CS 361
Software Engineering I

Individuals and Interactions

The Prey, the Pack, and the Hunt

- Your goal is to meet your customer's needs.
- That goal, and nothing else, is the prey.
  - Not throwaway prototypes
  - Not documentation
  - Not models

- You will work as a team to obtain your goal.

Identifying Your Prey

- Extreme Programming offers three ways to identify your goal.
  - User stories
  - Acceptance tests
  - Unit tests

User Stories Review

- How big should a story be?
  - 3x5” card
- How many user stories is the customer allowed to generate?
  - As many as desired
- When can the engineer prioritize the stories?
  - Never. The customer does the prioritization

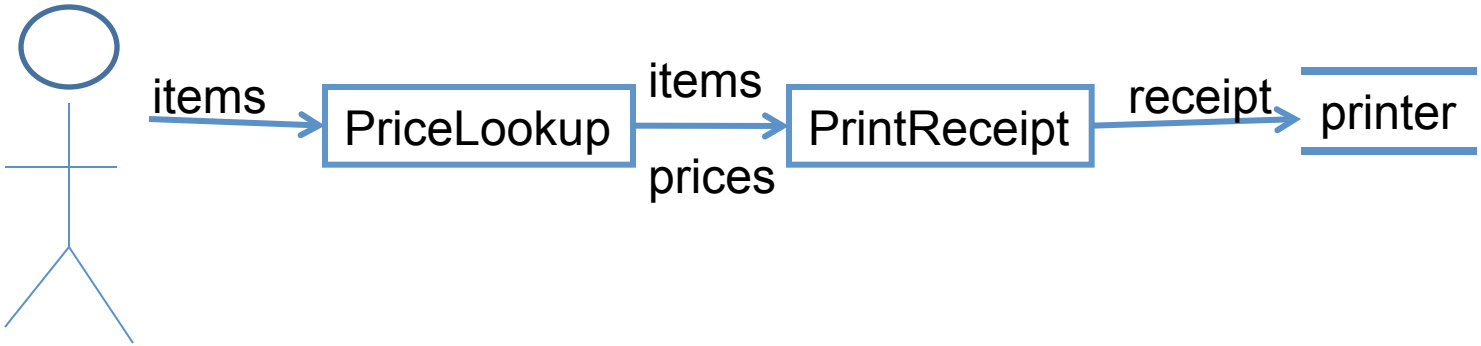
Acceptance Tests & User Stories

- Each user story may have multiple acceptance tests.
- A single acceptance test validates an aspect of a user story by exercising the system the way that a user would.
- Acceptance tests tell what the customer will use to judge success.
- Main benefit of acceptance tests: you know when you can stop developing!

Example Tests

- User Story: Generate a Receipt

"As items are entered by the clerk, their name and price are added to a receipt, and the price is added to the subtotal."



Example Tests

- User Story: Generate a Receipt

"As items are entered by the clerk, their name and price are added to a receipt, and the price is added to the subtotal."

- Unit tests for PriceLookup component

Input: 6-pack of beer; Verify: $6.47

Input: milk; Verify: $2.00

Example Tests

- User Story: Generate a Receipt

"As items are entered by the clerk, their name and price are added to a receipt, and the price is added to the subtotal."

- Unit tests for PrintReceipt component

Input: Beer $6.47, Milk $2.00, Beer $6.47

Verify: Receipt lists items and total is $14.94

Example Tests

- User Story: Generate a Receipt

"As items are entered by the clerk, their name and price are added to a receipt, and the price is added to the subtotal."

- Acceptance test (combines several unit tests)

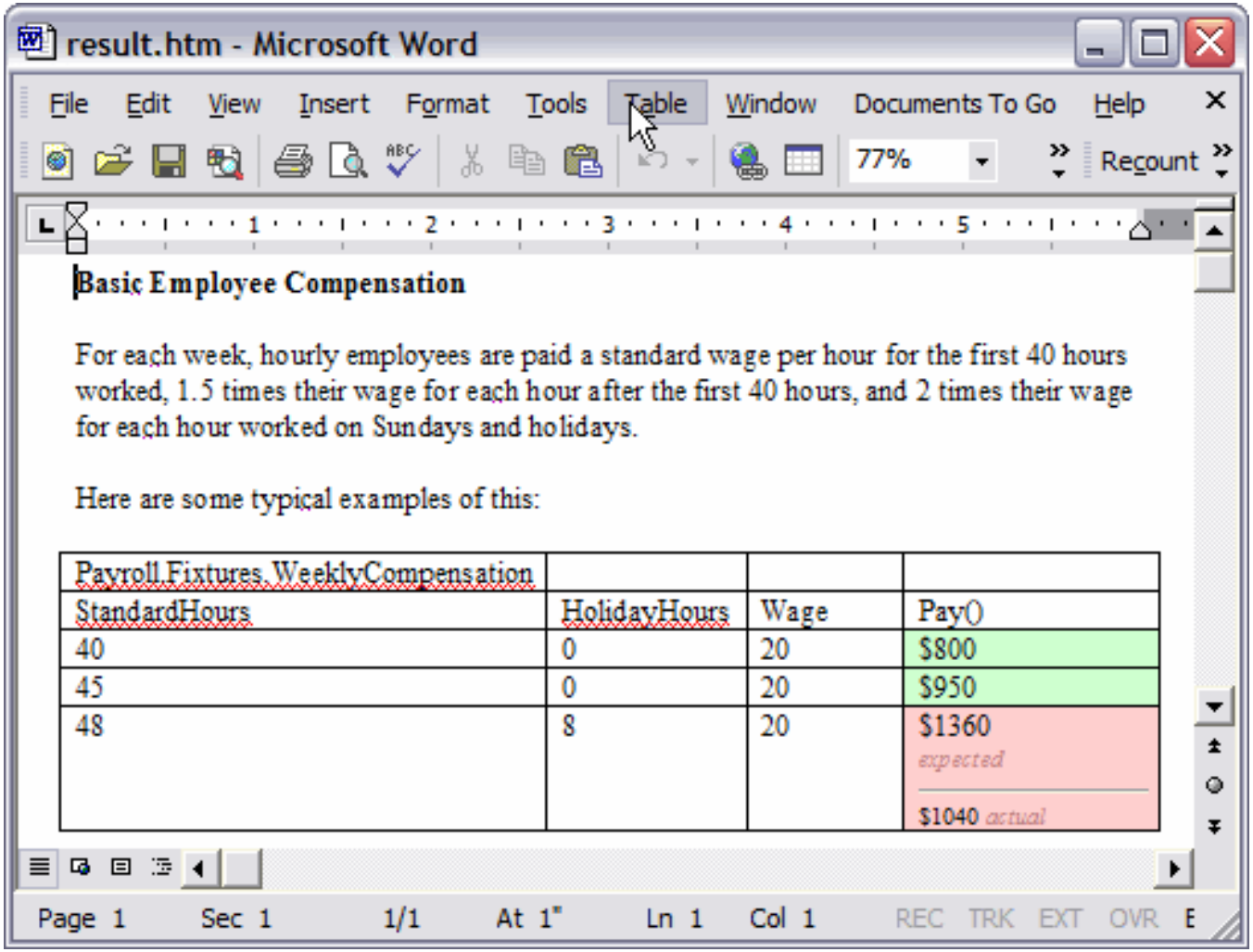
Test: Scan 1 6-pack, 1 milk, 1 6-pack

Verify: Receipt lists items and total is $14.94

About Creating Acceptance Tests

- Ideally, the customer writes the acceptance tests
- Acceptance tests are blackbox tests
  - The test writer doesn't get to read the tested code
- Tests should be unambiguous

FIT Framework



http://fit.c2.com/

Junit Fixture

```
package junitfaq;

  import org.junit.*;
  import static org.junit.Assert.*;
  import java.util.*;

  public class SimpleTest {

     private Collection<Object> collection;

     @Before
     public void setUp() {
        collection = new ArrayList<Object>();
     }

     @Test
     public void testEmptyCollection() {
        assertTrue(collection.isEmpty());
     }


     @Test
     public void testOneItemCollection() {
        collection.add("itemA");
        assertEquals(1, collection.size());
     }
  }
```

About Running Acceptance Tests

- Run acceptance tests at least once a week, preferably more often.

- Often useful to have an integration machine where you run acceptance tests.
  - Representative of what the customer would have.

Unit Tests Review

- Who writes the unit tests?
  - Programmer
- When do the unit tests get written?
  - Before writing the application code

Benefits of Unit Tests

- Clarifies the task at hand
- Frees you from writing perfect code (temporarily)
- Makes reliable refactoring possible

About Creating Unit Tests

- Unit tests start out as blackbox tests and become whitebox tests
  - They are first created for non-existent code.
  - At any time, you can read your (new) application code and revise the test code

- Exhaustive testing is usually infeasible
- So use representative testing instead
  - Typical inputs
  - Boundary cases

Writing the First Test is Hardest

- But keep in mind…
  - "Your test suite is more valuable than your code"
  - "If you want a good suite of tests next year you must start collecting them today."

Time to Code

- You only code when the test suite is broken or you are refactoring
- Your twin goals
  - Pass the unit tests for today's user stories
  - Refactor to keep the bad smells under control

The Hunt is Not Yours Alone

- It's not your code. It's the team's code.
  - "Egoless programming doesn't work; expand your ego to include everyone's code."

- ANYBODY can edit ANY code that has been checked in.

Pair Programming- Benefits

- Real-time code reviews
- Copilot can help to catch implicit assumptions
- Pair programming takes 15% longer but leads to 15% fewer bugs… a "win" for many projects
- Ideas and techniques spread throughout the group

Pair Programming- The Driver

- The driver…
  - controls keyboard and mouse
  - is actively talking to the copilot
  - explains intent of code and where s/he is going

Pair Programming- The Copilot

- Copilot is not a passenger! Copilots pay attention, watch for mistakes, offer ideas
  - Talk about and agree upon best course of action
  - If you can't agree, then take turns yielding to one another's preferences
- At any moment, copilot can request to switch places and become the driver.
  - Driver can say, "hold on a second" but should switch as soon as possible
  - You are working as a pair, a team.

Pair Programming- Principles

- If someone asks you to pair with him or her and you can, then you must say yes.
- Code formatting: pick a standard as a team, enforce it, and move on
- Font size and eyesight: choose a font large enough for the copilot to see, too
- Take turns pairing with lots of different people
- If you write code alone, then it *must* be rewritten.
  - If you have a brilliant idea when you're alone, then write it down.

Interacting With the Team

- Periodically communicate your progress
  - An end-of-day email is usually perfect
  - Perhaps track progress with wiki, emails or Google Docs?
- Use models judiciously
  - Draw diagrams to help you reason
  - Draw diagrams to communicate
  - But only draw diagrams if it helps your team to get the prey

In agile processes, models…

- Must provide **value**
- Must fulfill a purpose
- Must be **understandable**
- Must be as **simple** as possible
- Must be **sufficiently**
  - accurate
  - concise
  - detailed

Not true about models in agile

- Models equal documentation.
- Modeling implies a heavyweight process.
- Modeling freezes the requirements.
- Models never change.
- Models are complete.
- Models must be created with a tool.
- All developers know how to model properly.
- Modeling is a waste of time.
- The data model is the only one that matters.

Tips for Agile Modeling

- Don't let the models distract from the hunt
- Model iteratively and incrementally
- Model with other people
  - Involve stakeholders
  - Own models collectively
  - Display models publicly
    - (share on Google Docs or Office on the Web?)
- Create simple content
- Use the simplest tools

More Tips About Agile Modeling

- Don't spend time getting every line straight.
- Follow standards
- Reuse existing resources
- Discard temporary models