Users and Use Cases

- User Characterization
- Use Cases
  - what are they, why they are interesting
  - how they differ from specifications
- Use Case Development
  - identifying and characterizing classes of users
  - identifying and elaborating usage scenarios
- Representing Use Cases
  - XP user stories
  - UML (use case, activity, state) diagrams

User Characterization

- User Characterization
  - identifying distinct sub-classes of users
- Common Criteria
  - product use roles (or intentions)
  - knowledge (domain or technology)
- Results in a profile for each user class
  - goals and expectations
  - knowledge and experience
  - typical usage scenarios

Use Case

A stylized story (from a user's perspective) about how an end user (in a role, with a goal) interacts with the system under a specific set of circumstances.

In requirements, it captures a contract that describes (at a high level) the way the system should function in the face of a specified user goal.

Use Cases vs. Specifications

- Specifications: component descriptions
  - required behavior and other characteristics
- Use Cases are "user-centric stories"
  - scenarios the product must support
  - key instances of user/product interactions
- Use Cases are a better starting point
  - they are easier to gather and review
  - they more honestly capture requirements
  - entry-points for exploration of users' world
  - they do not describe implementations

Scenarios, Tasks, & Steps

- Scenario … representative work session
  a sequence of related tasks to solve a problem
  (e.g. handle a customer phone call)
- Task … smallest interesting unit of work
  sequence of steps culminating in useful a result
  (e.g. scheduling an appointment)
- Step … smallest interesting unit of action
  one individual action in a sequence
  (e.g. entering a password)

Use Cases

- Are a really good form for requirements
  - they tend to be concise
  - they tend to be very concrete
  - they are tied to real-world problems
  - they are expressed from the users' perspective
  - they describe functionality (vs. design)
  - they help us understand the user's world view
- They are also easy to develop/validate
  - ask people what they do, or simply watch them

Developing Use Cases (i)

- They can be based on existing processes
  - interview potential users
  - ask them to describe the things they do
  - identify the tasks within their scenarios
  - get descriptions of steps within those tasks
  - get descriptions of problems and exceptions
- Use these as a starting point
  - design to enable the same tasks & scenarios
  - design to deal with the problems & exceptions

Developing Use Cases (ii)

- They can be brain-stormed
  - who are the primary actors?
  - what tasks do they need to perform?
  - what information is needed to perform task?
  - what information is user interested in?
  - what could go wrong in performing task?
  - how would user want to be informed of these?
- These suppositions must be validated
  - by interviews
  - by usability testing with prototypes

Representing Use Cases

- Use cases capture product capabilities
  - describe what product should be able to do
  - may describe detailed user/system interactions
- They can be represented in many ways
  - XP user stories overview of an operation
  - UML Diagrams
    - use case actors, objects, operations among them
    - behavior detailed interactions among actors & objects
    - state observable states and transitions
  - user interface prototypes or mock-ups

XP User Story Cards

- Brief summary of a desired capability
  - name of story
  - brief general description of a useful behavior
  - user assigned priority
  - development estimated cost
- All on a single 3x5 card
  - just enough to capture the concept
  - specifically not a complete specification
    - this will be established between user and developer

Sample User Story Card

Story #107: buy parking permit

Any registered student can buy a parking permit.
Can choose semester or annual.
Need to provide car license plate number.
Payment options:
credit card, PayPal, charge to student account

Priority: high Estimate: 9 points

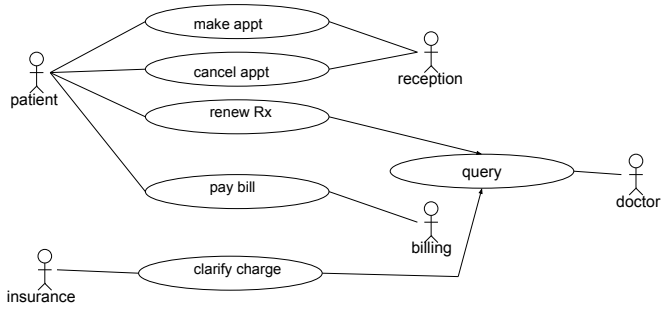
Using User Story Cards

- Product Descriptions
  - a product is a collection of features/capabilities
- Work planning tokens (the planning game)
  - each represents a requested development task
  - they can be laid out on a table
  - they can be laddered, grouped into releases
- Project Management metrics
  - completed cards record work accomplished
  - completion rate measures project velocity

UML Use Case Diagrams

- A graphical overview of capabilities
  - a list of defined actions (use cases)
  - shows boundary between system and actors
- Use Case diagrams identify
  - classes of actors who can initiate actions
  - actions (use cases) each can initiate
  - other people (or objects) that will be affected
- They do not describe the interactions
  - that is left to more detailed behavioral models

Sample UML Use Cases



Using UML Use Cases

- an overview of system functionality
  - what are the general classes of users
  - what are the operations each can perform
- a summary of system capabilities
  - what are the basic things it can do
- a table of contents for the use cases
  - enumerating the things to be defined
- May be an introduction to the solution
  - external object classes and methods

How they compare

- User Story Cards
  - summary of a specific functional capability
  - not a description, a discussion/planning token
- UML Use Case Diagrams
  - enumerate the actors, objects and operations
  - a single snapshot of the major capabilities
- Use Case/Scenario Descriptions
  - detailed behavior descriptions/specifications
  - basis for designs and acceptance tests

Modeling more detailed behavior

- Use-cases and story cards are imprecise
  - they capture intent, but not details
- Activity Diagrams
  - steps in a (potentially distributed) process
  - sequences of events
  - key decisions
- State Models
  - can be user-visible or internal states
  - events that cause changes between them

Universal Modeling Language

- a family of related graphical notations
  - for representing software system designs
  - particularly those built in object oriented style
- supports a wide range of uses
  - a design sketching language
  - a system specification language
  - a programming language
- also has a textual (XML) representation
  - enabling development of CAD tools

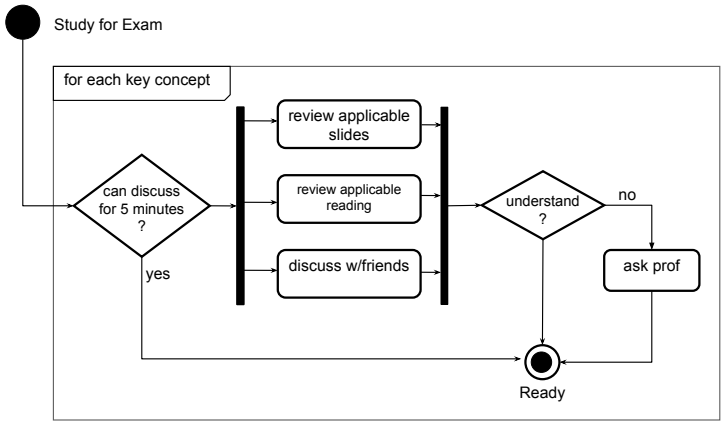
UML General Conventions

- square boxes represent things
  - classes, objects, packages, components
  - 3D boxes represent physical things
- round boxes represent processes/states
- circles represent class interfaces
  - entry-points
- arrows represent relationships
  - flow, communication, dependency
  - arrow heads determine type and direction
- UML diagrams can be nested/composed

(UML Activity Diagrams)

- Describe processes (not just algorithms)
  - in terms of steps (decisions and actions)
- a standard UML representation
  - rounded “capsules” represent activities
  - arrows represent temporal sequencing
  - diamonds represent decision & merge points
  - bars delimit parallel activities
- excellent for behavioral requirements
  - illustrative sample usage scenarios
  - additional detail for a use-case or story card

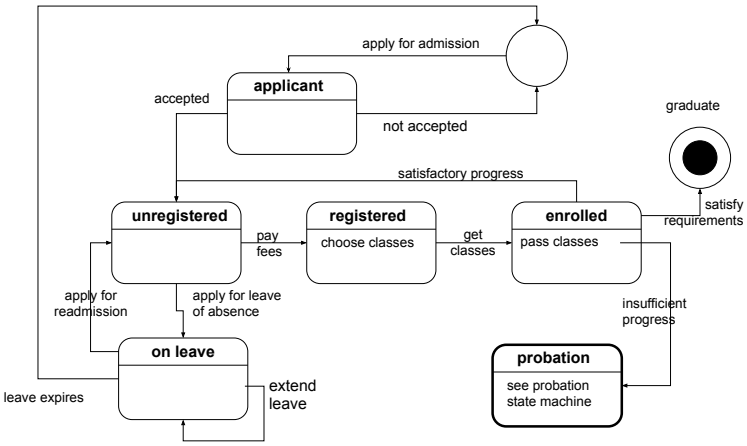
UML Activity Diagrams



(UML State Models)

- describe state/transition models
  - where events drive state changes
- a type of activity diagram
  - activity boxes have two compartments
    - state name in the top portion
    - processing steps in the bottom portion
  - arrows represent state transitions
    - from previous state, to next state
    - labels describe conditions triggering the transition
    - processing steps can also be placed on lines

UML State Diagrams



Team Exercise

- identify distinct classes of your users
  - what usefully distinguishes each sub-class
- identify key goals/operations for each
  - and the participants in each
  - describe/diagram their use cases
- you will be asked to briefly discuss
  - why you chose those classification criteria?
  - how well this diagram captures your product?
  - what this diagram tells you about your product?

Requirements Elicitation Session

- when
  - next lecture – in class
- goals
  - to demystify the process
  - to allow you to observe the process
  - provide an example we can discuss
  - to prepare you for your own sessions
- preparation
  - review chap 3 of my User Requirements paper

For the Next Lecture

- Kampe: Why we Model
  - different goals and approaches
- Gabler: Prototyping
  - principles for building game prototypes
- Ambler:
  - Agile Modeling Principles
  - UML component diagrams
  - UML deployment diagrams

Backup Slides

Discussion Slides

Deeper B_1

- Why is it useful to identify different sub-classes of users?

They may need the product to provide different functions.

They are likely to use the system to accomplish different goals.

They are likely to have different world-models and experience.

We may want to present different interfaces to different types of users.

Deeper B_2

- How might role based classification be useful?

There may, in fact, be fundamentally different types of users, who will use the product in different ways.

- Why might role based classification be invalid?

Some roles may distinguish momentary activities rather than people (e.g. shopper vs. purchaser).

Deeper

- Why might knowledge based classifications be important?

  If knowledge differences between users give rise to different models of the system, we might want to show each the expected model.

- Why might knowledge differences not greatly affect scenarios?

  If the difference in knowledge is just a matter of experience we might provide the same basic scenarios, but with optional help/guidance for trainees.

Deeper

- How are "softer" user characterizations like Rouse's useful?

  They establish "fundamental goals" … which are probably more important than "sample use cases".

  They give us a richer model of the user, whom we are trying to satisfy.

  They do give us usage scenarios, and goal- (rather than task-) oriented use-cases.

Deeper

- Why is a good use case easier to get than a good declarative requirement?

  It is more natural for people to describe what they do, than an imaginary system.

- Why are use cases easier to review than declarative requirements?

  Use cases are stories, and users can easily see how well the story matches their own experience.

Deeper

- How might a use case more honestly capture requirements than declarative statements?

  Because the real requirements may actually be to facilitate the specified use cases.

  Decoupling the requirements from the use cases adds opportunities for translation errors and/or the introduction of new elements that are not necessarily implied by those use cases.

Deeper

- What is good about a prototype as an interface specification?

  Potential users can review and comment on it.

- What is bad about a prototype as an interface specification?

  It goes well beyond requirements, perhaps specifying an almost complete implementation.

Deeper

- Why should user stories be kept to one 3x5 card?

  To prevent calling large projects one feature.

  To force us to distill what we want to simple, clear and concise statement.

  To prevent us from calling it a complete specification (those come from users).

  To give us a useful management token.

  … but the truth is that we will associate task descriptions and test cases with them.

Deeper

- Why are detailed behavioral expectations not included in User Story cards?

(a) *This information is expensive to develop, and is not needed for planning purposes.*

(b) *This information is evolved between the developer and user, and becomes the basis for the acceptance test suite. There is no need for any intermediate representation.*

Deeper

- When would we choose a state diagram rather than an activity diagram?

If we expected to implement the algorithm as a state machine.

Even systems that are implemented with normal code, may have several interesting and distinct operational states or modes. Diagramming those states and the events that trigger transitions between them may be a very useful (and revealing) way to view the system's dynamics.

Deeper

- How would you generate a list of different types of users for a product?

Some roles are obvious.

You can ask someone with domain expertise if there are different sub-classes of potential users.

You could interview potential users, ask them what they do, and learn about different roles from them.

Deeper

- Why is a graphical design language interesting?

Any language can be evaluated on the basis of range, ease of expression and precision of expression.

While graphical languages are (in general) less precise and powerful than textual ones, the human mind can often absorb structures and relationships <u>much</u> more easily and quickly from a visual representation than a textual one.

Deeper

- Why are use cases:
  - concise … *simple list of steps*
  - real-world … *scenario has a purpose*
  - what, not how … *story only tells "what"*
  - user world-view . *from user perspective*

Deeper

- How is UML different from other graphical flow-charting & design systems of the last 50 years?

It supports precise and complete high level descriptions of objects and their relationships.

Its wide range of sub-languages permit meaningful description of many more aspects of system behavior and structure.

The sub-languages are related in ways that enable easy integration of multiple models.

Deeper C4

- What distinguishes a sketching language from a specification language?
  - *A sketching language can quickly and easily describe general concepts.*
  - *A specification language can precisely describe objects and their behavior.*
- What distinguishes a specification language from a programming language?
  - *A programming language can completely specify data types and flow control.*

Deeper C1

- When does it makes sense to develop use cases by observing or interviewing potential users?
  - *When the product is intended to assist, enhance, or extend an existing process, users of the current process can provide good descriptions of how they use it.*

Deeper C3

- Why is it useful to standardize a graphical design language?
  - *People can more quickly understand a language they already know.*
  - *Designs captured in a standardized language can be more easily shared and reused.*
  - *A standardized language can express much more nuanced concepts (compare AMSLAN with charades).*
  - *A large population will attract tool developers.*

Deeper C2

- When does it makes sense to develop use cases by brain storming?
  - *If the whole point of the new product is to enable new usage scenarios, there may be no people who are familiar with them.*
  - *When the new product use will be very different from existing processes, there are limits to what we can expect from users of the existing products.*

Deeper C3

- If existing users don't know how they will use a product they've never seen, why should we interview them?
  - *To understand the needs to which our new product will respond.*
  - *To understand the world-view into which our new product may need to integrate.*
  - *To understand what users of products find to be convenient and inconvenient.*

Deeper D1

- If use case diagrams do not describe the interactions, what are they useful for?
  - *They are the basis for discussions of the basic sub-classes of users, and the way that each will use the system.*
  - *They provide a good high level overview of the system's expected capabilities.*
  - *They could be a table of contents for more detailed behavioral descriptions of each interaction.*

Deeper D2

- What does it mean for one use case to extend another?
  - *A more complex use case might be a superset of the steps of a simpler (and previously defined) use case.*
  - *Variations on a common underlying mechanism might front-end the simpler use case.*
  - *This both shows those relationships, and simplifies the overall use-case collection.*

Deeper D3

- How would it affect understandability if it was necessary to represent a system in multiple distinct use case diagrams?
  - *If they were truly distinct (different actions on different types of objects) separating them out to separate diagrams would probably make the system easier to understand. Packages can be used to for hierarchical groupings.*
  - *Otherwise, one larger diagram would probably make more sense than breaking things arbitrarily into smaller diagrams.*

Deeper F1

- Would it make sense to use both UML use case diagrams and XP user story cards?
  - *Each story card might show up as a use case in the use case diagram.*
  - *The use case diagram would provide an overview of the capabilities in a graphical and more structured (broken down by user types) way that is much easier to grasp.*