CS 320

Introduction to Software Engineering
Spring 2017

February 22, 2017

Overview: the big picture

- **Software processes, requirements, and specification**
  - Learn about different software development processes.
  - Learn how to write a requirements document and a specification.
- **Software development**
  - Learn how to decompose a complex problem and build abstractions.
  - Learn about best practices.
  - Improve your coding skills.
- **Software testing and debugging**
  - Learn how to write (unit) tests.
  - Hands-on experience, using testing and debugging techniques.

Overview: the big picture

- **Software processes, requirements, and specification**
  - Learn about different software development processes.
  - Learn how to write a requirements document and a specification.
- **Software development**
  - Learn how to decompose a complex problem and build abstractions.
  - Learn about best practices.
  - Improve your coding skills.
- **Software testing and debugging**
  - Learn how to write (unit) tests.
  - Hands-on experience, using testing and debugging techniques.

Today

Recap: Entity-Relationship (ER) diagrams

Use cases

- Textual vs. graphical representation

Paper discussion

- Purposes, Concepts, Misfits, and a Redesign of Git

Open discussion

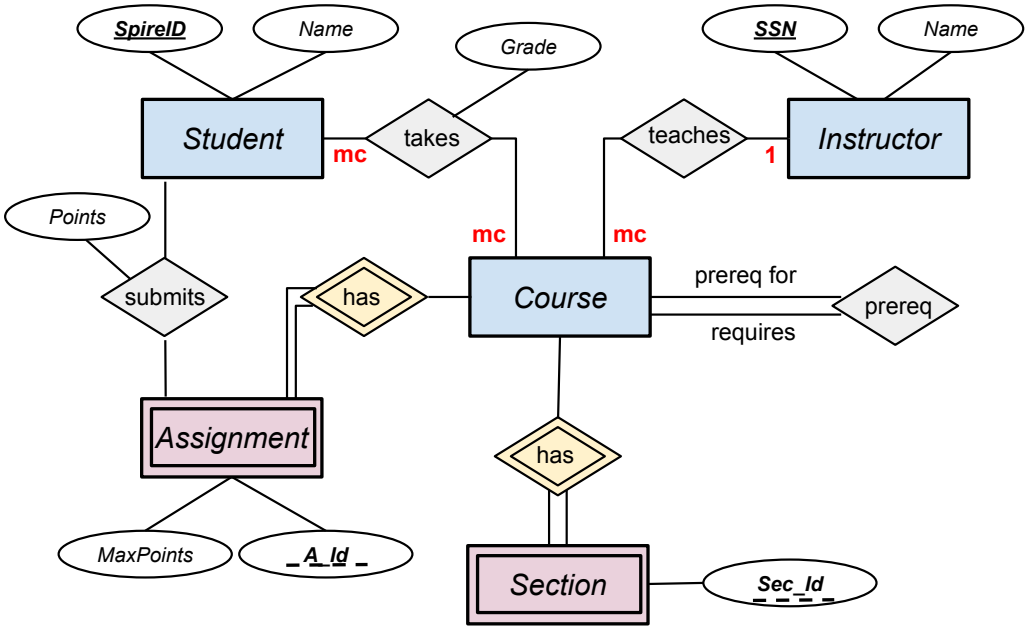
- Purposes and concepts in your class projects.

Recap: ER diagrams

Recall our model for a simple course registration system at UMass:

- Students
- Instructors
- Courses
- Sections
- Prerequisites
- Assignments
- Points/grades

Recap: ER diagrams



Use cases: overview

"A use case captures a contract between the stakeholders of a system about its behavior. The use case describes the system's behavior under various conditions as it responds to a request from one of the stakeholders, called the primary actor. The primary actor initiates an interaction with the system to accomplish some goal. The system responds, protecting the interests of all the stakeholders. Different sequences of behavior, or scenarios, can unfold, depending on the particular requests made and conditions surrounding the requests. The use case collects together those different scenarios."

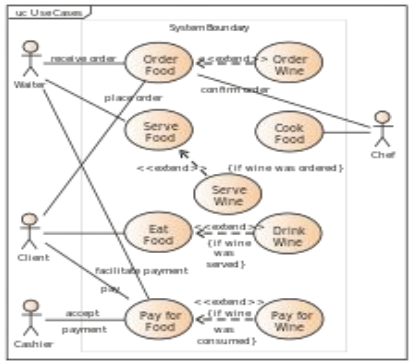
Writing Effective Use Cases, Alistair Cockburn, 2000.

See *Writing effective use cases* on Moodle.

Use case diagrams

"For reasons that remain a mystery to me, many people have focused on the stick figures and ellipses in use case writing since Jacobson's first book came out, and neglected to notice that use cases are fundamentally a text form."

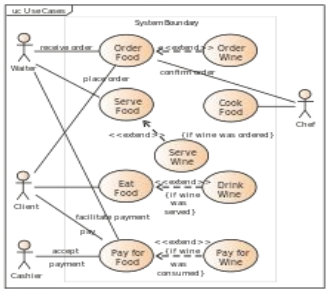
Writing Effective Use Cases, Alistair Cockburn, 2000.



Textual use case

Name	The Use Case name. Typically the name is of the format <action> + <object>.
ID	An identifier that is unique to each Use Case.
Description	A brief sentence that states what the user wants to be able to do and what benefit he will derive.
Actors	The type of user who interacts with the system to accomplish the task. Actors are identified by role name.
Organizational Benefits	The value the organization expects to receive from having the functionality described. Ideally this is a link directly to a Business Objective.
Frequency of Use	How often the Use Case is executed.
Triggers	Concrete actions made by the user within the system to start the Use Case.
Preconditions	Any states that the system must be in or conditions that must be met before the Use Case is started.
Postconditions	Any states that the system must be in or conditions that must be met after the Use Case is completed successfully. These will be met if the Main Course or any Alternate Courses are followed. Some Exceptions may result in failure to meet the Postconditions.
Main Course	The most common path of interactions between the user and the system. 1. Step 1 2. Step 2
Alternate Courses	Alternate paths through the system. AC1: <condition for the alternate to be called> 1. Step 1 2. Step 2 AC2: <condition for the alternate to be called> 1. Step 1
Exceptions	Exception handling by the system. EX1: <condition for the exception to be called> 1. Step 1 2. Step 2 EX2 <condition for the exception to be called> 1. Step 1

Use case diagrams



vs.

Name	The Use Case name. Typically the name is of the format <action> + <object>.
ID	An identifier that is unique to each Use Case.
Description	A brief sentence that states what the user wants to be able to do and what benefit he will derive.
Actors	The type of user who interacts with the system to accomplish the task. Actors are identified by role name.
Organizational Benefits	The value the organization expects to receive from having the functionality described. Ideally this is a link directly to a Business Objective.
Frequency of Use	How often the Use Case is executed.
Triggers	Concrete actions made by the user within the system to start the Use Case.
Preconditions	Any states that the system must be in or conditions that must be met before the Use Case is started.
Postconditions	Any states that the system must be in or conditions that must be met after the Use Case is completed successfully. These will be met if the Main Course or any Alternate Courses are followed. Some Exceptions may result in failure to meet the Postconditions.
Main Course	The most common path of interactions between the user and the system. 1. Step 1 2. Step 2
Alternate Courses	Alternate paths through the system. AC1: <condition for the alternate to be called> 1. Step 1 2. Step 2 AC2: <condition for the alternate to be called> 1. Step 1
Exceptions	Exception handling by the system. EX1: <condition for the exception to be called> 1. Step 1 2. Step 2 EX2: <condition for the exception to be called> 1. Step 1

What is the main purpose of a use case?

Which representation should you choose?

Are the graphical and textual representation interchangeable?

Paper discussion

Purposes, Concepts, Misfits, and a Redesign of Git

(Quotes and graphs in the slides are taken from the paper.)

Concept and motivating purpose

"A **concept** is something you need to understand in order to use an application (and also something a developer needs to understand to work effectively with its code) and is invented to solve a particular problem, which is called the **motivating purpose.**"

Concept and motivating purpose

“A **concept** is something you need to understand in order to use an application (and also something a developer needs to understand to work effectively with its code) and is invented to solve a particular problem, which is called the **motivating purpose.**”

What are other examples for concepts and motivating purposes?

Operational principle and misfit

“A concept is defined by an **operational principle**, which is a scenario that illustrates how the concept fulfills its motivating purpose.”

Operational principle and misfit

“A concept is defined by an **operational principle**, which is a scenario that illustrates how the concept fulfills its motivating purpose.”

“A concept may not be entirely fit for purpose. In that case, one or more **operational misfits** are used to explain why. The operational misfit usually does not contradict the operational principle, but presents a different scenario in which the prescribed behavior does not meet a desired goal.”

Relationship between concepts and purposes

Motivation

Each concept should be motivated by at least one purpose.

Relationship between concepts and purposes

Motivation
Each concept should be motivated by at least one purpose.

Coherence
Each concept should be motivated by at most one purpose.

Relationship between concepts and purposes

Motivation
Each concept should be motivated by at least one purpose.

Coherence
Each concept should be motivated by at most one purpose.

Fulfillment
Each purpose should motivate at least one concept.

Relationship between concepts and purposes

Motivation
Each concept should be motivated by at least one purpose.

Coherence
Each concept should be motivated by at most one purpose.

Fulfillment
Each purpose should motivate at least one concept.

Non-division
Each purpose should motivate at most one concept.

Relationship between concepts and purposes

Motivation
Each concept should be motivated by at least one purpose.

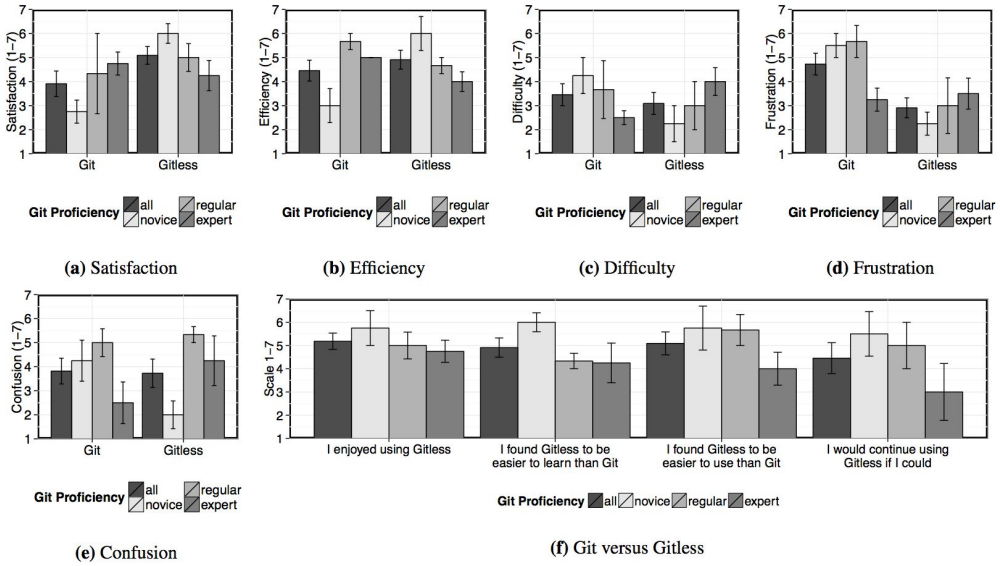
Coherence
Each concept should be motivated by at most one purpose.

Fulfillment
Each purpose should motivate at least one concept.

Non-division
Each purpose should motivate at most one concept.

Decoupling
Concepts should not interfere with one another’s fulfillment of purpose.

Gitless vs. Git: evaluation results



(a) Satisfaction (b) Efficiency (c) Difficulty (d) Frustration

(e) Confusion (f) Git versus Gitless

Gitless vs. Git: questionnaire results

- "Gitless was easier to use for the tasks these sessions asked me to perform, but I really like having a Git stash and staging area to work with in Git"
- "the ability to walk away from a branch in any state is very useful and would go far in helping new git users"
- "However, I make heavy use of the staging area and interactive rebase and I would not be willing to part with either..."

Purposes, Concepts, Misfits, and a Redesign of Git

Paper discussion

- Do you agree with the authors' reasoning about git?
- What are the main conclusions of the evaluation?
- Would you prefer gitless over git? Why or why not?

Misfits

- Switching Branches
- Renaming a file
- Tracking a new file
- Tracking an empty directory

Violated principles

- *Unmotivated concept*: stash
- *Unfulfilled purposes*: renaming a file and tracking and empty directory
- *Coupled concepts*: tracking a new file

Purposes and concepts in your class projects

Open discussion

- Name one purpose and one corresponding concept for your class project.
- How are purposes and concepts related to requirements and software design?

"A **concept** is something you need to understand in order to use an application (and also something a developer needs to understand to work effectively with its code) and is invented to solve a particular problem, which is called the **motivating purpose.**"