Week1:

Estimated time to complete: 3 hours

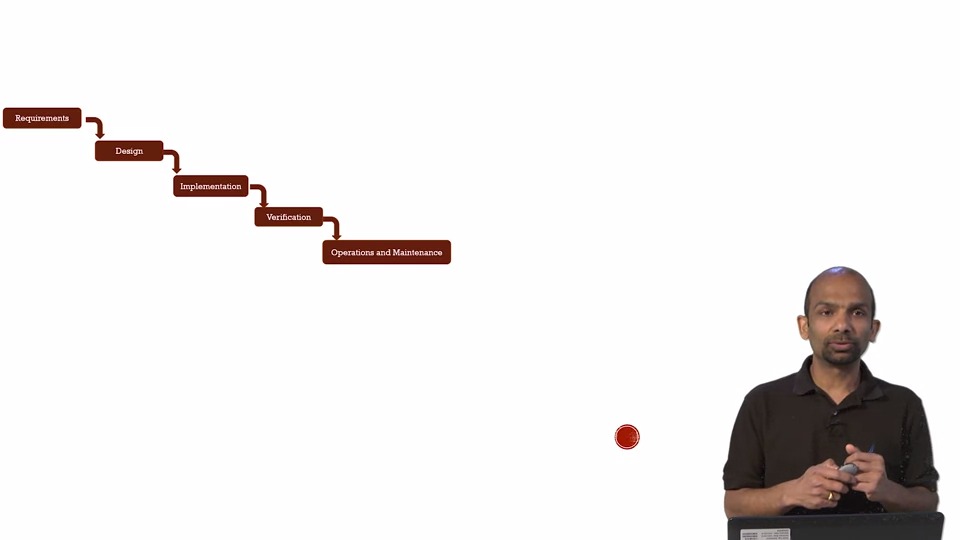
Actual time:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Start Time | End Time | Accomplishment | Time(min) |
| 7/01/2020 | 12:06PM | 12:20PM | 2 video completed. 3rd ongoing. | 14 |
| 7/01/2020 | 1:15PM | 2:20PM | Week1-Requirement-3 video completed | 65 |
| 7/01/2020 | 4:50PM | 6:00PM | Week 1- Requirements topic closed | 70 |
| 7/01/2020 | 6:45PM | 7:00PM | Week1- Software architecture started | 15 |
|  |  |  |  | 164 |

**Introduction to course:**

**What software development looks like?**

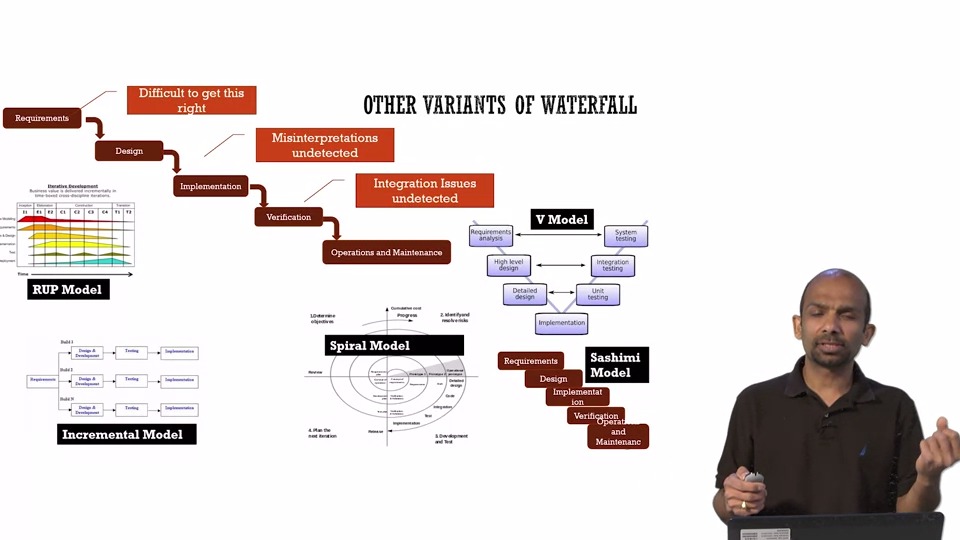
Waterfall:



The model that we just reviewed is called, a waterfall method, where you go from phase to phase to phase. Requirements, design, implementation.

Other Variants of waterfall:

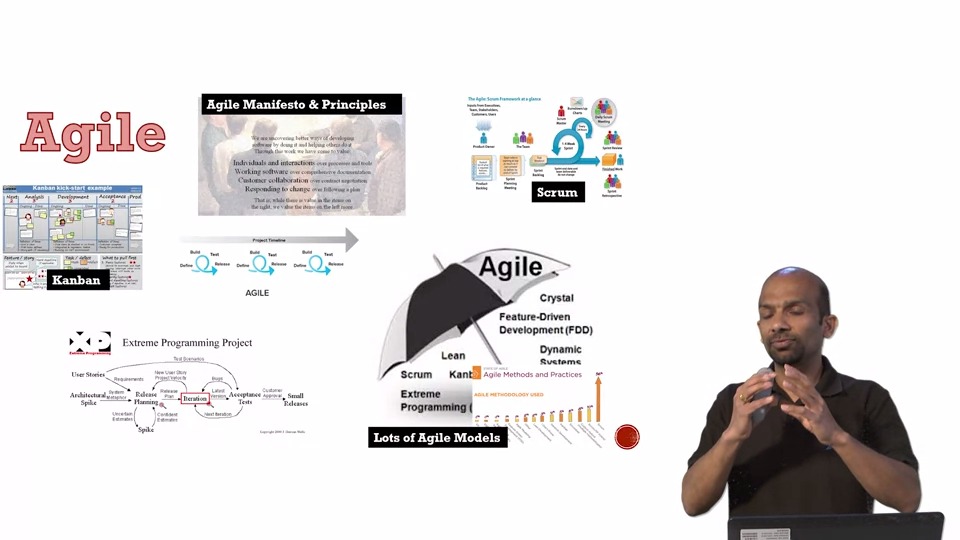
[V-Model, RUP Model, Incremental Model, Spiral Model, Sashimi Model]



As all of these variants were coming along, there was another thought process that were

evolving in the software industry, called Agile.

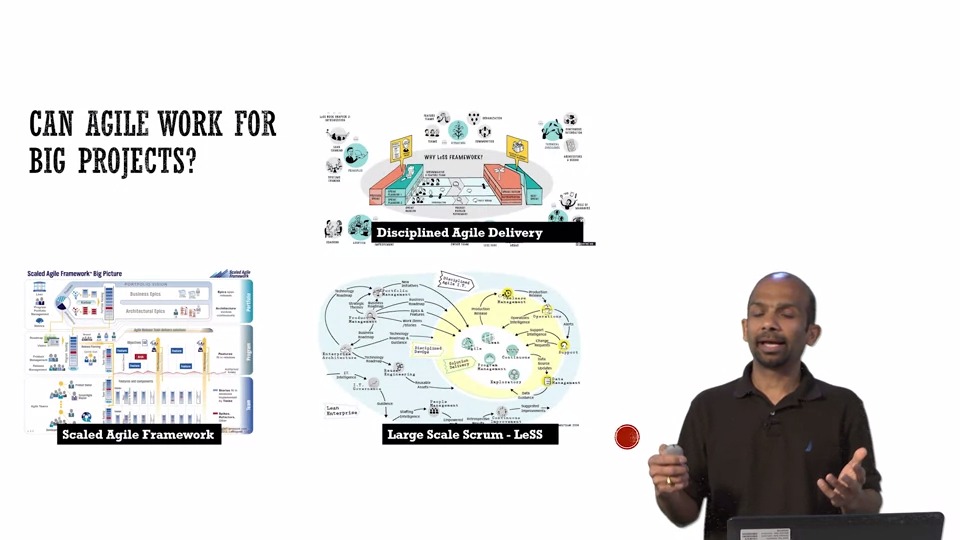
So, Agile is not a model. It's a mind-set.

Agile Model:

The basic idea behind all of this model, was that instead of building this whole one year cycle,

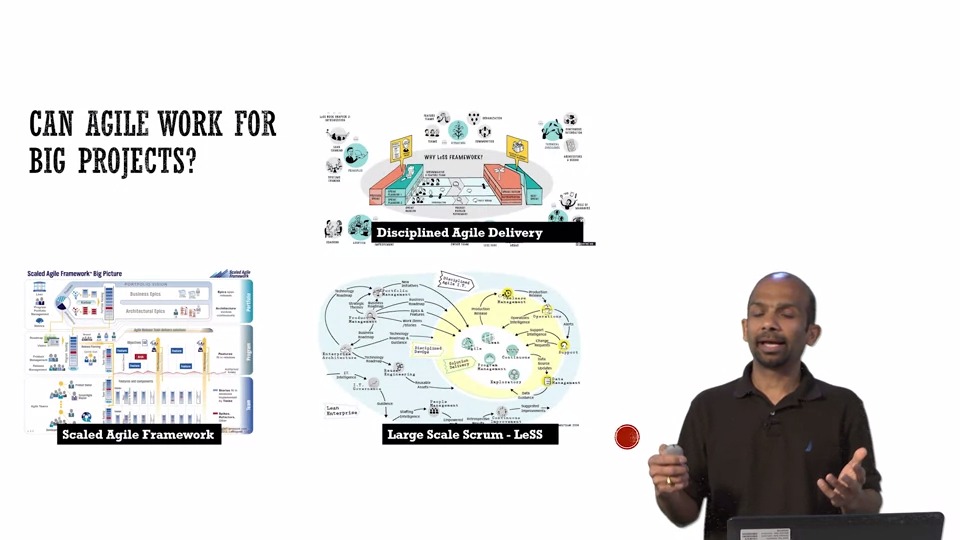
you build in short cycles.

So, you define little, you build, you test, and you learn. And then again, you just keep doing this. That way you can adjust to the market quickly.

DevOps:

They march towards a common goal.

And, that led to a culture of DevOps.

Models for bigger projects:

Now, as this model of Agile was getting popular in industry in a smaller scale,

people started to wonder  "Can we apply this on a bigger projects?"

And hence, many new models evolve, like Scale Agile Framework, Large Scale Scrum, and Disciplined Agile Framework.

Cheaper Models:



They wanted to learn if there is a cheaper and a faster way to learn.

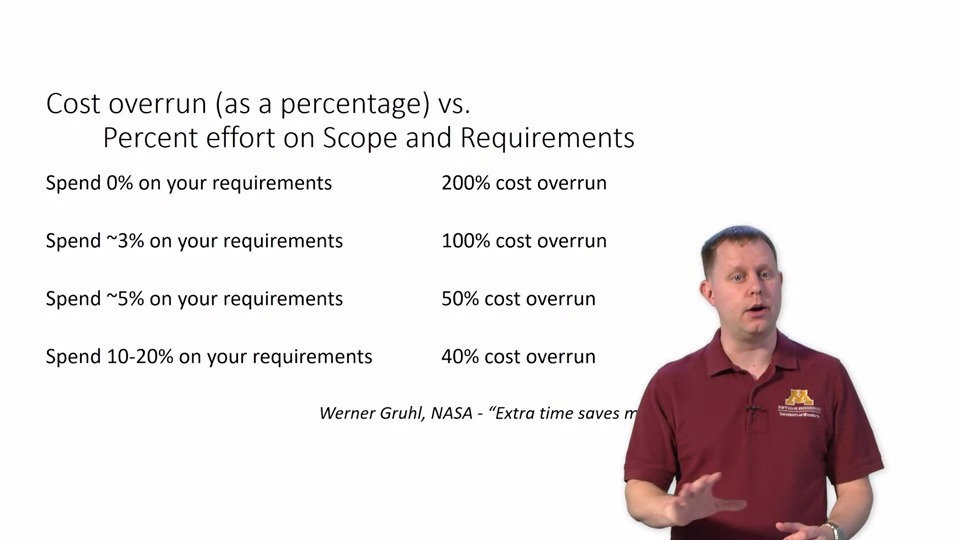
And, the concepts like Lean Startup and Design Thinking came into being, where, you find the cheapest way to learn the one cycle.

**What is Requirement Specification?**

1. Process
2. Product of process

Importance of Requirement Specification:

1. Engineering Argument-



1. Economic Argument: Problem found after deployment cost 1000 times more to solve than the problem found during staging area.

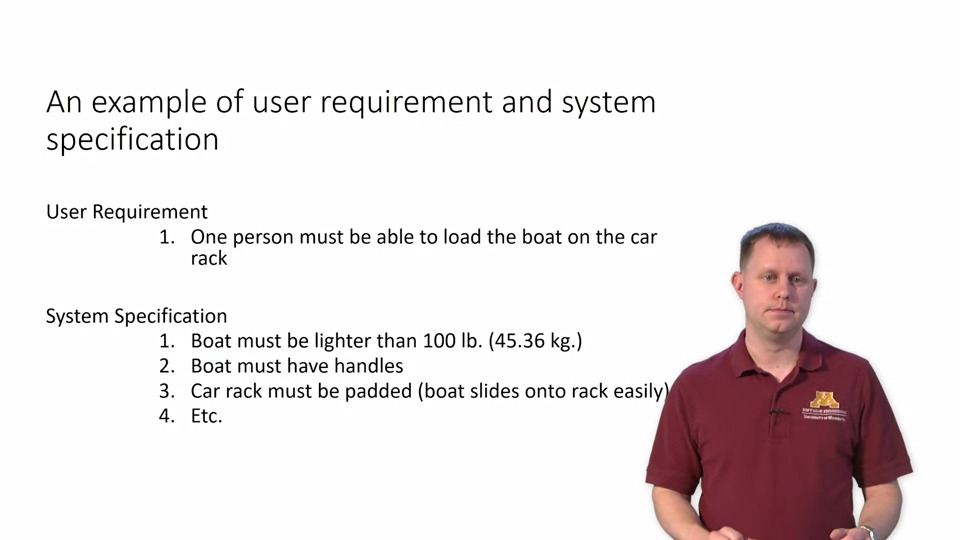
Requirement vs Specification:

1. User Requirement:
2. Requirement is for user or client.

2. Write in user language

1. System Specification:
2. System spec is the usually more precise or constraining statement of how the system will meet the user requirements.
3. Write in system language.
4. Specification is for Developing team.

Example:



**Non-functional Requirement:**

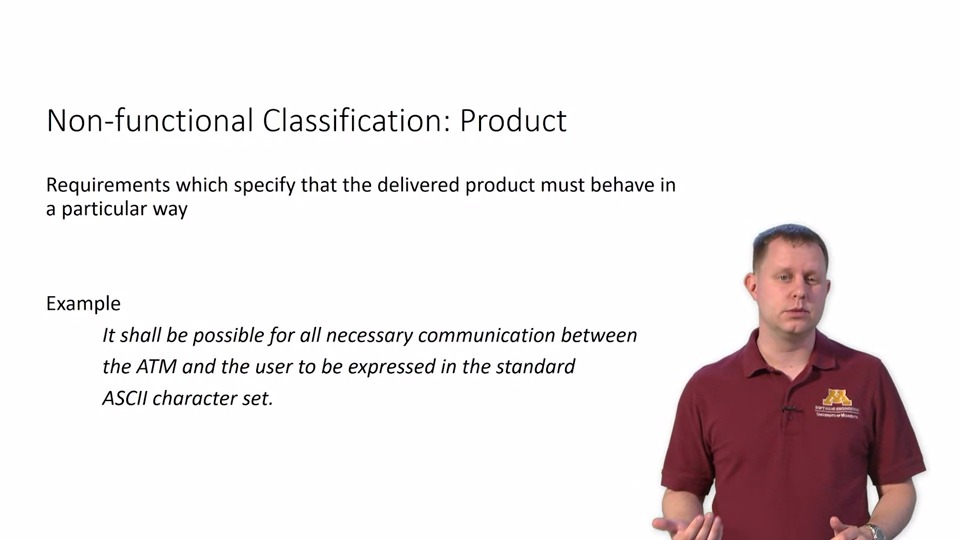
Requirements which don't specify what the system will do, but rather how the system will perform the behaviours.

Properties:

1. Define system properties and constraints.
2. Process requirement: Process requirements may be things like mandating a particular case system, that is, a computer-assisted software engineering tool like Microsoft Project or Ajira, a bug-tracking software.
3. More critical than functional requirements

Classification:

1. Product Requirement
2. Organization Requirement
3. External Requirement
4. Product Requirement:

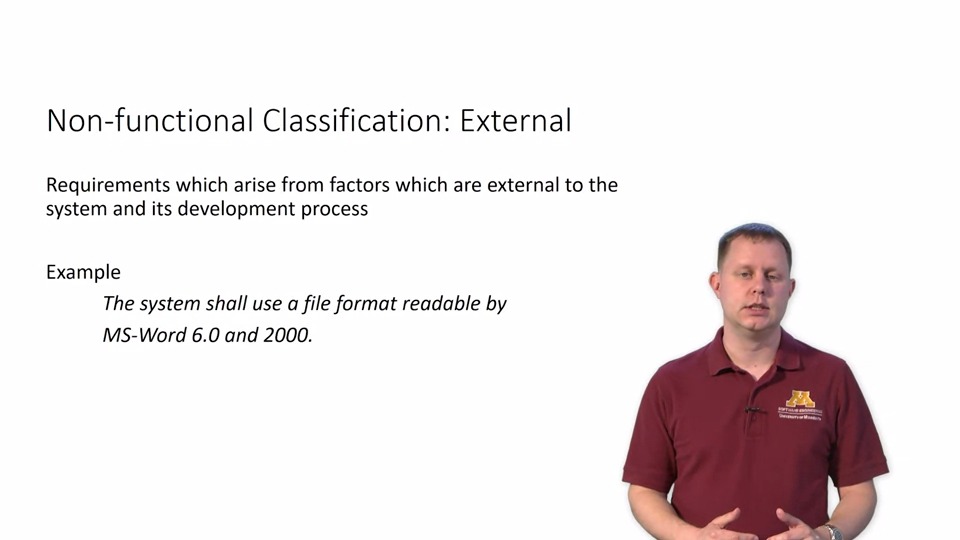


1. Product requirements which are non-functional talk about specific behavior.
2. This is often in the form of protocol requirements, encodings, or encryption requirements, that sort of thing.
3. They are requirements on the product itself.
4. Organizational requirements:



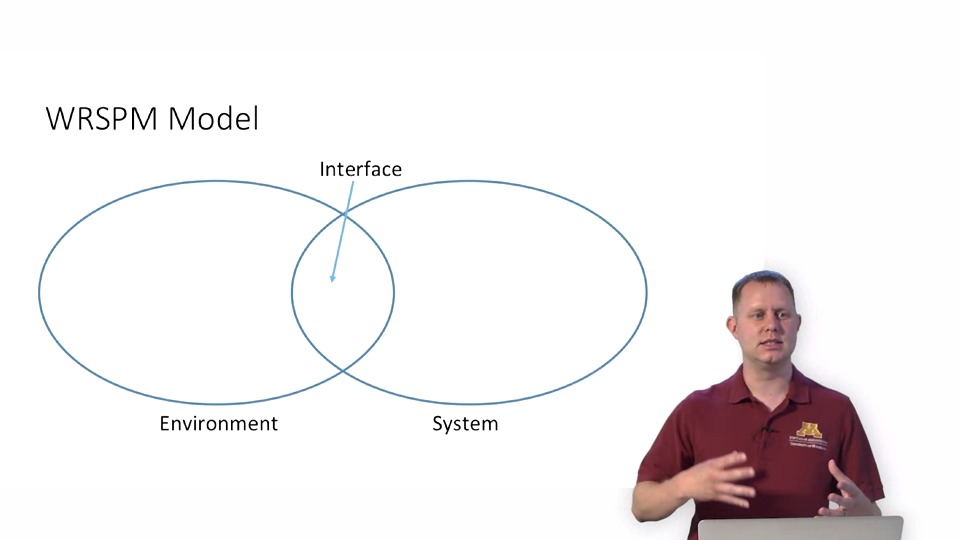
1. Organizational requirements are those that are defined by the company.
2. Company standards, your development team's code style requirements, even the development process itself like using SCRAM could be defined as something like this.

3. External Requirement:



1. And then external constraints are a big factor, especially in regulated industries.
2. When the FAA says you have to use this development process or meet these code coverage testing metrics, that's all there is to it.
3. You have to.
4. And if they change, you have to change.
5. This is the kind of control we're talking about, being impacted by external entities.
6. That's something you want to document early on in the process.

**WRSPM Refrence Model (world machine model) :**



The WRSPM model is one way of looking at this system in order to determine what the requirements specifications might be.

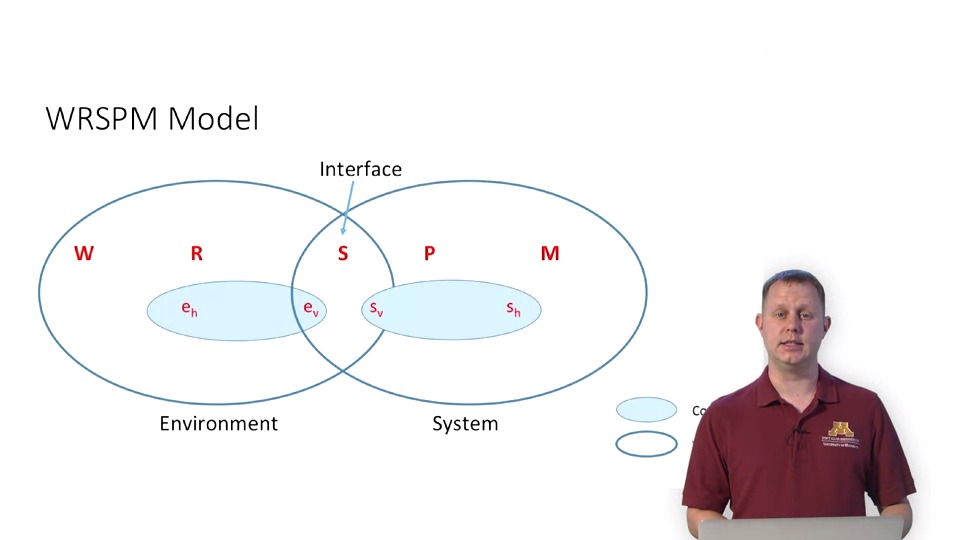
1. Environment:

So the environment is all the user visible elements of everything in the world, which includes some part of the system where that overlaps and that overlap is called the interface.

1. Interface:

It's the location of meeting between what the user can do and what the system can do or display to the user in order to capture input and provide output.

1. System:



1. World:

world understanding

1. Requirement:

the requirements which sit within the environment that deal with the user requirements

1. Specification:

the specification that defines how the two will meet

1. Program:

, the program that meets that specification in order to meet the requirements

1. Machine:
2. the machine that it all runs on.

Variables in WRSPM:

1. EH:

EH are the elements of the environment that are hidden from the system. So there very well may be things that we care about outside of the system that we still have to care about. It's the parts of the environment that the user wants.

1. EV:

The EV, the parts that are visible to the system in the environment. The data that the user enters then is the visible part of the environment in that case.

1. SV:

The SV is the system elements that are visible in the environment.

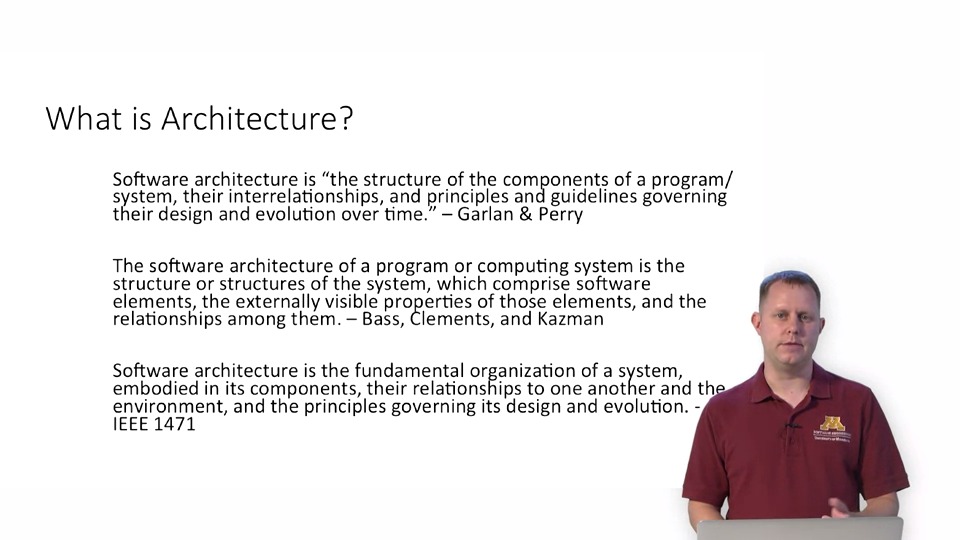
1. SH:

The SH is the system element that is hidden to the environment.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Start Time | End Time | Total Time | Particulars |
| 8/01/2020 | 11:10 AM | 11:25PM | 15 | Half a video |
| 8/01/2020 | 1:30PM | 2:45PM | 75 | Completed Week 1 |
|  |  |  | 90 |  |
|  |  |  |  |  |

**Software Architecture Development:**

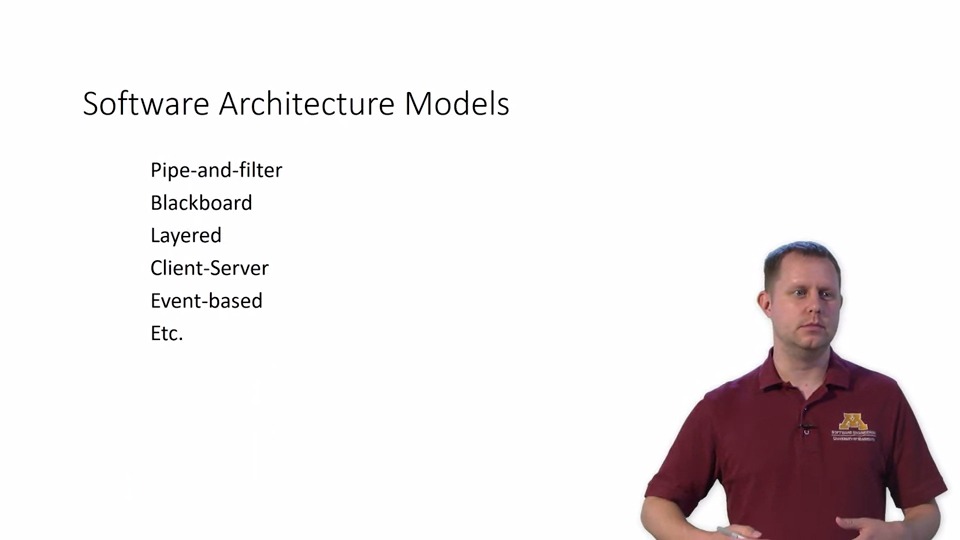
**Definition:**

****



**Software Architecture Models:**

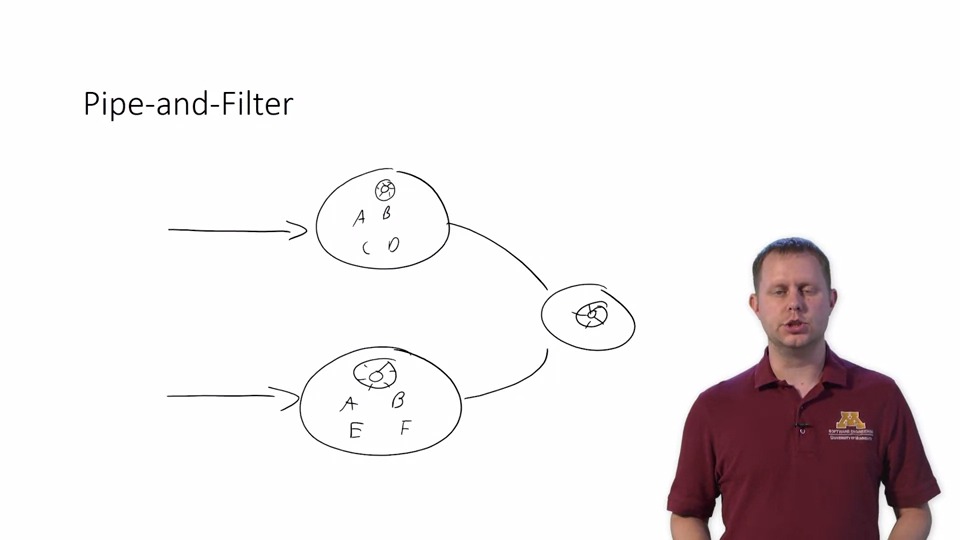
So there's a variety of models that have become essentially go-to best practice models for a number of different common problems.

****

Examples of Software Architecture:

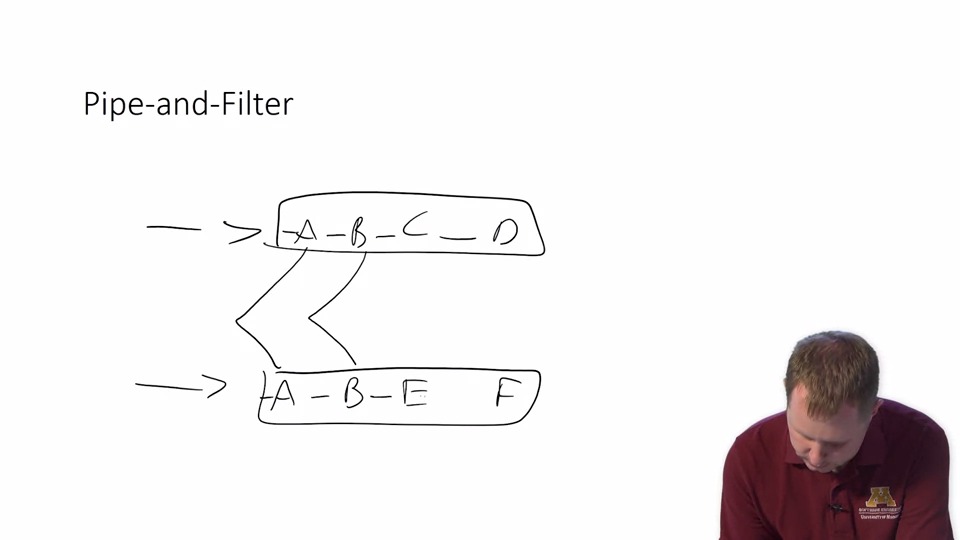
1. Pipe Filter Mode:

There are two stream of data and two sub systems and they connect to final business logic.



Our initial case.

Above case in Pipe-Filter Model

****

A and B are shared process, and we want to build this shared subsystems in such a way that A and B are shared. This can happen with Pipe-Filter.

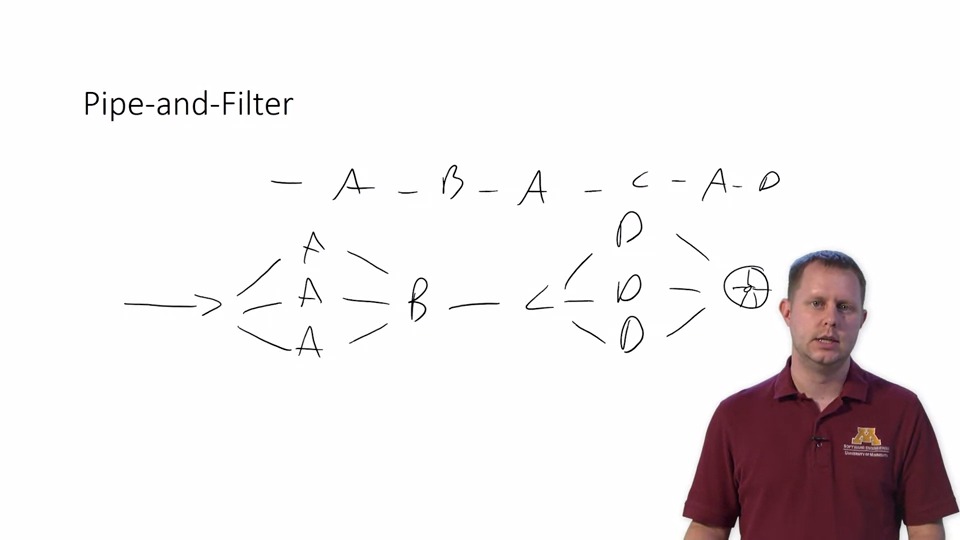
“So, Pipe-and-Filter tends to mean that we eventually start to build and look at these interfaces between the individual elements.”- Instructor.

Let's say that later we realize, we want to apply Element D of first subsystem, but in between parts E and F in the second subsystem. If and only if we are using a Pipe-and-Filter system, we can nearly insert D directly into the process flow without having to change anything.

**And that only works if you are particularly focused on making sure that the input and output formats are exactly the same.**

Another Example:

Pipe-filter helps in very parallelization.



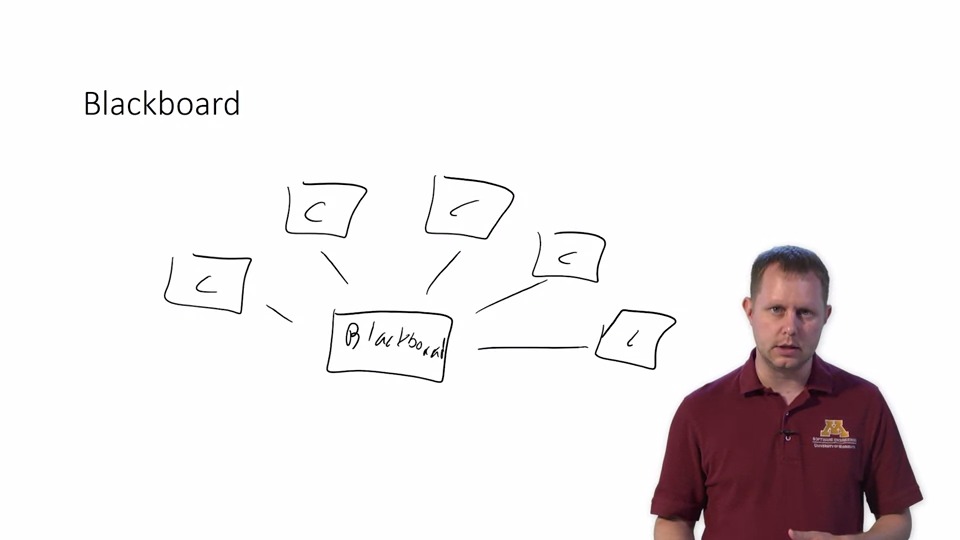
The idea that the same input and output formats are being used across all processes or all modules within the components.

This Pipe-and-Filter architecture is particularly useful in those cases where we may want to expand, parallelize, or reuse components across large systems like this.

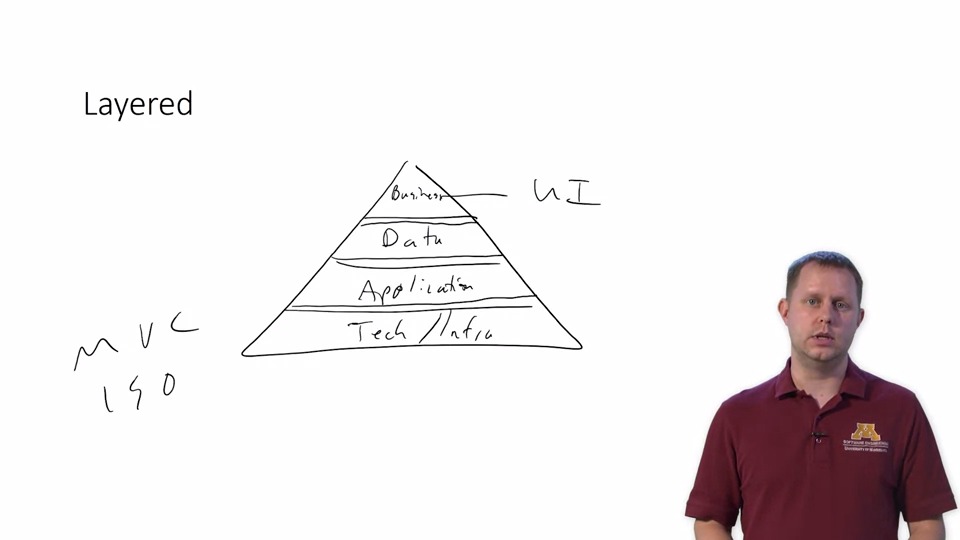
Examples where Pipe-Filter is used:

1. Compilers: ( they have )
   1. Logical Analysis
   2. Pair parsing
   3. Semantic Analysis
   4. Code Generation
2. Blackboard Model

It has a central global variable (a blackboard) and many components and all of them work on some shared data and communicate with each other using blackboard. The blackboard has provided interface which helps all components to communicate with the blackboard.

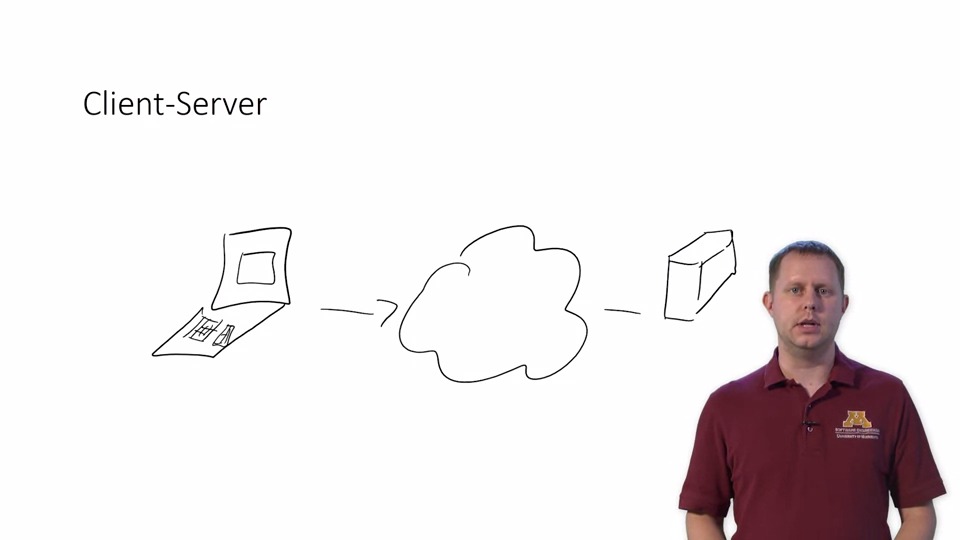


1. Layered



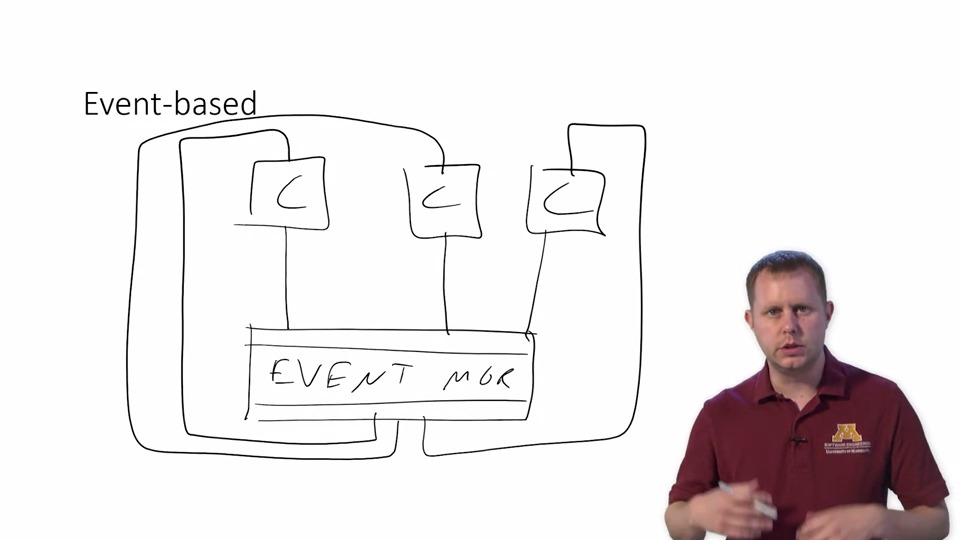
It has different layers and each layers are separated by UI. These UI make sure that the different layers do not mix with each other. If we modify any one layer the other layer doesn’t have to be changed.

1. Client Server



Eg. Internet

1. Event – Manager:



It is very similar to Blackboard architecture.

Event manager is connected to each component. When any component triggers an event, the manager decide which component should handle this event. It is useful When you're looking at waiting for some event to occur, and then having some action in response to that.

**Software Architecture Process:**

Three concerns:

1. System Structuring

Systems structuring refers to how the system is decomposed into these several principal subsystems and communications between those subsystems are then identified.

1. Control Modeling

Control modelling then, is how architectures create a model of the control relationships between the different parts of the system that's established.

1. Modular Decomposition

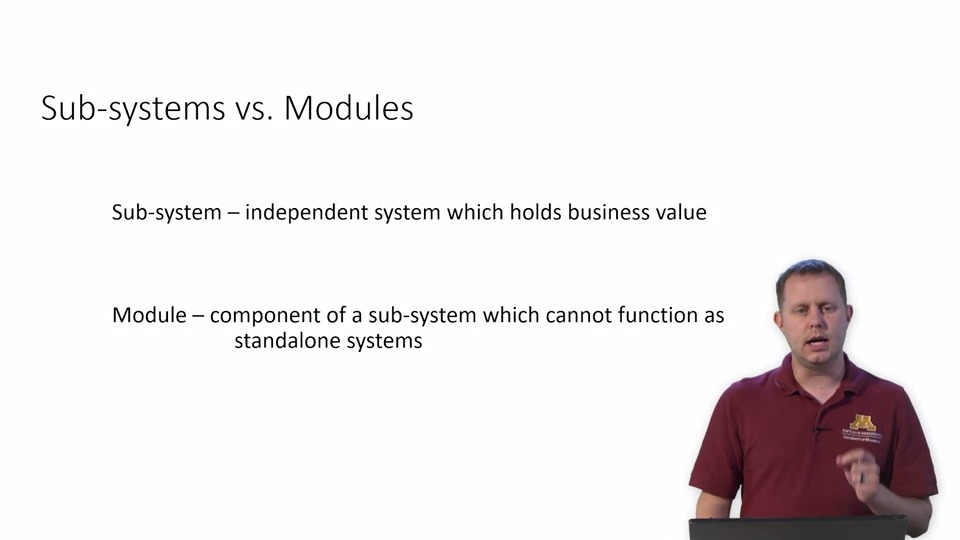
And then, modular decomposition is how we identify those subsystem partitions.

Thing like maintainability, security, reliability and so on

Software Quality Attributes: (ensured at architectural stage)

1. Performance
2. Reliability
3. Testability
4. Security
5. Usability

Subsystem vs Module:



**Estimated time to finish Week1: 3 hour**

**Actual Time Taken: 4 hour 20 minutes.**