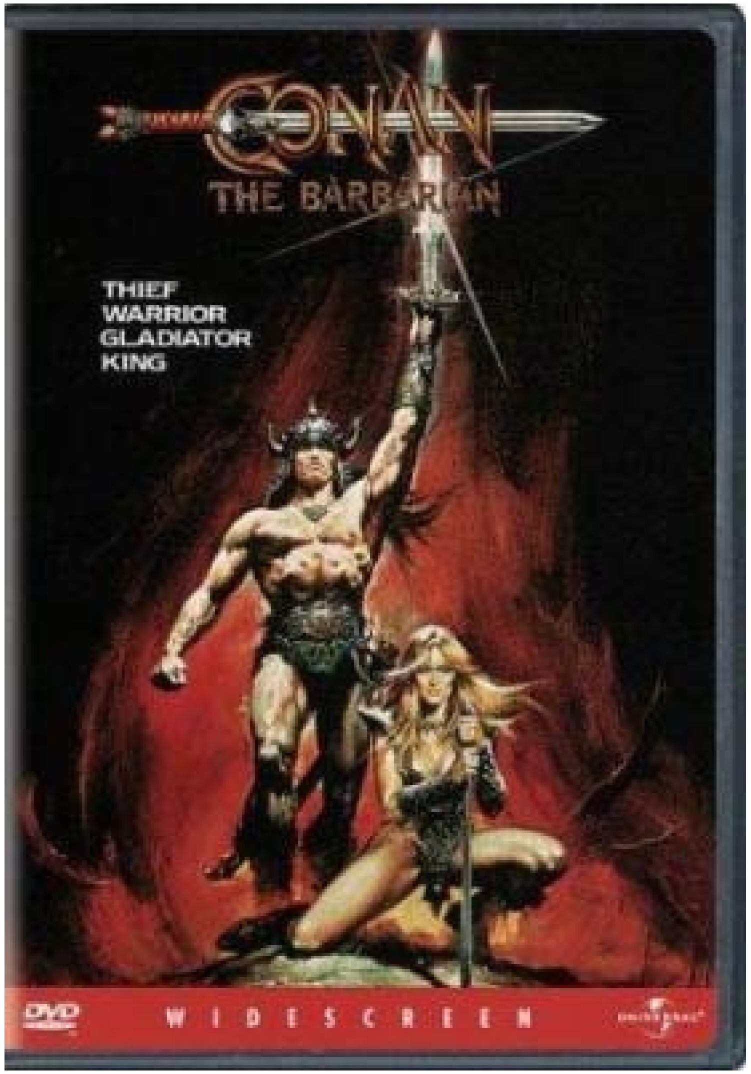
- My main goal of the class is to facilitate and guide everyone through the implementation of a larger scale software project using agile development
- You will learn by ***doing***
- Feel free to suggest advanced topics that you would like to cover in class:
  - Java web applications
  - Cool threading stuff
  - Java generics/C++ templates
  - Java annotations
  - Etc.
- I am also free to help outside of class with any questions you have
- **Every** member of each team must contribute
- Although I will be focused on groups as a whole, I will also pay attention to each team member's individual effort
  - I will look at Google code/SVN to see who committed what code
  - I will look at the bug tracking system to see who was reporting errors
  - I will look at project wikis to see who posted what
  - I will pay attention in class to who is contributing to the discussion

CS 279 Course Work

- There will be ~6 build cycles
  - All projects must be implemented in Java or C++
  - Can be done on Windows, Linux, Mac, etc.
  - **Must be done as a team**
- Your grade will be based on:
  - 70% bi-weekly build cycle execution
  - 20% final project demo/presentation
  - 10% in-class participation
- Waiting until the end of the course and trying to code everything (regardless if it works) **will** produce a poor grade
- A key part of the course is staying on the development schedule, following the development guidelines, and contributing each class period
- Feel free to use any open source code that you want (as long as you aren’t just ripping it off or writing a wrapper around it)

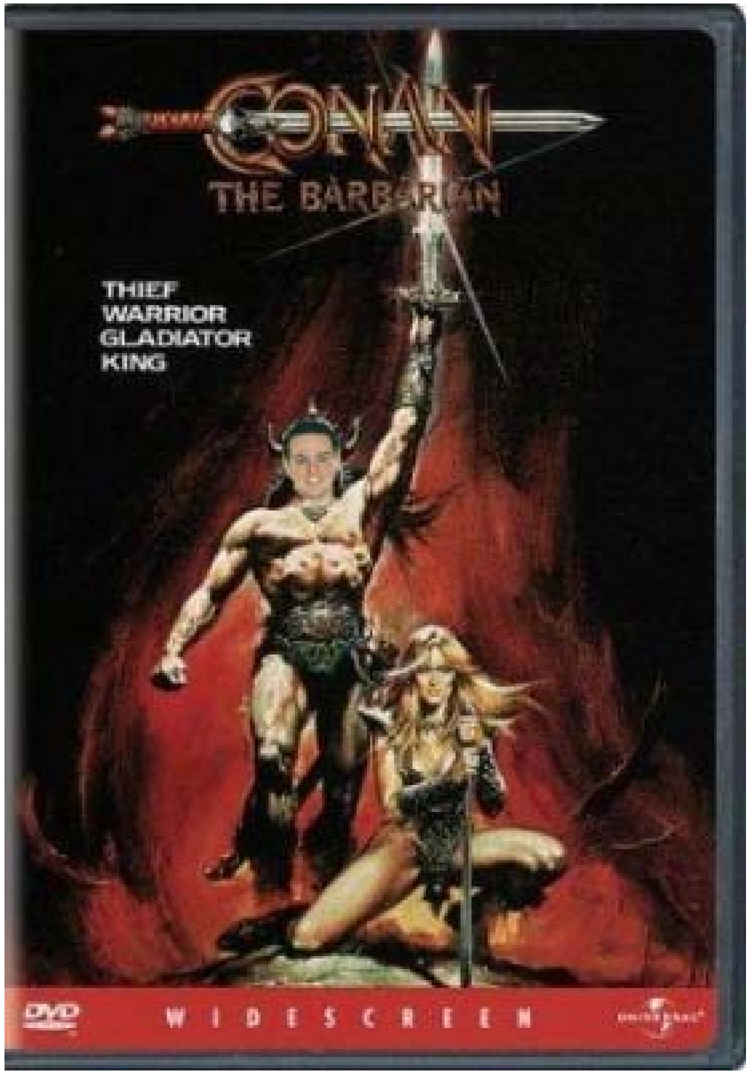
Lessons from Conan

- The secret of steel has always carried with it a mystery. You must learn its riddle, Conan. You must learn its discipline. For no one - no one in this world can you trust. Not men, not women, not beasts. **Steel you can trust**



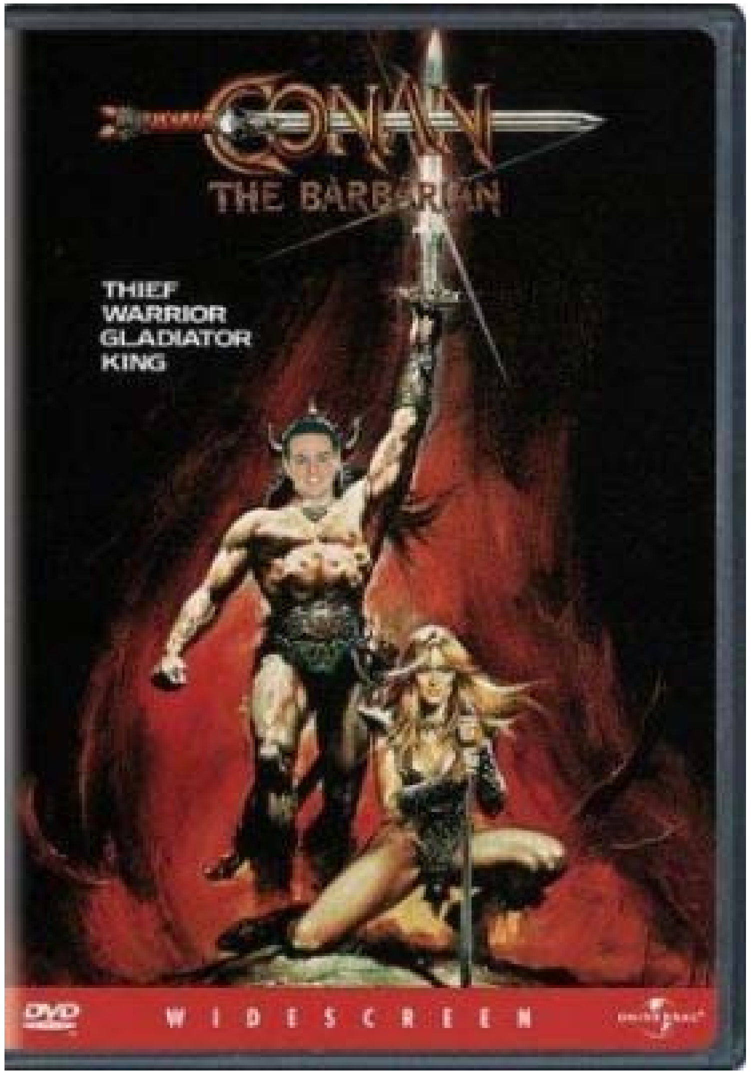
Lessons from Agile Development

- The secret of *code* has always carried with it a mystery. You must learn its riddle, undergrad. You must learn its discipline. For no project manager - no developer in this world can you trust. Not UML diagram, not test plan, not architect hype. **Code you can trust**
- **(if it is thoroughly tested)**



Lessons from Agile Development

- We will be using an Agile development process in CS279
- Short concentrated build cycles that focus on working code
- Client-focused, we will be demoing each others' software at the end of each build cycle



CS279 Development Cycle

- **We will use a 2 week development cycle that will start on Tuesdays**
- **1st Tuesday of cycle:**
  - Discuss/select user stories in class (rough drafts prepared before class)
  - Discuss code design for selected user stories
- **1st Thursday of cycle:**
  - Barebones code skeletons for user stories checked in before class
  - Each group designs tests for another group's user stories (your barebones code needs to be sufficient for others to design tests for)
  - Discuss test coverage and testing strategies
  - Advanced Java or C++ topic introduced (time permitting)
- **1st cycle starts Tuesday, Jan. 26**

CS279 Development Cycle (1st Tuesday)

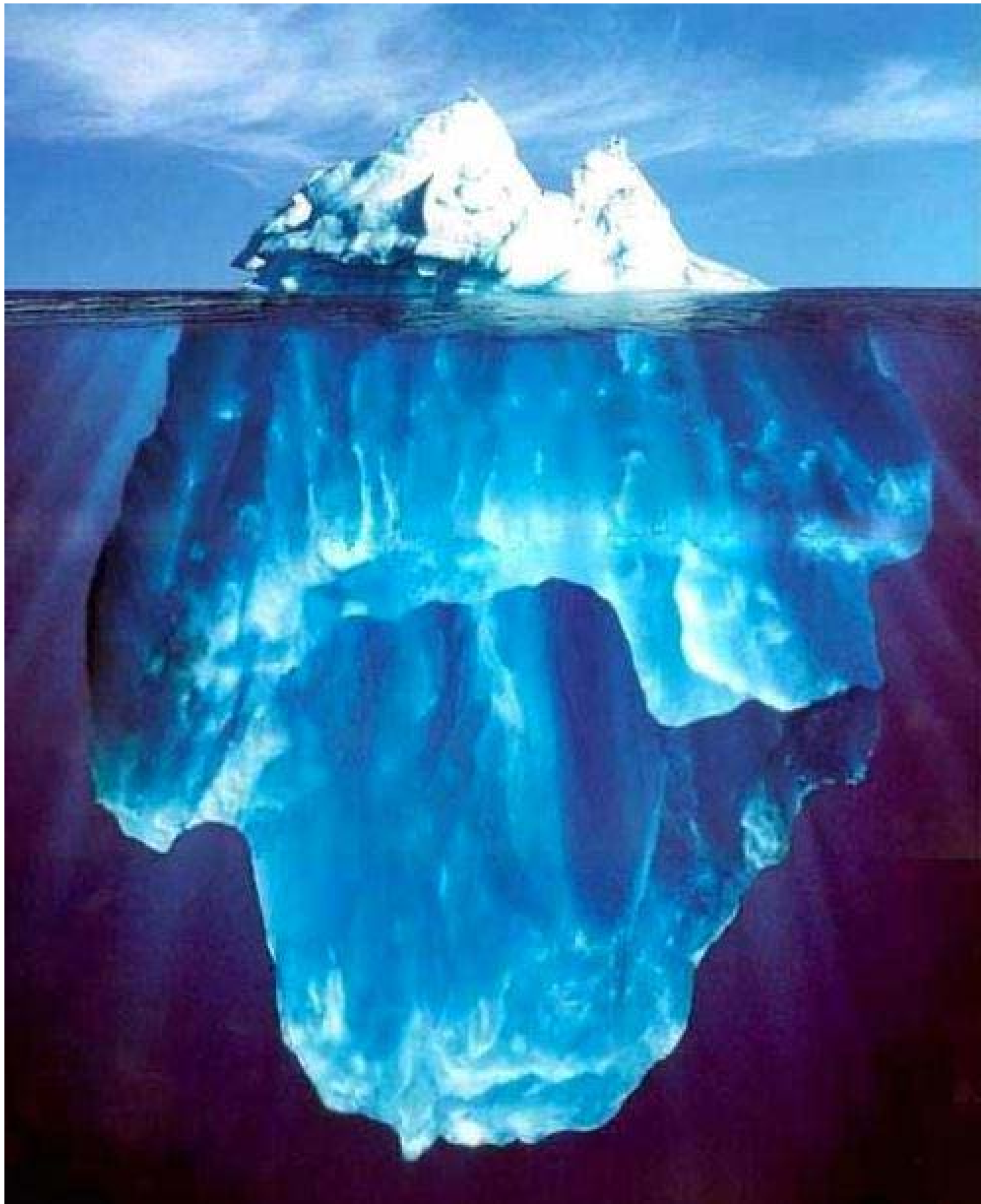
- **1st Tuesday of cycle:**
  - Discuss/select user stories in class (rough drafts prepared before class)
    - Each team member presents a user story.
    - Appropriate scope for each story?
    - Appropriate number of user stories?
    - User stories assigned to team members?
    - How do user stories fit with end-semester user stories?
  - Discuss code design for selected user stories
    - What design approach makes sense?
    - Patterns appropriate for a user story?
    - What kind of infrastructure is needed?
    - Potential problems?

CS279 Development Cycle (1st Thursday)

- **1st Thursday of cycle:**
  - Barebones code skeletons for user stories checked in before class
    - Any superfluous code for the current (and past) user stories?
    - Does all code relate to a user story?
    - Patterns used/appropriate?
  - Each group designs tests for another group's user stories (your barebones code needs to be sufficient for others to design tests for)
    - What design approach makes sense?
    - What kind of infrastructure is needed?
    - Potential problems?
  - Discuss test coverage and testing strategies
    - Automation/scripting (e.g., ACE "push button" tests)
    - Who should write tests?
    - Who should run tests?
    - What attitude should the tester(s) have (e.g., cooperative, antagonistic)?
    - Regression tests
    - Profilers

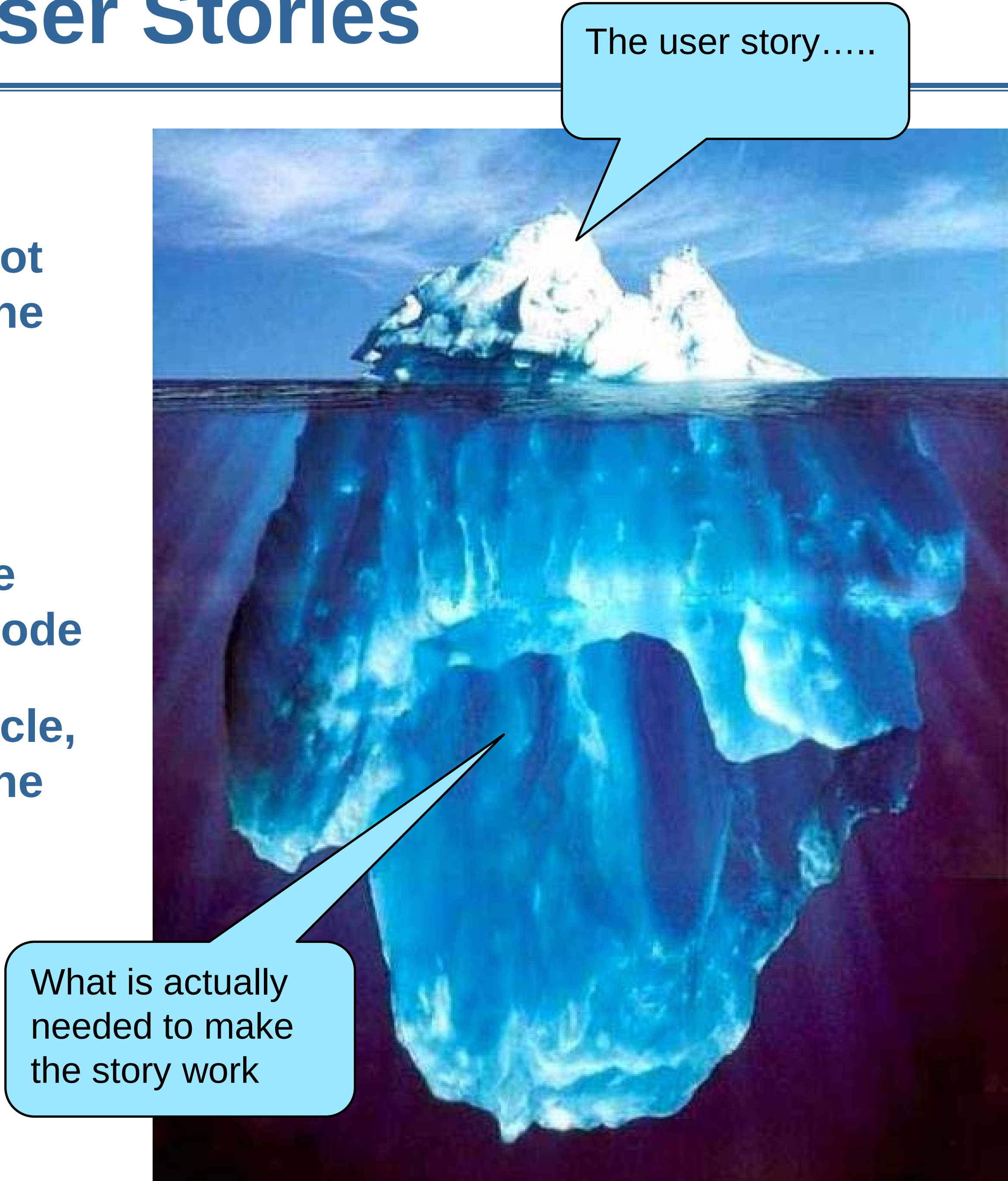
User Stories

- **What is a "user story"?**
- **A user story should be a short 1-2 sentence explanation of something that a user can do with the software:**
  - A student can add a new course to his/her schedule
  - A player can view the results of a match
- **User stories must be assigned to team members**
- **Team members will be graded on their assigned user stories & integrated functionality**



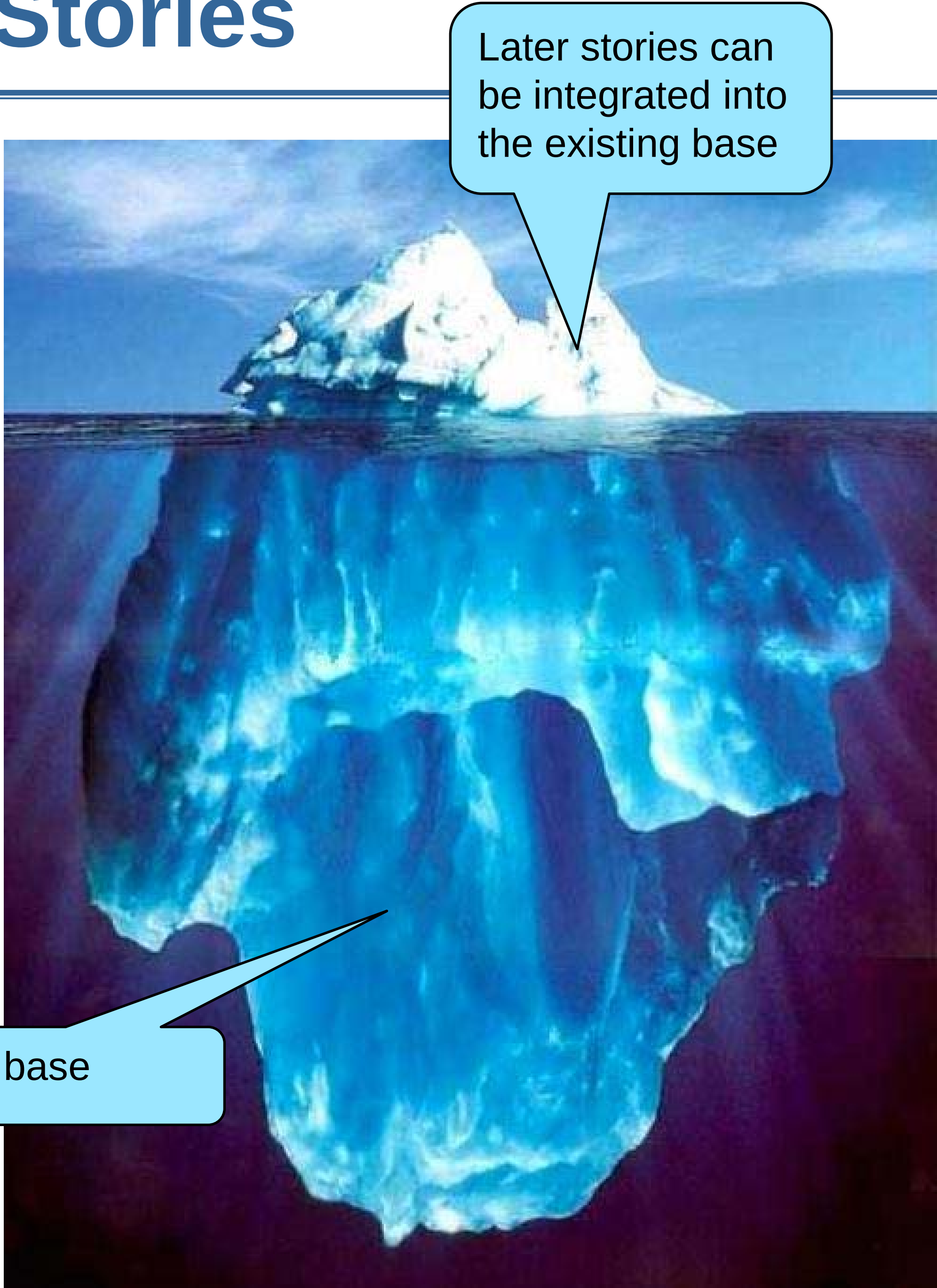
User Stories

- Each user story will be simple but will require a lot of things to work under the hood
- User stories emphasize working fully integrated software rather than large bodies of un-integrated code
- At the end of the build cycle, if a user can't complete the story, it isn't finished



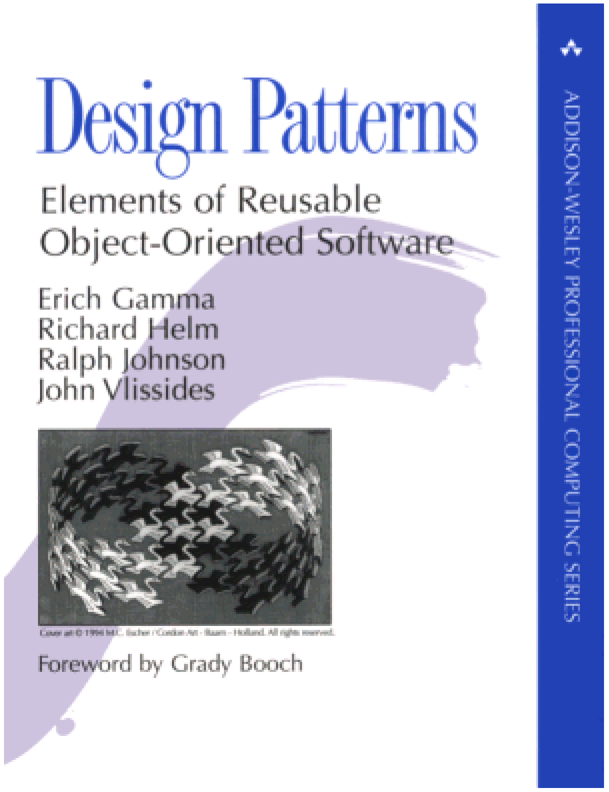
User Stories

- At the beginning, you should pick fewer user stories since you will need to build the “hidden base” of software beneath it
- Later, you can increase the # of user stories per build cycle because the bulk of your base is complete



Code Design

- **Patterns should be used wherever possible**
  - We will learn new patterns as needed in class
- **Testing is critical, your code must be designed so that it can easily be tested**
  - Plan to use mock objects early on for complex parts (e.g., faking remote server interaction)
- **Agile development assumes that *code will be refactored and extended***
  - Make sure that your code doesn't exhibit tight coupling
  - You will be refactoring your code after code reviews….tightly-coupled code will land you in a world of painful code rewriting


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Coding Standards

- **Basic coding standards:**
  - The code format standard should be what you get when run the Eclipse automated code formatter (ctrl + shift + f)
  - Groups should agree on variable naming conventions. I recommend all lowercase letters for local variables, all caps for static variables, and one of the following for member variables:
    - Foo myVariable; //All references to foo use "this"
    - this.myVariable = ….;
    - Foo myVariable_;
    - myVariable_ = ….;
  - Proper Java package naming
    - org.myprojectname.foo.bar
- **You must use an open source license**
  - License headers should be at the top of each source file!
  - I recommend the Apache License v2

Example Apache License Header

```
/***************************************************************************
* Copyright 2010 Joe Hoffert                                              *
*                                                                         *
* Licensed under the Apache License, Version 2.0 (the "License");         *
* you may not use this file except in compliance with the License.        *
* You may obtain a copy of the License at                                 *
*                                                                         *
* http://www.apache.org/licenses/LICENSE-2.0                              *
*                                                                         *
* Unless required by applicable law or agreed to in writing, software     *
* distributed under the License is distributed on an "AS IS" BASIS,       *
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.*
* See the License for the specific language governing permissions and     *
* limitations under the License.                                          *
***************************************************************************/
```

Project Requirements

- Every group must maintain their project in Google code
- You must use SVN or CVS (preferably SVN)
- You must maintain a wiki that provides detailed instructions on how to build, run, and test your code
- You must produce a binary distribution at the end of each build cycle

SVN Commit Rules

- **Rule #1: Never ever ever commit code that doesn't compile**

NFL equivalent of checking in code that doesn't compile

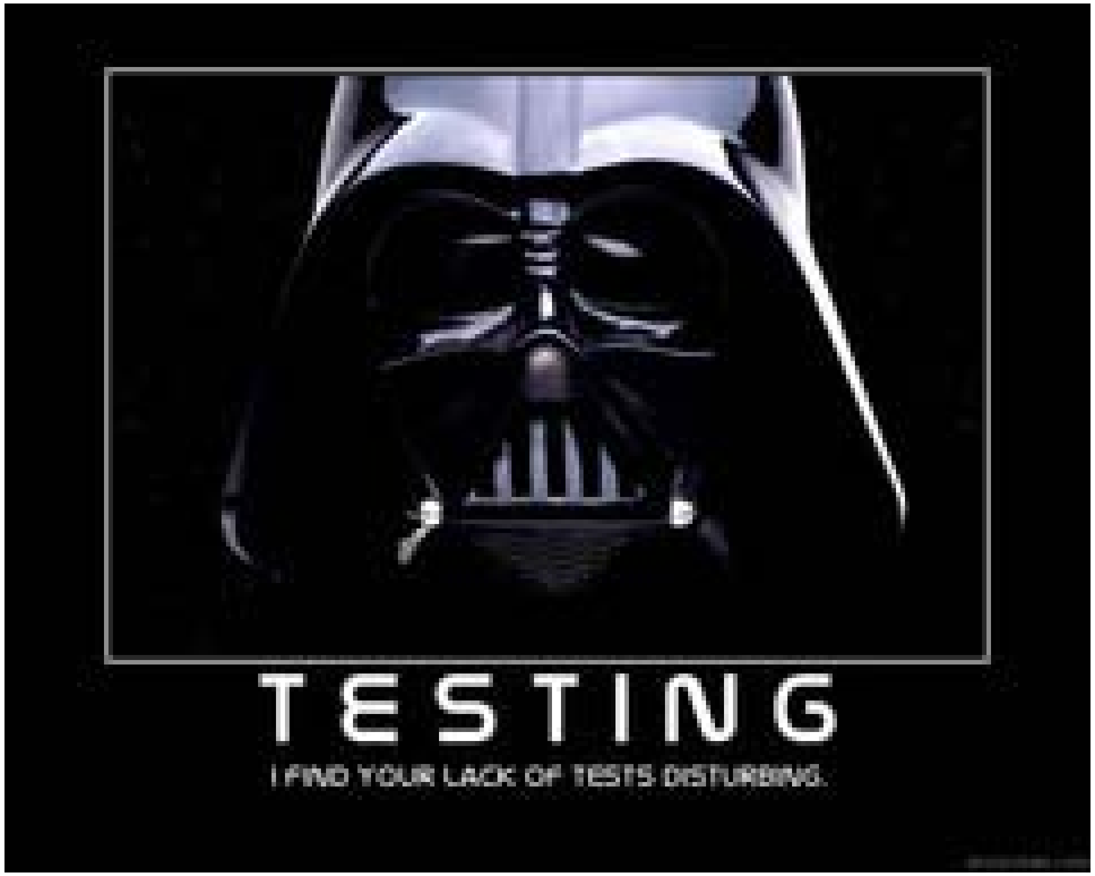
THAT'S GOTTA HURT ... Pittsburgh Steelers running back Destry Wight lies injured on the field Sunday night after he dislocated his right ankle and broke his right leg.

SVN Commit Rules

- **Rule #2: Always include a commit comment that briefly summarizes what changes you are checking in**

SVN Commit Rules

- **Rule #3: Always try to make sure your code passes the unit tests before checking it in**



SVN Comment Conventions

When you commit code use the following conventions

- For user stories, prefix with “US” + cycle + “:” + and user story number followed by normal text comments (e.g., US3:2 …).
- For unit tests, prefix “UT” + cycle + “:” + and user story number followed by normal text comments (e.g., UT3:2 …).
- For integration tests, prefix “IT” + cycle + “:” + and user story number followed by normal text comments (e.g., IT3:2 …).
- For bugs/issues use “B” prefix followed by issue ID plus normal text comments (e.g., B7 …).

CS279 Development Cycle Requirements

- **Every user story needs at least one associated unit test or integration test.**
- **SVN comments need to be specified accurately (e.g., using “UT3:2” for unit tests).**
- **Again, all issues/bugs need to be either**
  - Resolved by the end of the cycle OR
  - Justification for rescoping

Bugs

- If a team member checks in code and you notice that it breaks something, you must report it as a bug in the bug tracker (e.g., issues in google code)
- Make sure that you provide sufficient information to reproduce the bug
- All bugs either
  - must be cleaned up by the end of the build cycle or
  - used as a rational for rescoping a user story

CS279 Development Cycle

- **2nd Tuesday:**
  - Initial story implementations turned in (checked into SVN before class)
  - In-class code reviews of user story implementations
  - Bug/Issue discussions
  - Advanced Java/C++ topic introduced (time permitting)
- **2nd Thursday:**
  - Code refactored per code review recommendations (checked into SVN before class)
  - Binary distributions made available as file releases (checked into SVN before class)
  - User stories demoed
  - In-class user acceptance testing
  - Advanced Java/C++ topic introduced (time permitting)

CS279 Development Cycle (2nd Tuesday)

- **2nd Tuesday:**
  - Initial story implementations turned in (checked into SVN before class)
  - In-class code reviews of user story implementations
    - Teams make presentations
    - Any in-cycle refactoring/changes of direction?
    - What (potential) problems are there?
    - Any patterns used?
    - Does all the code relate to the user stories?
  - Bug/Issue discussions
    - Were any bugs found
    - Did any issues or concerns arise while coding?
  - Advanced topic, e.g., patterns (time permitting)

CS279 Development Cycle (2nd Thursday)

- **2nd Thursday:**
  - Code refactored per code review recommendations (checked into SVN before class)
    - Teams present refactoring work
    - Briefly describe bugs reported
  - Binary distributions made available as file releases (checked into SVN before class)
  - User stories demoed
    - Each team demos the user stories for the cycle
  - In-class user acceptance testing
    - One team runs the user stories for another project
  - Lessons learned for projects
  - Lessons learned for cycle
    - Different structure, interaction, format helpful in class
  - Advanced topic, e.g., patterns (time permitting)

Implementing User Stories

- Only build the minimum of what is needed to realize the user story
- All code created during the build cycle should be directly traceable back to a user story
- On the 2nd Tuesday, we will do in class code reviews
  - I will do code reviews for anyone who doesn't have their code reviewed in class
- Code will need to be refactored by the following Thursday per the code review recommendations

Implementing User Stories

- At the beginning, it is ok to “fake” or use mock objects for parts of the implementation
- For example, you may want to fake the communication with a remote server by creating a mock object that automatically returns the expected answers or stock data

What if I Just Can’t Get X to Work?

- **If you realize that a user story is much harder than expected to implement, don’t panic**
  - Discuss the issue with your group and send me email saying that you are going to postpone the user story until the next build cycle
  - Prioritize your other user stories and finish them
  - At the latest, you must notify me by the start of class on the 2nd Tuesday
- **Start early so that you can predict if you aren’t going to finish a user story**
- **If you have a midterm, etc. during a build cycle, go easy on yourself and pick easier/fewer user stories**

In-class User Acceptance Testing

- On the last class of a build cycle, we will first let each team demo their working user story implementations
- Groups will then test each others' user story implementations
  - Every group will be required to have a binary distribution that other groups can download to test
  - Groups must have all usage directions posted on their project wiki (i.e., no hand holding)
  - Groups can bring in user surveys to get feedback from users (optional)

Binary Distributions

- **A binary distribution should be a compiled version of the code that can be run fairly easily by a user**
- **Examples:**
  - A jar file, launch script, and instructions (always include a license file too)
  - A Java launcher, such as launch4j
  - An Eclipse plugin distribution
  - A set of project binaries and an ANT file to run them
  - A C++ executable for the target environment

Bi-weekly Grading

- **(20pts) Were all of the user stories completed or properly postponed?**
- **(20pts) Were adequate tests created and executed for the code?**
- **(20pts) Were bugs properly reported and addressed?**
- **(20pts) Did the new features pass user acceptance testing?**
- **(20pts) Does the code adhere to the development standards and was it refactored after the code review?**
- **(10pts Bonus) Did you bring up an interesting new topic in class and provide examples for it (e.g., code/uml)?**
- *****I reserve the right to change the weighting/grading criteria during the semester**