Design in Software Construction

Contents

- Introduction
- Design Challenges
- Key Design Concepts

Introduction

- Some people might argue that design isn't really a construction activity, but on small projects, many activities are thought of as construction, often including design.
- Even in some larger projects, a formal architecture might address only the system-level issues and much design work might intentionally be left for construction.

Introduction

- “Design” might be
  - just writing a class interface in pseudo code before writing the details.
  - Drawing diagrams of a few class relationships before coding them.
  - Asking another programmer which design pattern seems like a better choice.
- Regardless of how it’s done, small projects benefit from careful design just as larger projects do.
- Recognizing design as an explicit activity maximizes the benefit you will receive from it.

Design Challenges

- The phrase “software design” means the conception and invention of a scheme for turning a specification for computer software into operational software.
- Design is the activity that links requirements to coding and debugging.
- Good design is useful for small projects and essential for larger projects.

Design Challenges

- Design is marked by numerous challenges, which are (details from book Pg. 74-76)
  - Design is a wicked problem
  - Design is a sloppy process
  - Design is about tradeoffs and priorities
  - Design Involves Restrictions
  - Design Is Nondeterministic
  - Design Is a Heuristic Process
  - Design Is Emergent

Key Design Concepts

- Good design depends on understanding a handful of key concepts.
- Some of these key concepts are
  - Role of complexity
  - Desirable characteristics of designs
  - Levels of design

Managing Complexity

- When software-project surveys describe the causes of project failure, they rarely identify technical reasons as the primary causes of project failure.
- Projects fail most often because of poor requirements, poor planning, or poor management.
- But when projects do fail for reasons that are primarily technical, the reason is often uncontrolled complexity.
- The software is allowed to grow so complex that no one really knows what it does.
- When a project reaches the point at which no one completely understands the impact that code changes in one area will have on other areas, progress grinds to a halt.

Managing Complexity

- Managing complexity is the most important technical topic in software development.
- Dijkstra (Computing Pioneer) pointed out that no one's skull is really big enough to contain a modern computer program, which means that we as software developers shouldn't try to cram whole programs into our skulls at once.
- We should try to organize our programs in such a way that we can safely focus on one part of it at a time.
- The goal is to minimize the amount of a program you have to think about at any one time.

Managing Complexity

- At the software-architecture level, the complexity of a problem is reduced by dividing the system into subsystems.
- The goal of all software-design techniques is to break a complicated problem into simple pieces.
- The more independent the subsystems are, the more you make it safe to focus on one bit of complexity at a time.

Managing Complexity

- Overly costly, ineffective designs arise from three sources:
  - A complex solution to a simple problem
  - A simple, incorrect solution to a complex problem
  - An inappropriate, complex solution to a complex problem

Desirable Characteristics of a Design

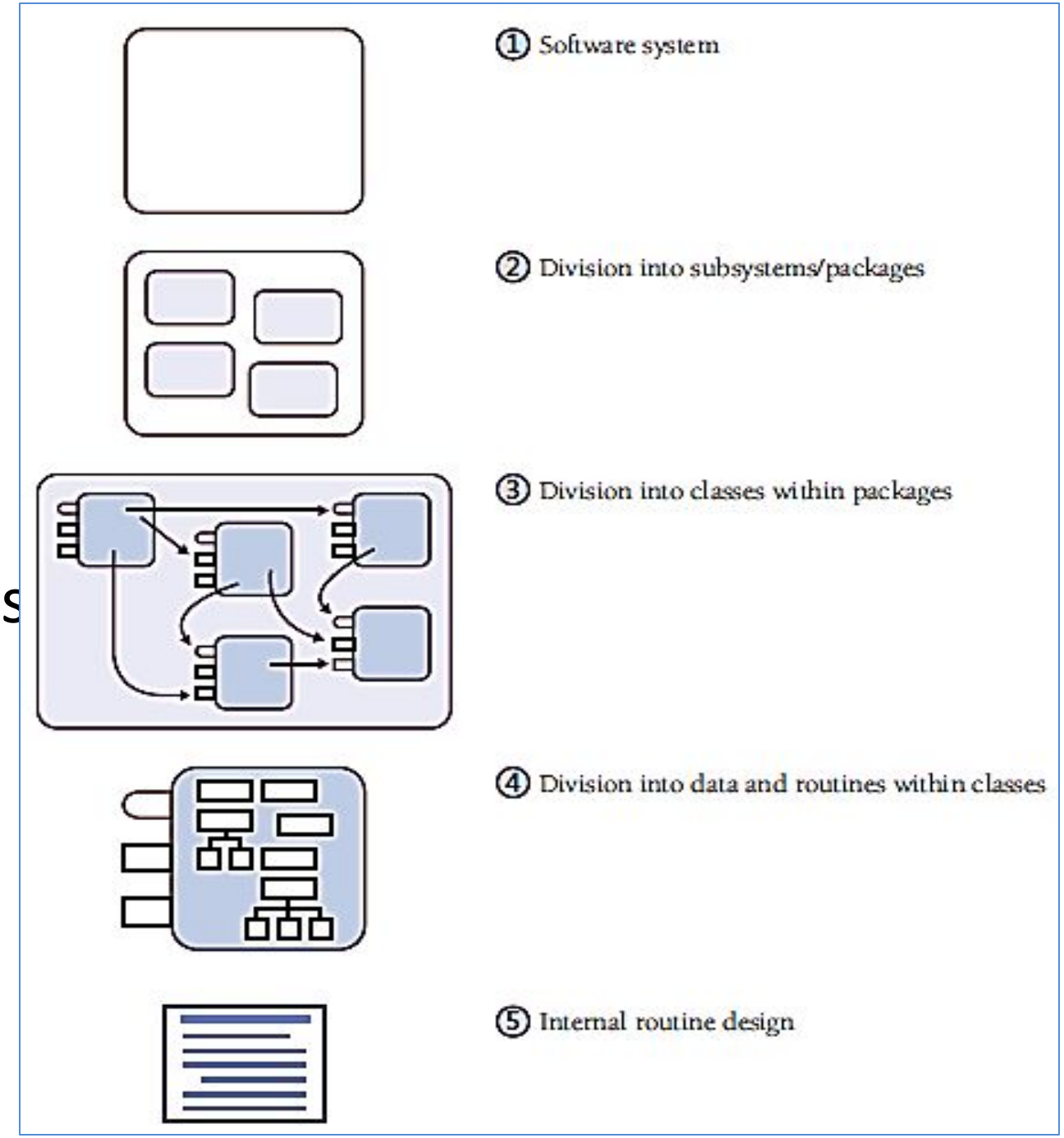
- A high-quality design has several general characteristics.
- If you could achieve all these goals, your design would be very good indeed.
- Some goals contradict other goals, but that's the challenge of design—creating a good set of tradeoffs from competing objectives.

Desirable Characteristics of a Design

- Here's a list of required design characteristics (details from book Pg. 80, 81)
  - Minimal Complexity
  - Ease of Maintenance
  - Loose Coupling
  - Extensibility
  - Reusability
  - Portability
  - LeStandard Techniques

Levels of Design

Design is needed at several different levels of detail in a software system.



Levels of Design
Level 1: Software System

- The first level is the entire system.
- Some programmers jump right from the system level into designing classes, but it's usually beneficial to think through higher level combinations of classes, such as subsystems or packages.

Levels of Design

Level 2: Division into Sub Systems/Packages

- The main product of design at this level is the identification of all major subsystems.
- The major design activity at this level is deciding how to partition the program into major subsystems and defining how each subsystem is allowed to use each other subsystem.
- Division at this level is typically needed on any project that takes longer than a few weeks.
- Within each subsystem, different methods of design might be used—choosing the approach that best fits each part of the system.

Levels of Design

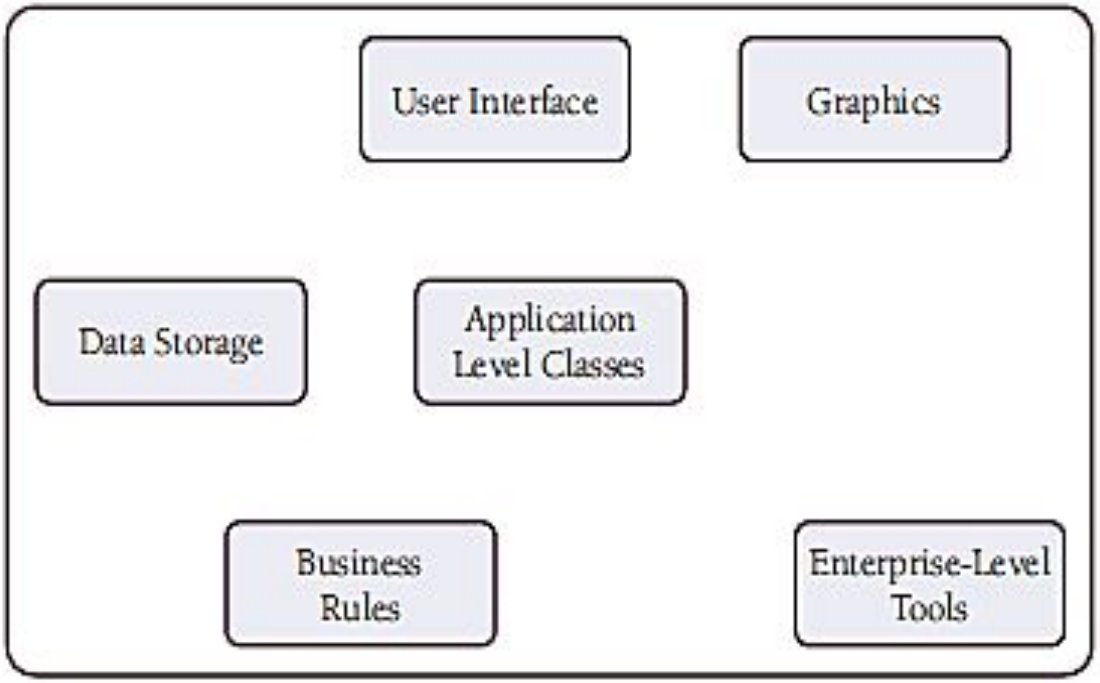
Level 2: Division into Sub Systems/Packages

- Of particular importance at this level are the rules about how the various subsystems can communicate.
- If all subsystems can communicate with all other subsystems, you lose the benefit of separating them at all.
- Make each subsystem meaningful by restricting communications.

Levels of Design

Level 2: Division into Sub Systems/Packages

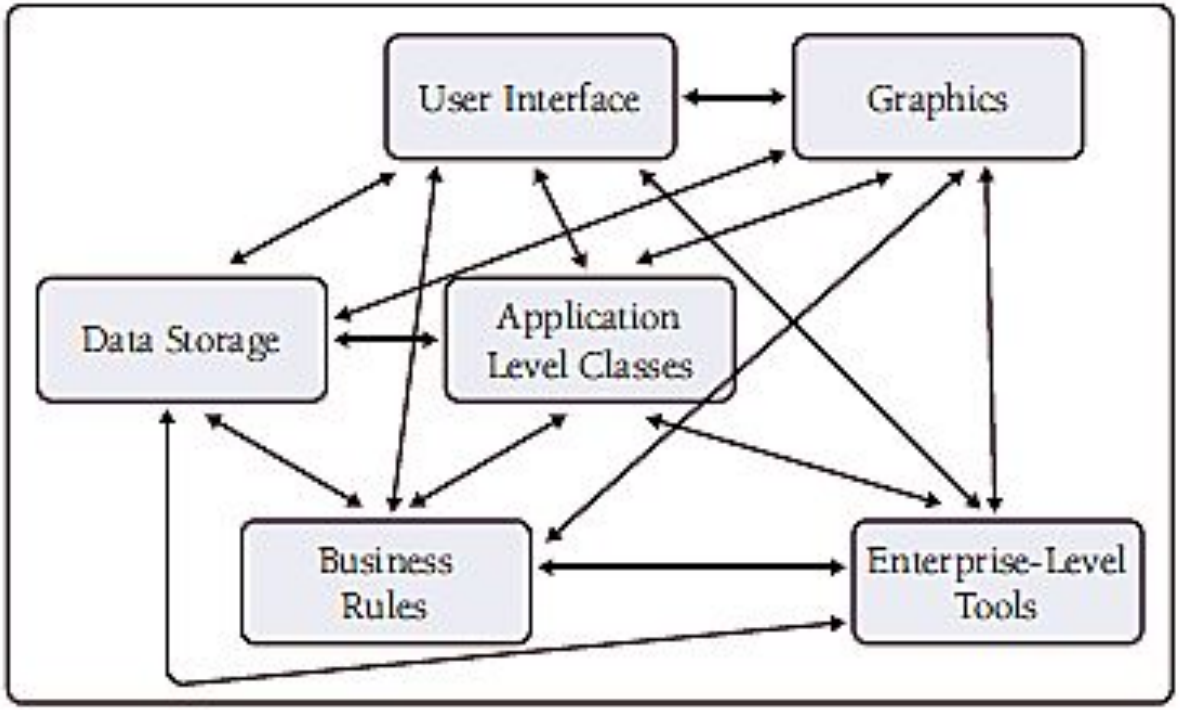
- Consider the following example. A system is divided into six sub systems



Levels of Design

Level 2: Division into Sub Systems/Packages

- This will happen when there is no restriction on inter subsystems communication



Levels of Design

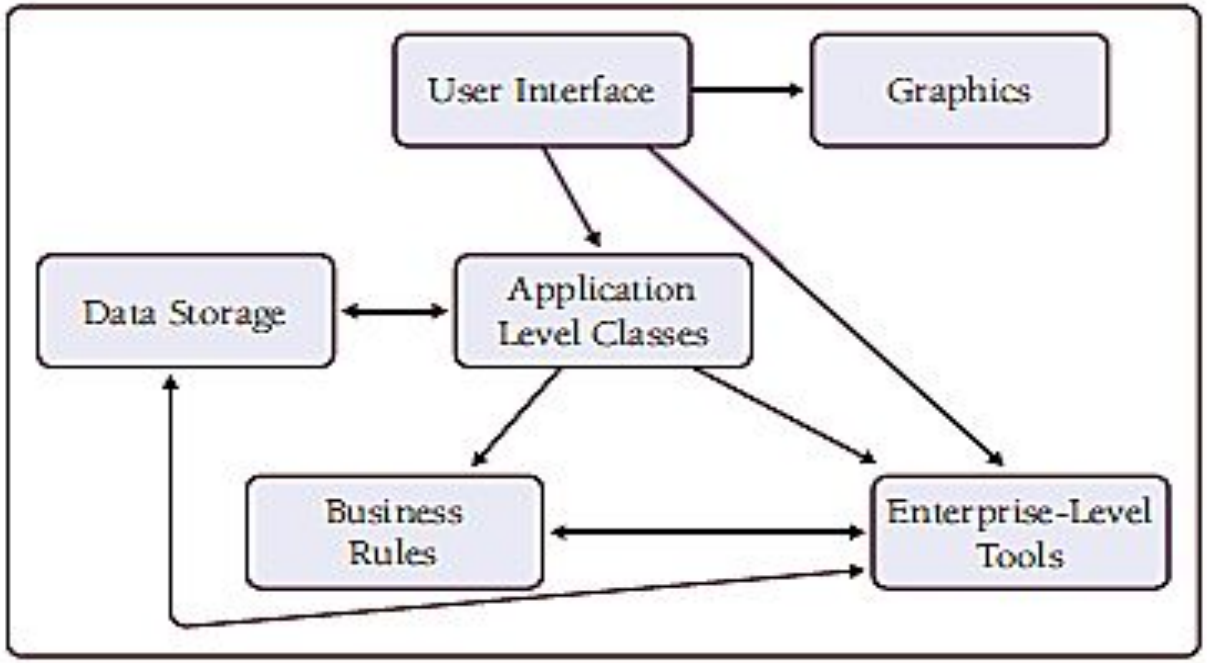
Level 2: Division into Sub Systems/Packages

- Every subsystem ends up communicating directly with every other subsystem, which raises some important questions:
  - How many different parts of the system does a developer need to understand at least a little bit to change something in the graphics subsystem?
  - What happens when you try to use the business rules in another system?
  - What happens when you want to put a new user interface on the system, perhaps a command-line UI for test purposes?
  - What happens when you want to put data storage on a remote machine?

Levels of Design

Level 2: Division into Sub Systems/Packages

- All of these issues can be addressed with little extra work.
- Allow communication between subsystems only on a “need to know” basis.



Levels of Design

Level 2: Division into Sub Systems/Packages

- A good general rule is that a system-level diagram should be an acyclic graph. In other words, a program shouldn't contain any circular relationships in which Class A uses Class B, Class B uses Class C, and Class C uses Class A.

Levels of Design
Level 3: Division into Classes

- Design at this level includes identifying all classes in the system.
- Details of the ways in which each class interacts with the rest of the system are also specified as the classes are specified.
- In particular, the class's interface is defined.
- Overall, the major design activity at this level is making sure that all the subsystems have been decomposed to a level of detail fine enough that you can implement their parts as individual classes.

Levels of Design
Level 4: Division into Routines

- Design at this level includes dividing each class into routines.
- The class interface defined at Level 3 will define some of the routines.
- Design at Level 4 will detail the class's private routines.
- When you examine the details of the routines inside a class, you can see that many routines are simple boxes.

Levels of Design
Level 4: Division into Routines

- The act of fully defining the class's routines often results in a better understanding of the class's interface, and that causes corresponding changes to the interface—that is, changes back at Level 3.
- This level of decomposition and design is often left up to the individual programmer, and it's needed on any project that takes more than a few hours.
- It doesn't need to be done formally, but it at least needs to be done mentally.

Levels of Design

Level 5: Internal routine Design

- Design at the routine level consists of laying out the detailed functionality of the individual routines.
- Internal routine design is typically left to the individual programmer working on an individual routine.
- The design consists of activities such as writing pseudo code, looking up algorithms in reference books, deciding how to organize the paragraphs of code in a routine, and writing programming-language code.