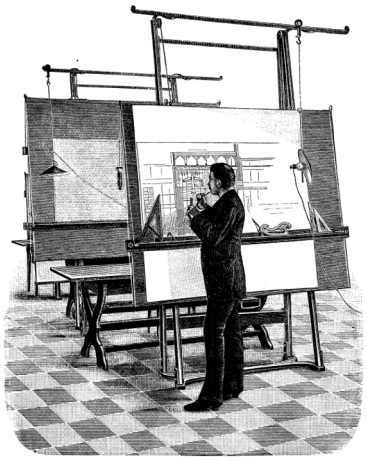
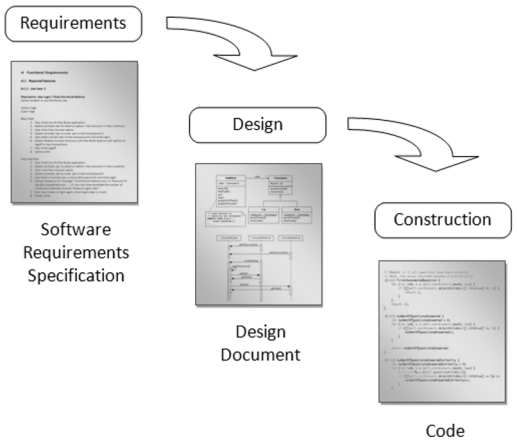
Software Design



"You can use an eraser on the drafting table or a sledgehammer on the construction site.“
--Frank Lloyd Wright

What is Software Design?

- Design bridges that gap between knowing what is needed (software requirements specification) to entering the code that makes it work (the construction phase).
- Design is both a verb and a noun.



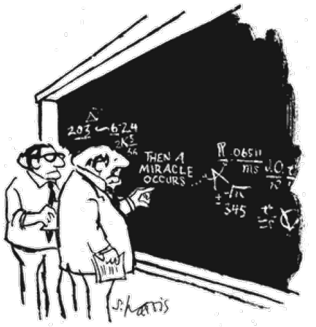
What is Software Design? [cont]

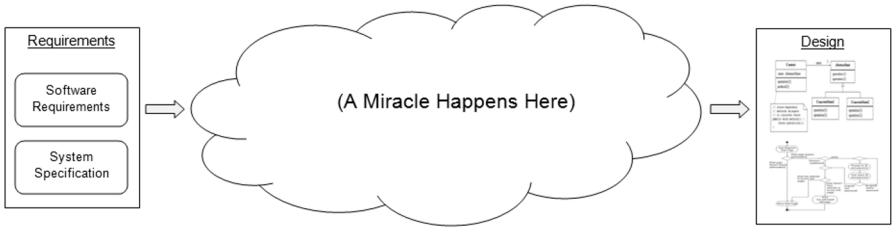
- During the design phase, software engineers apply their knowledge of the problem domain and implementation technologies in order to translate system specifications into plans for the technical implementation of the software.
- The resulting design expresses the overall structure and organization of the planned implementation. It captures the essence of the solution independent of any implementation language.

3

There is a famous cartoon showing two professors at a chalkboard examining a proof that includes the step "then a miracle occurs".

At times it seems like this is the most that can be hoped for during software design.

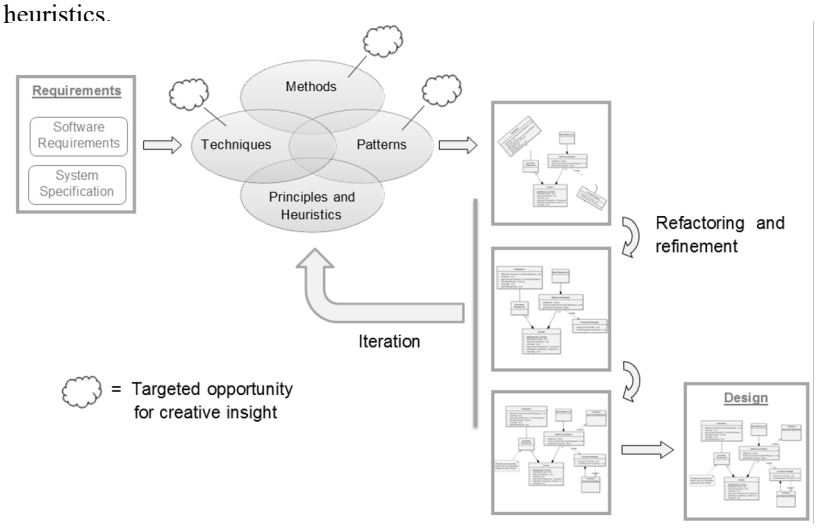




4

Relying on miracles or exceptional ingenuity isn't a reliable (predictable or repeatable) way of arriving at a design.

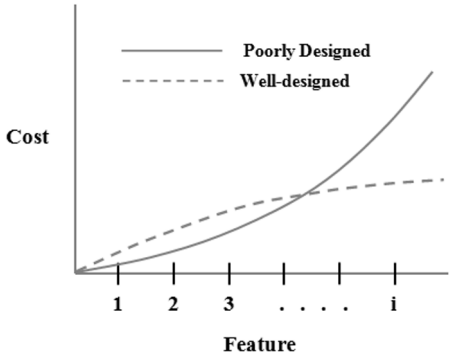
The design process can be made more systematic and predictable through the application of methods, techniques and patterns, all applied according to principles and heuristics.



5

Importance of Managing Complexity

- Poorly designed programs are difficult to understand and modify.
- The larger the program, the more pronounced are the consequences of poor design.



Cost of adding the i^{th} feature to a well-designed and poorly designed program

6

Two Types of Complexity in Software

- To better understand how good design can minimize technical complexity, it's helpful to distinguish between two major types of complexity in software:
  - Essential complexities – complexities that are inherent in the problem.
  - Accidental/incidental complexities – complexities that are artifacts of the solution.
- The total amount of complexity in a software solution is:

  Essential Complexities + Accidental complexities

7

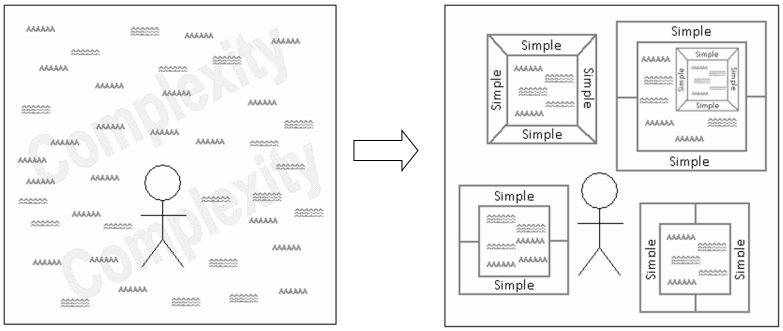
Design: An Antidote to Complexity

- Design is the primary tool for managing essential and accidental complexities in software.
- Good design doesn't reduce the total amount of essential complexity in a solution but it will reduce the amount of complexity that a programmer has to deal with at any one time.
- A good design will manage essential complexities inherent in the problem without adding to accidental complexities consequential to the solution.

8

Design Techniques for Dealing with Software Complexity

- Modularity – subdivide the solution into smaller easier to manage components. (divide and conquer)
- Information Hiding – hide details and complexity behind simple interfaces



9

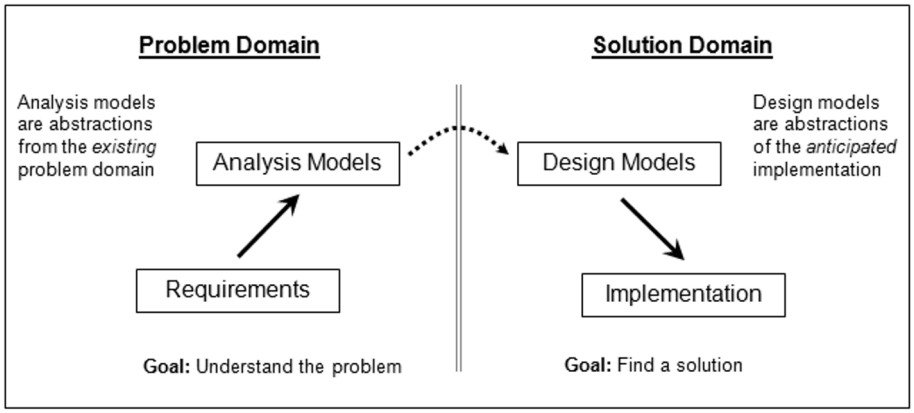
Additional Techniques for Dealing with Complexity

- Abstraction – use abstractions to suppress details in places where they are unnecessary.
- Hierarchical Organization – larger components may be composed of smaller components. Examples: a complex UI control such as tree control is a hierarchical organization of more primitive UI controls. A book outline represents the hierarchical organization of ideas.

10

Why Design is Hard

- Design is difficult because design is an abstraction of the solution which has yet to be created.



11

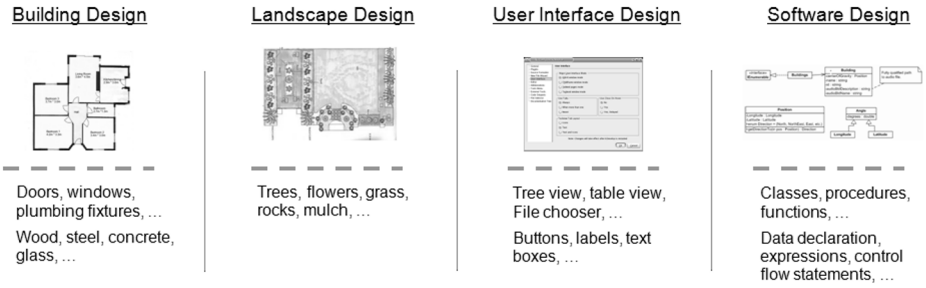
Design is a wicked problem

- The term wicked problem was first used to describe problems in social planning but engineers have recognized it aptly describes some of the problems they face as well.
- A wicked problem is one that can only be clearly defined by solving it.
- Need two solutions. The first to define the problem, and the second to solve it in the most efficient way.
- Fred Brooks could have been talking about wicked problems when he advised: "Plan to throw one away; you will anyhow."

12

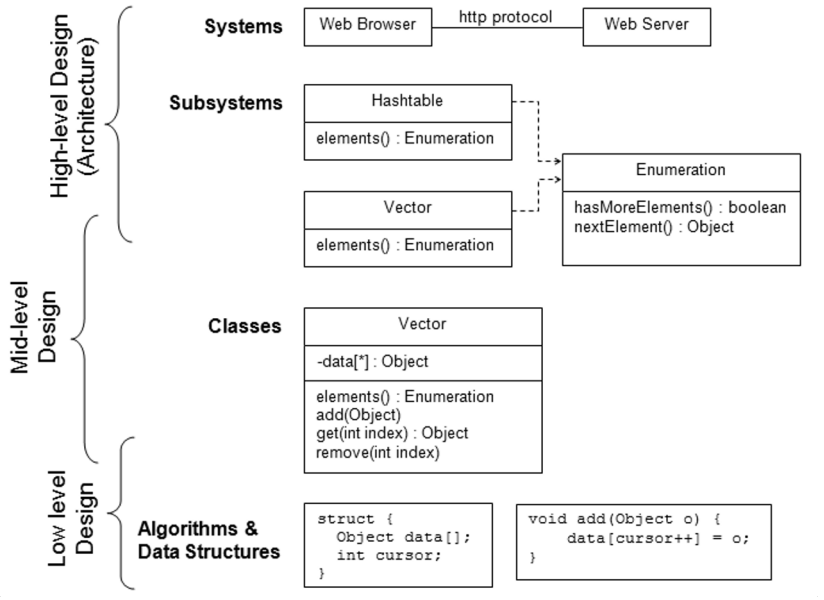
Design is a Universal Activity

- Any product that is an aggregate of more primitive elements, can benefit from the activity of design.



13

Design Occurs at Different Levels



Standard Levels of Design

14

Characteristics of Software Design

- Non-deterministic – A deterministic process is one that produces the same output given the same inputs. Design is non-deterministic. No two designers or design processes are likely to produce the same output.
- Heuristic – because design is non-deterministic design techniques tend to rely on heuristics and rules-of-thumb rather than repeatable processes.
- Emergent – the final design evolves from experience and feedback. Design is an iterative and incremental process where a complex system arises out of relatively simple interactions.

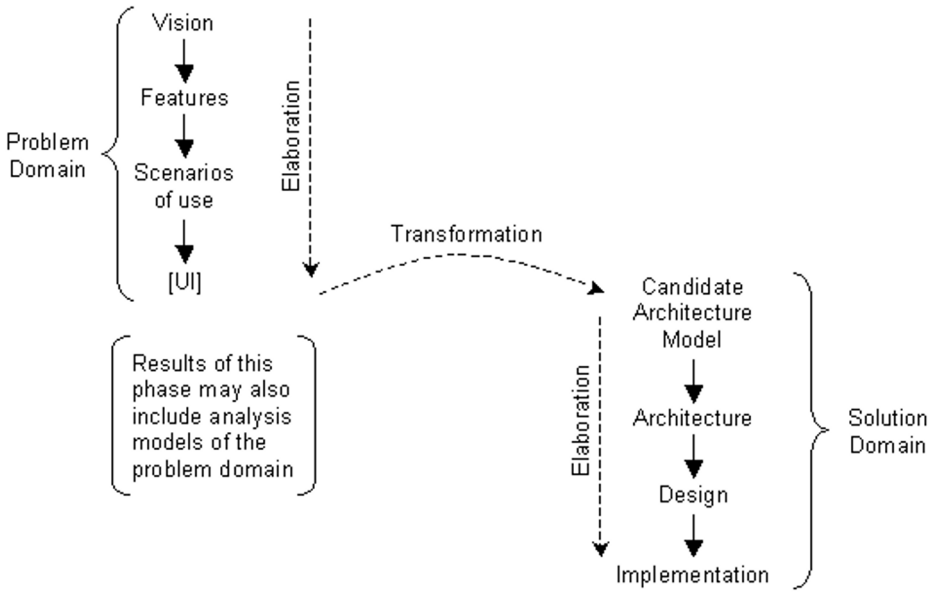
15

The Evolution of Designs

- Design as a single step in the software life cycle is somewhat idealized.
- More often the design process is iterative and incremental.
- Designs tend to evolve over time based on experience with their implementation.

16

Elaboration and Transformation



17

The Benefits of Good Design

- Good design reduces software complexity which makes the software easier to understand and modify. This facilitates rapid development during a project and provides the foundation for future maintenance and continued system evolution.
- It enables reuse. Good design makes it easier to reuse code.
- It improves software quality. Good design exposes defects and makes it easier to test the software.
- Complexity is the root cause of other problems such as security. A program that is difficult to understand is more likely to be vulnerable to exploits than one that is simpler.

18

A Generic Design Process

1. Understand the problem (software requirements).
2. Construct a "black-box" model of solution (system specification). System specifications are typically represented with use cases (especially when doing OOD).
3. Look for existing solutions (e.g. architecture and design patterns) that cover some or all of the software design problems identified.
4. Design not complete? Consider using one or more design techniques to discover missing design elements
   - Noun-verb analysis, CRC Cards, step-wise refinement, etc.
   - Take final analysis model and pronounce a first-draft design (solution) model
5. Consider building prototypes
6. Document and review design
7. Iterate over solution (Refactor) (Evolve the design until it meets functional requirements and maximizes non-functional reqs)

19

Inputs to the design process

- User requirements and system specification (including any constraints on design and implementation options)
- Domain knowledge (For example, if it's a healthcare application the designer will need some knowledge of healthcare terms and concepts.)
- Implementation knowledge (capabilities and limitations of eventual execution environment)

20

Desirable Internal Design Characteristics

- Minimal complexity – Keep it simple stupid (KISS). Maybe you don't need high levels of generality.
- Loose coupling – minimize dependencies between modules
- Ease of maintenance – Your code will be read more often then it is written.
- Extensibility – Design for today but with an eye toward the future. You ain't going to need it (YAGNI). Note, this characteristic can be in conflict with "minimize complexity". Engineering is about balancing conflicting objectives.

21

Desirable Internal Design Characteristics

- Reusability – reuse is a hallmark of a mature engineering discipline
- Leanness – when in doubt, leave it out. The cost of adding another line of code is much more than the few minutes it takes to type.
- Stratification – Layered. Even if the whole system doesn't follow the layered architecture style, individual components can.
- Standard techniques – sometimes it's good to be a conformist! Boring is good. Production code is not the place to try out experimental techniques.

22

The Object-Oriented ... Hype

- What are object-oriented (OO) methods?
  - OO methods provide a set of techniques for analysing, decomposing, and modularising software system architectures
  - In general, OO methods are characterized by structuring the system architecture on the basis of its *objects* (and classes of objects) rather than the *actions* it performs

- What is the rationale for using OO?
  - In general, systems evolve and functionality changes, but objects and classes tend to remain stable over time
  - Use it for **large systems**
  - Use it for **systems that change often**

OO Design vs. OO Programming

- Object-Oriented Design
  - a method for decomposing software architectures
  - based on the objects every system or subsystem manipulates
  - relatively independent of the programming language used

- Object-Oriented Programming
  - construction of software systems as
    - Structured collection of Abstract Data Types (ADT)
    - Inheritance
    - Polymorphism
  - concerned with programming languages and implementation issues