What’s So Hard About Writing Good Use Cases?

I understand the requirements, but what does it actually do?

- Lack of context with traditional functional requirements.
  - What do you really use the system for?
  - How does it work?
  - Why are you paying me to build this?

People like stories…

- From one point of view, use cases are just stories about how people (or other things) use a system to perform some task.
- But stories about requirements are nothing new.
- What makes use cases special?

Use cases are a good way to tell stories about requirements

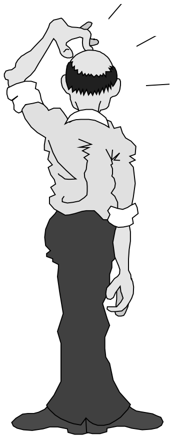
- A “liberating form” which releases our creativity by give us a semiformal framework for structuring the stories.
  - Actors,
  - use cases,
  - Associations
- Describe the system requirements for the error situations.

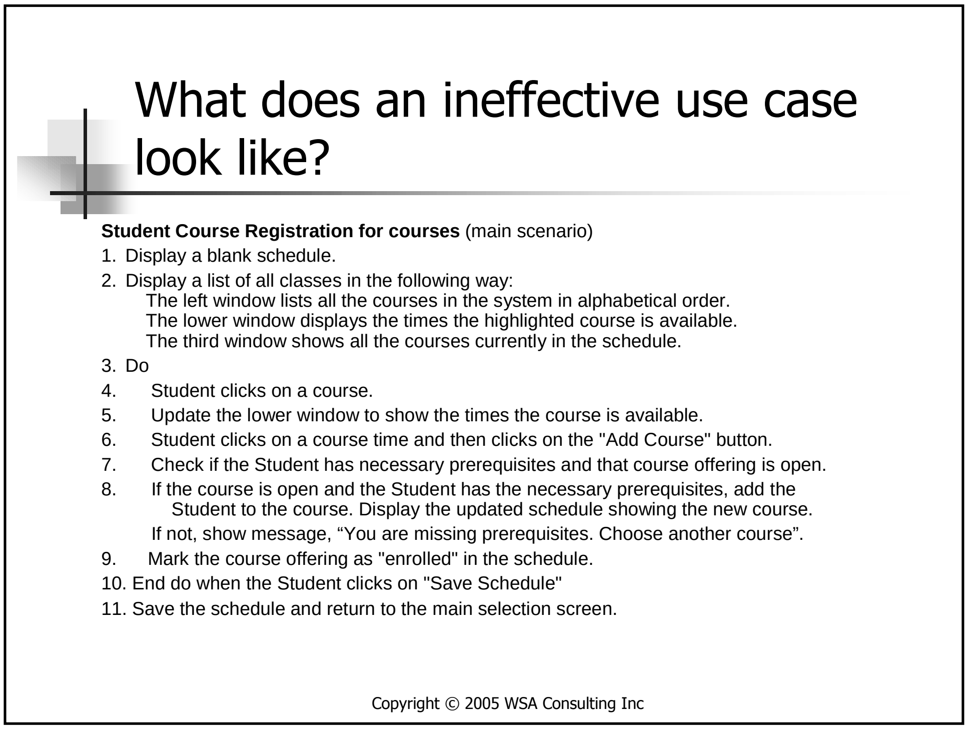
Use cases are a good way to organize requirements.

- Provide good scaffolding on which to hang other project information.
  - Business rules
  - External interfaces
  - Data requirements
- A popular element of object-oriented software development methodologies (Jacobson, Larman).

However, good stories are hard to write.

- Everyone thought writing use cases was going to be easy.
- Easy to read, does not mean easy to write.
- Use cases are stories, prose essays, and so bring along all the associated difficulties of story writing in general.

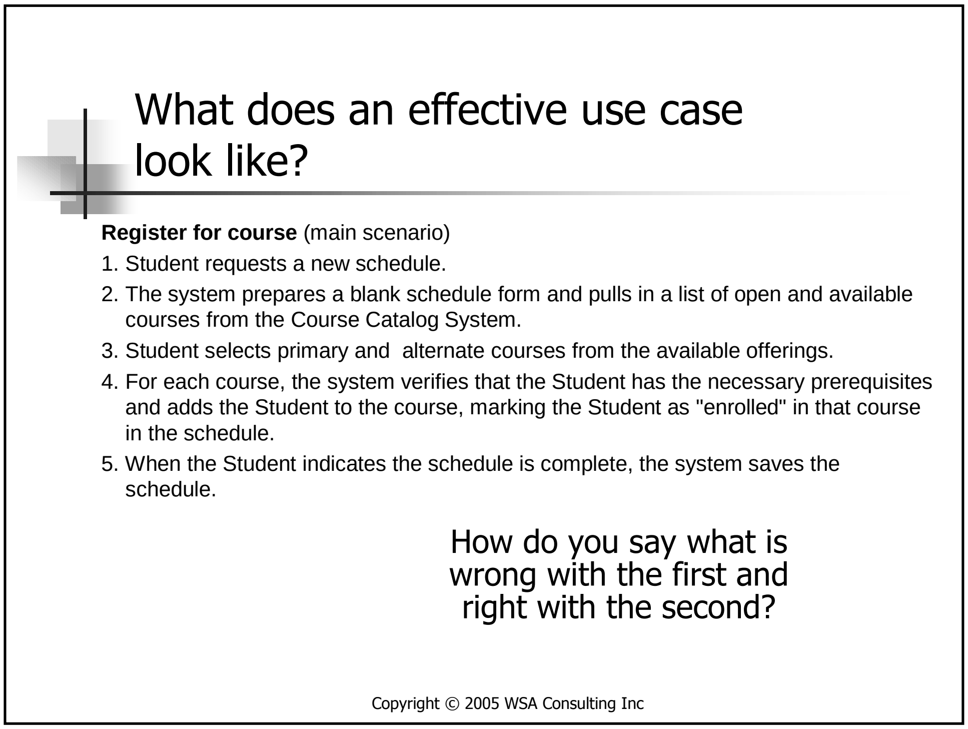




What does an ineffective use case look like?

Student Course Registration for courses (main scenario)

1. Display a blank schedule.
2. Display a list of all classes in the following way:
   The left window lists all the courses in the system in alphabetical order.
   The lower window displays the times the highlighted course is available.
   The third window shows all the courses currently in the schedule.
3. Do
4. Student clicks on a course.
5. Update the lower window to show the times the course is available.
6. Student clicks on a course time and then clicks on the "Add Course" button.
7. Check if the Student has necessary prerequisites and that course offering is open.
8. If the course is open and the Student has the necessary prerequisites, add the Student to the course. Display the updated schedule showing the new course.
   If not, show message, “You are missing prerequisites. Choose another course”.
9. Mark the course offering as "enrolled" in the schedule.
10. End do when the Student clicks on "Save Schedule"
11. Save the schedule and return to the main selection screen.



What does an effective use case look like?

Register for course (main scenario)

1. Student requests a new schedule.
2. The system prepares a blank schedule form and pulls in a list of open and available courses from the Course Catalog System.
3. Student selects primary and alternate courses from the available offerings.
4. For each course, the system verifies that the Student has the necessary prerequisites and adds the Student to the course, marking the Student as "enrolled" in that course in the schedule.
5. When the Student indicates the schedule is complete, the system saves the schedule.

How do you say what is wrong with the first and right with the second?

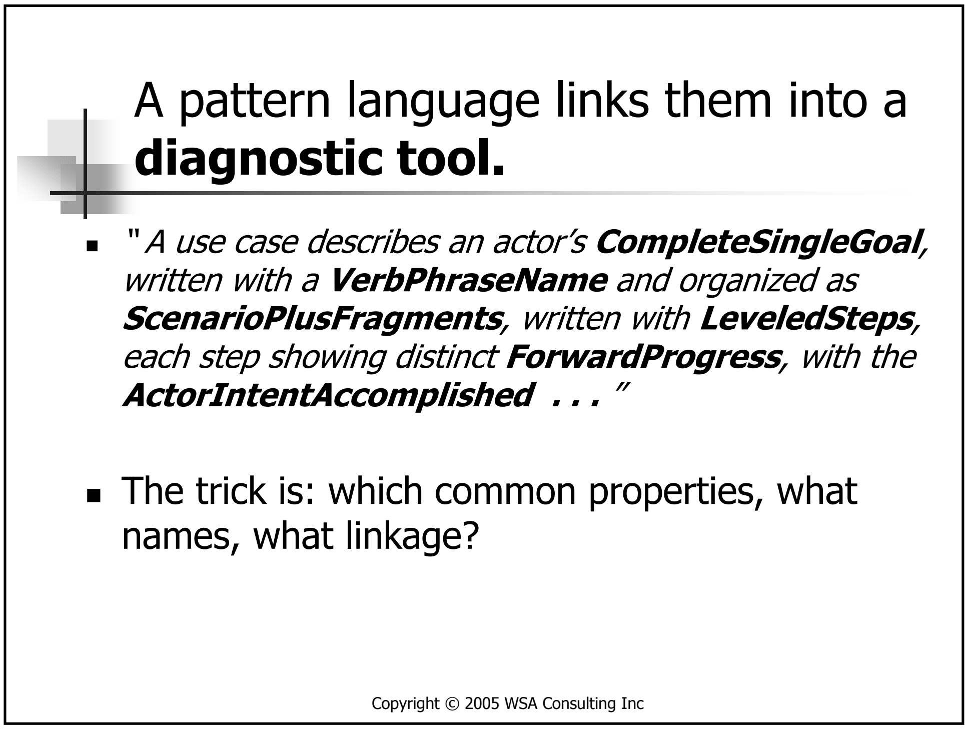
What are you using use cases for? Who is using them?

- MultipleForms:
  - Different projects need different degrees of formality in their work, and various people have different preferences for the template. Requiring everyone to use the same use case template is counterproductive.

Patterns describe **properties of good examples**: signs of quality**.**

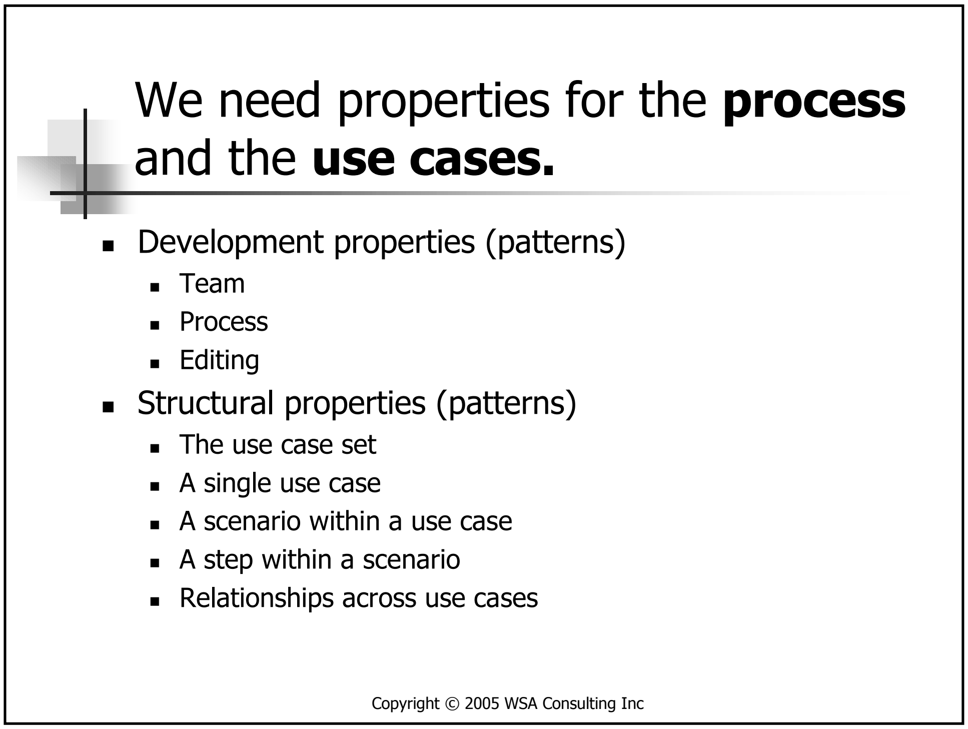
- The take-home value of a pattern is its **name**!
  - Content should be fairly clear from the name.
  - The names let you connect the ideas:

*"Split a large **ParticipatingAudience** into **SmallWritingTeams**; then use a **TwoTierReview** to gather feedback economically."*



A pattern language links them into a **diagnostic tool.**

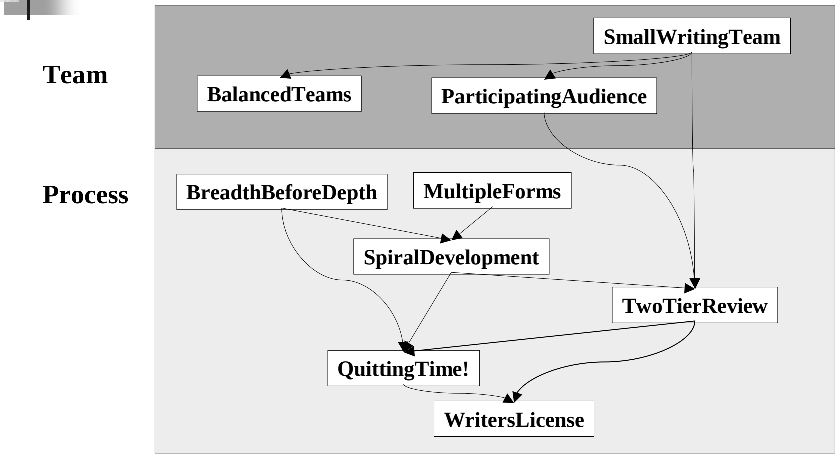
- "*A use case describes an actor's **CompleteSingleGoal**, written with a **VerbPhraseName** and organized as **ScenarioPlusFragments**, written with **LeveledSteps**, each step showing distinct **ForwardProgress**, with the **ActorIntentAccomplished . . .** "*
- The trick is: which common properties, what names, what linkage?



We need properties for the **process** and the **use cases.**

- Development properties (patterns)
  - Team
  - Process
  - Editing
- Structural properties (patterns)
  - The use case set
  - A single use case
  - A scenario within a use case
  - A step within a scenario
  - Relationships across use cases

The development patterns say what’s present during writing.



Team

Process

SmallWritingTeam

BalancedTeams

ParticipatingAudience

BreadthBeforeDepth

MultipleForms

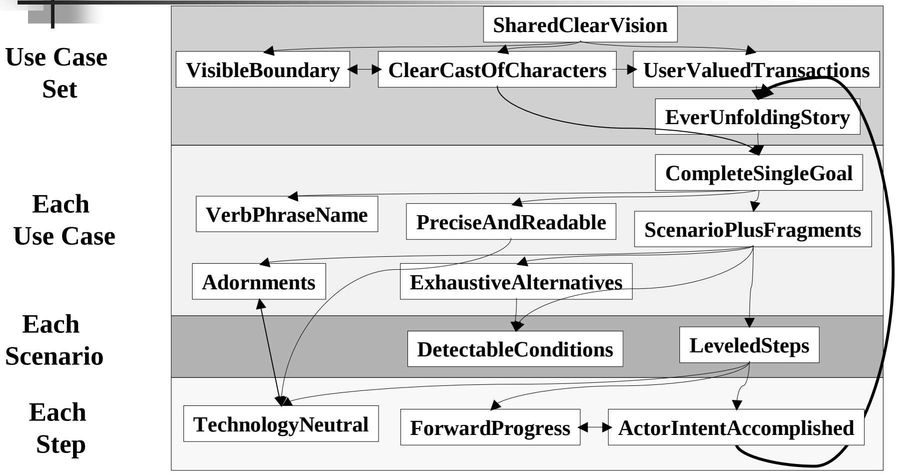
SpiralDevelopment

TwoTierReview

QuittingTime!

WritersLicense

The structural patterns say what is present in the use cases



Use Case Set

Each Use Case

Each Scenario

Each Step

SharedClearVision

VisibleBoundary

ClearCastOfCharacters

UserValuedTransactions

EverUnfoldingStory

CompleteSingleGoal

VerbPhraseName

PreciseAndReadable

ScenarioPlusFragments

Adornments

ExhaustiveAlternatives

DetectableConditions

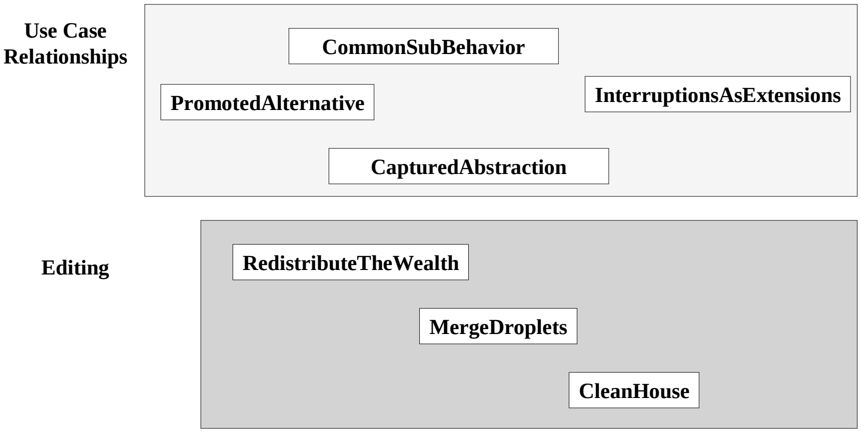
LeveledSteps

TechnologyNeutral

ForwardProgress

ActorIntentAccomplished

More Structural patterns . . .
More Development patterns...



Use Case Relationships

CommonSubBehavior

PromotedAlternative

InterruptionsAsExtensions

CapturedAbstraction

Editing

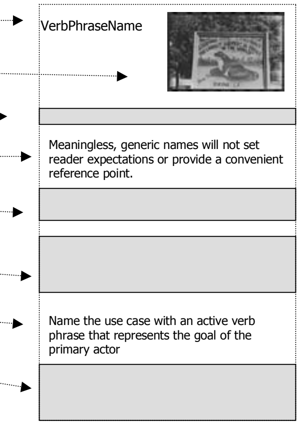
RedistributeTheWealth

MergeDroplets

CleanHouse

Use Case Pattern Form

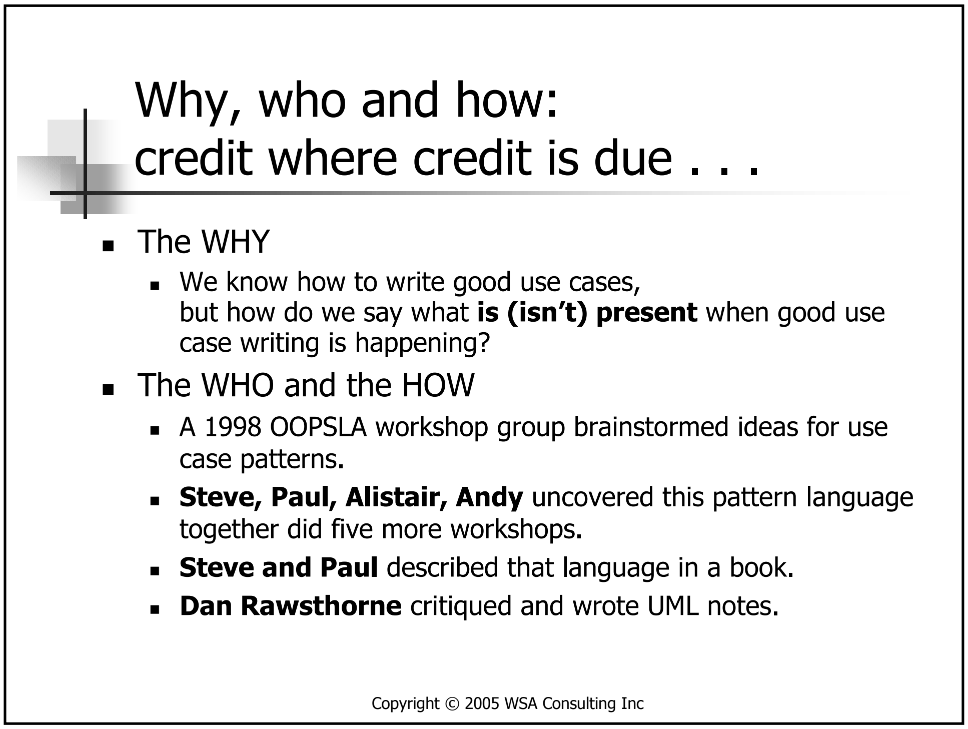
- Name
- Photograph
- Context
- Problem
- Metaphoric story
- Forces
- Solution
- Examples



VerbPhraseName

Meaningless, generic names will not set reader expectations or provide a convenient reference point.

Name the use case with an active verb phrase that represents the goal of the primary actor



Why, who and how: credit where credit is due . . .

- The WHY
  - We know how to write good use cases,
    but how do we say what **is (isn't) present** when good use case writing is happening?
- The WHO and the HOW
  - A 1998 OOPSLA workshop group brainstormed ideas for use case patterns.
  - **Steve, Paul, Alistair, Andy** uncovered this pattern language together did five more workshops.
  - **Steve and Paul** described that language in a book.
  - **Dan Rawsthorne** critiqued and wrote UML notes.