Group Projects

- A 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***
- 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 GitHub 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.

Project Work

- There will be 5 2-week build cycles
  - Can be done on Windows, Linux, Mac, web server, etc.
  - **Must be done as a team**

  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 session.
- 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).

2 Week Development Cycle

- **We will use a 2 week development cycle (that will initially start on a Thursday)**
- **1st class session of cycle:**
  - Discuss/select user stories in class (rough drafts prepared before class)
  - Discuss code design for selected user stories
- **2nd class session 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

2 Week Development Cycle (1st Class Session)

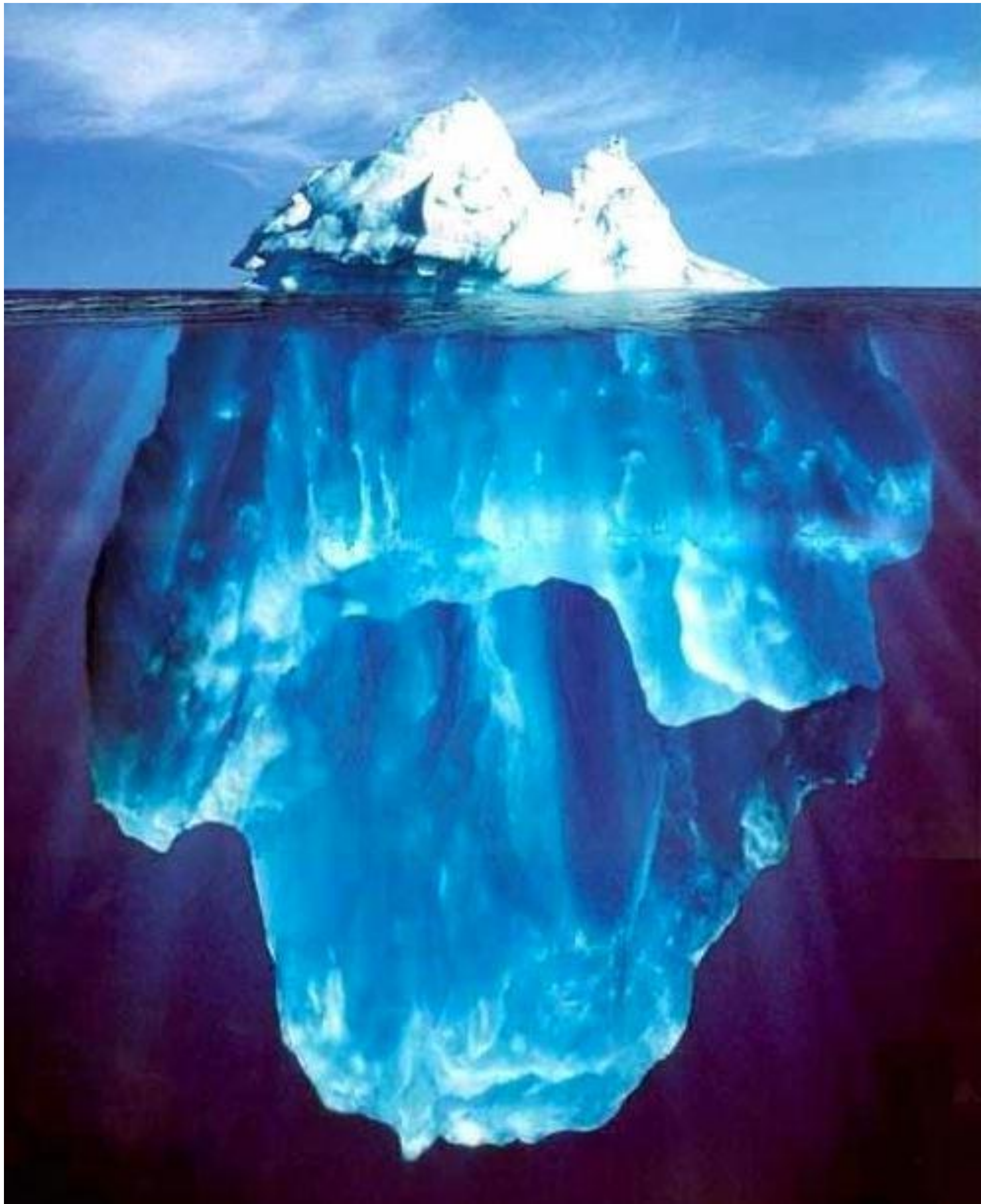
- **1st class session of cycle:**
  - Discuss/select user stories in class (rough drafts prepared before class)
    - Each team member presents a user story.
    - Appropriate scope for each story?
    - Appropriate number of user stories?
    - User stories assigned to team members?
    - How do user 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?

2 Week Development Cycle (2nd Class Session)

- **2nd class session of cycle:**
  - Barebones code skeletons for user stories checked in before class
    - Demo (high-level) design using code skeletons
    - 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., "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

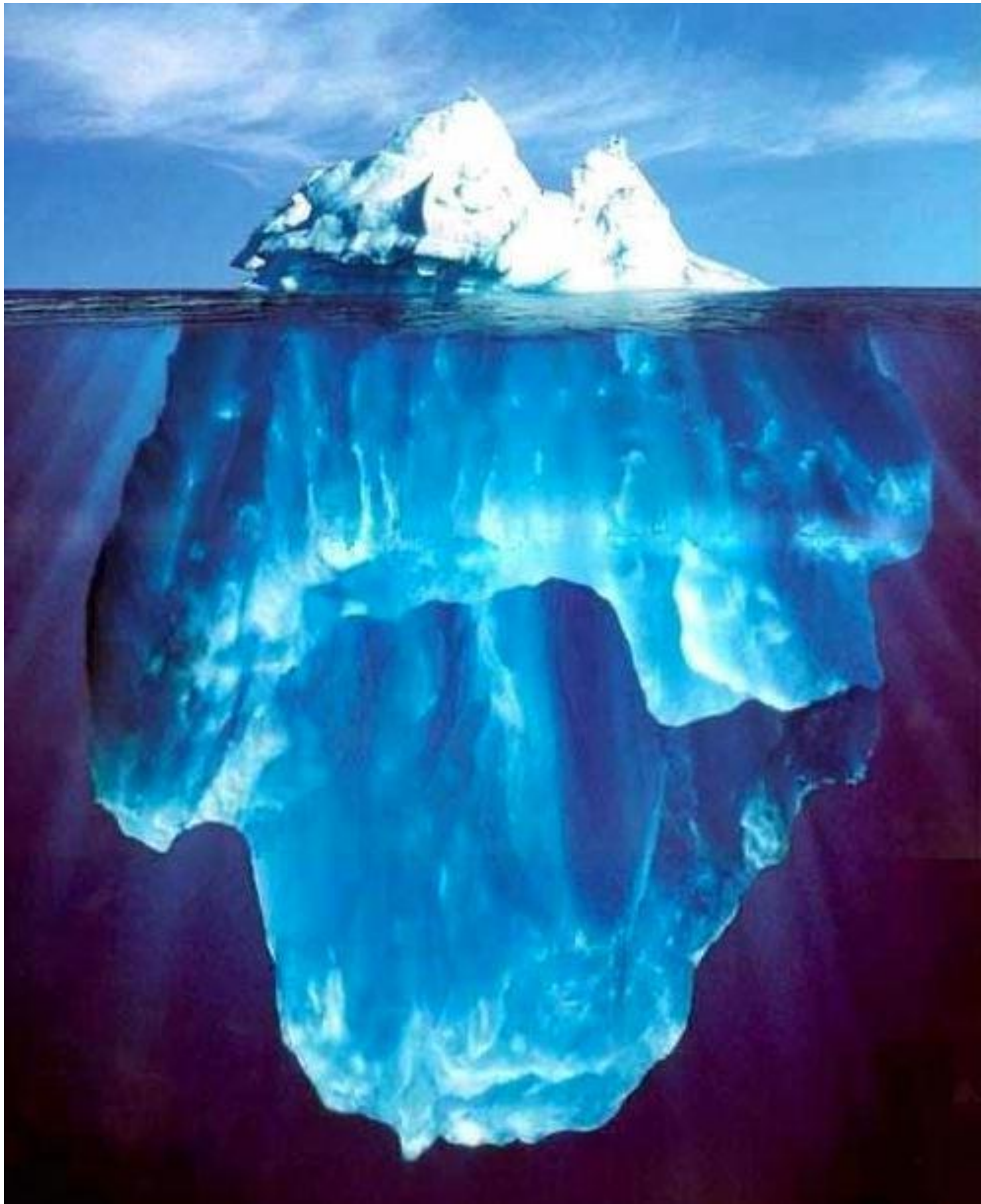
User Stories (CIS 320 Review)

- **What is a "user story"?**
- **A user story should be …:**
  - Example?
- **User stories must be assigned to …?**
- **Team members will be graded on …?**



User Stories Review

- **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 & that the user can verify in some way:**
  - A student can add a new course to his/her schedule and see it on the screen
  - A player can view the results of a match after clicking the “Results” button
- **User stories must be assigned to team members**
- **Team members will be graded on their assigned user stories & integrated functionality**

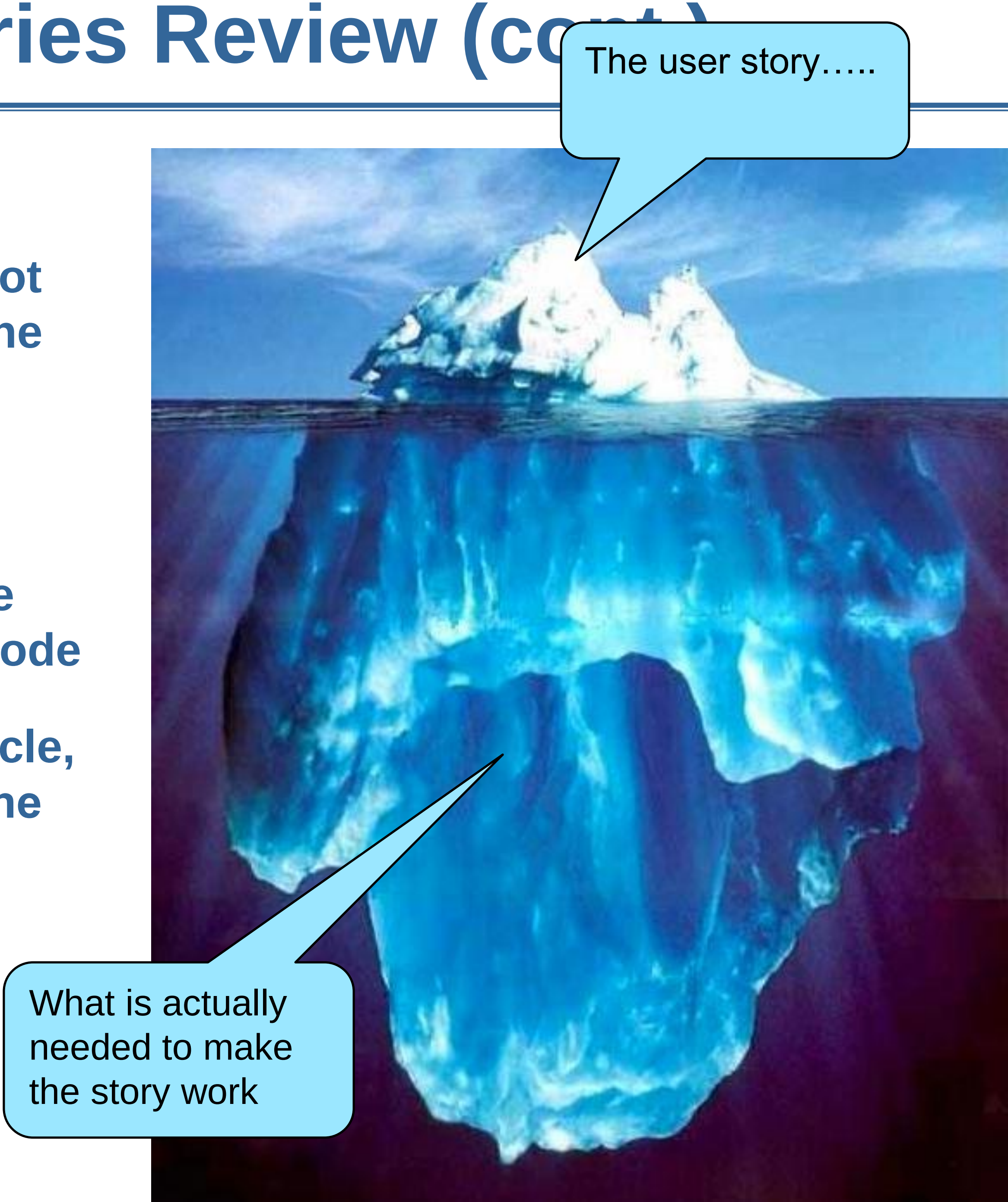


User Stories Review (cont.)

- Each user story will be simple but will require a lot of things to work under the hood

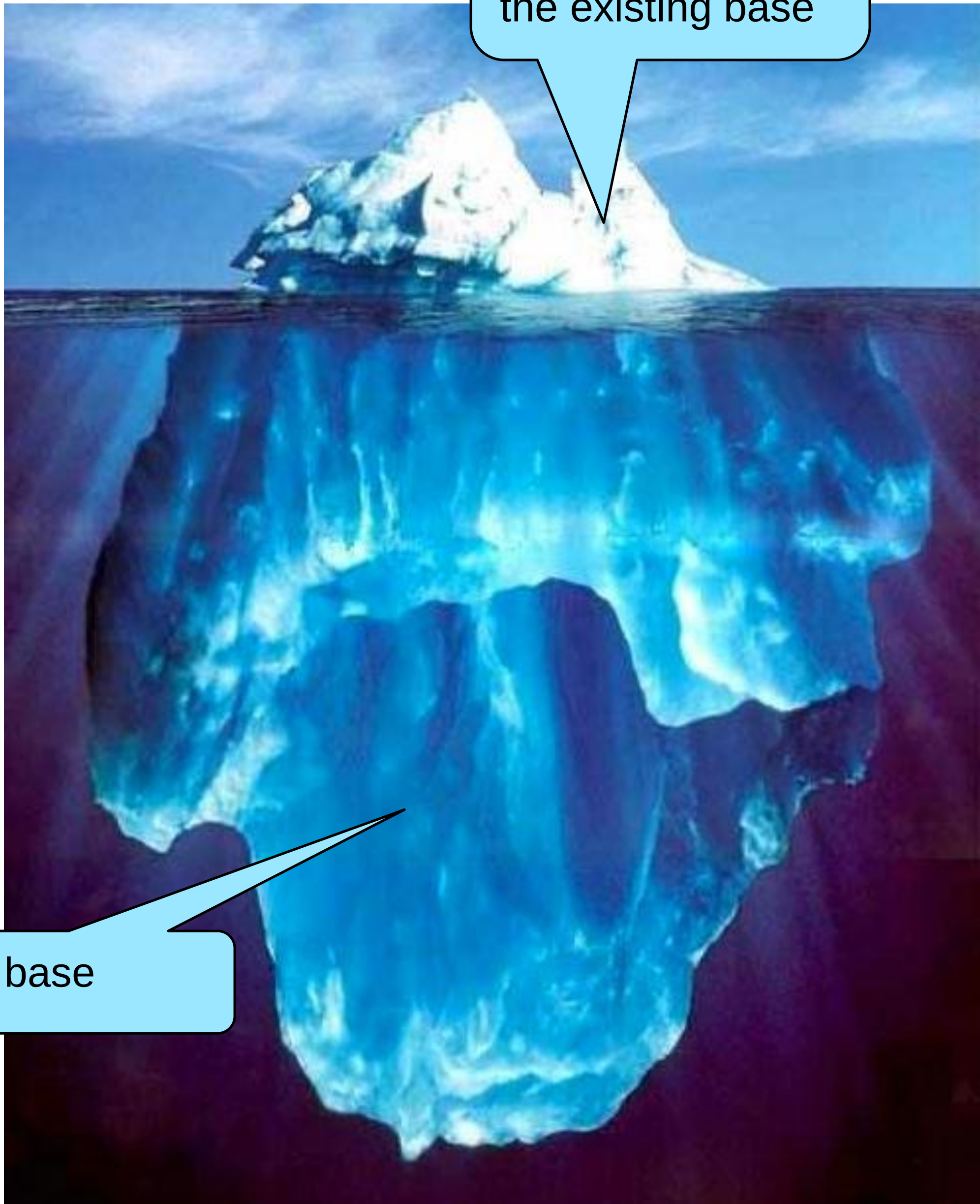
- User stories emphasize working fully-integrated software rather than large bodies of un-integrated code

- At the end of the build cycle, if a user can't complete the story, it isn't finished



User Stories Review (cont.)

- 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

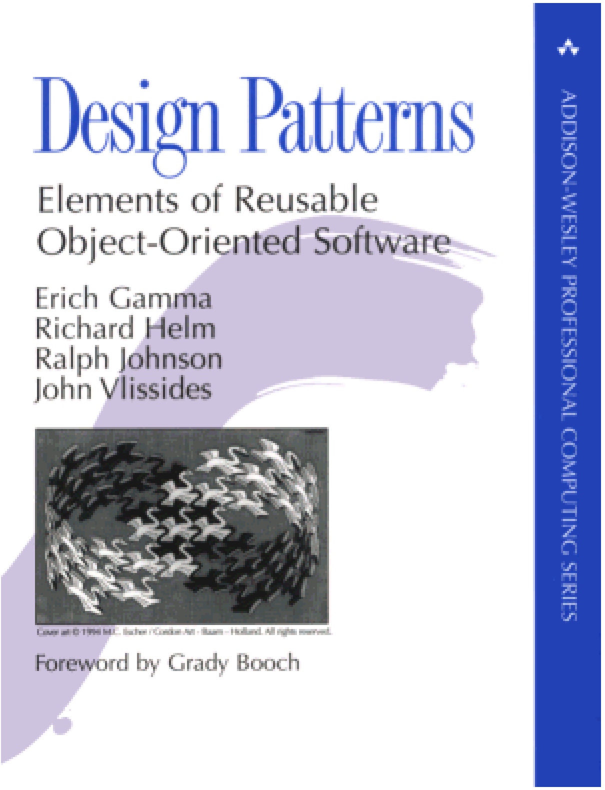


Later stories can be integrated into the existing base

Hidden base

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 (and it must 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 could 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 consistent (*e.g.*, what you get when run the Eclipse automated code formatter or indent-region in Emacs)
  - 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_ = ….;
- **You must use an open source license**
  - License headers should be at the top of each source file!
  - I recommend the Apache License v2
    - http://www.apache.org/dev/apply-license.html
  - Talk to me if you *<u>need</u>* closed source.

Example Apache License Header

```
/***************************************************************************
 * Copyright 2019 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 GitHub
- You must maintain a wiki (in GitHub) 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

Commit Rules for Artifacts

- 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.

Commit Rules (cont.)

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

Commit Rules (cont.)

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

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 …).

2 Week Development Cycle Reqts

- **Every user story needs at least one associated unit test or integration test.**
- **Comments need to be specified accurately (*e.g.*, using “UT3:2” for unit tests).**
- **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 GitHub)**
  - Issues/bugs should be showing up and getting resolved.

- **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**

2 Week Development Cycle

- **3rd class session:**
  - Initial story implementations turned in (checked into GitHub before class)
  - In-class code reviews of user story implementations
  - Bug/Issue discussions
  - Test skeletons/coverage/strategies
  - Advanced Java/C++/Software Engineering topic introduced (time permitting)

- **4th class session:**
  - Code refactored per code review recommendations (checked into GitHub before class)
  - Binary distributions made available as file releases (checked into GitHub before class)
  - User stories demoed
  - In-class system testing *(performed by non-team members)*

2 Week Development Cycle (3rd Class Session)

- **3rd class session:**
  - Initial story implementations turned in (checked into GitHub 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?
  - Testing
    - Coverage
    - Skeletons
    - Unit tests, system tests

CIS320 Development Cycle (4th Class Session)

- **4th class session:**
  - MAIN EMPHASIS: In-class system testing
    - One team runs the user stories for another project
  - Code refactored per code review recommendations (checked into Github before class)
    - Teams present refactoring work
    - Briefly describe bugs reported
  - Binary distributions made available as file releases (checked into Github before class)
  - User stories demoed **(as time allows)**
    - Each team demos the user stories for the cycle
  - Lessons learned for projects
  - Lessons learned for cycle
    - Different structure, interaction, format helpful in class

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.**
- **In the 3rd class session, 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 4th class session 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/We 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 for the 3rd class session.

- **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 System Testing

- On the last (i.e., 4th) class session of a build cycle, as time allows we will let each team <u>*briefly*</u> 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
  - An Eclipse plugin distribution
  - A set of project binaries and an ANT file to run them
  - A C++ executable for the target environment
  - Working website

Bi-weekly Grading

- **(16 pts) Were all of the user stories completed or properly postponed?**
- **(16 pts) Were adequate tests created and executed for the code?**
- **(16 pts) Were bugs properly reported and addressed?**
- **(16 pts) Did the new features pass system testing?**
- **(16 pts) Does the code adhere to the development standards and was it refactored after the code review?**
- **Time documented/spent – I will check this from Weekly Hours**
  - 12 hours/2-week cycle is the expected minimum
- *****I reserve the right to change the weighting/grading criteria during the semester**