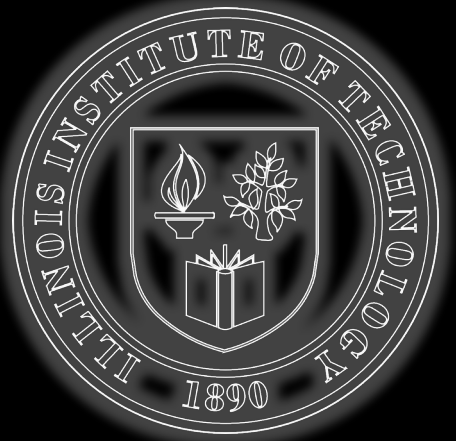
ILLINOIS TECH | College of Computing

ITMD-511 : Chapter 3

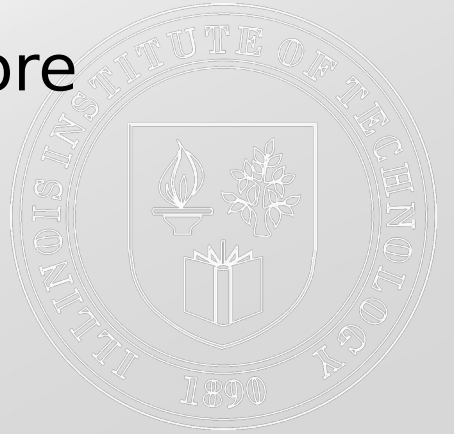
Chapter 3 : Measure Twice, Cut Once!

Upstream Prerequisites



Chapter 3 Introduction

- What do we do before construction of our software?
- **Prerequisites**
  - Exploring terminology and methods before development
  - Each project is different, thus the preparation for a project must be different
  - This is referred to as an upstream activity (architecture, design, and project planning)
- The success or failure of a project can be determined before construction begins.
- *Measure twice, cut once*
  - Be prepared for your project!

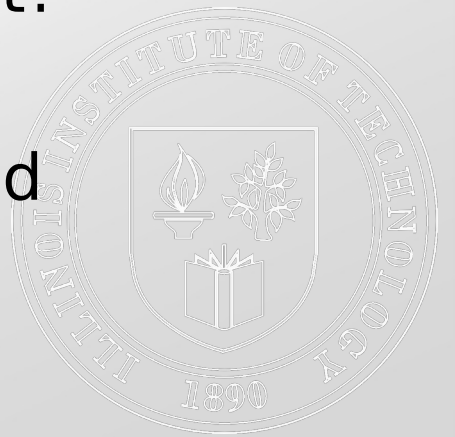


What are Prerequisites?

- Prerequisites are a form of high-quality practices that can help build high quality software!
- We can attempt to use high-quality practices at various parts of our code.
  - End of project: system testing is emphasized
  - Middle of project: construction/coding practices are emphasized
  - Beginning of project:
    - ►We define a problem, determine the solution, and design the solution

Do We Really need Prerequisites?

- Yes! Not enough data supports ignoring upstream activities (prerequisites)! We need to use them to prepare our project
- The overarching goal of prerequisites is **risk reduction.**
  - Good project planning clears a huge chunk of major risks out of the way as early as possible.
- What are the most common risks in software development?
  - Poor project requirements and poor project planning
- There is not a single way to reduce risk, it must be decided project by project



Causes of Incomplete Preparation

Some developers are assigned to create prerequisites and don’t have the expertise

Individuals ready to code jump right into the project and ignore the starting phases

Sometimes managers do not care about the prep, they just want the product to be!

Why we need Prerequisites before Construction

- Planning a project avoids wasting money or time on building incorrect systems
  - Always think about **how** to build the system before building it. Don’t spend time and efforts on areas that don’t matter.
- Construction projects need architectural drawings, blueprints, and reviews to be done before we start working.
  - We don’t start putting up wood and pouring concrete before these are approved, we would get in trouble!
- If we don’t plan properly, we will most likely run into **defects**

The cost of defects!

- Leaving defects to be found until later can be very costly
  - A defect found during the architecture phase can cost $1000 to fix, whereas if the same defect is found during construction it will cost $15000 to fix.

Table 3-1 Average Cost of Fixing Defects Based on When They're Introduced and Detected

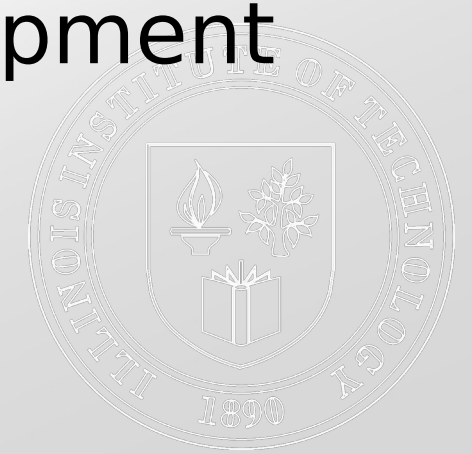
	Time Detected				
Time Introduced	**Requirements**	**Architecture**	**Construction**	**System Test**	**Post-Release**
Requirements	1	3	5–10	10	10–100
Architecture	—	1	10	15	25–100
Construction	—	—	1	10	10–25

Source: Adapted from "Design and Code Inspections to Reduce Errors in Program Development" (Fagan 1976), *Software Defect Removal* (Dunn 1984), "Software Process Improvement at Hughes Aircraft" (Humphrey, Snyder, and Willis 1991), "Calculating the Return on Investment from More Effective Requirements Management" (Leffingwell 1997), "Hughes Aircraft's Widespread Deployment of a Continuously Improving Software Process" (Willis et al. 1998), "An Economic Release Decision Model: Insights into Software Project Management" (Grady 1999), "What We Have Learned About Fighting Defects" (Shull et al. 2002), and *Balancing Agility and Discipline: A Guide for the Perplexed* (Boehm and Turner 2004).

The data in Table 3-1 shows that, for example, an architecture defect that costs $1000 to fix when the architecture is being created can cost $15,000 to fix during system test. Figure 3-1 illustrates the same phenomenon.

Determine the Software we are working on

- Different projects call for different preparation and construction styles
  - While very project is unique, they do tend to fall into general development styles.
- The next slide shows these general development styles



Determine the Software we are working on

1. Business Systems
2. Mission-Critical Systems
3. Embedded Life-Critical Systems.

Table 3-2 Typical Good Practices for Three Common Kinds of Software Projects

	Kind of Software		
	Business Systems	**Mission-Critical Systems**	**Embedded Life-Critical Systems**
Typical applications	Internet site Intranet site Inventory management Games Management information systems Payroll system	Embedded software Games Internet site Packaged software Software tools Web services	Avionics software Embedded software Medical devices Operating systems Packaged software
Life-cycle models	Agile development (Extreme Programming, Scrum, time-box development, and so on) Evolutionary prototyping	Staged delivery Evolutionary delivery Spiral development	Staged delivery Spiral development Evolutionary delivery

Determine the Software we are working on

- These three themes will have an infinite number of variations and will require different approaches.
  - Business system projects benefit from highly **iterative** approaches
    - Planning, requirements, and architecture are put together with construction, system testing etc.
  - Life critical systems benefit from **sequential** approaches
    - A sequential approach allows the projects requirement to be stable and lead to a very high level of reliability for the project

Iterative & Sequential Approaches?

- In simpler terms
  - Iterative approaches discover defects in our code as we progress through development
  - Sequential approaches rely on testing to discover defects in our code during development
- Most projects aren’t completely sequential or iterative.
  - We usually want to identify the most critical requirements and architecture early but may add onto them later.

Iterative vs Sequential : When to use them?

- Choose sequential (up-front) approach when
  - The requirements are stable
  - Design is straightforward and well understood
  - Development team is familiar with the applications
  - Little risk in the project
  - Long-term predictability is important
  - Cost of changing requirements, design, and code later is high

Iterative vs Sequential : When to use them?

- Choose an iterative (as-you-go) approach when
  - Requirements are not well understood or are unstable
  - Design is complex or challenging
  - Development team is not familiar with the applications
  - Project contains a lot of risk
  - Long-term predictability is not important
  - Cost of changing requirements, design, and code later is low.

Which one do we use a lot?

- Usually, iterative approaches are more useful than sequential approaches.
  - Can adapt prerequisites to the specific project a bit easier

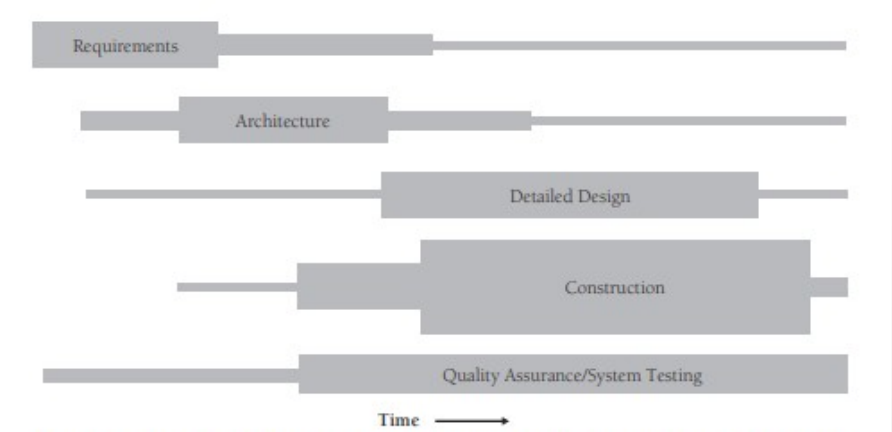


Figure 3-2 Activities will overlap to some degree on most projects, even those that are highly sequential.

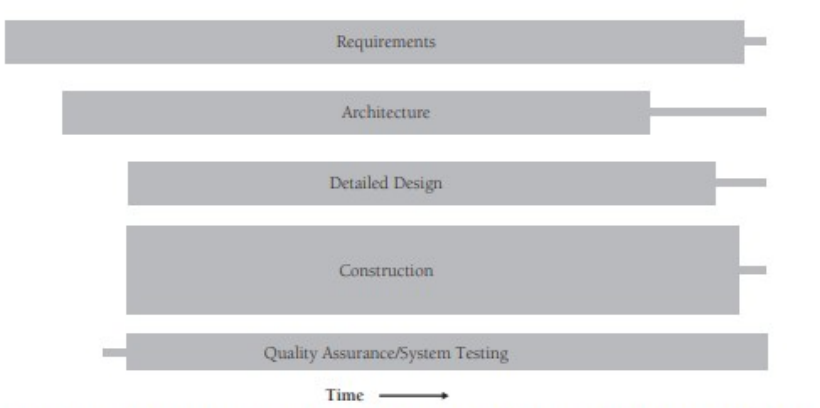


Figure 3-3 On other projects, activities will overlap for the duration of the project. One key to successful construction is understanding the degree to which prerequisites have been completed and adjusting your approach accordingly.

Problem-Definition Prerequisite

- Before jumping into coding, we need to think about the problem.
- A problem definition defines what the **problem** is without any reference to possible solutions
  - Example: We can’t keep up with orders from Apple!
- What is a bad problem definition?
  - Example: We can’t keep up with orders from Apple, so we need to optimize our automated data-entry system
  - This is bad because it states a possible solution

Problem-Definition Prerequisite

- Why do we not want to say a solution?
- The problem definition comes **before** the detailed requirements work, which goes more in-depth for solutions.
- The problem definition should be from the aspect of the user.
  - Use the user language and their point of view
  - This avoids approaching all solutions from a programmer's mindset
    - If we state a solution in the problem definition, our entire project will be geared towards finding that solution. It might not even be correct!

Problem-Definition Prerequisite

- If you fail to define a problem, you can waste a lot of time solving the wrong problem.

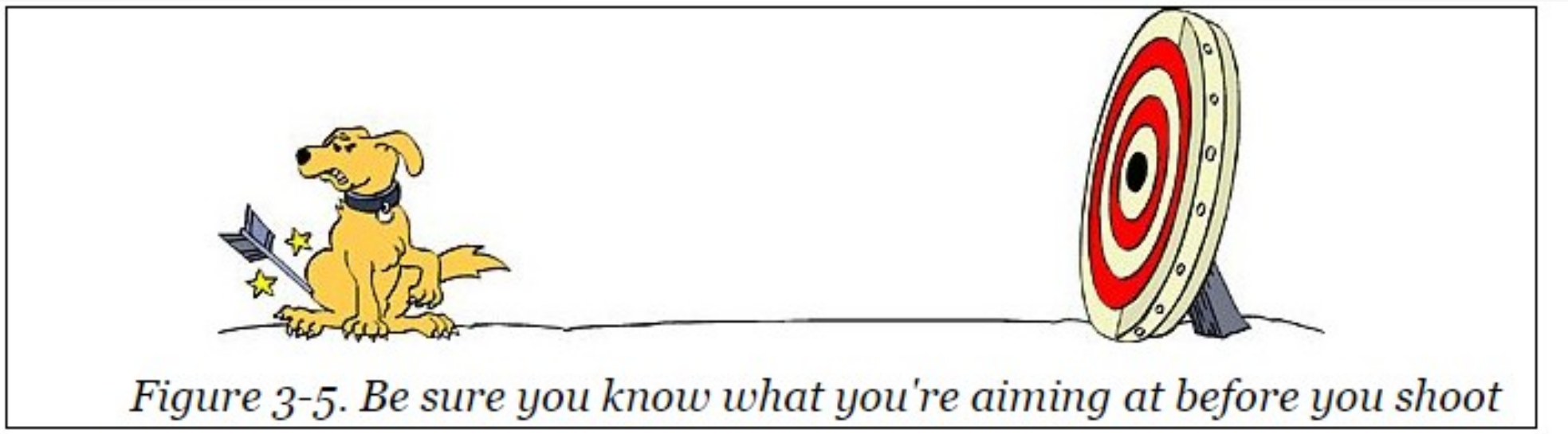


Figure 3-5. Be sure you know what you're aiming at before you shoot

Requirements Prerequisite

- This occurs **after** we figure out our problem definition!
- Requirements describe in **detail** what a software system is supposed to do.
  - These are the first step toward a solution
  - This can also be referred to as
    - Requirements development, Requirements analysis, analysis, requirements definition, software requirements, specification, spec, functional spec.

Requirements Prerequisite

- Why do we need official requirements?
  - Explicit or official requirements allows the user to agree to them and help us narrow down we were are supposed to develop
  - This allows the **user** to drive the requirements, rather than the programmer.
    - Avoids guessing what the user wants
  - This decides the **scope** of the program.
  - If you are confused on what the software needs to do, review the requirements that the user wants.
    - This helps minimize changes to a system after development, which can be extremely expensive to change.

Requirements Prerequisite

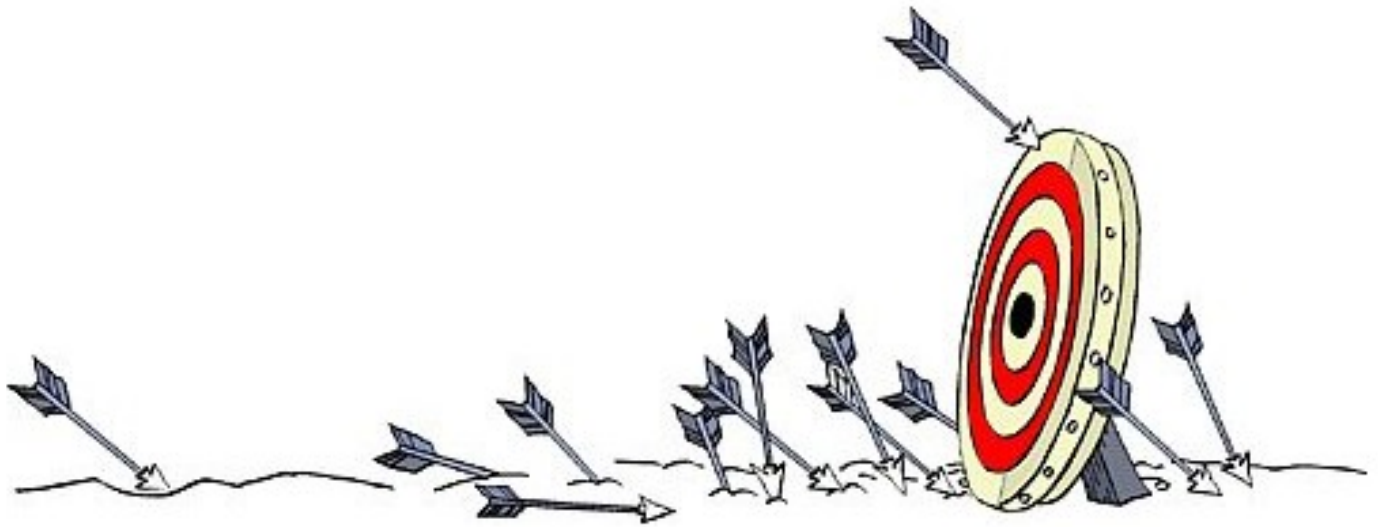


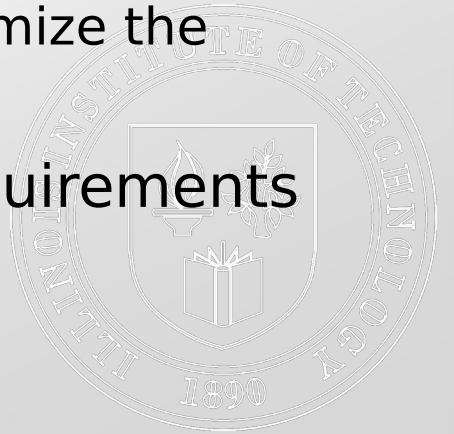
Figure 3-6. Without good requirements, you can have the right general problem but miss the mark on specific aspects of the problem

Myth of Stable Requirements

- It is hard to describe what will always be needed before code is written
  - The more a project is worked on the more it is understood.
- Studies report that a 25% change in requirements will occur during development.
  - Didn’t we state that changes in requirements will be costly?
  - How can we minimize the impact of this?

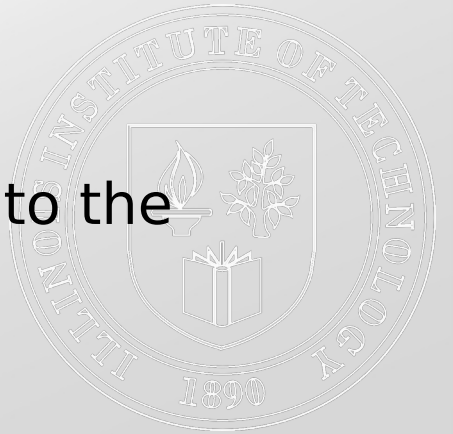
Handling Requirement Changes during Development

- Book provides a checklist to aide this process
- Ensure that you are always headed in the right direction
  - If you need to travel East and see you are travelling West, don’t debate anything, just stop travelling West.
- Ensure your team and clients know about the cost of requirement changes
  - I want XYZ on my software, let them know it will cost money to do this!
  - Set up a procedure if changes are a constant occurrence to help minimize the impact
- Business requirements may not match up with prerequisite requirements
  - Cool code ideas may be terrible for the business value of the product.



Architecture Prerequisite

- There is a LOT of information here, so don’t feel overwhelmed!
- This is the high-level part of software design, the frame that holds detailed parts together.
- The quality of architecture determines the conceptual integrity of the system, which determines overall quality.
  - Provides a top-down level of guidance for programmers.
- Good software architecture makes construction easy
  - Bad software architecture makes construction impossible!
- Changes to the architecture will incur a similar cost like changes to the requirements.
- This will act as our **blueprint** during our development process!



Architecture Prerequisite

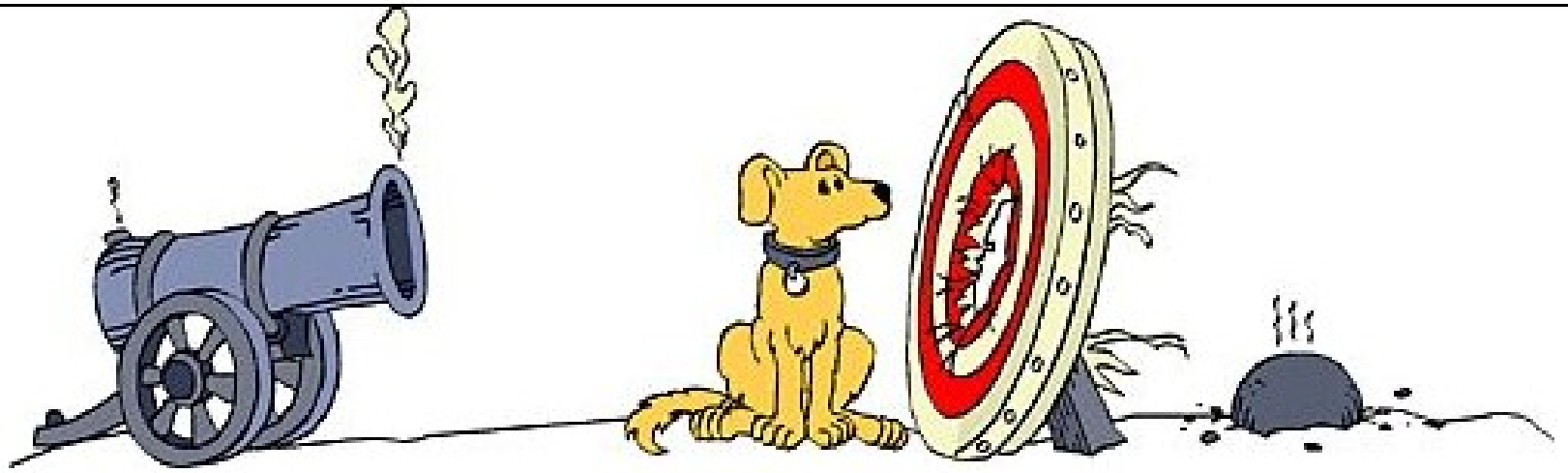


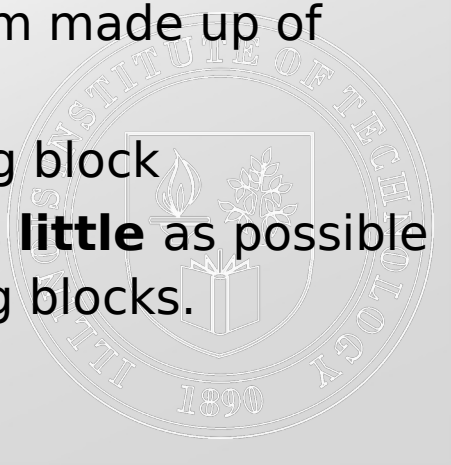
Figure 3-7. Without good software architecture, you may have the right problem but the wrong solution. It may be impossible to have successful construction

Typical Architectural Components

- The following slides details what some aspects of a blueprint for software architecture might look like.
- There is a lot of information on the next slides, but they are accounting for various scenarios that may occur during software development.

Typical Architectural Components

- Program Organization
  - A broad overview that describes the system
    - 12 puzzle pieces that fit together. Ensure you can explain it easily
  - Frustrating to work on classes that aren't really described properly.
    - I don't understand where the class fits?
  - Define the major building blocks in a program.
    - Depending on the size, each block may be a single class, or a subsystem made up of multiple classes.
    - Every feature listed in the requirements should be covered by a building block
    - Each building block should have **one** area of responsibility and know as **little** as possible about the other building blocks. Localize information into single building blocks.
    - Communication rules for each building block should be well defined



Typical Architectural Components

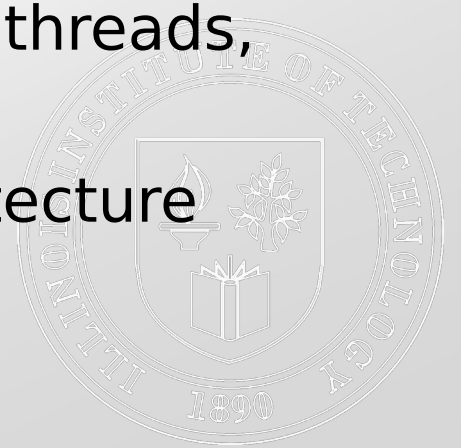
- Major Classes
  - Architecture should specify the major classes to be used
    - Identify the responsibilities of each major class and how the class interacts with other classes
    - Include Hierarchies, state transitions, and object persistence
    - Giving reasons for why it is organized in such a way will also help
    - 80/20 rule:
      - Specify 20 percent of the classes that make up 80 percent of the system's behavior

Typical Architectural Components

- Data Design
  - Should describe the major files and table designs to be used.
  - Specify the high-level organization and contents of any database used.
  - Explain why to use a single or multiple databases
- Business Rules
  - If architecture depends on certain rules, it should identify them.

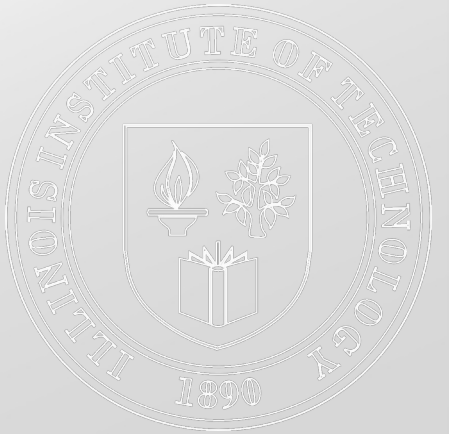
Typical Architectural Components

- User Interface Design
  - Specify major elements of web page formats, GUI's, command lines etc.
  - Careful user interface architecture helps a lot!
- Resource Management
  - Managing scare resources such as database connections, threads, handles etc.
  - Memory management is another important area for architecture to treat memory-constrained applications



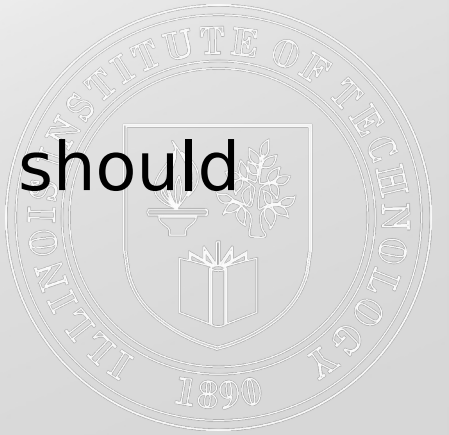
Typical Architectural Components

- Security
  - Code should be constructed with security in mind
  - Guidelines to handle buffers, rules for untrusted data, encryption, level of detail for error messages, secret data
- Performance
  - If a concern, goals should be specified here
  - Including resource use, resource priorities, speed vs. memory vs. cost



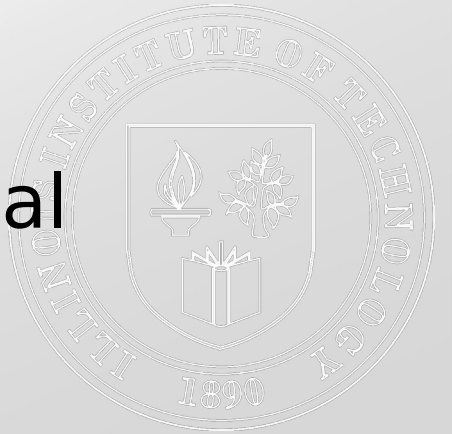
Typical Architectural Components

- Scalability
  - Ability for a system to grow to meet future demands
  - Describe how the system will address growth in user, server, network, databases, transactions and so on.
  - If the system is not expected to grow, then don’t worry!
- Interoperability
  - If the system is expected to share data or resources, should describe how this is accomplished



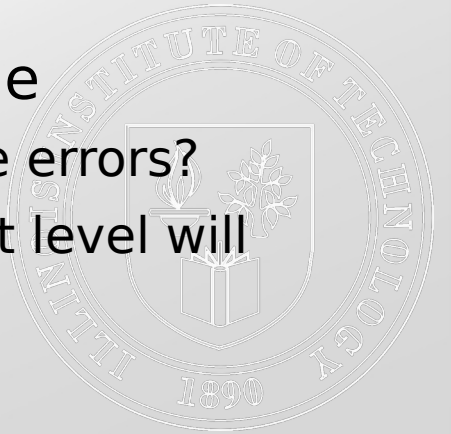
Typical Architectural Components

- Internationalization
  - Technical activity of preparing a program to support multiple locales.
    - If a system is interactive with various prompts and messages this would be a crucial thing to develop!
- Localization
  - Translation of a program to support specific local languages



Typical Architectural Components

- Input/Output
  - Describe how we read in inputs and how errors are detected
    - At the field, record, stream, or file level
- Error Processing
  - Larger issue that can’t be dealt with easily
    - Sometimes 90% of a program’s code is written for exception, error-processing cases or housekeeping.
  - This has systemwide implications, so don’t just treat it as code
    - Is it corrective or detective? Active or passive? How do we propagate errors?
    - How do we handle error messages, will exceptions be handled? What level will errors be handled at?
    - Level of responsibility for each class with input validation

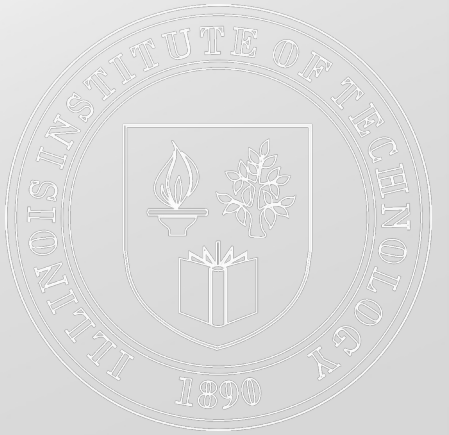


Typical Architectural Components

- Fault Tolerance
  - Architecture should indicate the kind of fault tolerance expected
  - A collection of techniques that increase a system's reliability, and how to recover from them.
- Architectural Feasibility
  - Describe how the system is technically feasible, or if there are drawbacks it should describe those as well.

Typical Architectural Components

- Overengineering
  - Describe how to overengineer class so programmers don’t do it automatically.
- Buy-vs.-Build decisions
  - Describe products to buy rather than having to build them
- Reuse Decisions
  - Describe how to reuse software if we are doing so



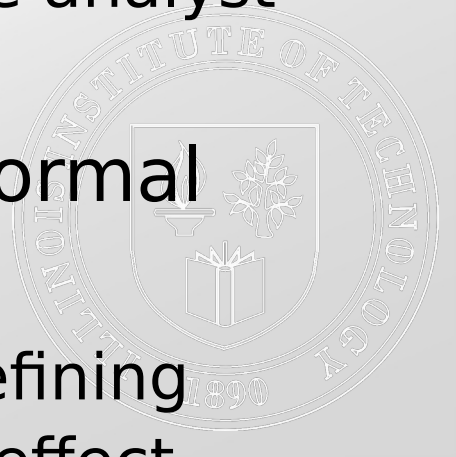
Typical Architectural Components

- Change Strategy
  - Describe a strategy for handling changes
    - Putting version numbers in data files
    - Reserve fields for future use
    - Design files so you can add new tables

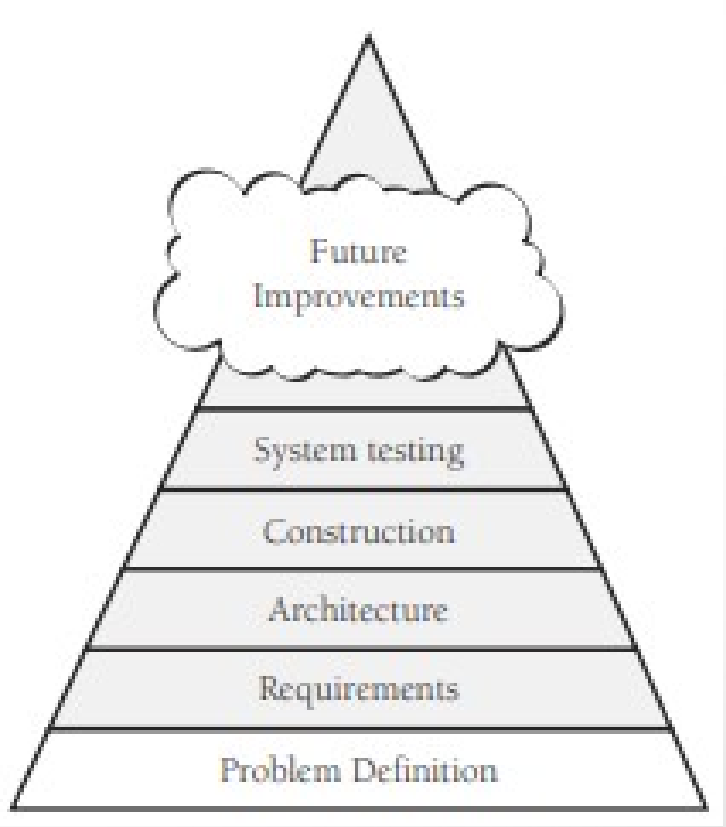
"Design bugs are often subtle and occur by evolution with early assumptions being forgotten as new features or uses are added to a system" –Fernando J. Corbato

How much time do I spend on this?

- Amount of time varies according to the Project's needs
  - Requirements are unstable and it's a large, formal project?
    - Work with requirements analyst. Allow time with the analyst to revise requirements before starting work
  - Requirements are unstable and it's a small, informal project?
    - Solve requirement issues yourself, allow time for defining



The overall design process



Key Points

- Overarching goal of preparing for construction is risk reduction
- If you want to develop high quality software, attention to quality must be part of the development process from the start to end
- Help educate fellow programmers and bosses about this process
- The type of project you're working on significantly affects construction prerequisites
  - Iterative vs. sequential
- If a good problem definition isn't made, you might be solving the wrong problem

Key Points

- If good requirement work isn’t done, you might have missed important details of the problem
- If good architectural design isn’t done, you might be solving the right problem in the wrong way
- Understand what approach must be taken to the construction prerequisites on your project, and choose the approach accordingly.