Software Development Processes and Methodologies

Week 1

What software development looks like

Waterfall model

Go from phase to phase: requirements, design, implementation,...

Issues:
+ Very difficult to predict the requirements 1 year or 2 years ahead, as the market changes, or sometimes it's just really difficult to predict what a user will like or not like.
+ Misinterpretations undetected.
+ Integration issues undeteced

Other variants of waterfall

V model:
+ focus on testing
Sashimi model

RUP model -

Incremental model:
+ do the requirements in 1 shot
+ do design, implementation in increment

Spiral model: risk-driven approach

Agile

is not a model, it’s a mindset

They articulated what we call now the **Agile Manifesto and Principles**.

Models like Scrum, and Kanban, and XP - Extreme Programming Project, that helped implement this mindset

Why do we need requirements?

What is a Requirements Specification?

It’s actually 2 things:
1) Process
Requirements specification (the process):
+ Create high-level descriptions
+ Distinguish between ‘right’ and ‘wrong’ system
+ Capture WHAT not HOW of solution

2) Product (of that process)

Why are RS important?

Engineering argument

Economic argument

Requirements vs Specification

Writing the Software Requirements Specifications (SRS)

2 audiences:
+ Users
+ Developers

User requirements

Are exactly what the user wants the solution to do in user’s language

System specifications

The required action of the system solution we’re developing.
Usually more precise or constraining statement of **how the system will meet the user requirements**.
(still WHAT the solution will do, not HOW)

→ Which model will be constructed? Will it be OO dessign and dev?

Requirements and Specifications are closely related concepts in defining your solution

Requirements for user, specification for developers

Write your requirements in the user language

Write your specification in the system language

Be sure that your specification meet the requirements

Non-functional requirements

Define system properties and constraints (security, performance, usability,...)
Process requirements

Often more critical than function requirements

Three classifications of non-functional requirements

`Product requirements`: specify that the delivered product must behave in a particular way.

- Specific behavior
- In the form of protocol requiremetns, encoding, encryption,...

`Organizational requirements`: which are a consequence of organizational policies and procedures

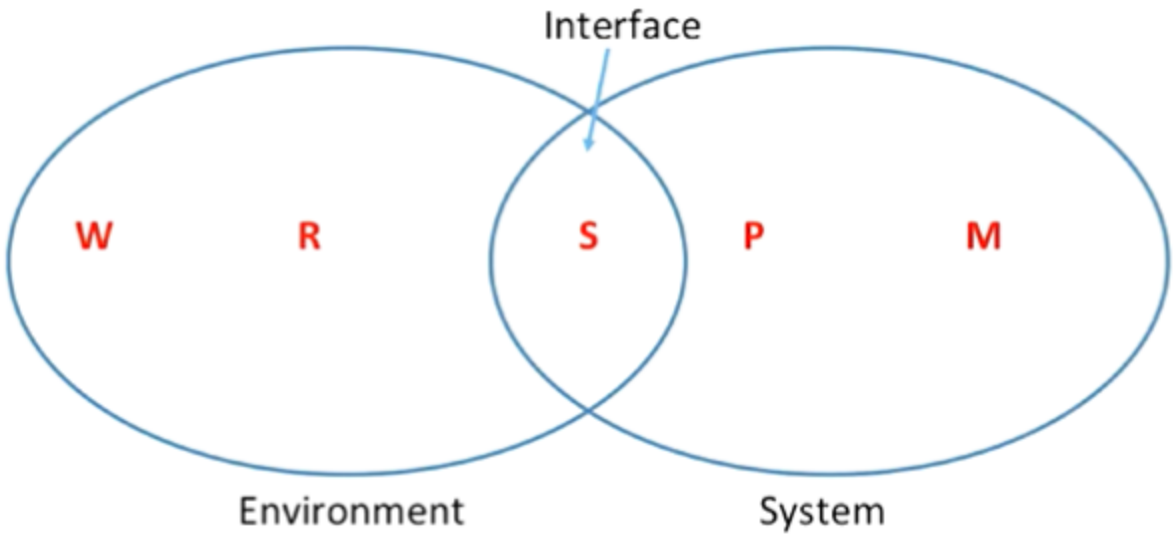
`External requiremens`: which arise from factors which are external to the system and its developments process.

> 💡 Never to be overlooked!
> Very important on nearly every project
> Be sure to consider them separatel from the functionality
> Consider each classification separately

WRSPM

WRSPM Reference Model: The World Machine Model

Capture the ‘Right’ thing:
+ Requirements are always in the problem domain
+ Software specification is in the solution (computer) domain
+ Serveral layers of abstraction can exist in between



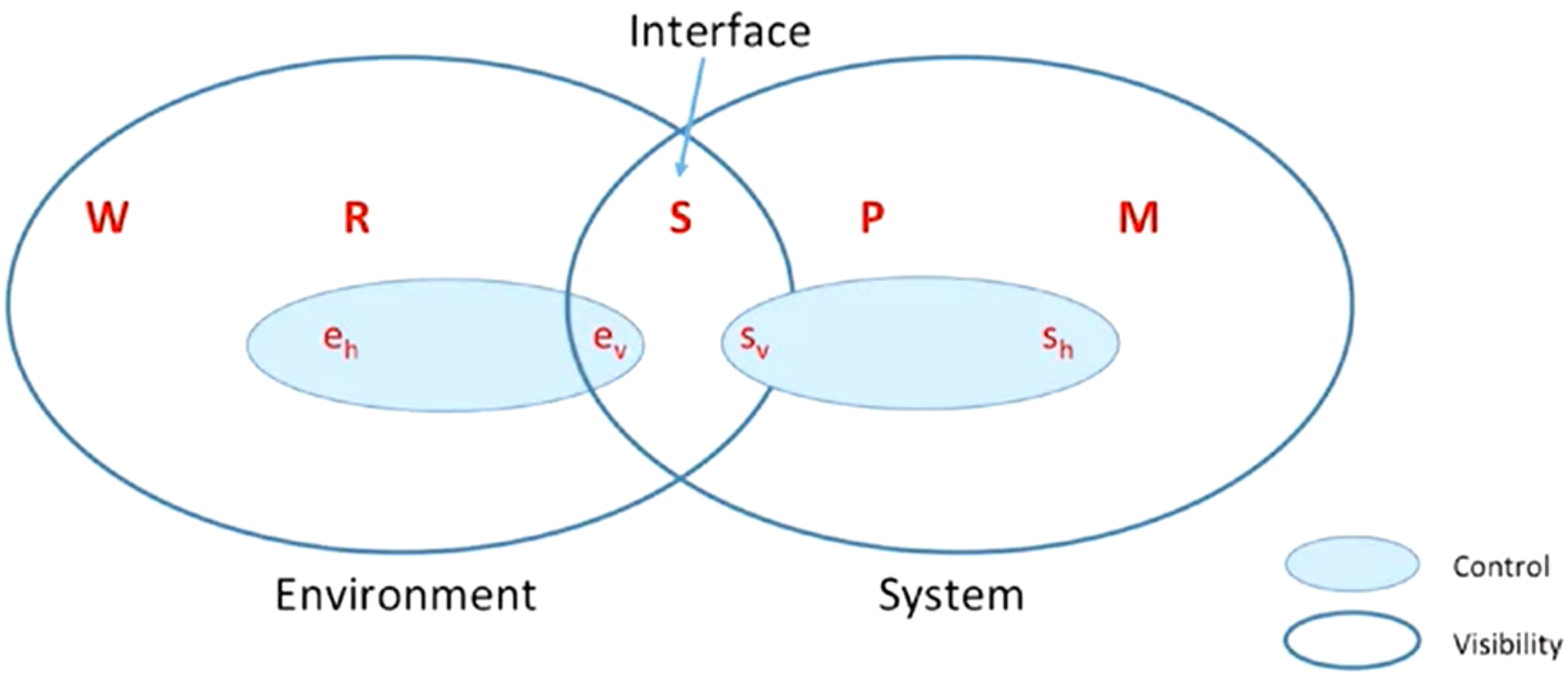
W - World: these are things that we know are true, have an impact on our system, our problem domain

R - Requirement: the user’s langague understanding of what the user wants from the solutions

S - Specification: lies in the inteferace area, what do system will do

P - Program: what the software developer will write, is all the code,

M - Machine: the hardware behind the system



e_h - elements of the environment that are hidden from the system

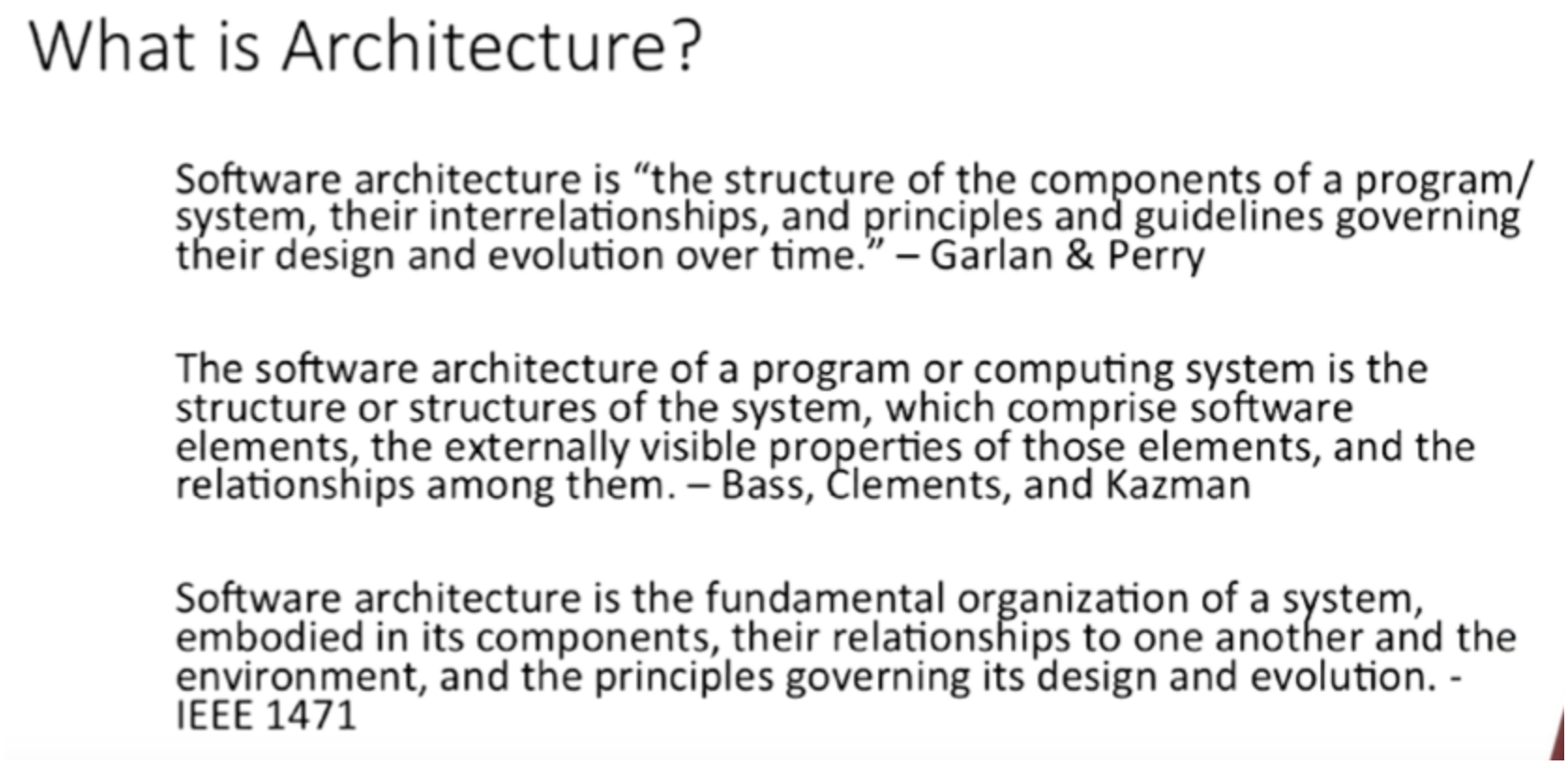
e_v - parts that are visible to the system in the environment
the data generated when you read (PIN, mag card info)

s_v - the system elements that are visible in the environment
(button, information on the screen, the prompts,...)

s_h - all the system data that is hidden from the environment
(approval number from the actual bank before distributing money)

> 💡 WRSPM Model is a reference model for how we understand problems in real world:
> + Helps identify the difference between requirement and specification
> + Requirements are in the user (problem) domain
> + Specifications are in the system (solution) domain
> + One must be careful to ensure that the specification meets the requirements

Software Architecture: Definition



Why should we care?

Good things are well-architectured

Good architecture is hard

Why else should we care?
MISTAKES in architecture

Conclusion

Software Architecture mainly about decomposing the system int components

- Each components must have individual business value
- Help organize workforce and resources
- Allows for parallelization
- Helps define the build vs. buy question and getting funding

Software Architecture: Models

Pipe-and-Filter

Blackboard

Layered

Client-Server

Event-based

Software Architecture Models are best practice solutions to commonly encountered problems
+ Start from the pure model, and adjust
+ If ur goals align with the goals of the model, use it
+ Customize the ‘pure’ form to ur needs
+ Each models solves one particular large-scale problem

Software Architecture: Process

3 major concerns talking about design process

System structuring

Refers to how the system is decomposed into these several principal subsystems, and communications between those subsystems are then identified

Control modeling

Is how architectures create a model of the control relationship between the different parts of the system that’s established

Modular decomposition

is how we identify those subsystem partitions (simplicity, maintainability, reliability, security, all those kinds of quality attributes, resource management)

Sub-system vs. Modules

Sus-system - Independent system which holds business value

Module - component of a sub-system which cannot function as standalone systems

Software quality attributes

Performance

Reliability

Testability

Security

Usability

> 💡 Sofware Architecture Process concerns itself with estimation, quality, partitioning
> + Partitioning the responsibilities of the software is only the beginning
> + Determine the costs and secure funding
> + Divide work and system with quality in mind
> + Apply software architectural models to common problems

Week 2

Software Design: Introduction

What is “software design”?

A process: the creative process of transforming the problem into a solution

A product of the process

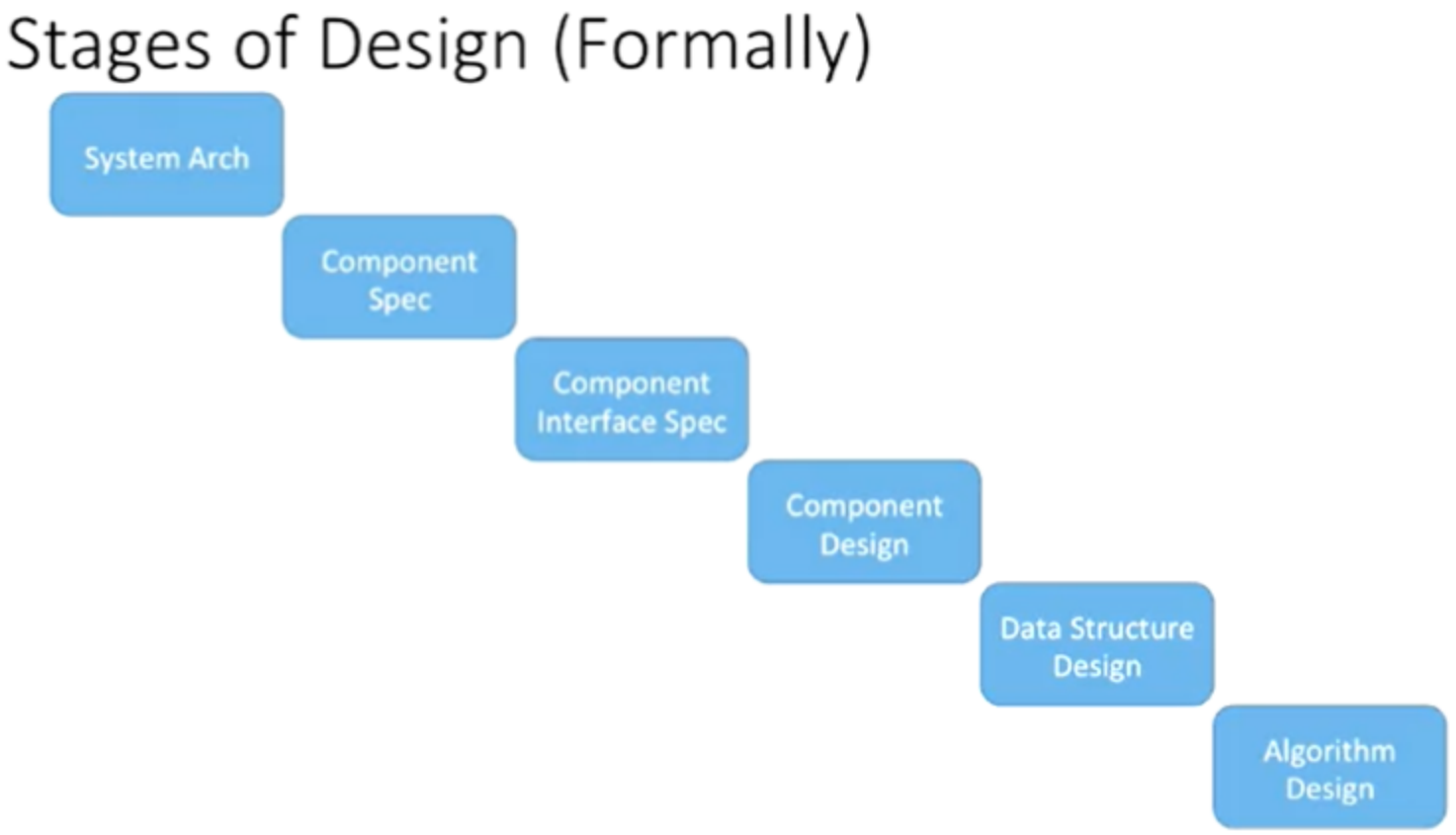
States of Design

Problem understanding

Identify one or more solutions
(TMTOWTDI - There’re more than one way to do it!)

Describe solution abstractions

Repeat process for each identified abstraction until the design is expressed in primitive terms



Software Design takes abstract requirements and creates detail ready to be developed

- Decide classes, methods, data types to be used in the solution
- Separate Architecture from Design (or don’t)
- Provde detail which is implementation-ready
- ... but doesn’t include implementation detail/constrictions

Software Design: Modularity

What are the aspects of modularity?

Coupling

Cohesion

Information Hiding

Data Encapsulation

Definitions and Principles of Modularity

Complex systems must be broken down into smaller parts

Three primary goals

1. Decomposability
2. “Composability”
3. Ease of understanding

Information hiding

Hide complexity in a “black box”

Examples: Functions, Macros, Classes, Libraries,...

Data encapsulation

Encapsulate the data: protecting the data from unauthorized access and maintaining integrity is a key point

Helps find where problems are

Makes designs more robust

Conclusion

Modularity is about the breakdown and reassembly of components

- Many aspects at play
- Coupling and cohesion
- Information hiding
- Data encapsulation

Software Design: Coupling

Coupling is how tightly coupled one module is to another.

Measuring the strength of connections between (sub-)system components.

Loose coupling allow for changes to be unlikely to propagate across components.

Shared variables and control information lead to tight coupling.

Loose coupling achieved by state decentralization and message passing.

Tight coupling

Content coupling: happens when module A directly relies on local data members of module B (rather than relying on some access or a method)

Common coupling: happens when module A and module B both rely on some global data or global variable.

External coupling: is a reliance on an externally imposed format protcol, interface

Medium coupling

Control coupling: happens when a module can control the logical flow of another by passing in information.

Data structure coupling: occurs when 2 modules rely on the same composite data structure, especially if the parts the modules rely on are distinct

Loose coupling

Data coupling: is when parameters are shared

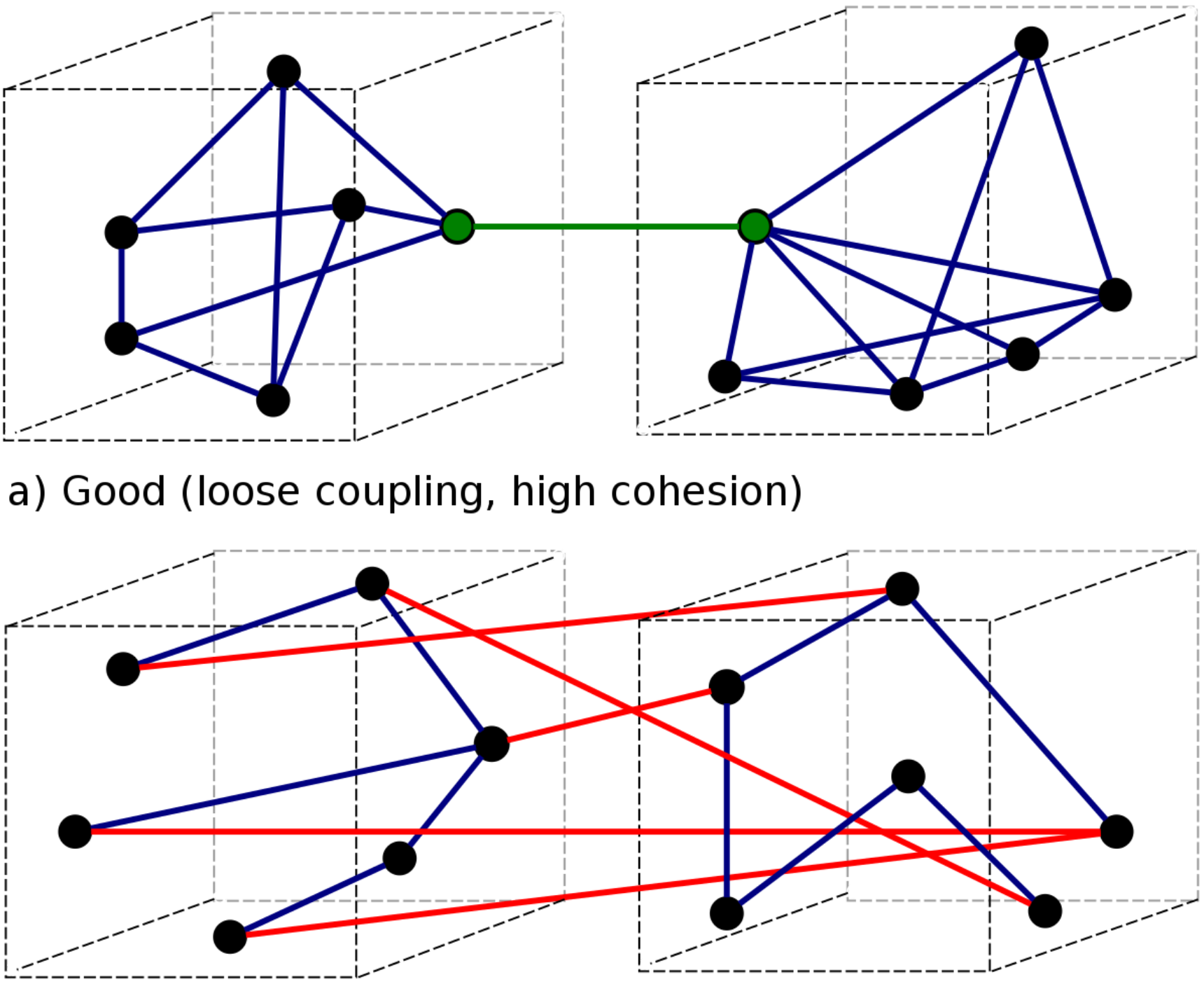
Message coupling: it’s primarily achieved through state decentralization and component communication is only accomplished.

No coupling:

Conclusion

Coupling measures communication:

- How well are we protected against requirements change propagation
- Measuring the various types of coupling is important
- Aim for data and message coupling
- Justify anything tighter



Software Design: Cohesion

Measures how well a module's components 'fit together'

Implements a single logical entity or function

Represents a desirable design attribute

Divides into various levels of strength

Weak cohesion

Coincidental cohesion: is effectively the idea that parts of the module are together just because they are.

Temporal cohesion: means that the code is activated at the same time, but that's it

Procedural cohesion: is similarly time-based and not very strong cohesion.

Logical association: components which perform similar functions are grouped.

Medium cohesion

Communicational cohesion: all elements of the component operate on the same input or produce the same output

Sequential cohesion: the stronger form of procedural cohesion; achieved when one part of the components is the input to another part of the component

Strong cohesion

Object cohesion: each part is specifically designed for purpose within the object itself

Functional cohesion: go above and beyond sequential cohesion to assure that every part of the component is necessary for the execution of a single well-defined function or behavior.

Cohesion and Inheritance

Technically speaking, inheriting attributes from a super class weakens cohesion

To understand the component, one must examine the super class as well.

Conclusion

Cohesion is concerned with how well the components of a module serve a single goal

- Just having high cohesion is not sufficient
- Measuring the various types of cohesion is important
- Aim for object and functial cohesion
- Justify anything less

Implementation

Deployment

Deployment planning

- Largely determined by project scope
- Includes planned steps, problem areas, plans to recover

Deployment plant concerns (sections)

- Physical environment
- Hardware
- Documentation
- Training
- Database-related activities
- 3rd party software
- Software being deployed

Software deployment is the event of launching your product to users

- Primarily focused on how to deliver and how to revert on failure
- Deployment without rollback plans shouldn’t happen
- Document your rollback plans for error-reduced action under fire
- Consider all aspects of the environment when writing the plan

Deployment: Rollback

What is a rollback?

The reversal of actions, completed during a deployment, with the intent to revert a system back to its previous working state.

Why rollback?

Installation did not go as expected.

Problems would take longer to debug/fix than installation window

Keep production system alive

Know your point-of-no-return

At some point, rolling back will take longer (and/or cost more) than pushing through

If at all possible, know this point prior to initializing the deployment

Just prior to reaching the point, verify that a rollback is not the best course of action.

Rollback is the primary concern of deployment

A matter of “when” not “if”

Deployment without rollback plans shouldn’t happen

Deployment without the possibility of rollback need special attention

Be sure to determine your point of no return before you begin

Deployment: Cutover Strategies

Hot failover (<30 minutes)

Warm standby (2-4 hours)

Cold back-up (storage) (~24 hours)

Conclusion

Software cutover is a trade-off between cost and preparedness

- Speed of recovery is directly proportional to cost recovery
- Hot failover allows you to redirect live data with minimal
- Warm standby has services ready for initialization
- Cold back-up is a replacement machine needing full setup

Software Testing: Introduction

Testing: Definitions

The process of executing a program (or part of a program) with the intention of finding errors.

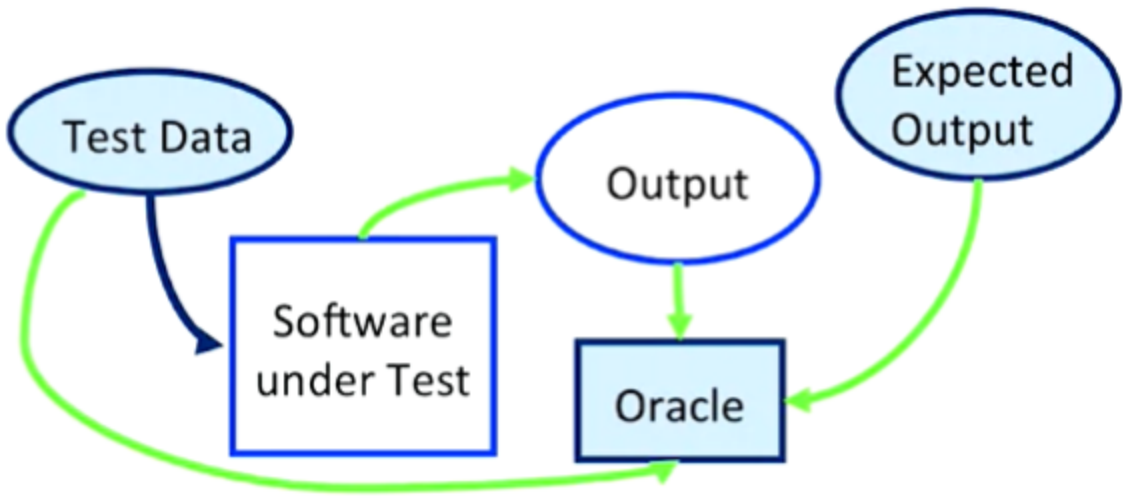
The purpose of testing is to find errors

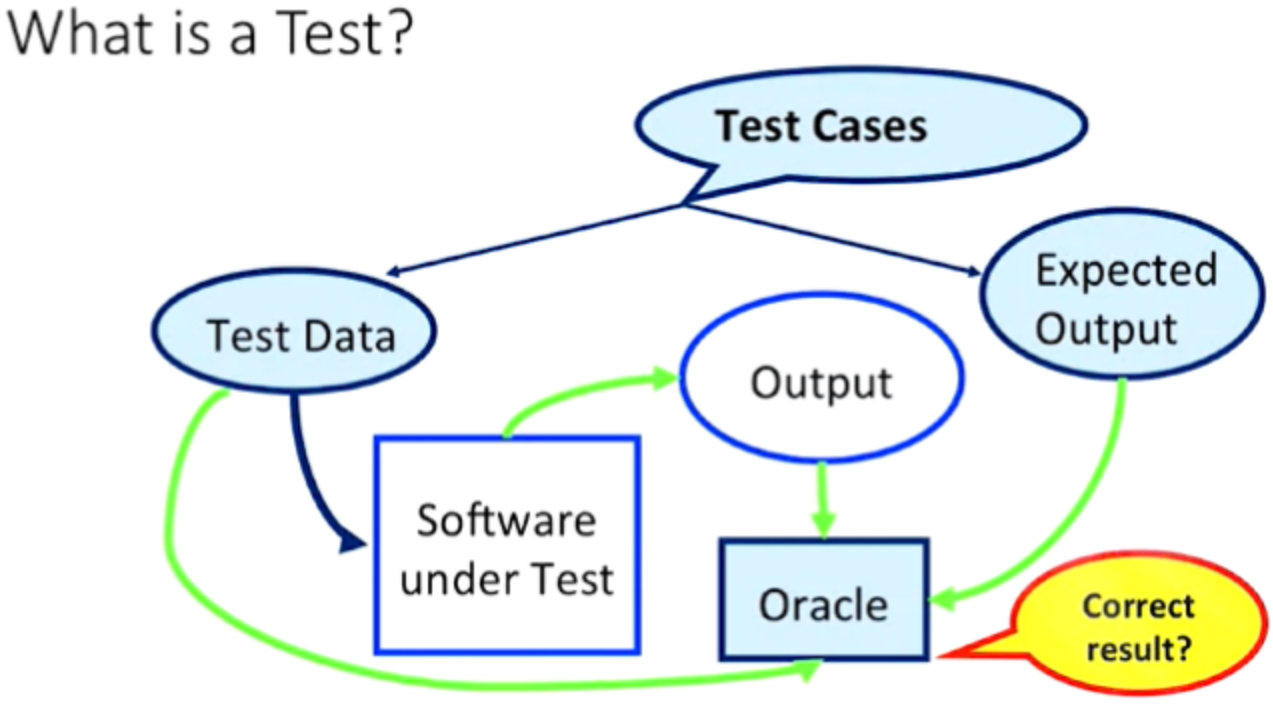
Testing is the process of trying to discover every conceivable fault or weakness in a work product

The process of searching for error

What’s a TEST?

Software under Test: whatever part or subset of the program we have completed, where we can exercise something, some behavior (unit test)





Test Data: Inputs which have been devised to test a system

Test Cases: Inputs to the system and the predicted outputs from operating the system on these inputs, if the system performs to its specification.

What’s a “Bug”?

Failure:

a system failure occurs when the delivered service deviates from the specified service. That means that something didn't happen the way it was supposed to. The failure occurred because the system was **erroneous**.

When the error then affects the delivered service, that is when the error actually causes a change in behavior that the user did not expect to see, that is a failure.

Error: Latent, Effective

An error, is that part of the system state which is liable to lead to a failure.

Upon occurrence, a fault, a mistake, creates a latent error. That latent error sits in the code and becomes effective when it's activated

Fault

A fault is what actually happened.

So, the error is a manifestation of a fault.
And a failure is the manifestation of an error on the servers.

Axiom of Testing

Program testing can be used to show the presence of bugs, but never their absence.

Conclusion

Software testing is a process to find (and fix) defects in a system’s implementation

- There is no way to prove a program correct through testing
- Still an essential part of the process
- Must ensure quality of input selection and correct expected output
- Understanding testing is key to a successful project

Software Testing: Definitions

Verification

`IEEE` The process of evaluating a system or component to determine whether the products... satisfy the conditions imposed.

`Kaner` Checking a program against the most closely related design documents or specifications.

`Myers` An attempt to find errors by executing a program in a test or simulated environment.

`Correct` **Confirm that the software performs and conforms to its specifications;**
“Are we building the thing right?”

Validation

`IEEE` The process of evaluating a system or component... to determine whether it satisfies specified requirements.

`Kaner` Checking the program against the published user or system requirements.

`Myers` An attempt to find errors by executing a program in a real environment.

`Correct` **Confirm that the software performs to the user’s satisfaction;**
Assure that the software system meets the user’s needs;
“Are we building the right thing?”

Dynamic vs. Static Verification

Dynamic V & V

Concerned with exercising and observing product behavior;
Testing (in all forms)

Static V & V

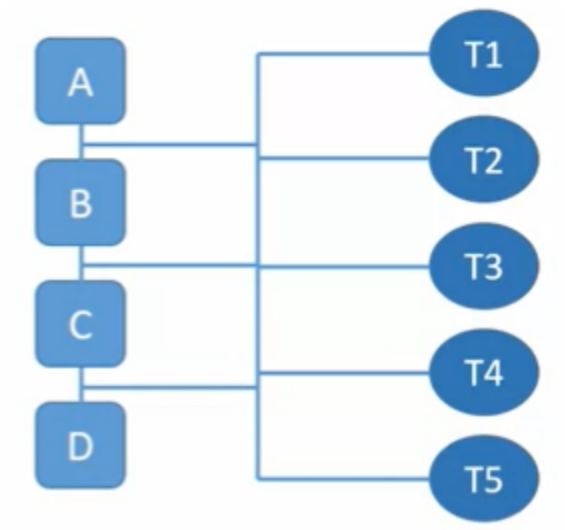
Concerned with analysis of the static system representations to discover problems;
Proofs and inspections

Conclusion

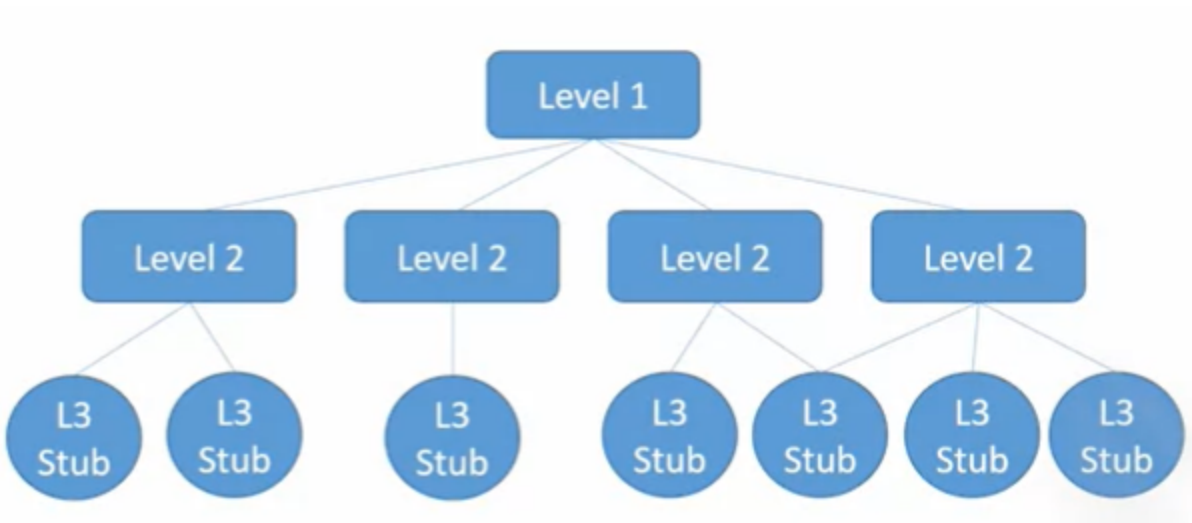
- Building the thing right and building the right thing
- Regrardless of definition, an important aspect of testing
- Dynamic and static V&V aren’t exclusive
- Verfication is cheaper/easier than Validaiton

Software Testing: Strategies

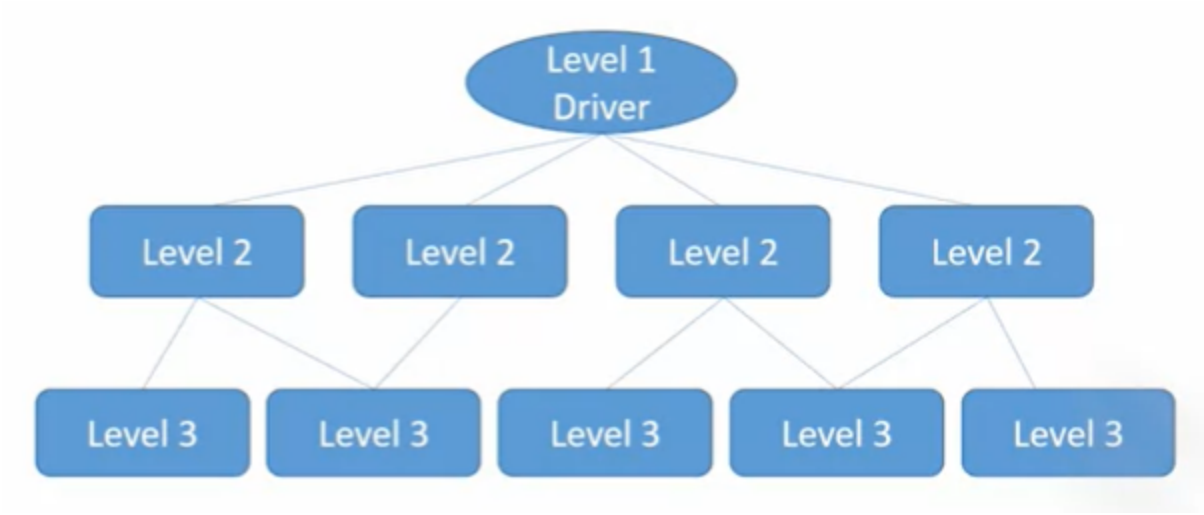
Incremental testing



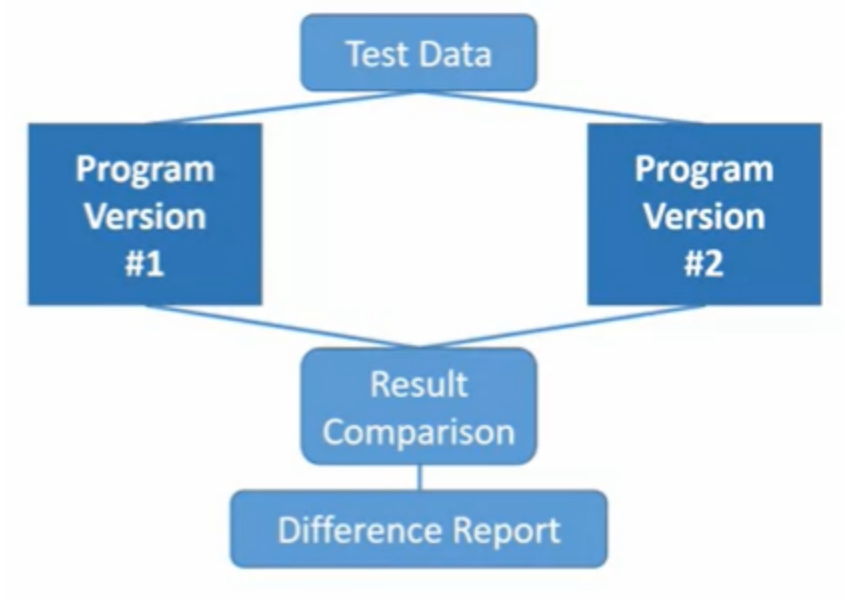
Top-down testing



Bottom-up testing



Back-to-Back testing



Test Scaffolding

GOAL: Setup the environment for executing tests

- Driver: initializes non-local variables, params, activates units under test
- Stub: use templates of modules
- Program Unit
- Oracle: verifies the correspondence between produced and expeceted results

Who should test?

Developer:
+ “this is my baby!”
+ Understands the system
+ Will test gently
+ Driven by deadlines

Tester
+ “Smashy smashy!”
+ Must learn the system
+ Will attempt to break it
+ Driven by “quality”

Axiom of Testing

As the number of defected defects in a piece of software increases, the probability of the existence of more undeteced defects also increase.

Assign your best programmers to testing

Exhaustive testing is impossible

You cannot test a program completely

Even if you do find the last bug, you will never know it

It takes more time than you have to test less than you would like

You will run out of time before you run out of test cases

Conclusion

Strategies of testing drive the actual act of testing units

- Pure top-down or bottom-up doesn’t exist
- Stubs are created to allow higher level units to execute
- Drivers organize and execute the units under test
- Programs cannot be tested completely

Software Testing: Perspectives

Black-box testing

Designed without knowledge of program’s internal structure

Based on functional requirements

White-box testing

Examines the internal design of the program

Requires detailed knowledge of its structure

V & V process

Must be applied at each stage of the process

Has 2 principal objectives:
+ Discovery of defects in a system
+ Assessment of whether or not the system is usable

Stage of Testing

Unit testing

Module testing

Sub-system testing

System testing

Acceptance testing

Conclusion

Many software testing perspectives exist

- Each has its own benefits
- Use them all to get a better picture of how testing works
- It is never one or the other; just use each in isolation
- Even all of them combined will likely not be sufficient

Question 11
In your current project, you have access to some intern development resources, which are not currently operating at full capacity. You also know that the testing timeline will be truncated, due to delays in critical-path module development. Which strategy should you employ?

- [ ] Make no changes to the current project testing or development allocations, utilizing intern resources to create documentation.
- [x] ~~Utilize the intern resources to design and develop drivers and stubs, while work continues on critical-path module development~~
- [ ] Allow the testing team to work without (or with quick-to-develop) drivers and stubs, and utilize the intern resources to aid testing once all critical-path development is complete
- [ ] Allow the testing team to work without (or with quick-to-develop) drivers and stubs, while using intern resources to aid critical-path development

Week 3

Software Development Models

Classification of models: `predictive` vs `adaptive`, `incremental` vs `iterative`

Predictive vs Adaptive

Predictive

You have good understanding of the requirements of the software or the product that you are building.

Adaptive

The customer or the client generally has an idea of what they want to build, but not quite there (they are not 100% sure what they want to build)

Models are not always 100% predictive or 100% adaptive

Incremental vs Iterative

Incremental

Have fairly good idea of what wo build, but instead of building it in one shot, you build in increments.

Iterative

You don’t have clear of an idea, but some idea.

You sometimes replace what you’ve built with something different.

> 💡 The different between **incremental** vs **iterative** is that you are building on top of the existing product.
> In incremental, you breaking the product into smaller pieces.
> There is enhancements happen during iterative.

Incremental vs Iterative vs Both vs None

Waterfall Model

We put all of these software engineering processes one after the other in a logical sequence

Requirements → Design → Implementation → Testing → Deployment → Maintenance

Add this feedback loop from each of the phases to the previous phase.

If you find something wrong in the implementation phase, you will go back to the design phase if the problem was in the design phase, or you can go into the requirement phase if the problem was in the requirement phase

Some assumptions:

- We know requirements very well. They won’t change.
- Team has experience building similiar software
- Translation from requirements to product is going to be perfect

This model is pretty much a very **predictive model**
(cuz you don’t start until ur requirements are done)

Pros/cons

Pros

- Simple, Easy to understand
- Predictability
- Efficient

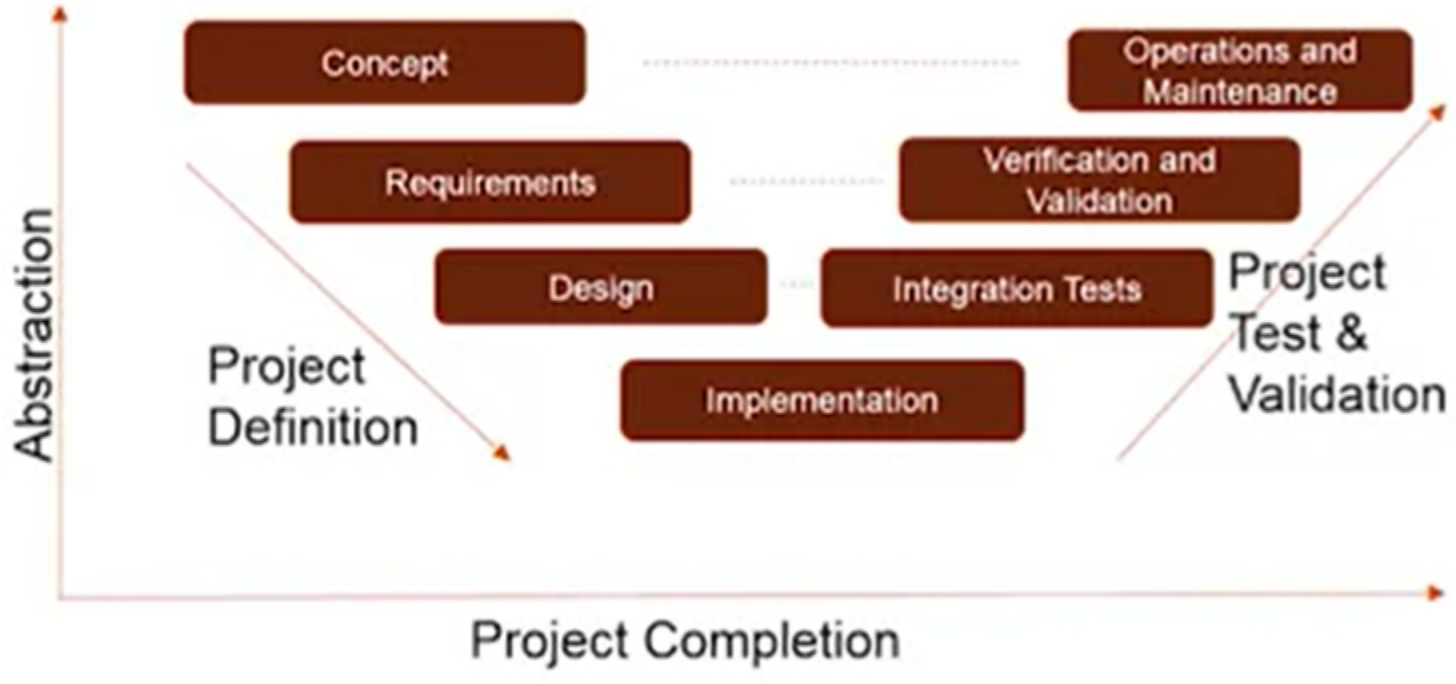
Cons

- Not flexible for change
- First release takes a long time

V-model

Looks very similar to waterfall model but it bent into a shape of V

Concept → Requirements → Design → Implementation → Integration Tests → Verification & Validation → Operation & Maintenance



Left-side is about **project definition**

Right-side is about **project test & validation**

Emphasis on validation earlier in the process

Very much a **predictive model**

Pros/cons

Pros

- Earlier detection of potential defects/issues

Cons

- More upfront work

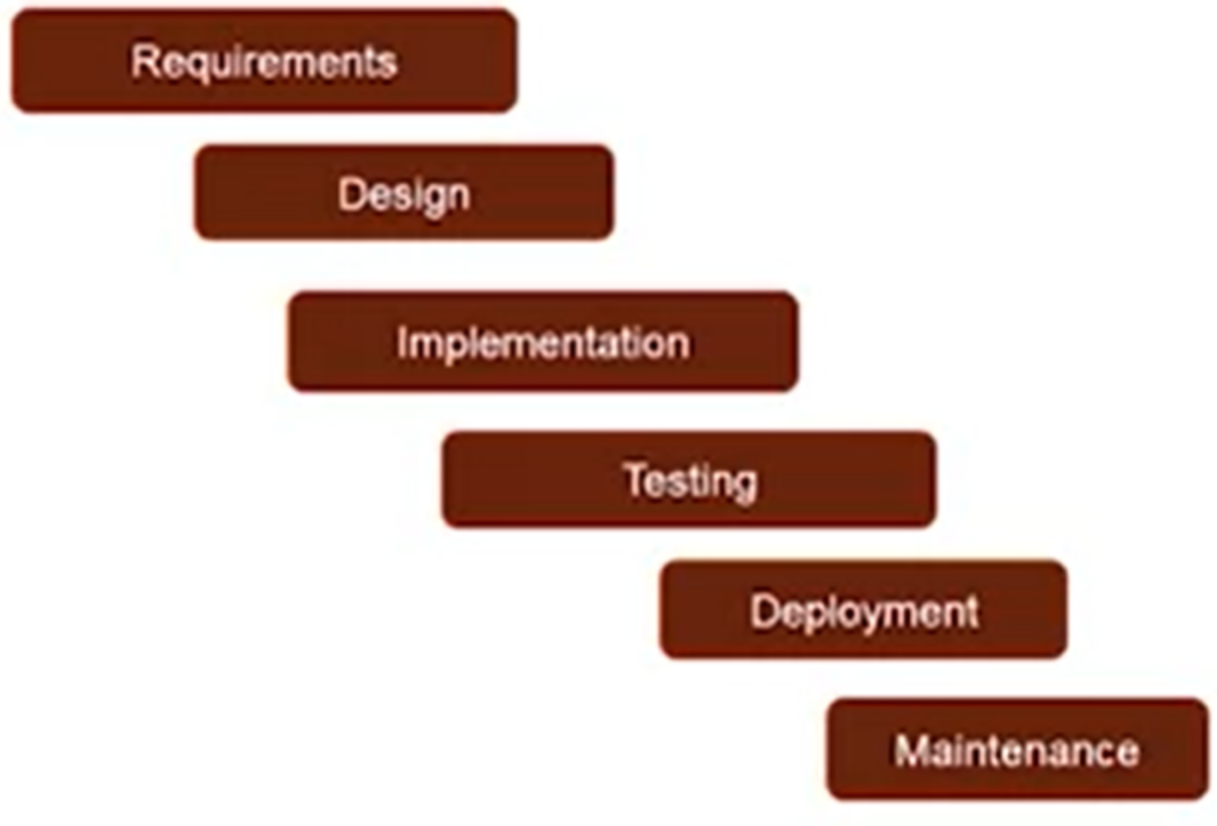
When to use?

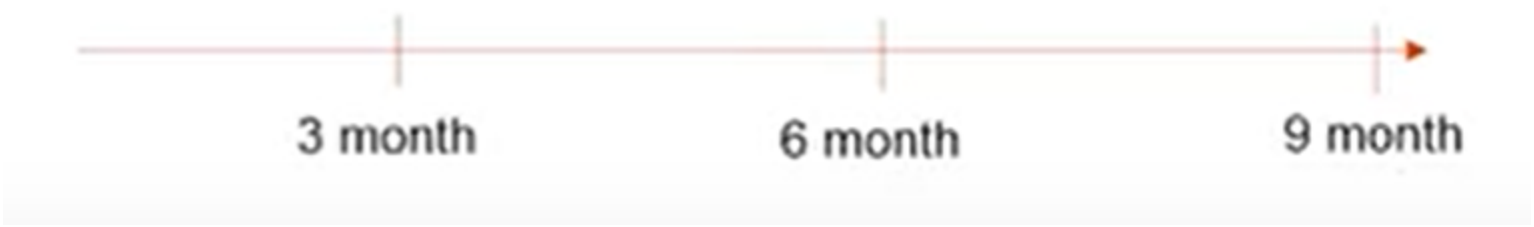
If there is ambiguity in the requirements, and then you want some kind of early validation.

Sashimi model

We allow to overlap the different phase of SDLC

A phase can start before previous phase ends





Not 100% predictive, a little bit toward adaptive

Pros/cons

Pros

- Shorten the development time
- People with different skill can start working without waiting
- Can do a learning spike early

Cons

- May result in rework

When to use?

Want to shorten time scale.

Have all the resource available, you want them to get started on the project soon.

Incremental models

Still use waterfall, but only for that particular increment.

So you will do steps but only for part of the application.
If that increment is useful for users, can deploy and have users use it.

Build in parts. Multiple mini waterfall.

May overlap.

Can use different model for each increment.

Base on apply feedback or not

Pros/cons

Pros

- Deliver value earlier
- Get feedback and make necessary changes between increments

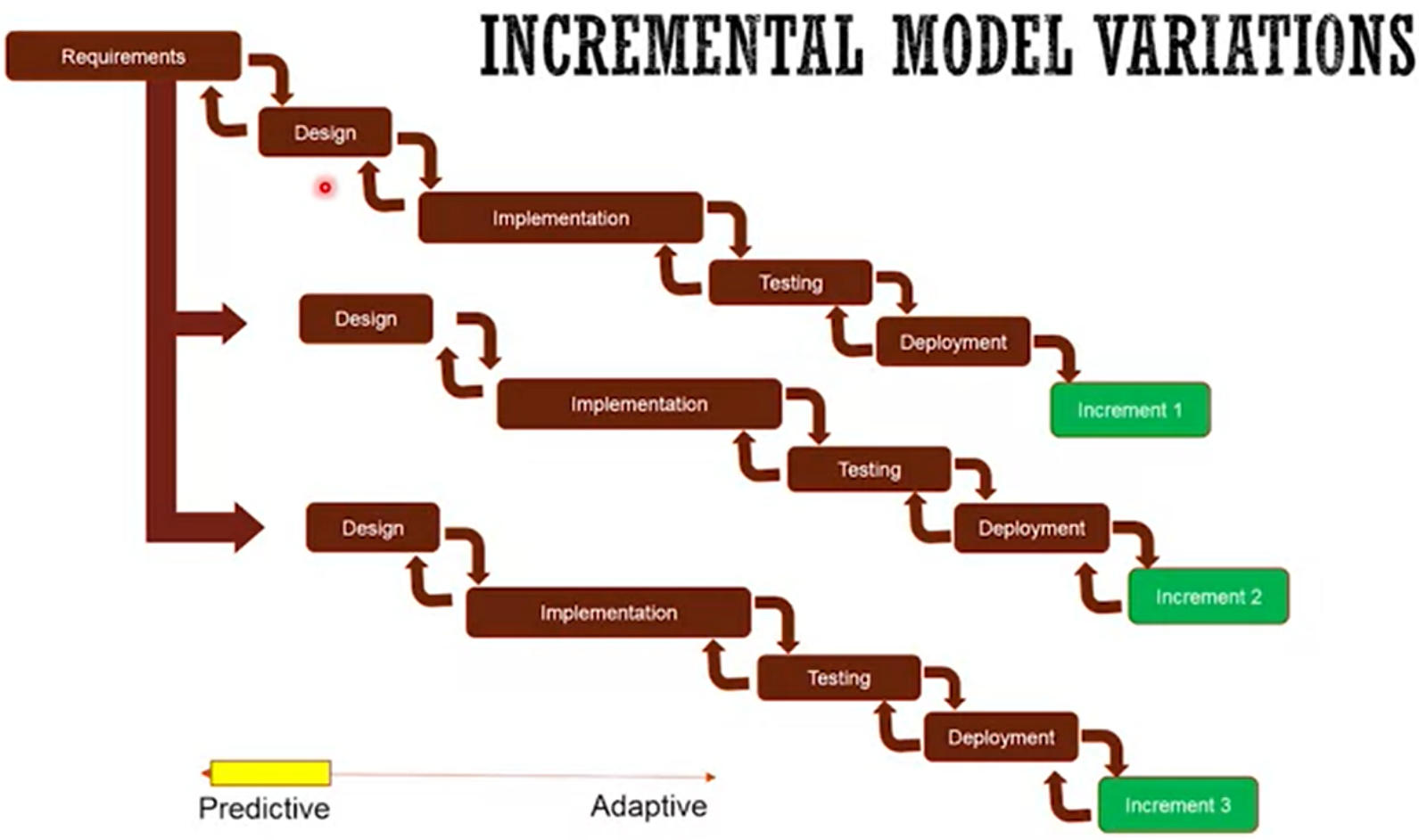
Cons

- May result in rework
- May cost more

When to use?

If organization may benefit from early delivery of part of product

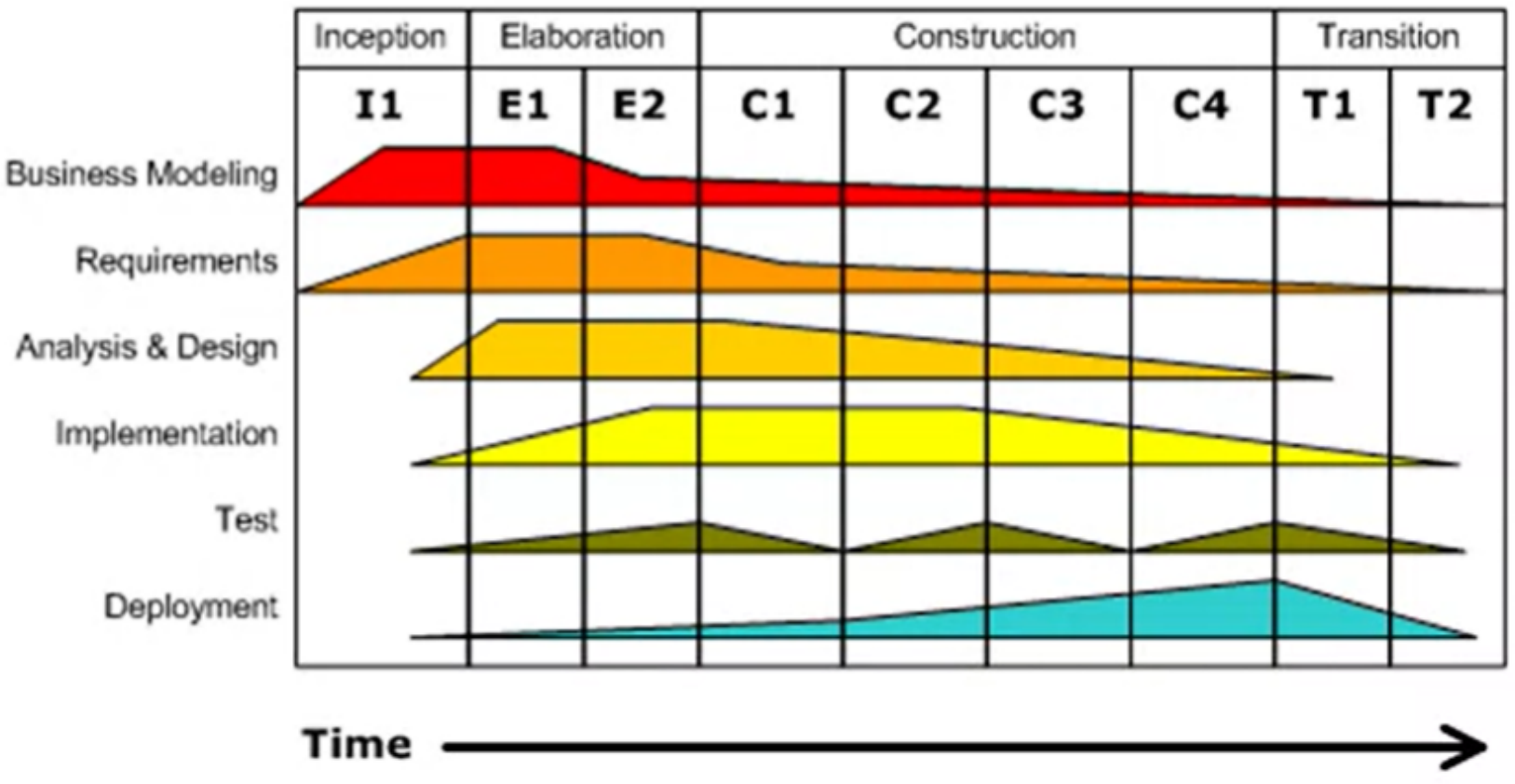
If building one increment will help define future increments



Section: Iterative models

Unified Process and its Variants

What does it look like?



Inception

A lot of focus it on the Business Modeling and Requirements

Shortest phase

What happens?

- Establish business case, scope
- Feasibility
- Build vs Buy
- Preliminary schedule and cost

Milestone: Lifecycle objective

Elaboration

What happens?

- Capture requirements
- Address known risks
- Validate the system architecture by building Executable architecture baseline
- Credible construction estimates

Milestone: Lifecycle architecture

Construction

Largest phase

What happens?

- Software is built
- Multiple iterations - each results in release
- Iterative and incremental

Milestone: Initial Operational capability

Transition

What happens?

- Deployment
- Get feedback and refine
- System conversion and user training

💡 **Characteristics**

- It is a framework. It incorporates other methods/models
- Any step may involve different kinds of work (requirements, design etc) but relative effort and emphasis might be different
- Architecture centric
- Use case centric
- Focus on risk mitigation

Conclusion

Pros

- Adaptive
- Quality and reuse
- Focus in risk mitigation
- Increase chances of success
- Flexible to incorporate other s/w dev models

Cons

- Complicated
- Need more resources
- Too much overhead for smaller project

Unified process variants

Rational Unified Process - 9 disciplines, 6 best practices + tools

Enterprise unified process

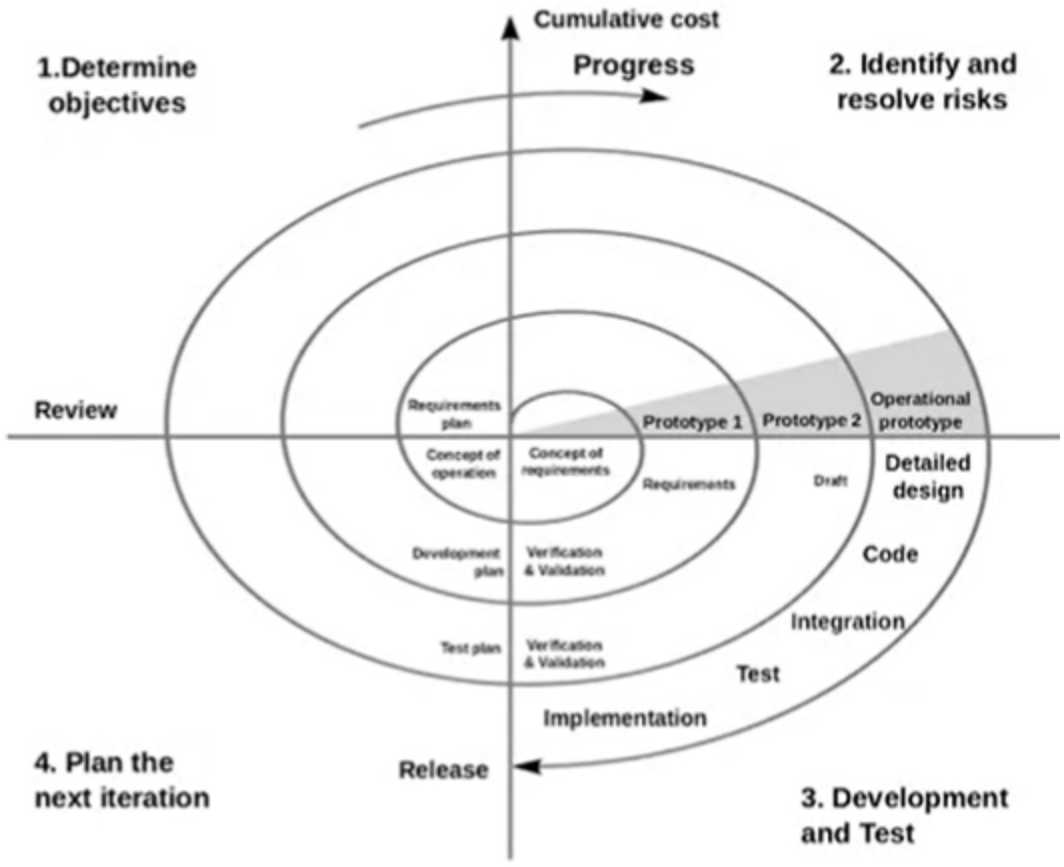
Open UP - lighter version

Agile UP - lighter version, agile focused

Spiral model

Risk driven approach

What does it look like?



1. Cyclic
   Go in cycles and then keep improving or iterating
2. 4 basic steps
3. Not every activity need to be performed

Determine objectives

Have to define:

1. Objectives
2. Constraints (cost, time,..)
3. Alternatives

Identify and resolve risks

1. Identify risks
2. Resolve what you can

Development and Test

Work done to meet objectives

Plant for next iteration

Review work done and commitment for next iteration

Start the next cycle

Some question

How do you track progress?

- Track progress through milestone
  - **Life cycle objective:** sufficient def of a technical and management approach
  - **Life cycle architecture:** sufficient def of the preferred approach + significant risks eliminated or mitigated
  - **Initial operational capability:** sufficient preparation of the software, site, users, operators, maintenance

Characteristics

- Risk driven model
- Effort and detail driven by risk

- Process model generator. Incorporates other models

Conclusion

First describe in 1986 by Barry Boehm as Process Model Generator

Risk driven

4 basic activities in every cycle

Risk determines level of effort and degree of details

Not every activity in the diagram need to be performed

Pros

- Adaptive
- Risk focus increase chances of success
- Flexible for using any model
- Minimizes waste
- Options for go/no-go

Cons

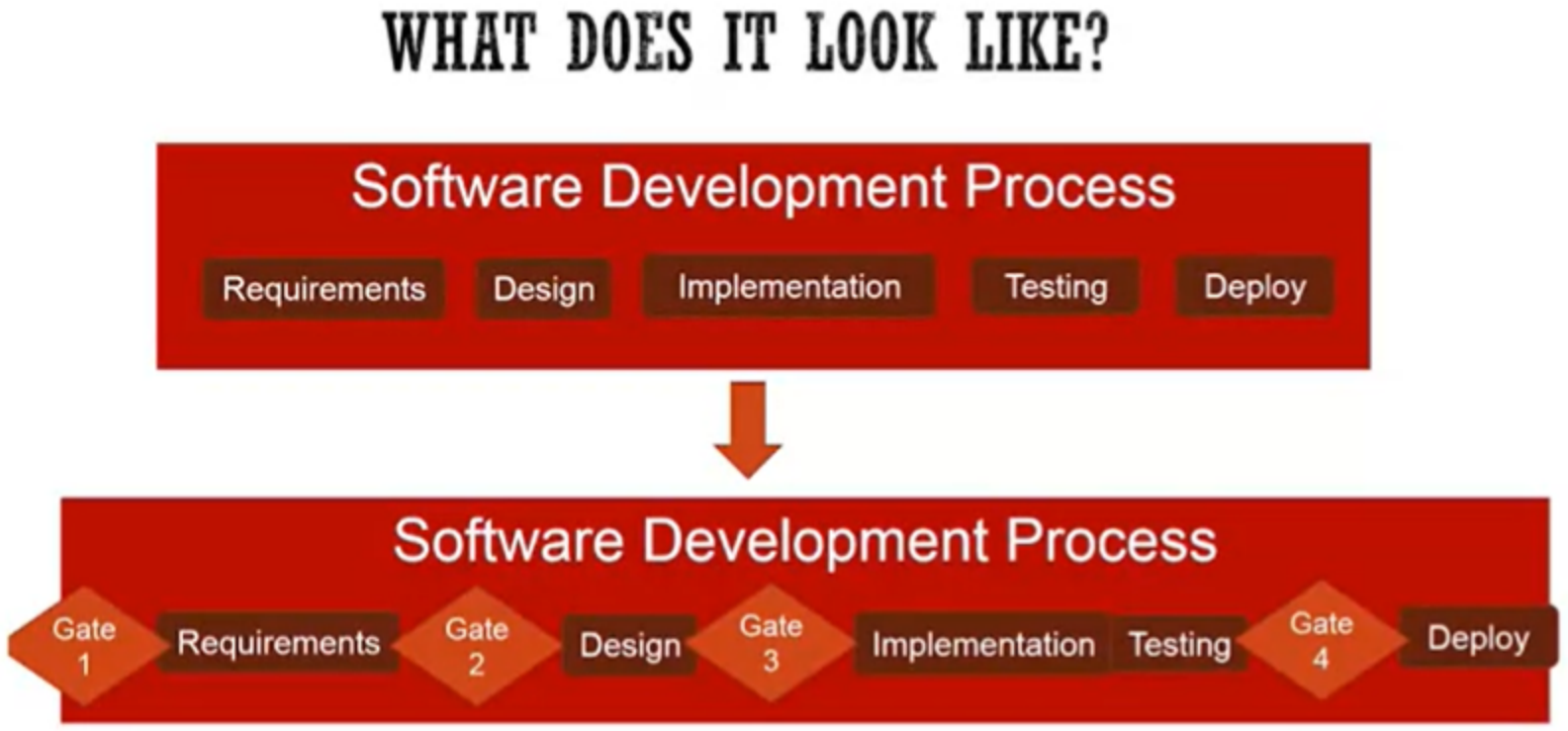
- Complicated
- Cost more to manage
- Need stakeholder engagement

High risk project would be use this model

Applying traditional software development models

Phase gates / Stage gates

Apply checkpoints into differrent parts or different places



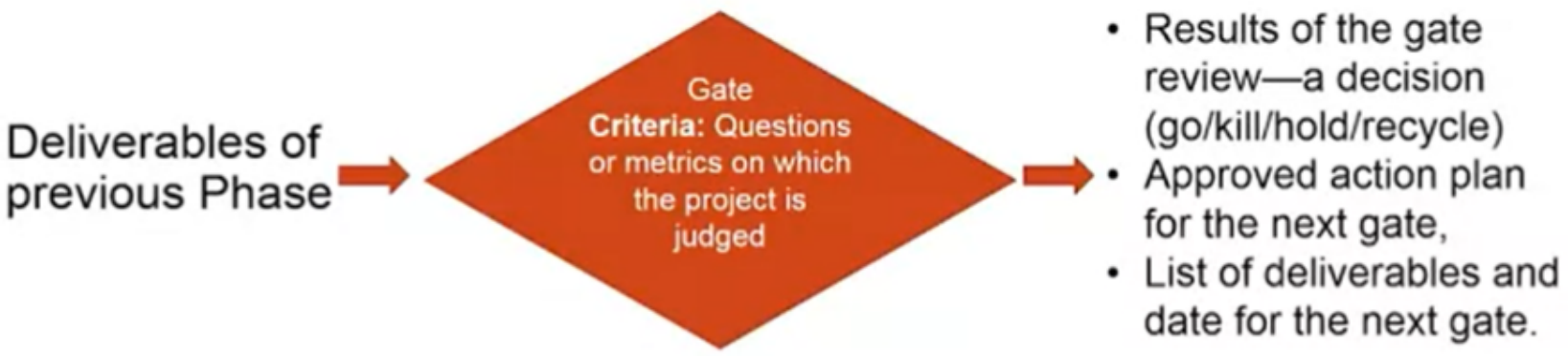
What do you check at gate check?

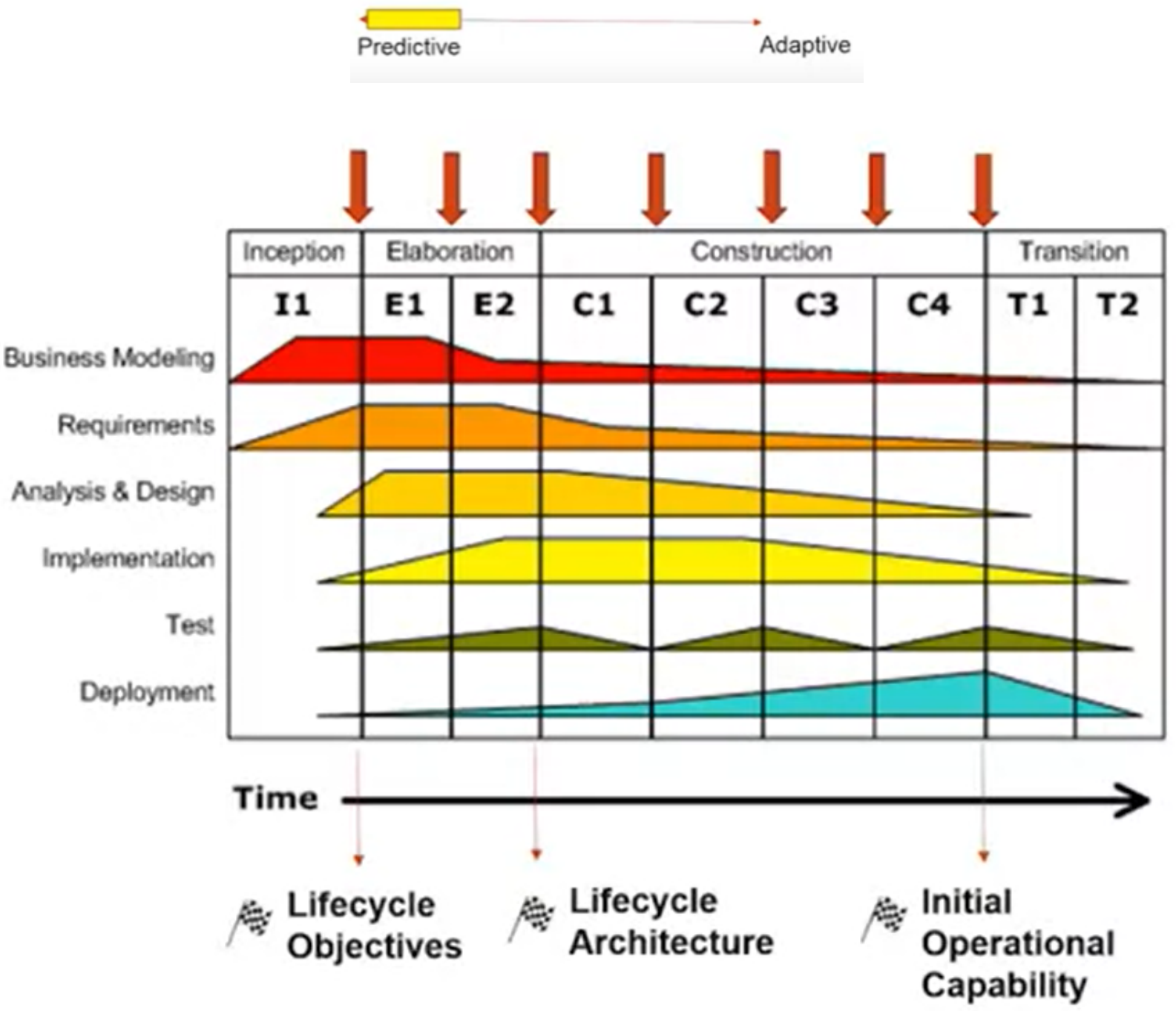
3 things:

- Check quality of execution of previous step
- Business case still make sense?

- If yes, does the action plan for the next step reasonable and sound?

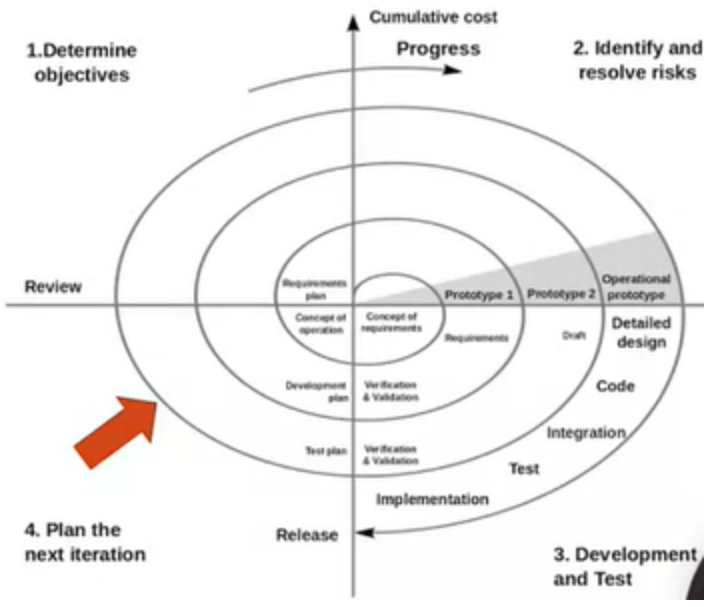
Gate structure and elements





SPIRAL – POTENTIAL PHASE GATES

Step 4 of each cycle by definition is a phase gate.



Conclusion

Technique to provide opportunity to reflect on current progress and decide if we need to change/continue or terminate

Organization customize it to their needs

Pros

- Poor projects can be quickly rejected
- Cost and fiscal analysis provides quantitative info around feasibility
- Provides opportunity to validate the updated business case

Cons

- Potential for structral organization to interfere with creativity and innovation

Applying Software Development Models

Model selection

Assignment

What software development methodology would you suggest for this situation and why?

Step 1: Start analyzing the scenario by identifying the characteristics of this situation and specify the logic behind the selection of characteristics. For example, you may identify "User Needs Unknown" as a characteristic based on statement X, Y, and Z in the scenario.

Step 2: Select a model that best fits the characteristics you identified in step 1. Justify your choice by providing by the logic behind your selection. For example, you may say that since the scenario has characteristics X and Y, models A and B are potential candidates. Additionally, since the scenario has characteristic Z, model A is the best option.

The situation involves a 2 year old product that needs new architecture to handle growing demand. The client needs are well known as the system must perform the exact functions. The best design model for this scenario is the Incremental Model where the Requirements and Design are done upfront and each branch afterwards will follow the Waterfall Model. These designs work best with low-risk, predictive situations. Since the system is performing the same exact needs, the Requirements and Design will be consistent

Imagine that you were the lead or project manager for this project. For the selected model, take us through a simulated/fictitious journey on how this project will be completed all the way from defining requirements to deployment. You are free to make up characters as you feel appropriate to fit your story. Please watch the video on "Applying Software Development Models" to get an idea. The video stays at a high level, but you can go into further detail as you feel necessary. In your story, please make sure to talk about artifacts and practices followed by the team on this project

The PM and Architect will deploy the Incremental Model with the Requirements and Design are done upfront. After this is deployed to our remote team, each team can work on the 4 components of the system simultaneously in seperate incremental branches and be launched when ready to meet rising demand. Testing can be done individually since each component functions relatively independent. The 1 component that requires replacement can be worked on first to allot for

time testing while the other components are much more predictive and can be relaunched. This design will allow for development of multiple parts at one time since there is a high demand and will result in a faster development time.

Assume that you are the quality lead or technical lead on this project. What kind of testing would you suggest the team to do? Be sure to justify your answer. To answer this question, first, list down the key things from the use case above that are really important. For e.g. scalability, performance, usability, integration between components, etc. After that, identify what type of testing would you want the team to do to make sure that upgraded product is high quality and deployed defect free. Please refer to the "Testing and Verification" section in module 2. Also, please watch the following videos to learn about various types of testing methods: https://www.coursera.org/learn/softwareprocesses/lecture/G30EZ/software-testingperspectives

Testing would be important for scalability, handling high traffic, and performing the exact requirements and needs. Each component will be tested for scalability and v & v thorugh unit and module testing. The system as a whole will also be tested for performance with system testing. And finally, to test for meeting requirements and client needs it will be tested through acceptance testing live, with users, especially those who are familiar with the software.

Continuing your role as a quality lead or technical lead for the project. Write a few examples of test cases or a descriptive narrative for what you expect the testing team to use when testing this product. Please refer to the "Testing and Verification" section in module 2. Feel free to make assumptions about the functionality of the system to come up with a scenario.

The system will be tested through acceptance testing by various users to verify and validate the systems functions are performing the same. Users can easily visit the site with the same expectations and test the site through behavior. Since this is a service based system, users with various needs and utilizing various functions can be final verification based on their knowledge of the previous system.

Week 4

Why Agile?

Challenges with waterfall method

- Pieces don’t work together
- Clients problems:
  - This is not what I need
  - After all this investment, we didn’t get the benefit we predicted
- The market shifted

Necessity is the mother of invention

The idea is reduce the learning cycle, how can we learn faster, how can we increase the collaboration of team members

Manifesto for Agile Software Development: 4 values, 12 principles

It is not a specific model/process. Agile is a mindset!

Agile Manifesto and Principles

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

4 values

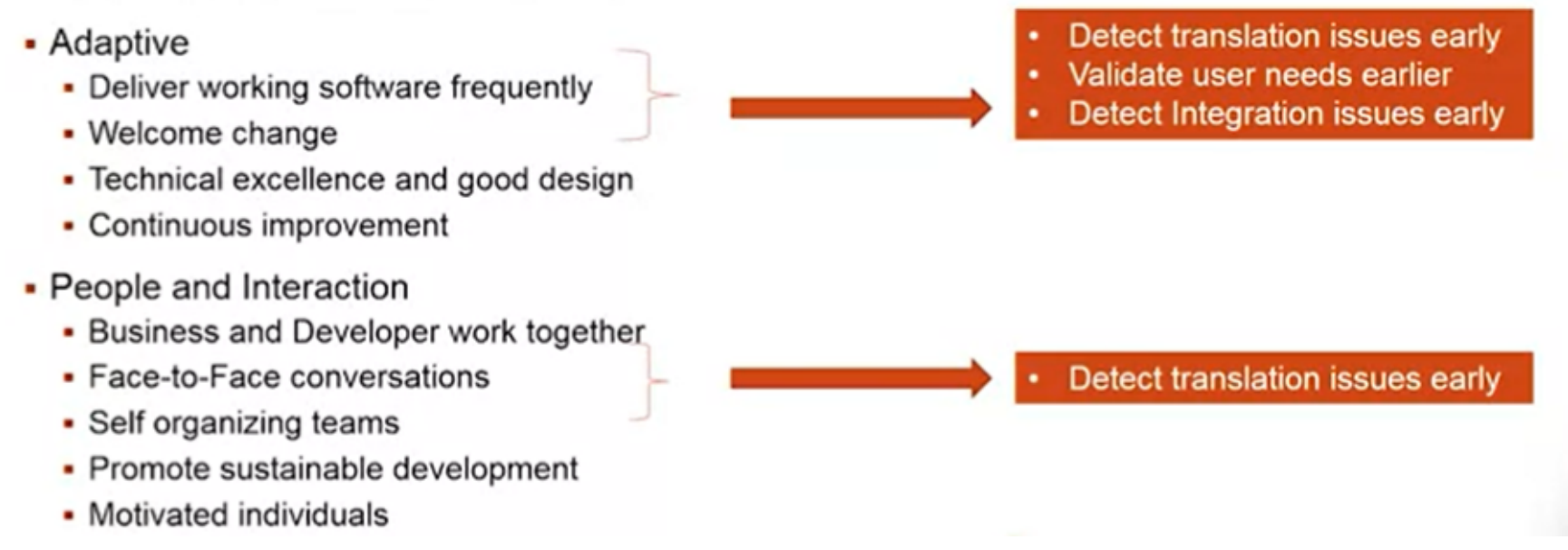
- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Principles behind agile manifesto

1. Our highest priority is to **satisfy the customer through early and continuous delivery** of valuable software.

2. **Welcome changing requirements** even late in development. Agile processes harness change for the customer's competitive advantage.
3. **Deliver working software frequently** from a couple of weeks to a couple of months with a preference of a shorter time scale.
4. **Business people and developers must work together** daily throughout the project.
5. Build projects around **motivated individuals**. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within the development team is **face-to-face conversation.**
7. **Working software is the primary measure of progress.**
8. Agile processes **promote sustainable development**. The sponsors, developers and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to **technical excellence** and good design enhances agility. - cost of exploration.
10. **Simplicity** - The art of maximizing the amount of work not done - is essential.
11. The best architectures, requirements and designs **emerge** from **self-organizing teams.**
12. At regular intervals, the **team reflects** on how to become more effective, then tunes and adapts its behavior accordingly.

How does it solve problems of waterfall model?



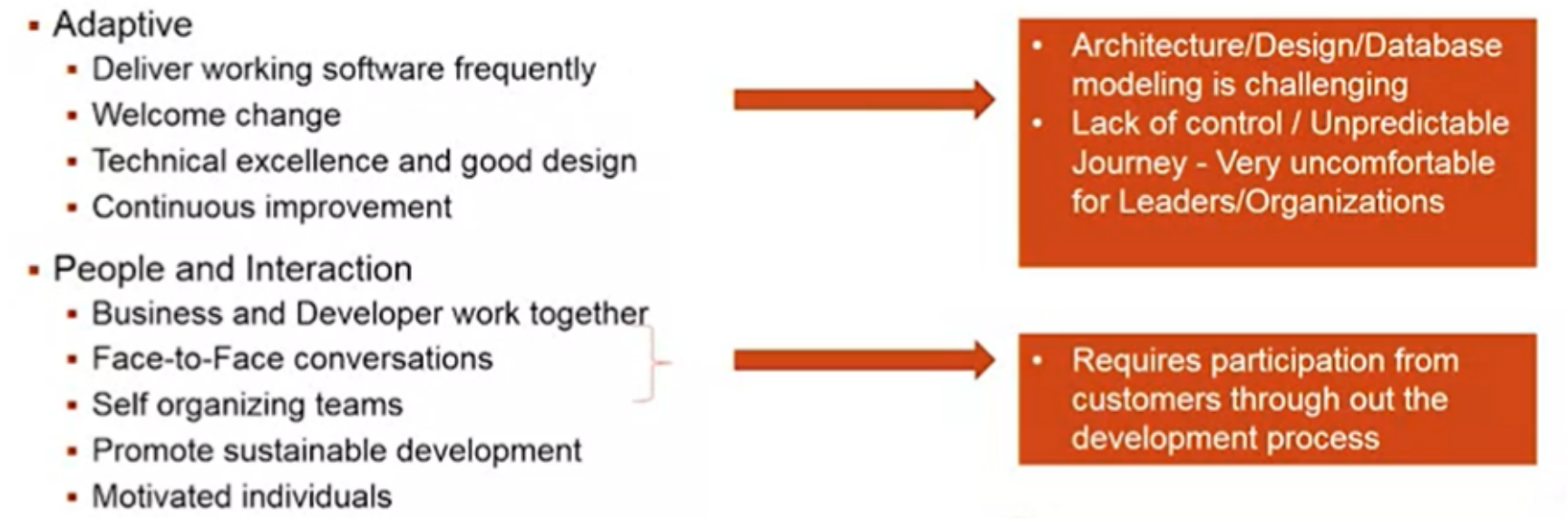
What new problems does it bring?

Adaptive

- Architecture/Design/Database modeling is challenging
- Lack of control/Unpredictable Journey - Very uncomfortable for Leaders/Organizations

People and Interaction

- Requires participation from customers through out the development process



Agile Frameworks

Scrum

Based on 1-4 week cycle where you take part of your project and do your define, develop, design, test

Most popular of all. About 70% of the Agile teams use either Scrum or one of these variants

Kanban

Based on continuous flow model where you basically try to optimize your existing software development process.

Scrumban

Combination of scrum and kanban

XP - Extreme programming

Scrum XP Hybrid

Lean startup

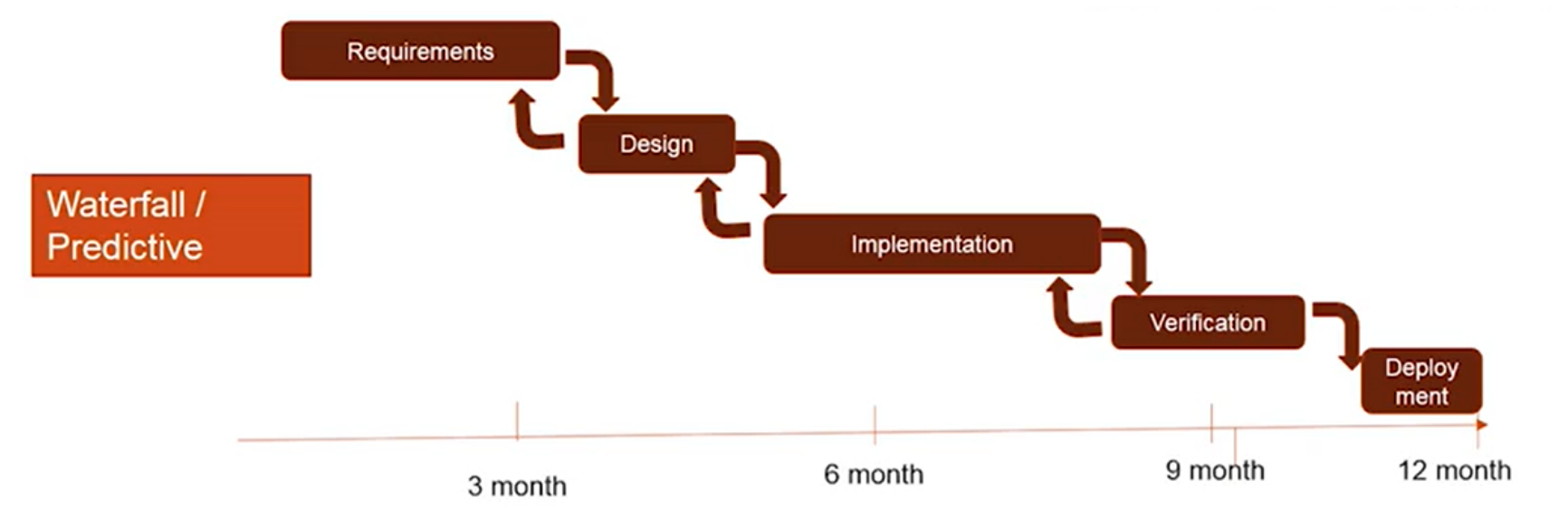
Help you if you have a lot of unpredictable market or industry and you want to really prove your solution before implement it.

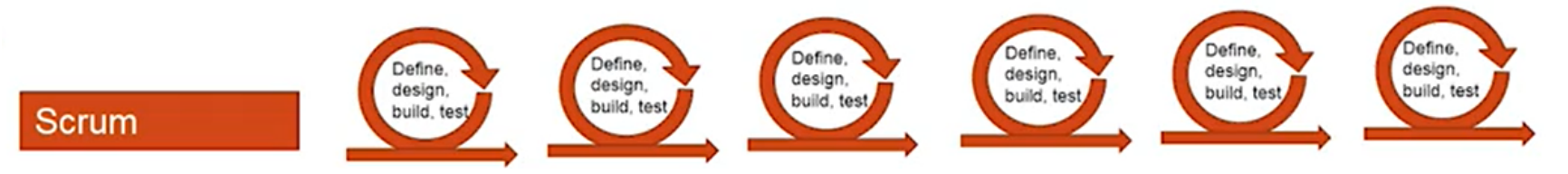
Crystal

Feature Driven Development (FDD)

Dynamic Systems Development Methodology (DSDM)

Scrum





Plan, Build, Learn — Repeat

Scrum works on 1-4 week sprint where you take part of your product

After every two or between one to four weeks, you get some finished product and then you just keep on adding to your existing product until you get your final product.

Scrum framework at a glance

3 roles: product owner, Scrum master, the team

Product owner:

- Define what needs to be done, in what order

Scrum master:

- Helps the team stay true to the Scrum values and principles

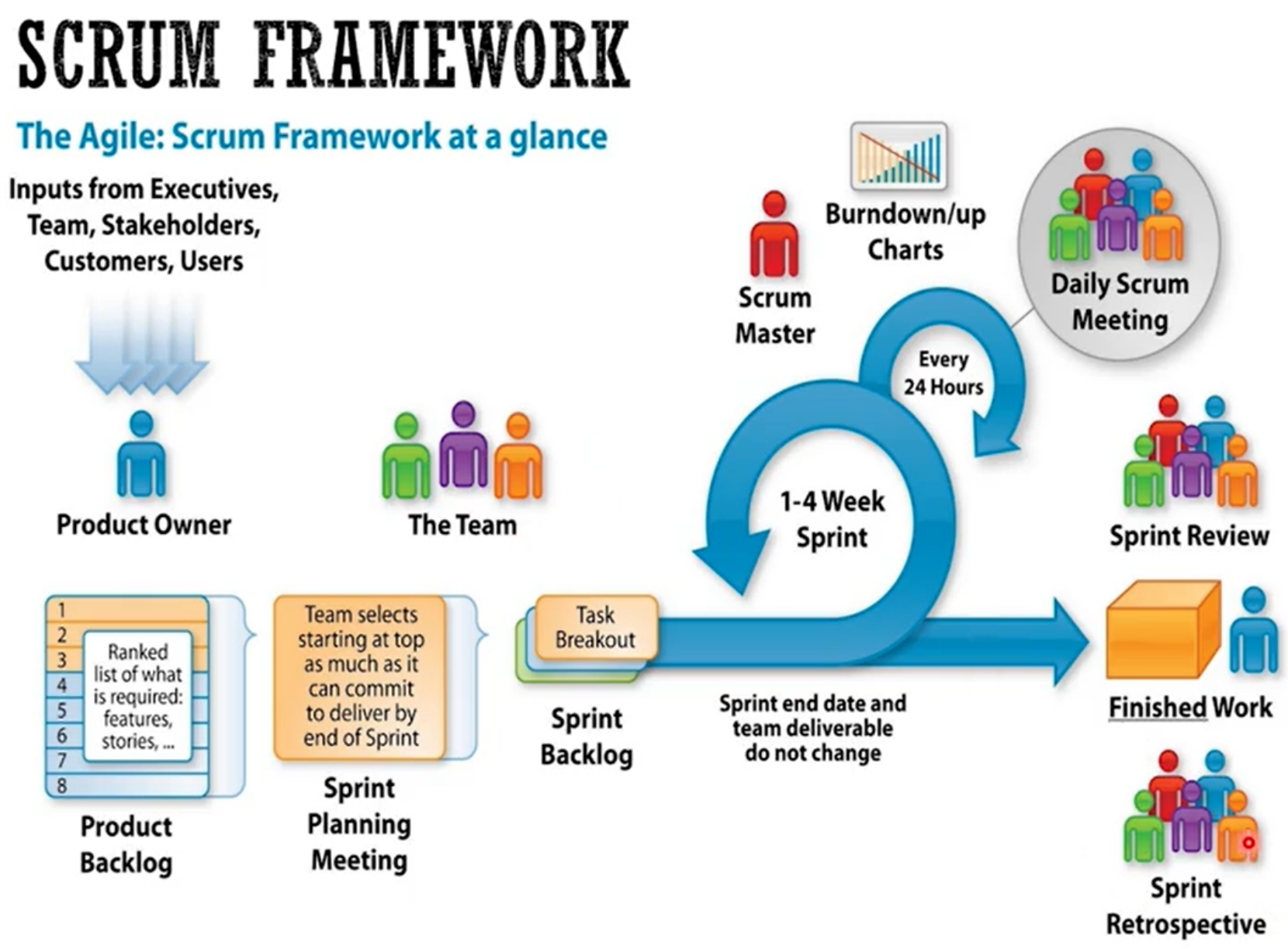
- Helps facilitate most of the meetings in the team

The team:

- Self organizing
- Do most of the building of the software

Process:

- The product owner get inputs from Executive, Team, Stakeholders, Customers, Users
  Create a `product backlog`
- **Product Backlog:**
  - Basically a list of user stories which are prioritizedand defines what needs to be done
  - At high level
  - Can change over time
- **Sprint Planning Metting:**
  - The whole team gets together, pick the top stories they can work
  - PO reviews the stories, helps answer any question/clarifies anything that not clear
- **Sprint Backlog**
- **Finished work**
- Sprint review:
  - whole team gets together with the stakeholders, with the client and demonstrate the work that they have done and get the feedback.
- Sprint retrospective:
  - they talk about the process and not about the product. So they talk about like how can we do better?



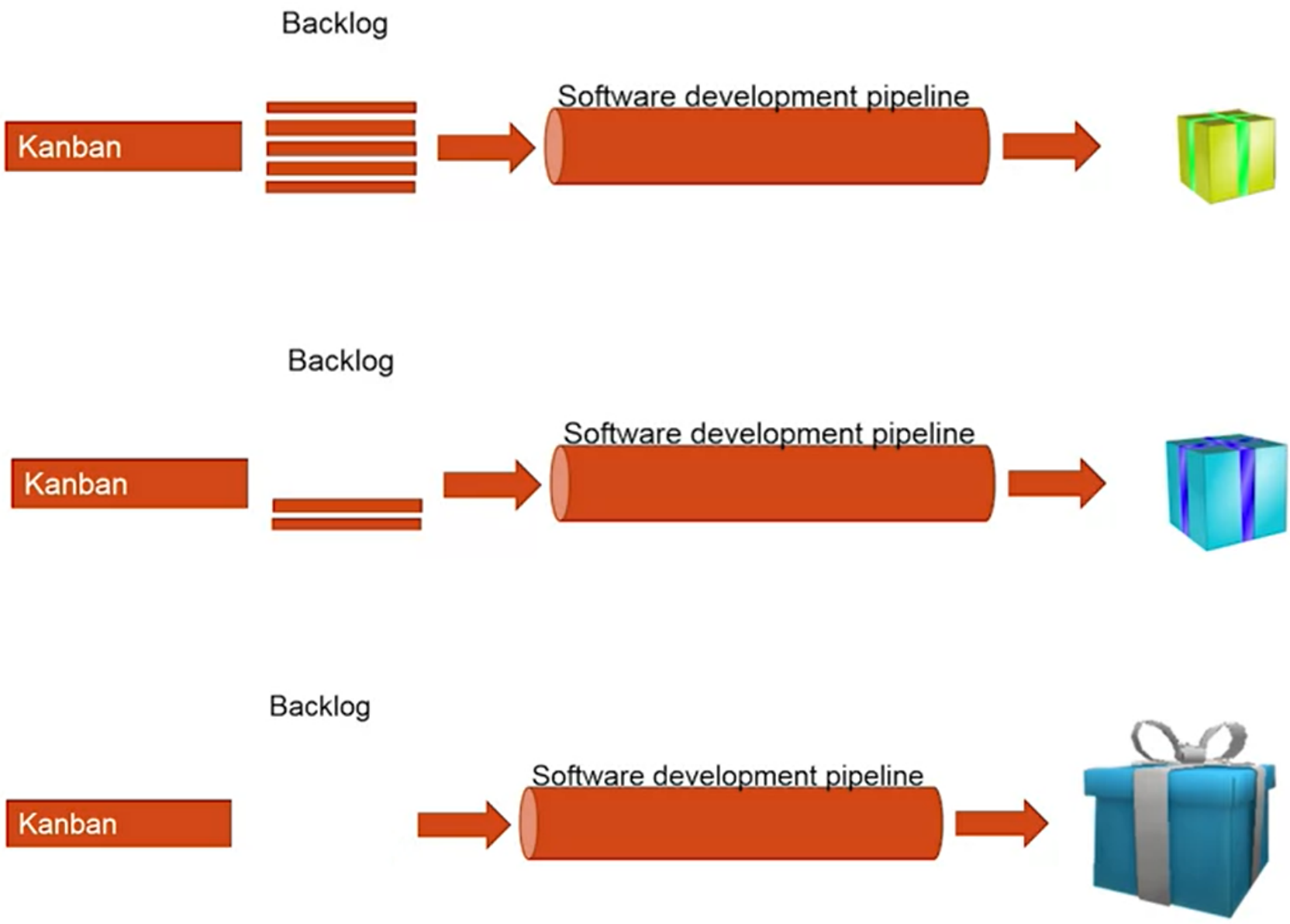
Burndown/up Chart

- Show the amount of work left or the days of the sprint

Kanban

It's just a set of properties and principles that help optimize software development process as long as that process is a `continuous flow`.

things from the backlogs are moving through this software development pipeline and its finished product is coming out at the other end of the pipeline.

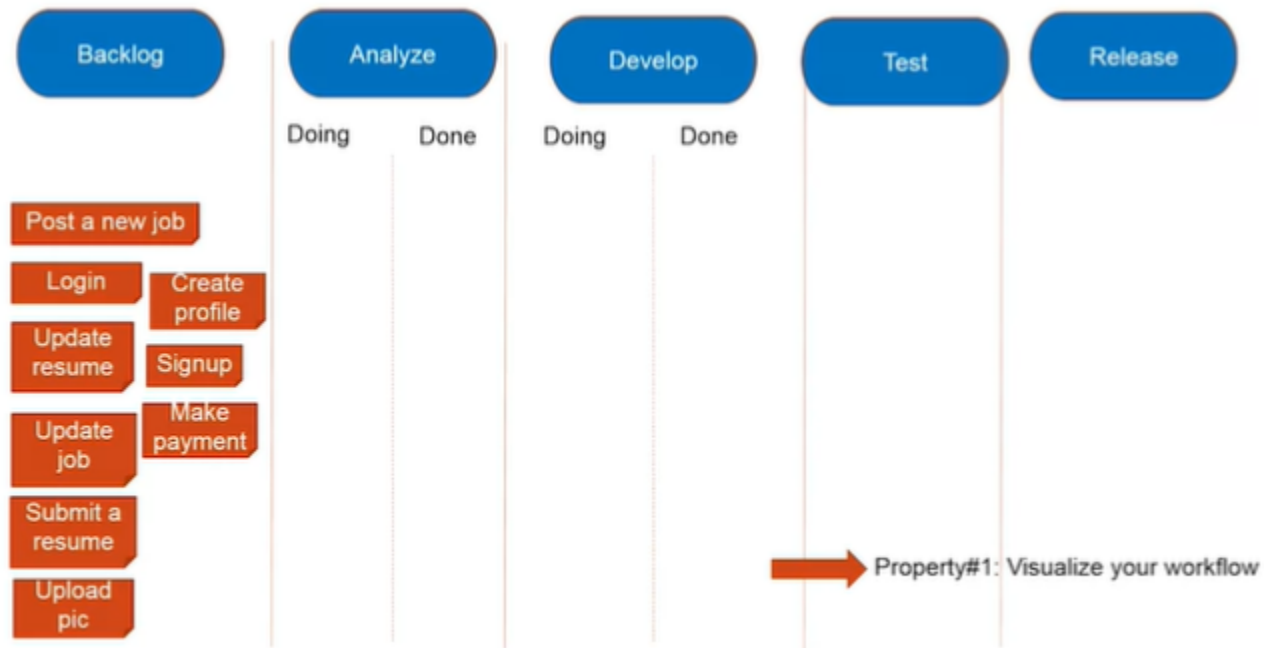


Kanban optimize workflow

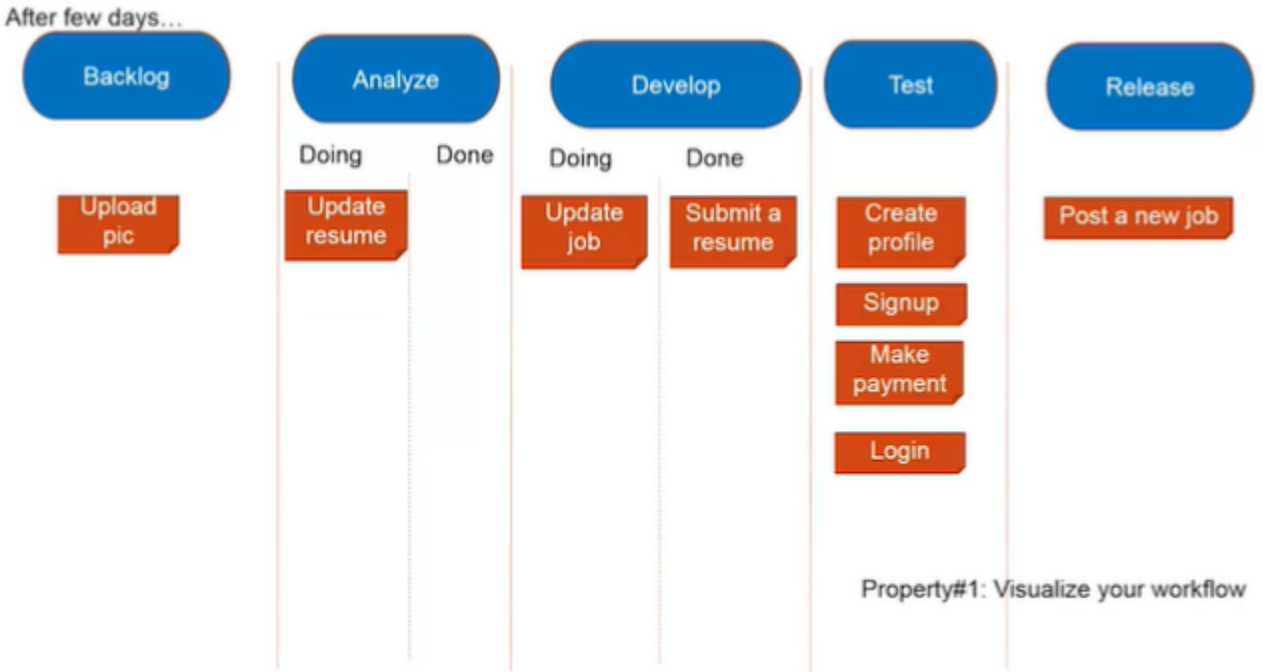
Backlog → Analyze → Develop → Test → Release

Property

- Visualize the workflow
- Limit work in progress (WIP)
- Manage the flow
- Make process policies explicit



After few days…



Kanban principles

1. Start with what you do know
2. Agree to pursue incremental, evolutionary change
3. Respect the current process, roles, responsibilities & titles

Kanban properties

1. Visualize the workflow
2. Limit WIP (work in progress)
3. Manage flow
4. Make process policies explicit
5. Improve collaboratively

Agile and Lean Summary

Agile is mindset not process/model

Incremental and Iterative

Very short development cycle

Pros

- Flexible to change increase chances of building the right product
- Speed to market
  Quickly get something out into the market, beat your competitor

Cons

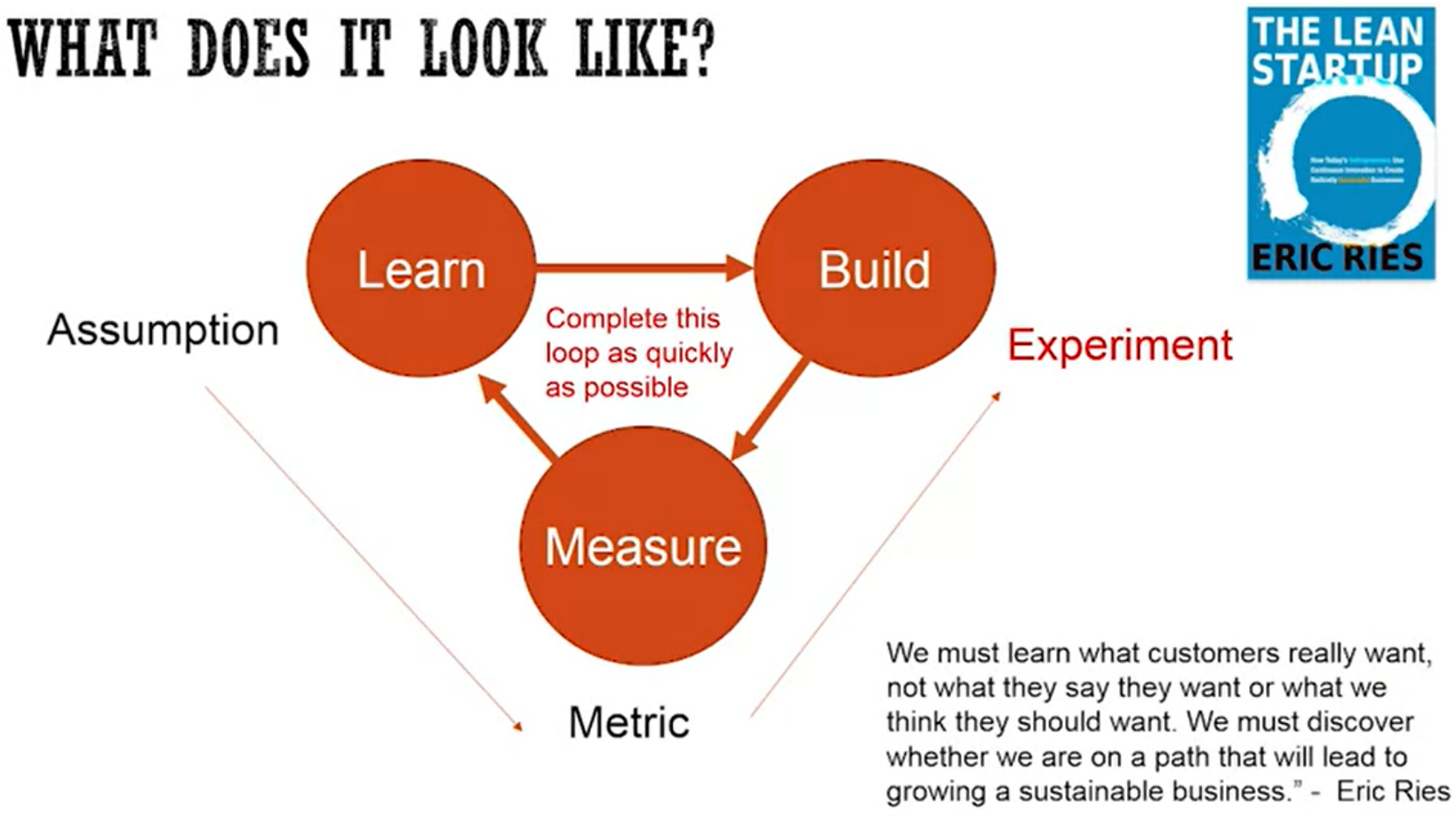
- May result in rework
- Requires close collaboration with clients and users

When use?

- Where requirements may change
- Team using unproven technology (unknown/ haven't used)

Lean Startup

The basic idea behind this approach is, how can you learn faster about your market or your user need?



Concept 1: Validated learning

"We must learn what customers really want, not what they say they want or what we think they should want. We must discover whether we are on a path that will lead to growing a sustainable business."

Concept 2: Time - complete this cycle as quickly as possible.

Characteristics

Incremental and iterative

Very short development cycle

Pros

- Helps you learn faster and build the right product
- Speed to market

Cons

- May result in rework
- Requires you to experiment iteratively with clients and users.

When use?

- Doubtful business case / user need
- Lots of high probability risks

Assignment: Project Scenario 2

As a software development professional, you will run into all kind of projects and situations within those projects. This assignment is designed to present you with a fictitious situation and ask you to recommend an approach to software development for that situation.

Here is the situation:

Georgia school of Arts has been the choice for higher education in the state. The college has been serving the state for last 50 years. College has mostly operated using manual processes and state has been expecting the college to match the other colleges in the country in terms of automation. College leadership also sees great benefit by automation as it hopes to keep tuition in check and provide better service to student and faculty.

The college wants to automate most of its processes in next 5 years. College do not have budget to shorten the timeframe and wants to do this automation mostly in-house with hired consultants when needed.

The IT department has lot to learn in terms technology to build this system. Also, college leadership has some idea on what to build but not sure what exactly are college needs in terms of automation. College has formed a group from various departments of college that will help define and drive the development. This group will work closely with IT department on this effort.

College wants to reap the benefit of automation as it is being built and use feedback from system users to guide future automation efforts.

The automation will start with processes that impact the students for e.g. admissions, class registration, grading, learning management system etc. Some of the key things identified for this software are the privacy and security concerns. College wants to make sure that systems can't be hacked and only the right people have access to the info.

To start with, the team working on this project consists of 5 developers, 3 QA and a Team Lead. Organization had signed a contract with a local company to provided additional resources when needed.

Assessment

What software development methodology would you suggest for this situation and why?

Step 1: Start with analyzing the scenario and identifying characteristics of this situation and specify logic behind the selection of characteristics. Example: You may identify "User Needs Unknown" as a characteristic based on statement x, y and z in the scenario.

Step 2: Map the characteristics to selection of model and provide your logic to make that conclusion. For e.g. you may say that since scenario has x and y characteristic, model A and B would be potential candidate. Additionally, since scenario has characteristic z, model A would be best option.

For the selected model, take us through a simulated / fictitious journey on how this project will be done all the way from requirements to deployment. You are free to make up characters as you feel appropriate to fit your story. Please watch the video on Applying software development models to get an idea. The video stays at high level. You can go in further details as you feel necessary. In your story, please make sure to talk about artifact and practices followed by the team on this project.