

CSE 403  
Lecture 5 [Notkin]

---

Software requirements  
&  
Software design

Today's educational objectives

- Be able to distinguish a system's requirements from a program that realizes those requirements
- Understand that the driving force behind design is managing complexity
- Non-objectives for today: make you great requirements engineers and/or great designers

Software requirements:  
your definitions?

Software design:  
your definitions?


What vs. How?

- A classic notion is that the requirements are the "what" and the program (design) is the "how"
- But "what" and "how" are relative terms
  - Parsing is the what, a stack is the how
  - A stack is the what, an array or a linked list is the how
  - A linked list is the what, a doubly linked list is the how

The Machine and the World:  
Michael Jackson


- The requirements are in the application domain
- The program (design) defines the machine that has an effect in the application domain
- Ex: Imagine a database system dealing with books

The diagram consists of two speech bubbles. The left bubble is labeled 'The World' and contains the text 'Books, Authors, Titles, etc.'. The right bubble is labeled 'The Machine' and contains the text 'Records, databases, pointers, etc.'.




## Not a perfect mapping

- There are things in the world not represented by a given machine
- Examples might be:
  - Book sequels or trilogies
  - Pseudonyms
  - Anonymous books
- There are things in the machine that don't represent anything in the world
- Examples might be:
  - Null pointers
  - Deleting a record
  - Back pointers




## Success

- A system is judged not by properties of the program, but by the effects of the machine in the world
- You don't care how Caller ID works, just that it works
- TCAS is a collision-avoidance system for commercial aircraft
  - Pilots love it (on the whole) because it helps them fly more safely and easily — not because it has great data structures




## Software failures

- More software projects fail because they don't meet users needs than for any other reason




## Three challenges

- To figure out the desired effects (requirements) of the machine in the world
  - Beyond scope of class; not needed for GizmoBall
- To figure out how to write this down in an effective way
- To figure out how to make sure that the machine (program) you build satisfies the requirements
  - More later in the quarter




## Why write it down?

- It will help clarify what you think
- It is necessary to communicate with your users
- It is necessary to communicate with your team members
- It could form the basis for a contractual relationship



## How to write it down?

- Natural language
- Structured natural language
- A formal language
  - We'll come back to this later in the quarter




## Natural language

- n Inherently ambiguous
  - n If you don't believe it, make sure to teach or TA an undergraduate course sometime!
- n You all have your favorite examples
  - n The one on the right is from Jackson, found at the base of an escalator


Shoes Must Be Worn

Dogs Must Be Carried



## Well?


- n Must I carry a dog?
- n What about the shoes I just bought that are still in my shopping bag?
- n Do dogs have to wear shoes?
- n What does it mean to wear shoes?
- n What are shoes?
- n What are dogs?



## "dog" (noun):


OED has 15 definitions, Webster's has 11

- n a highly variable domestic mammal closely related to the common wolf
- n a worthless person
- n any of various usu. simple mechanical devices for holding, gripping, or fastening that consist of a spike, rod, or bar
- n FEET
- n an investment ... not worth its price
- n an unattractive girl or woman




## "shoe" (noun, 6 Webster's)

- n an outer covering for the human foot usu. made of leather with a thick or stiff sole and an attached heel
- n another's place, function, or viewpoint
- n a device that retards, stops, or controls the motion of an object
- n a device (as a clip or track) on a camera that permits attachment of accessory items
- n a dealing box designed to hold several decks of playing cards



## Structured natural language

- n I
  - n I.A
    - n I.A.ii
      - n I.A.ii.3
        - n I.A.ii.3.q

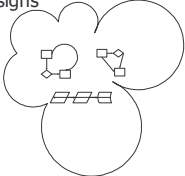
- n Although not ideal, it is almost always better than unstructured natural language
- n Unless the structure is used as an excuse to avoid content
- n You will probably use something in this general style


## What is design?

- n The activity that leads from requirements to implementation (verb)
- n A description that represents a key aspect of this activity (noun)

## Design space

- There are many designs that satisfy a given set of requirements
- There are also many designs that may at first appear to satisfy the requirements, but don't on further study
- Collectively, these form a design space
- A designer walks this space evaluating designs



## Design: managing complexity

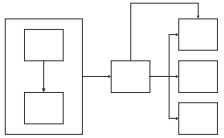
- Of course, this design space isn't just sitting out there for you to search like a library catalog or the WWW
  - Although limited progress is being made on this
- You must also generate the designs
- A key aspect of design generation is understanding that the goal is to achieve the requirements in the face of the limitations of the human mind and the need for teams to develop products

## Decomposition

- Design is largely a process of finding decompositions that help humans manage the complexity
  - Understand that the design satisfies the requirements
  - Allow relatively independent progress of team members
  - Support later changes effectively
- Not all decompositions are equally good!

## Decompositions

- A decomposition specifies a set of components (modules) and the interactions among those modules
- The degree of detail in the specification varies



## Comparing designs

- Not all decompositions are equally good
- So, on what basis do we compare and contrast them?

## Coupling and cohesion

- Given a decomposition of a system into modules, one can partially assess the design in terms of cohesion and coupling
  - Loosely, *cohesion* assesses why the elements are grouped together in a module
  - Loosely, *coupling* assesses the kind and quantity of interconnections among modules

## Kinds of cohesion

- Elements can be placed together to provide an abstract data type
  - if they are all executed about the same time (say, during initialization or termination)
  - because they will be assigned to a single person
  - because they start with the same letter
  - because...

## Coupling

- Coupling assesses the interactions between modules
- It is important to distinguish *kind* and *strength*
  - kind: A calls B, C inherits from D, etc.
    - And directionality
  - strength: the number of interactions

## "Good" vs. "bad" coupling

- Modules that are loosely coupled (or uncoupled) are better than those that are tightly coupled
- Why? Because of the objective of modules to help with human limitations
  - The more tightly coupled are two modules, the harder it is to think about them separately, and thus the benefits become more limited

## What kind (of relations)?


- There are many kinds of interconnections (relations) to consider
  - calls, invokes, accesses, ...
  - how about invokes in an OO language using dynamic dispatch?
  - others?
- Question: how many different relations are there among components of a software system?

## names vs. invokes

- Here is an example of a mix of relations
- Module A registers interest with an event that B can announce
- When B announces that event, a function in A is invoked
- A knows the name of B, but B doesn't know the name of A


## Hierarchical designs

- Loose coupling is often associated with hierarchical designs
  - They also tend to arise from repeated refinements of a design
- Hierarchies are often more coupled than they appear
  - Because of other relations




## Other dimensions (Bergland)

- n Complexity
  - n A design with really complex modules is worse than a design with simpler modules
  - n Complexity is brutally hard to measure
- n Correctness
  - n Correct designs dominate incorrect ones
- n Correspondence
  - n How closely does it map to the requirements specification? *This is critical to evolution!*




## Use a single module?

- n Great cohesion!
- n No coupling!
- n Where's the failure?



## Coupling and cohesion...

- n ...are just guidelines
- n Again, they don't by themselves give criteria for modular decomposition



## Whirlwind...

- n ...we'll do more later on!
- n Ask questions!