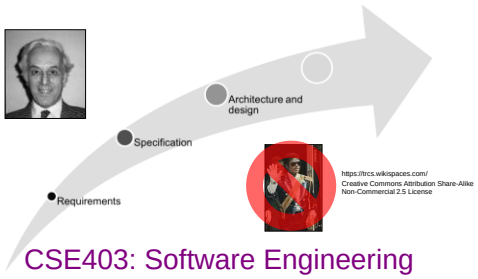


CSE403: Software Engineering

David Notkin
Winter 2009

Today

- Some requirements wrap-up (mostly from guess who?)
  - Forms of denial
  - Relation to specifications (note – there are many uses of the term "specification")
  - UML-ish diagrams
- Towards software architecture and design…

Denial by…

- …prior knowledge
- …hacking
- …abstraction
- …vagueness

Denial by prior knowledge

- Don't need requirements – they are clear and all we need to do is build it
- Specialized and standardized engineering
  - Most engineering is *specialized*, requiring some expertise (in stresses, in fluid flow, in computing at web-scale, in proteomics, etc.)
  - Some engineering is *standardized* as well: compare car design (standardized) to bridge design (non-standardized)
- Denial is dangerous unless a domain is both specialized and standardized

Denial by hacking

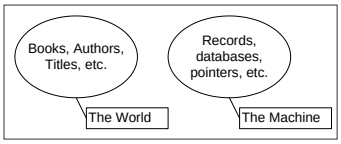
- Not bad hacking, but rather an interest in the computer itself more than in what the effect of the computer is
- Nothing wrong with this interest, unless it compromises the ability to create useful machines
- Applications can be more boring
- "I came into this job to work with computers, not to be an amateur stockbroker" –Member of a failed development team

Denial by abstraction

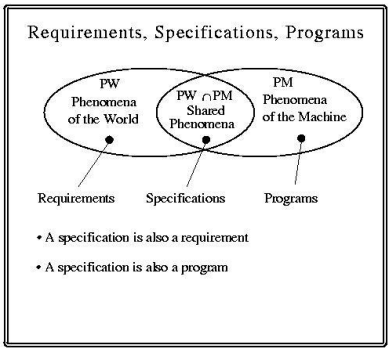
- Intentionally choosing to forget what the symbols represent
- Arguably, denial by hacking in the mathematical realm
- Abstraction is good – surely one of the most powerful tools in software engineering
- But "too much abstraction blinds you to the nature of many problems"

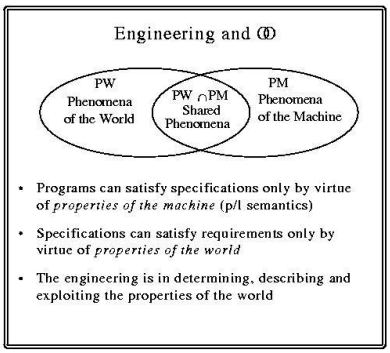
Denial by vagueness

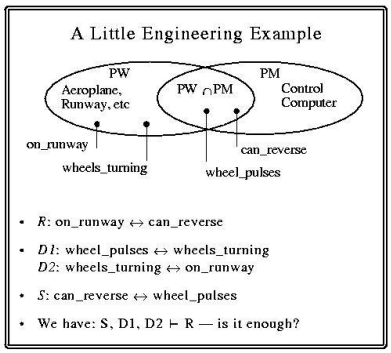
- "Describe the Machine, but imply that you're describing the World."

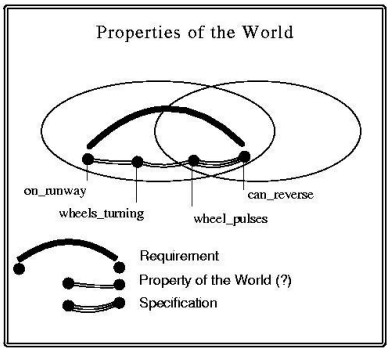


- No designations, avoid saying explicitly what is described, …









UML-ish models/diagrams

- Object models
- …fill in…
- Used, and perhaps useful, at multiple levels of software engineering – requirements, design, implementation
  - Clarity about the level of use is as important as clarity of the models

Towards software design

- ..via architecture (and perhaps back to architecture later on)

System architecture

System architecting, the planning and building of structures is as old as human societies and as modern as planning the exploration of the solar system. It arose in response to problems too complex to be solved by pre-established rules and procedures. It introduces heuristics as design guidelines and focuses on the art — in contrast with the science and mathematics — of conceiving and certifying systems of complexity too great to analyze. —E. Rechtin

Brooks [MMM after 20 years]

Today I am more convinced than ever. Conceptual integrity is central to product quality. Having a system architect is the most important step toward conceptual integrity.

Some Spinrad heuristics [Rechtin/Maier]

- "In architecting a new program, all the serious mistakes are made in the first day"
- "The test of a good architecture is that it will last. The sound architecture is an enduring pattern"

We're less grandiose

- This description of Rechtin's is fine and accurate
  - It's largely the basis for the field of systems engineering
- Indeed, lots of these ideas are applicable to systems with substantial software components
- But a less broad notion of system architecture is fine for this course

System architecture

- The really basic, essentially unchangeable structures of the system
- They can arise in at least two ways
  - These structures can be defined entirely beforehand: they are part of the customer's definition of the project
  - In other cases, they are the first, high-level choices you make

Ex: Customer imposed

- Build this system in Unix using X-windows
  - This sets the basic user interface engine and structure (client-server)
  - It also sets the basic internal structures and computations
    - Unix processes, byte streams, etc.
- Why might this kind of decision be imposed?

X Windows requirements [Lee]

- The system should be implementable on a variety of displays.
- Applications must be device independent.
- The system must be network transparent.
- The system must support multiple applications concurrently.
- The system should be capable of supporting many different application and management interfaces
- The system must support overlapping windows, including output to partially obscured windows.
- The system should support a hierarchy of resizable windows; an application should be able to use many windows at once.
- The system should provide high-performance, high-quality support for text, 2-D synthetic graphics, and imaging.

Visio (pre-purchase by Microsoft)

- Single platform (Windows) and first on the block with every new Microsoft-based development approach
  - "Visio has employed the latest Windows-based technology in every version of its software products."
  - OLE, COM, DCOM, etc.
- This imposes a system architecture on every Visio product and on every application built on Visio

Embedded systems

- Systems containing a combination of hardware and software must often make significant system architecture decisions
  - Airplanes are a great example: fly-by-wire is heavily motivated by the benefits of reducing hydraulics, which are heavier than coaxial cables (reducing fuel consumption)
- There is an area called HW/SW co-design that (in part) addresses this issue in computing

Developers' choice

- Sometimes the system architecture is selected by the developers (as opposed to imposed by the customers)
- The consequences of what architecture is selected is equally important
- It's "only" a question of who chooses

Benefits and costs

- A system architecture gives you a structure, along with some specific benefits (in principle, at least)
- And presumably you get some costs if you choose to go outside the architecture

Isn't it just design?

- No!
- It's at a completely different level (at least)
  - Surely not whether to use arrays or a linked list to represent a sequence
  - Not even how to design a symbol table
- It's high-level, very fundamental structural decisions

Examples

- A few architectures in a few domains
- The point is to show how high-level architectural decisions are so fundamental ("first day" decisions)
  - *Not* deep insights into those domains
- Ex: in your project, there is a major difference between a web-based and a non-web-based interface, which has consequences from the requirements to design to coding to testing to maintenance

Network architectures

- Circuit-switched
  - The telephone system
- Packet-switched
  - The Internet
- Rings
  - IBM SNA
- ...

Operating systems

- Layered architectures
  - THE, Venus, CP/M, etc.
- Benefits? Costs?
- This is not at the same level as design: i.e., these aren't abstract data types!

THE
- Level 5: User Programs
- Level 4: Buffering for I/O devices
- Level 3: Operator Console Device Driver
- Level 2: Memory Management
- Level 1: CPU Scheduling
- Level 0: Hardware

Operating systems

- Virtual machines
  - Build a kernel that provides a set of virtual machines, each of which is (almost) identical to the bare machine
  - Share the resources of the bare machine (CPU, disks, etc.)
- Virtual card readers
- Virtual printers
- Minidisks
- Etc.

How to select an architecture?

- So, what do we do in our project? There is no magic
  - Rechtin/Maier have about 200 heuristics!
- Part of it is developing as deep an understanding of the underlying technologies as you can
  - Risk: how much time can you spend?
- Then you have to in some sense "proof" your requirements against each choice

Which requirements?

- You should not consider only your behavioral requirements (what happens)
- But also issues such as, "How will we be able to deliver our minimal subset only in the face of time pressure?"

Risk-proofing

- You can never eliminate risks
- But you may find in choosing an architecture that you'll have some particular concerns
  - You may be able to build a quick-and-dirty prototype to address these concerns

Finally

- If you're worrying about data structures, data representations, algorithms, etc., then you're almost surely not thinking about your architecture
- First things first

Questions?