Week 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Start Time | End Time | Total Time | Particulars |
| 8/1/2020 | 11:14AM | 11:44AM | 30 | Week 2 video 1 and 2 |
| 12/01/2020 | 1:03 AM | 2:00 AM | 60 | Design |
| 13/01/2020 | 11:00 PM | 12:30 AM | 90 | Design completed. Implementation started |
| 15/01/2020 | 10:30PM | 11:00PM | 30 | Implementation continued. |
| 17/01/2020 | 10:50AM |  |  |  |

**Software Design Introduction:**

Software design is what we call the deliverable, design, the noun, and what we call the process to make that design.

Design, the verb, is the creative process of transforming the problem into a solution. In our case, transforming a requirement specification into a detailed description of the software that's code-ready.

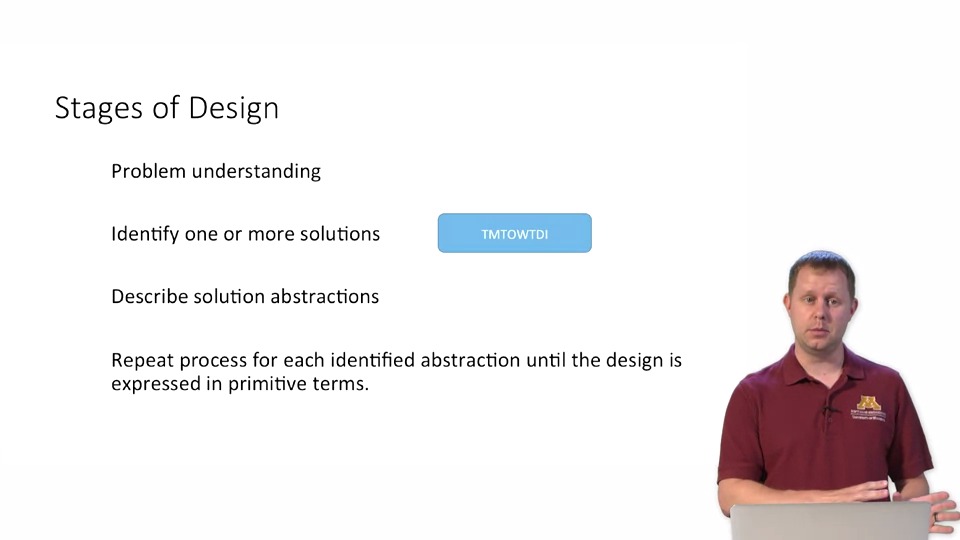
Architecture vs Design:

Architecture:

Large scale decision like

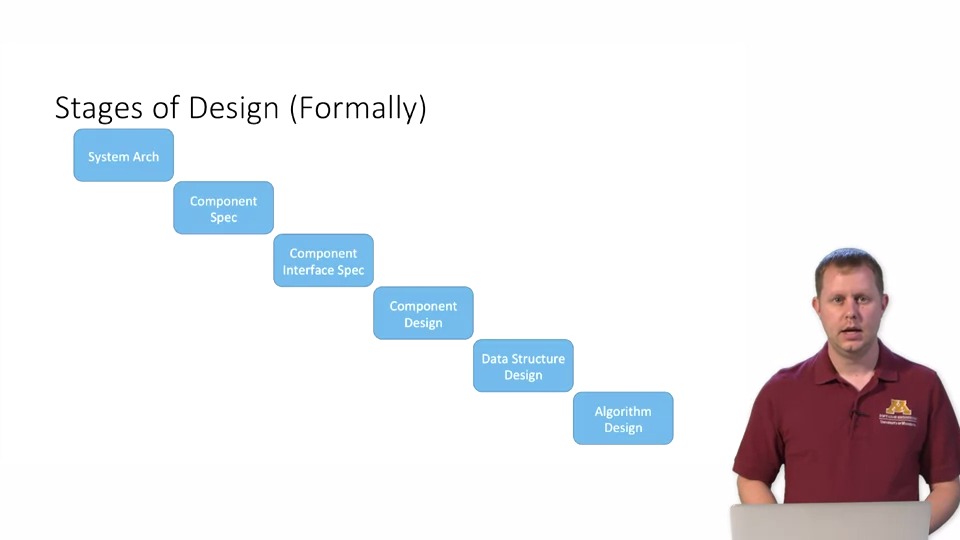
1. Buy or build software
2. Security
3. Appointing Resources
4. Fund related
5. Cost

Stages of Design:



TMTOWTDI: There’s more than one way to do it. There is almost always another way to reach the same singular goal, so consider multiple alternatives before deciding definitively which one to pursue.

Formal Stages of Design:



Architecture: Separate behaviour responsibility into components and, determine how those components will interact through interfaces.

Design: Design individual components in isolation.

**Software Design Modularity:**

Aspects of Modularity:

1. Coupling:

Defines how well does a module work together.

1. Cohesion:

Defines how well a module meets a single well defined goal.

1. Information Hiding
2. Information hiding describes our ability to abstract away information and knowledge in a way that allows us to complete complex work in parallel without having to know all the implementation details concerning how the task will be completed eventually.
3. Basically we know, what a module is doing and not how.
4. Data Encapsulation
5. data encapsulation refers to the idea that we can contain constructs and concepts within a module, allowing us to much more easily understand and manipulate the concept when we're looking at it in relative isolation.
6. It ensures protecting data from unauthorised access and maintaining integrity.
7. Only developer can modify the data.
8. And if in future we know that data is corrupted, it will happen only inside the module.
9. It makes software robust because later we can upgrade the module and

Coupling, cohesion are measures of how well modules work together and how well each individual module meets a certain single well-defined task and they tend to go together

Coupling

Primary Goals of Modularity:

1. Decomposability
2. Composability
3. Ease of Understanding

**Software Design: Coupling**

Coupling: Gives measure of how tightly coupled one module is to another.

Idea is when one module is changed, we hope/ensure that the change doesn’t affect other modules. It can be achieved only my low coupling. i.e. modules are not tightly bound.

Coupling levels:

1. Tight
2. Content coupling

Occur when module A relies directly on local data of module B.

1. Common coupling:

It occurs when both modules rely on common global data/variable.

1. External coupling:

It occurs when modules rely on external format e.g. protocol. It is highly tight coupling and can affect a large number of modules. Some abstractions are tried to create between module data and external format to minimise the impact.

1. Medium
2. Control coupling:

When one module control the logical flow of other module. Like what to do or how to do. Thus, changing first module affects the other module.

1. Data-structure coupling:

This occur when two modules rely on same composite data-structure.

1. Loose
2. Data coupling:

It occurs when modules share parameters i.e elementary data like integer.

1. Message Coupling:

Loosest type of coupling. It occurs when components interact only through message or parameters.

1. No coupling:

Trivial case. This case doesn’t require our attention because any modules of no coupling aren’t of great importance or complexity.

**Software Design: Cohesion**

It tells how well within a module does components fit together.

Levels of cohesion:

1. Weak: when components are similar but not enough to be put together.
   1. Coincidental cohesion:

Two piece of code are coincidental if they are in same file. They are cohesive only because of their proximity.

* 1. Temporal cohesion:

Codes are connected because they are activated at same time.

* 1. Procedural cohesion:

Also a time based cohesion. Components are activated one after other

* 1. Logical cohesion:

It occurs when two components have similar function.

1. Medium:
   1. Communicational cohesion: It occur when two components work on same input and have same format of output.
   2. Sequential cohesion:

It is better form of procedural cohesion. The components are not just one after another, the part of first component provides input to the other component.

1. Strong:
   1. Object cohesion:

Every operation in a module is allowed to modify object attributes. Each part is designed for purpose within the object itself.

* 1. Functional cohesion:

Better than sequential cohesion. Her, each part of the component is necessary for the execution of a single well defined function or a behaviour.

Cohesion and Inheritance:

Inheritance weakens cohesion.

In inheritance, the child class doesn’t have all functionality. It has to visit parent/super class. However, inheritance has its own benefit over cohesion.

**Inheritance:**

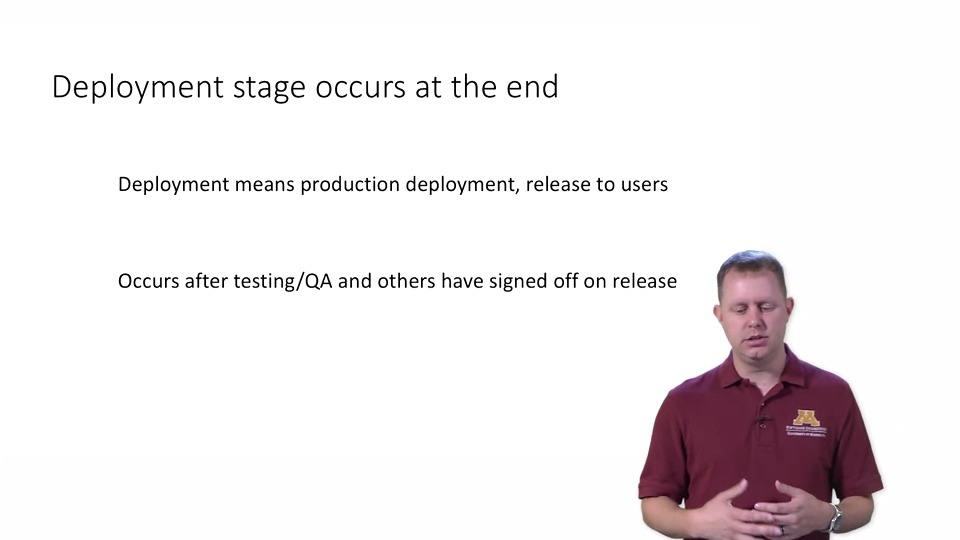
Let your comments describe the **why** and let the code describe the **how.**

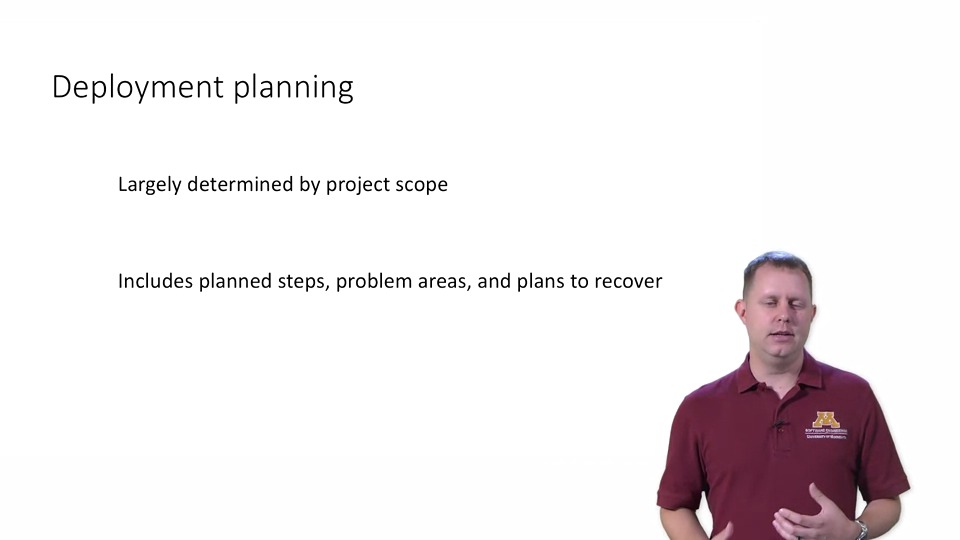
The Google style guide for C++ recommends that if a function is longer than 40 lines, you need to think about breaking it down.

So if you're using it twice, make it a method.

Optimise only when you are sure it is necessary.

Deployment:

****

****

Even more important than deployment, you have to make sure that you plan for recovery. One of the issues that we had in the business that I used to work in was that we were not allowed to even begin a production deployment unless our recovery plans had been approved by change management.

So there's an entire team of people that look over our plans for how we're going to install the software, but not only that, the plan set up in advance of how we're going to roll that back,

how we're going to get out of it if we somehow screw it up when we do

try to install it because if you're installing something on production, a production system that's supposed to be up 24/7, servicing millions of customers every hour,

you can't have the system entirely go down.

You need to have a plan to roll those changes

back to make sure that the system can get back into

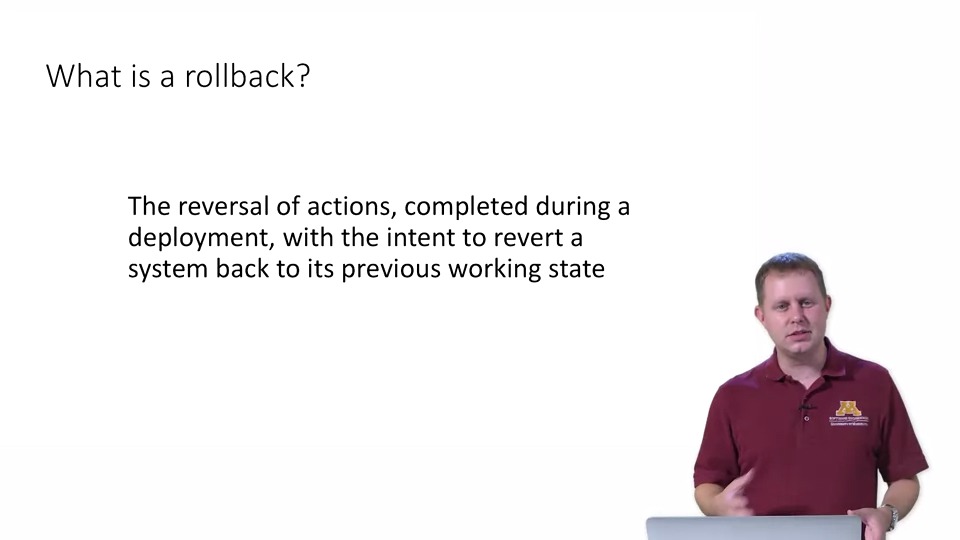
a working state if whatever you were planning to do doesn't end up working.

Deployment:

Rollback is what happens when that deployment doesn't go as you intended.

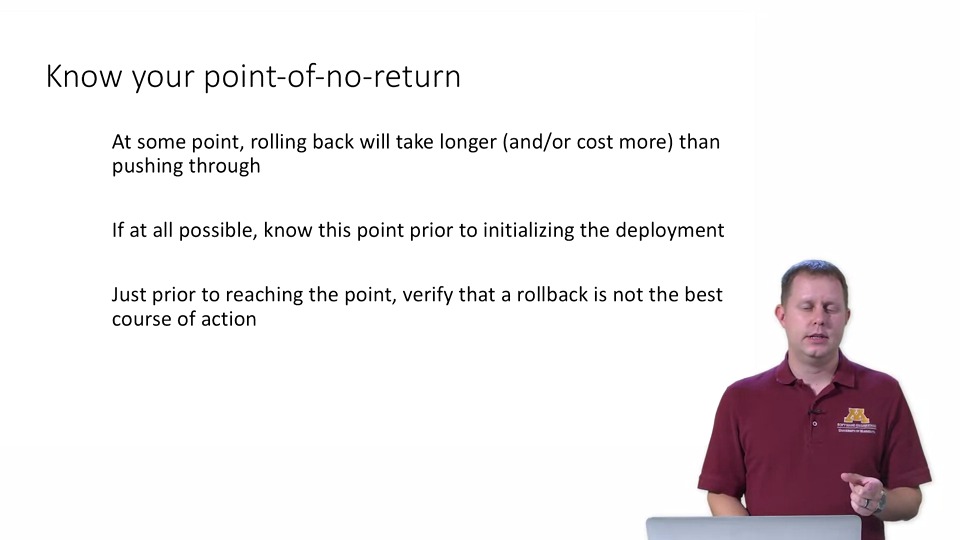
Why Roll-Back?

There's lots of reasons why you rollback, but most of them have something to do with things not going quite the way you expected them to.



Point of no-return:

There's at some point rolling back is going to take longer than you have.



Summary of Roll-Back:

