Gathering Requirements

Kenneth M. Anderson
University of Colorado, Boulder

CSCI 5828 — Lecture 5 — 01/27/2009

Goals

- Review material from Chapter 2 of Pilone & Miles
- Concepts include
  - Requirements and Requirements Gathering
    - User Stories
    - Brainstorming
  - Planning
    - Estimation Game
- Will also review requirements-related info not from textbook

Requirements Gathering

- Requirements gathering begins with a problem statement from your customer. Example:
  - We need a web site showing our current deals, and we want our users to be able to book shuttles and special packages, as well as pay for their bookings online. We also want to offer a luxury service that includes travel to and from the spaceport and accommodation in a local hotel
- Characteristics?
  - Loose, informal, unstructured, all over the place (deals, bookings, packages, payment, shuttle services, hotels)

First Step: Impose Structure

Title:
Description:

Title:
Description:

Title:
Description:

Title:
Description:

Identify all of the different things the system has to do.

In particular, find requirements

A requirement is a **single thing** that the software has to do

Title: Show Current Deals

Description: The website will show current deals to Orion's Orbits users.

Note:

Title: Show Current Deals

Description: The website will show current deals to Orion's Orbits users.

Written in User's Language

Informal: because we don't have a lot of information

But, allows us to validate initial understanding of domain

Translate Entire Problem Statement:

Title: Show Current Deals

Description: The website will show current deals to Orion's Orbits users.

Title: Book a shuttle

Description: An Orion's Orbits user will be able to book a shuttle between hotel and spaceport.

Title: Book package

Description: An Orion's Orbits user will be able to book a special package with extras online.

Title: Pay online

Description: An Orion's Orbits user will be able to pay for their bookings online.

Title: Arrange Travel

Description: An Orion's Orbits user will be able to arrange travel to and from the hotel.

Title: Book a hotel

Description: An Orion's Orbits user will be able to book a hotel.

Note: iteration

Then, return to customer and:

Title: Show Current Deals

Title: Book a shuttle

Title: Book package

Title: Pay online

Title: Arrange Travel

Title: Book a hotel

Title: New Requirement

Description: Pithy text describing new requirement...

- ask questions
  - Did I get this right?
  - What did you mean by…
- and gather more requirements
  - Is this really all of the functionality that you need?
  - If we built all of this, what would you want in version 2.0?

All this work will lead to new or clarified requirements

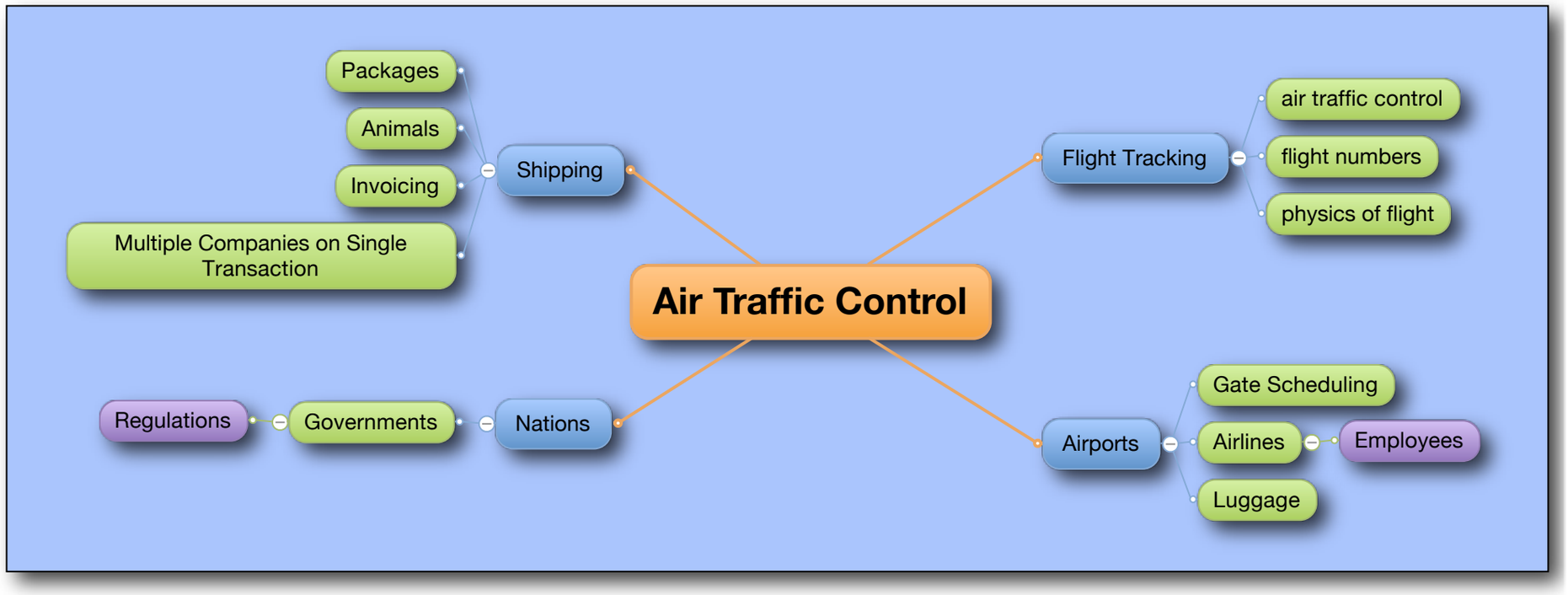
Problem: Not Enough?

- One problem that you'll encounter is that this back and forth may not be enough to get to crisp detailed requirements
  - or you feel that you just don't have a good grasp on the big picture
- This can be especially true if "customer" ≠ "end user"
- Next step is to hold a brainstorming session with as many different stakeholders as possible
  - what the book calls a "bluesky session" and is sometimes called "shooting for the moon"

Bluesky Session

- Brainstorming session
  - Goal: get stakeholders to generate tons of ***candidate*** requirements; not everything will make it into the final system
  - Secondary Goal: Capture everything for later analysis
- Things to Avoid
  - The Silent Tomb®: Leave job titles at the door, people should not feel afraid to speak up just because the boss is there
  - Criticizing people rather than ideas
  - Developer jargon “NOT ‘AJAX’ but ‘rich user interface’”

Tool Support



Airports

- 1. Airports
  - a. Airlines
    - i. Employees
      - 1. Scheduling
    - ii. Flights
      - 1. Scheduling
- 2. Luggage
  - a. Airline/Airport handoff
  - b. Bag numbers
  - c. Handling of oversize items
  - d. Handling of live animals

- Make use of outliners and other types of note taking applications during brainstorming sessions to capture the generated ideas, domain knowledge, and requirements; to the right is an example generated by Curio, a note taking application from Zengobi.

Gray Skies

- If things go wrong during the bluesky session: “bad boss”
- Make use of other techniques
  - Interview end users and have them pretend to interact with their “ideal system”, what the book calls “role playing”
  - Observe them working on tasks related to the system
    - how would the task change if the system were present?
  - Review the documents they use now
    - ask if the document would go away if the system were present
      - or how would it change?

Next? User Stories

- Transform requirements gathered so far into user stories
  - A user story describes how the user interacts with the software you're building
  - It should be written from your customer's perspective and describe what the software is going to do for the customer
- User stories are essentially **informal use cases**
  - See CSCI 6448 (changing to CSCI 5448 in Fall 2009) for more details on use cases

User Stories

- SHOULD
  - describe one thing the system should do for the customer
  - be written using language that the customer understands
  - be written by the customer
  - be short. No longer than three sentences
- SHOULD NOT
  - be a long essay
  - use technical terms unfamiliar to the customer
  - mention specific technologies (save those for design)

Requirements Life Cycle

- We now have a life cycle for use at the start of a project
  - Capture basic ideas from problem statement
  - Return with first pass, ask questions, set-up bluesky session
  - ITERATE
    - Construct User Stories
    - Find holes with stories and fix them with customer feedback, find new requirements, ask questions to assess completeness
  - Finish with initial set of clear, customer-focused user stories
- This defines the WHAT of the project, next up is the WHEN

Estimates

- At some point during the requirements gathering process, the customer will ask
  - How long will all of this take to develop?
- You need to supply a project estimate
  - which will be the sum of the estimates for your user stories
- So, now you need to supply estimates for each user story
  - How do we come up with this estimate?

Planning Poker

- A popular estimation technique in agile methods
- Addresses the problem in which two or more developers come up with wildly different estimates for a user story
  - i.e. when a single user story generates estimates of say “3 days”, “2 weeks”, and “3 months” from three different developers
- The underlying cause for these different estimates is assumptions; what did you assume was true or not true about the project to generate the number that you did?

Example

- “Add a comment on a product page”
- One developer might think:
  - “Simple. We need a form, a script to process the form, and a place to store the comment in the database. 3 days.”
- Another might think:
  - “Hmm. How do we relate the comment to the product? Do we have one comment table per product in the database? Will I need to change the product class? Maybe there is code from some other place in the system that I can re-use. 2 weeks.”

Example, continued

- Finally, another might think:
  - “Ugh. Complete database re-design. No code to re-use (this is the first time we’re allowing comments). What user interface should we use? Can the user embed HTML in their comments? How do we handle smileys? How will this impact the product model class? Do we keep the comments forever? Do we need moderation? Can a user edit a past comment? Who gets to delete comments? Yuck!! 3 months!”
- Based on your assumptions, you’ll get completely different numbers. How do you get these assumptions to the surface? Planning Poker!

Planning Poker (I)

- Create “deck” of cards. 13 cards per “player”.
  - Each card contains an estimate spanning from “already done” to “wow this is going to take a long time”.
    - 0, .5, 1, 2, 3, 5, 8, 13, 20, 40, 100 days
  - One card has a “?” meaning “not enough information”
  - One card has a coffee cup meaning “lets take a break”

Planning Poker (II)

- Place a user story in the middle of the table
  - Each developer thinks about the story and forms initial estimate in their heads
- Each person places the corresponding card face down on the table; note: estimate is for entire user story
- Everyone then turns over the cards at the same time
- The dealer marks the spread across the estimates

Planning Poker (III)

- The larger the difference between the estimates, the less confident you are in the estimate, and the more assumptions you need to highlight and discuss
- So, the next step in planning poker is
  - **Put assumptions on trial for their lives**
- Have each developer list the assumptions they made and then start discussing them
  - Again, you need to criticize the assumption not the developer
- Goal is to get agreement on what assumptions truly apply

Planning Poker (IV)

- If the assumptions reveal a misunderstanding of the requirements, then go back to the client and get that misunderstanding clarified
- Otherwise, start to eliminate as many assumptions as possible, then have everyone revise their estimates and play planning poker again to see if the spread has decreased
  - Your goal is convergence. Once everyone's estimates cluster around a common number, assign that number and move to the next story

Planning Poker (V)

- Your life cycle is thus
  - Talk to customer: clarify misunderstandings, assumptions
  - Play planning poker
  - Clarify assumptions, possibly by returning to step 1
  - Come to a consensus estimate for the user story
- Do this until all user stories have a consensus estimate assigned

Planning Poker (VI)

- Things to watch out for
  - Although implied in the previous slides, don't do one card at a time with multiple customer sessions each time
    - Value your customer's time
      - Process each card, identifying assumptions/misunderstandings that need clarification; THEN meet with customer
  - Big estimates (== bad estimates)
    - They indicate that the story is too big; decompose; try again
    - Remember, the book's ideal iteration is 20 work days (1 month)
    - Estimates longer than 15 days are more likely to be wrong than those shorter than 15 days; (others think 7 days is upper limit)

Requirements Life Cycle

- Capture basic ideas
- Bluesky Brainstorming
- Construct User Stories
- Find holes, get feedback
- Clear, customer-focused user stories
- Play planning poker
- Clarify misunderstandings and assumptions
- Develop project estimate

Iterate

May need to Iterate

After the estimate?

- At the end of this process, you may discover that your project estimate is too long
  - In the book, the developers came up with a 2-year estimate that the customer says is way too long
- We'll look at how to deal with that situation in Lecture 7
- Now, lets look at the requirements phase a little more broadly

Software Requirements

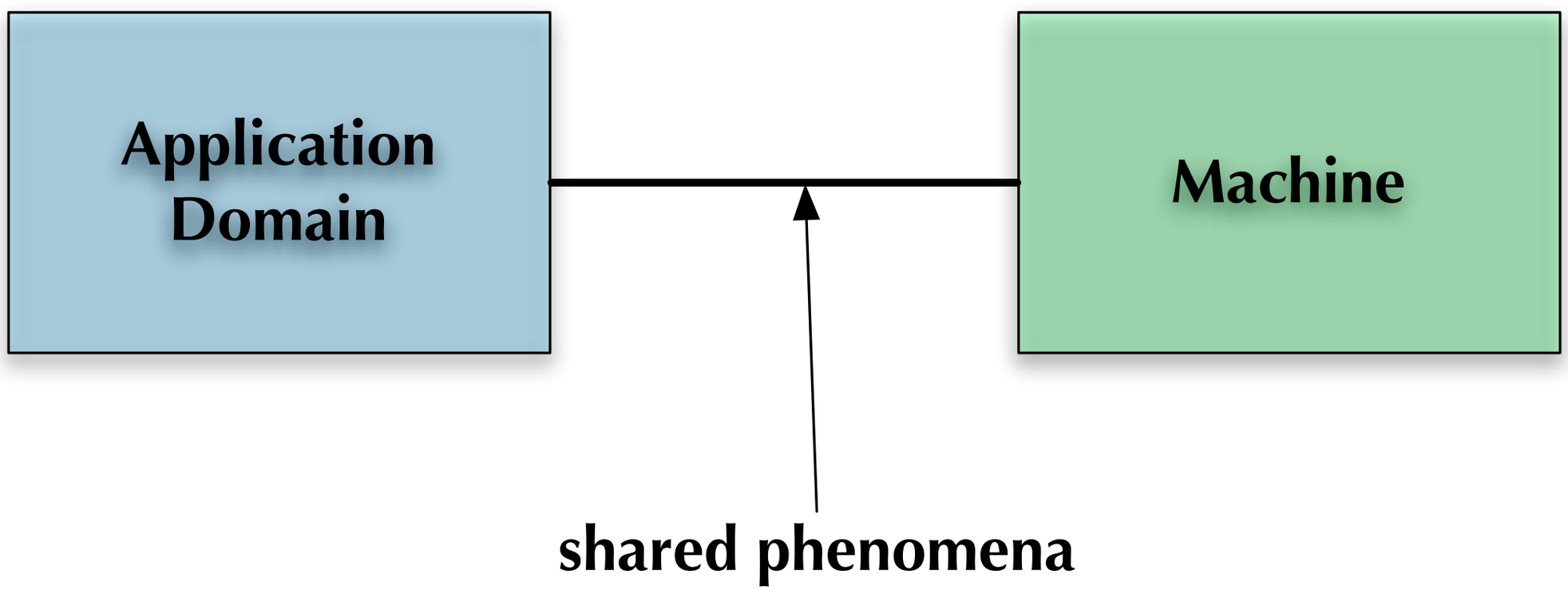
- Is the most critical task in software development
  - Its goal is to understand the problem that needs to be solved
- If you don't understand the problem, you can't solve it
- Example
  - There is a farmer with a fox, a rabbit and a prize cabbage. There is a row boat, complete with oars, on one side of the river. On the other side is a market. There is room in the boat for any two of the four. The fox is hungry, so is the rabbit. Foxes like to eat rabbits, rabbits like to eat cabbages.
- What's the Problem?

Problem Context vs. Problem

- In software development, there is typically one problem context but there may be more than one problem to be solved within that context
- During requirements, you need to
  - understand the problem context
  - determine which problem (or problems) need to be solved
  - document your understanding (with user stories)
- The problem context is also known as the **application domain** or **problem domain**

What about Design?

- After you understand the problem, you design a solution;
  - Need to find the intersecting concepts that allow our machine (i.e. software system) solve the problem in the app. domain



Shared Phenomena? (I)

- There will be some aspects of the application domain that will not be modeled, tracked, or otherwise handled by the machine
  - For instance, when controlling elevators, a software system may not care about the movement of cars between floors or the number of passengers in a car
- But there will be other elements of the application domain that are critical to the machine and will thus be tracked
  - Elevator Domain: button presses, door open, door close, number of cars, number of floors, etc.

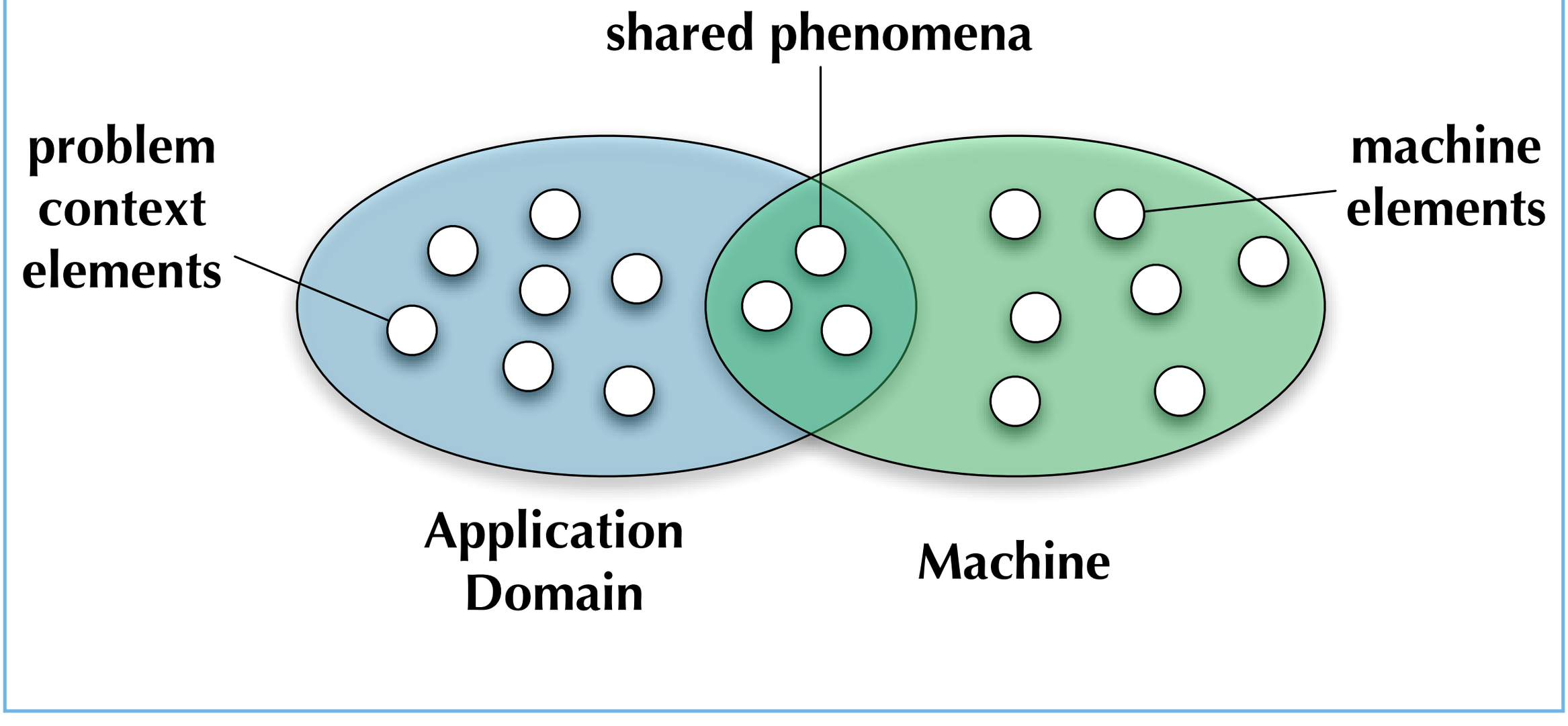
Shared Phenomena? (II)

- Similarly, there will be elements of the machine (aka solution domain) that will have no correspondence to elements in the application domain
  - The algorithm to schedule the movement of elevators is intangible and will only exist as code stored on a computer; it has no physical counterpart in the elevator domain

Shared Phenomena? (III)

- Those application domain elements that are modeled by the machine are shared; developers must perform work to enable that sharing
  - Elevator Domain: A call button must be wired to a hardware device connected to the software system such that pressing the button delivers a signal to the software system. Likewise, if the system determines that car A should go to floor B, there needs to be a mechanism that allows the software to send that command and have it be followed

Problem Context, Take 2



Requirements Engineering (I)

- We must understand the application domain and then understand the problem that needs to be solved
  - As understanding grows, we document it via requirements
- What's a requirement? IEEE definition
  - A condition or capacity needed by a user to solve a problem or achieve an objective
  - A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed documents
  - A documented representation of a condition or capability as in 1 or 2

Requirements Engineering (II)

- (One) Definition of Requirements Engineering
  - “The systematic process of developing requirements through an iterative cooperative process of analyzing the problem, documenting the resulting observations in a variety of representation formats, and checking the accuracy of the understanding gained.” K. Pohl, 1993

Two Phases

- Requirements Elicitation
  - The process whereby a development agency discovers what is needed and why (i.e. develops an understanding of the problem and the problem context)
- Requirements Analysis
  - The process of understanding the requirements
  - Asks questions about completeness and consistency
- As we've seen, we iterate between these two phases until we are "done"

Problems (I)

- It is important to understand that performing requirements engineering is HARD
  - Developers are not domain experts
    - need to learn jargon just to have conversations with domain experts
    - will have problems conveying solution-domain concepts to domain experts
    - will never fully understand the domain
    - must be careful not to alienate domain experts, who may be technology-phobic

Problems (II)

- Additional Problems
  - domain experts and end-users may not understand how software will change their environment
  - domain experts will not reveal all relevant domain information at once, will be surprised at how much they have to explain to bring developers up to speed
  - customer may not know what they want from a system, or may have ideas that are just plain wrong or infeasible
  - customer may change their minds about information provided previously

Problems (III)

- It is also difficult to transition between the two phases of requirements engineering
  - It may not be clear how to transform your understanding of the domain into specific requirements
  - What formalisms should be used?
  - Is natural language enough?
  - At some point, you need to be able to specify what the inputs of the system will be and what outputs it should produce WITHOUT specifying a solution to the problem
- In order to make progress, **iteration is essential!**

Formalisms

- Requirements can be captured in a variety of formalisms
  - Natural language text via the “big document” approach
    - Also known as the software requirements specification
  - Natural language via the user story / use case approach
  - Formal Methods: Z, StateCharts, etc.
    - Makes use of algebraic specifications, finite state machines, …
  - Data Flow Diagrams
  - Viewpoints
    - (See <http://www.requirementsviewpoints.blogspot.com/>)

Tools

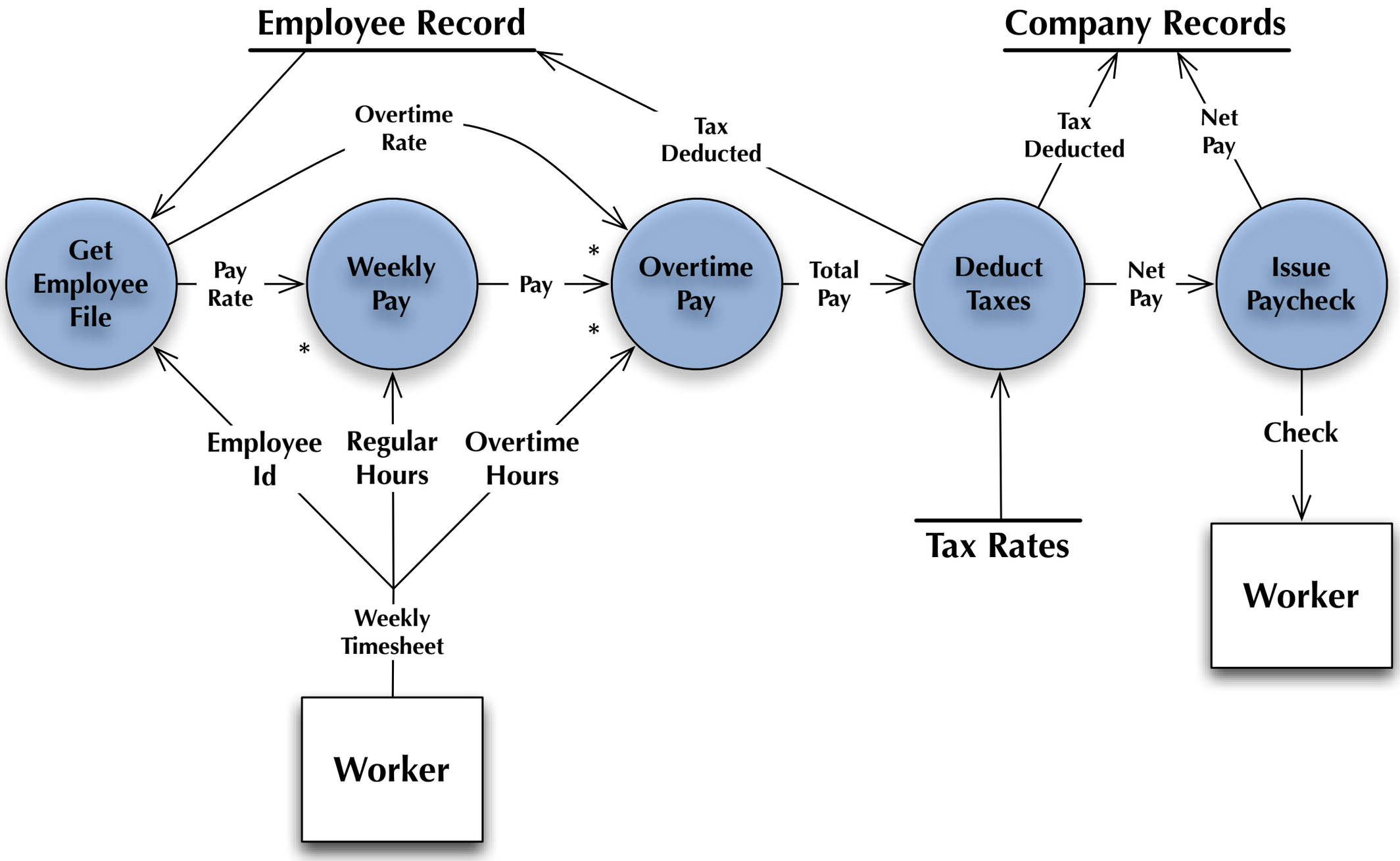
- Directory of Requirements Management Tools
  - <http://www.software-pointers.com/en-requirements-tools.html>
- IBM's Rational RequisitePro
  - <http://www-01.ibm.com/software/awdtools/reqpro/>
- IBM's Telelogic Doors
  - <http://www.telelogic.com/products/doors/index.cfm>
- Many more… note: agile methods shun the use of these tools and stick with user stories and (later) issue tracking

Data Flow Diagrams (I)

- Shows the flow of data through a system
  - Does not specify an ordering of operations, nor does it show looping, handling exceptions, etc.
- Instead specifies the input and output of each process in the application domain
  - It is often easy for humans to impose an ordering on the DFD

Data Flow Diagrams (II)

- Legend
  - Circles: Process or Function
  - Rectangles: Data source and/or sink
  - Arrows: Named data flows
  - External Repositories shown as labeled straight lines
  - Asterisk (*) indicates multiple required inputs to a process (process requires A AND B)
  - Plus (+) indicates multiple inputs to a process such than only one is required (process requires A OR B)


Employee Record
Company Records
Overtime Rate
Tax Deducted
Tax Deducted
Net Pay
Get Employee File
Pay Rate
Weekly Pay
Pay
Overtime Pay
Total Pay
Deduct Taxes
Net Pay
Issue Paycheck
*
*
*
Employee Id
Regular Hours
Overtime Hours
Check
Tax Rates
Weekly Timesheet
Worker
Worker

Wrapping Up

- Today, we saw the importance of requirements
  - Of identifying the problem to be solved and its problem context
  - Of the use of iteration in requirements gathering
  - How user stories can be used to document requirements
  - How planning poker can be used to generate estimates
  - Saw references to requirements management tools
  - Used DFD as an example of another formalism used to generate/elicit requirements

Coming Up

- Lecture 6: Processes and Threads
  - Chapter 2 of Magee and Kramer
- Lecture 7: Project Planning
  - Chapter 3 of Pilone & Miles