**Software Development Processes and Methodologies**

What software dev looks like:-

Waterfall – phase to phase to phase

* Requirements –
* Design - mis
* Implementation
* Verification
* Operations and Maintenance

Limitations of Waterfall – Difficult to respond to changes, misinterpretations of requirements or design can remain undetected until the later development phases, integration issues may remain undetected until the last phase.

V Model

Sashimi Model

RUP Model

Spiral Model

Agile – not a model – it is a mindset

Agile manifesto and principles

Scrum

Kanban

XP - Extreme Programming Project

Continuous Integration

Automated Testing

Automated Deployment

Automation

DevOps Culture

Can Agile work for big projects:-

Scaled Agile Frameworks

Disciplined Agile Delivery

Large Scale Scrum – LeSS

Smaller cylces – faster – cheaper?

Lean Startup

Design Thinking

Output => Outcome

Process is just part of the puzzle

Importance of Requirements

Software being intangible makes it hard

What is Requirement(s) Specifications?

2 things

**Process**

Create high level descriptions

Distinguis between ‘right’ and ‘wrong’ system

Capture WHAT not the HOW of the solution

**Product** (of the process)

What are the requirements specifications important?

Engineering Argument

Economic Argument

Poor requirements are the source of all evil

Requirements problems are

The most costly

The most difficult to identify

The most difficult to correct

**Requirements vs Specifications**

Writing the software requirements specifications (SRS)

* Users
* Developers

Natural language used to avoid miscommunication

* User requirements
* System specification

Requirements for the user – Specifications for the developer

Write your requirements in the user language

Write your specifications in the system language

Be sure that your specifications meet the requirement

**Non Functional Requirements**

Define System properties and constraints

Process requirements

Often more critical than functional requirements

Security, Performance, Usability

Non-functional requirements

* System Design Constraints.
* Quality related constraints like security, performance, and usability

Product Requirements

Organization Requirements

External Requirements

Product – Requierments which specify that the delivered product must behave in a particular way

Organizational – requirements which are a consequence of organizational policies and procedures

External – requirements which arise from factors which are external to the system and its development process

Non-functional requirements

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

A constraint to only use Microsoft Project during the system development is a non-functional requirement.- True

Which of the following are non-functional requirements:

Some product requirements, like using a specific encryption protocol, are non-functional requirements.  
Organization requirements imposed by the company, like a specific coding style, are non-functional requirements.  
External requirements imposed by external organization, like using a specific development style, are non-functional requirements.

WRSPM – Reference Model – The World Machine Model

Capturing the ‘Right’ Thing

* Requirement are always in the problem demain
* Software specifications is in the solution (computer) domain
* Several layers of abstraction can exist in between

Interface is the overlap between Environment & System

World – Assumptions to the ‘world’ situation  
Requirements – problem to be solved  
Specification (natural language  
Program – what software devs write  
Machine – hardware spec

eh – elements of the environment hidden from the system  
ev – data visible to the system  
sv – system elements visible to the environment  
sh - system hidden all the background detail, not visible to the user

WRSPM Model is a reference model for how we understand problems in the real world

* Helps identify the difference between requirements 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

WRSPM stands for: World, Requirement, Specification, Program and Machine

Looking at the difference between user requirements and system specifications in the ATM example, we know that swiping the card and prompting for a PIN are requirements, while reading the card details and a 4-digit PIN are specifications. – TRUE

The purpose of the WRSPM model is to ensure that: Specifications meet the requirements

WRSPM – Real World Example – the model helps identify the elements involved in solving problems

* Must be careful of world assumptions
* Requirements are met whne…
* Specification meet requirements when…
* If the W, S imply R and M, P imply S, your solution is ‘right’

**Software Architecture: Definition**

Parellels

* Architects are the technical interface between customer and contractor building <the thing>
* Bad architectural design cannot be rescued by good construction
* Specialist types of projects require architect expertise
* Schools and styles have emerged

Partitioning large systems into smaller ones that can be created sepearetly and integrated with one another and with existing systems

Good things are well architected

Good architecture is hard

Mistakes are the architectural level are hard to fix by just coding

Software Architecture mainly about decomposing the system into components

* Each component must have individual business value
* Helps organize workforce and resources
* Allows for parallelization
* Helps define the build vs buy question and getting funding
* A subsystem in an architecture must:
* be created separately and can operate individually.
* have business value.
* be integrated with one another or with existing subsystems.

Partitioning of a large system into smaller subsystems helps the buy-or-build decision because we can examine each subsystem and reason about possible buy-or-build options for each. – TRUE

A good software architecture is important because

* It helps organize the workforce and resources.
* It allows for parallelization in development.
* It helps build-or-buy decisions.
* It helps with funding decisions.

**Software Architecture: Models**

* Pipe and Filter
* Blackboard
* Layered
* Client server
* Event based

**Pipe and Filter**

Shared data between data sources which can be interspersed between data streams, components talk to each other

**Blackboard**

Using a hub of data to share data or process, no interaction between components all feed to main (blackboard) point

**Layered**

Triangle form, various forms top to bottom

Business  
Data  
Application  
Tech / Infrastructure

Systems are partitioned between layers

**Client Server**

Cloud based computing is the big one. Computer – Cloud – AWS

**Event Based**

Similar to blackboard, Event manager is a central point components feed into, Event manager has a separate connection to component

Software Architecture Models are best practice solutions to commonly encountered problems

* Start from the pure model, and adjust
* If your goals align with the goals of the model, use it
* Customize the ‘pure’ form to your needs
* Each models solves one particular large scale problem

A UNIX program where the output of one program is the input of another, is an example of which of the software architecture models below: -- Pipe and Filter Model

An online banking system is best modeled by: Client Server Model

A vehicle identification and tracking system, where each moving vehicle is tracked and monitored through a shared program, is best modeled by: Blackboard Model

**Software Architecture: Process**

System structuring

Control modelling

Modular decomposition

Subsystem vs Modules

Subsystem – independent system which holds business value

Module – component of a sub system which cannot function as a standalone system

Software Quality Attributes

* Performance
* Reliability
* Testability
* Security
* Usability

Software architecture process concerns itself with estimation, quality and 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

The difference between subsystems and modules are: Subsystems can independently comprise the business logic by itself while modules cant

Software architecture concerns itself with both estimation and quality but not partitioning. - FALSE

**Software Design: Introduction**

What is Software Design?

A process

A product of the process

What is it not?

Architecture

Design

Stages of Design:

* Problem understanding
* Identify one or more solutions – There is More Than One Way To Do It - TMTOWTDI
* Describe solution abstractions
* Repeat process for each identified abstraction until the design is expressed in primitive terms

Stages of design (formally)

* System Architecture
  + Component Spec
    - Component Interface Spec
      * Component Design
        + Data Structure Design

Algorithm Design

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

* Decide classes, method, data types to be used in the solution
* Separate architecture from design (or don’t)
* Provide detail which is implementation ready
* But doesn’t include implementation detail/constrictions

Software design is the process of transforming the stated problem into a ready-to-use implementation – FALSE

Abstract solutions do not require extensive domain knowledge and effectively reduce the costs during the software design phase. – FALSE

It is often advised that abstract solutions do not provide optimization details regarding the implementation. – TRUE  
When it comes to software design, it is always best to follow a solution that is widely popular in the industry – FALSE

While a solution coming from software design does not include implementation details, there are still common cases where pseudocode may be provided to correctly capture the sense of a complex algorithm. - TRUE

**Software Design: Modularity**

What are the aspects:

* Coupling
* Cohesion
* Information hiding
* Data encapsulation

Definition and principles of modularity

Complex systems MUST be broken down into smaller parts

3 primary goals:

* Decomposability – divide and conquer
* Composability
* Ease of understanding

Information Hiding:

Hide complexitiy in a ‘black box’

Examples: Functions, macros, classes, libraries

Example:

Void sortAscending (int \*array, int length)

Don’t know which sort it uses

Don’t really need to

Know how to use it

Data encapsulation

* Encapsulates the data
* Helps find where problems are
* Makes designs more robust

Which of the four aspects of modularity is defined as: How well modules work together. – Coupling

Which of the four aspects of modularity can be described as: Abstracting away implementation details. – Information Hiding

Three aspects of \_\_\_\_\_\_ can be described as (1) Decomposability, (2) Composability, and (3) Ease of Understanding. – Modularity

You have a sort function that provides no details on which sorting algorithm is used. This is an example of which aspect of modularity? – Information Hiding

A benefit to using \_\_\_\_\_\_ is that you know if your data is corrupted, then it must have been corrupted by the module. – Data Encapsulation

A benefit to using \_\_\_\_\_\_ is that you know if your data is corrupted, then it must have been corrupted by the module. – Cohesion

Which of the four aspects of modularity can be described as: Containment of constructs and concepts within a module. – Data Encapsulation

**Software Design: Coupling**

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

Loose coupling allow for changes to be unlikely to propogate across components

Shared variables and control information lead to tight coupling

Loose coupling achieved by state decentralization and message passing

Couling Levels – Tight – Medium – Loose

Tight

Content Coupling

Common Coupling

External Coupling

Medium

Control Coupling

Data Structure Coupling

Loose

Data coupling

Message coupling

No coupling

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

Choose the most accurate answer: Low coupling aids in... – ensuring that changes don’t cross boundaries of moduels

Which of the following statements about coupling are true? –

* Coupling measures the strength of connections between components.
* Loose coupling makes it unlikely that changes will be propogated across components.
* Shared variables and control information leads to tight coupling.
* Loose coupling is partly achieved through message passing.

The goal in low coupling is to ensure that changes don't cross the boundaries of modules. – TRUE

he three types of tight coupling are: Control, Common, External

Module A relies directly on local data of module B. This is an example of what type of coupling? - Tight content coupling

Modules A and B both rely on global data or a global variable. This is an example of what type of coupling? – tight common coupling

Modules rely on externally imposed format (or protocol or interface). This is an example of what type of coupling? – tight external coupling

The two types of medium coupling are: Control, data structure

Module A controls the logical flow of module B. This is an example of what type of coupling? – Medium control coupling  
Module A and B both rely on the same composite data structure. This is an example of what type of coupling? – medium data structure coupling

The three types of loose coupling are: data message none

Modules only share parameters. This is an example of what type of coupling? – loose data coupling

The loosest type of coupling; components only communicate through parameters or message passing. This is an example of what type of coupling? – loose message coupling

**Software Design: Cohesion**

Measures how well a moduel components fit together

Implements a single logical entity or function

Represents a desirable design attribute

Divides into various levels of strength

Weak

Coincidental cohesion

Temporal cohesion

Procedural cohesion

Logical association

Medium

Communicational cohesion

Sequential cohesion

Strong

Object cohesion

Functional cohesion

Cohesion and Inheritance

* Technically speaking inheriting attributes from a super class weakens cohesion
* To understand the component one must examine the super class(es) as well

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

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

Choose the most accurate answer: Cohesion describes... how well everying within a module fits together

Which of the following statements about cohesion is true? -

* Cohesion measures how well everything within a module fits together.
* Describes the implementation of a single logical entity or function.
* Represents a desirable design attitude.
* High cohesion is the aim in software development.

The goal in real-life software development is to aim for object and functional cohesion and anything less than this should be justified. – TRUE

The four types of weak cohesion are: Coincidental, temporal, procedural, logical association

Parts of modules are together in the same file. This is an example of what type of cohesion?

Coincidental cohesion

Different pieces of code are activated at the same time. This is an example of what type of cohesion? – temporal cohesion

One piece of code runs after the other. This is an example of what type of cohesion? Procedural cohesion

Components do similar but separate things. This is an example of what type of cohesion? – logical association cohesion

The two types of medium cohesion are: communicational sequential

All elements of a component operate on the same input or produce the same output. This is an example of what type of cohesion? Communicational cohesion

One component's output provides the input to another component. This is an example of what type of cohesion? Sequential cohesion

The two types of strong cohesion are: object, functional

Each operation in a module is provided to allow object attributes to be modified or inspected. This is an example of what type of cohesion? Object cohesion

Every part of a component is necessary for a single well-defined behavior. This is an example of what type of cohesion? Functional cohesion

True or False: inheritance strengthens cohesion. – FALSE

**Implementation**

Software development implementation tips

* Program when you are alert
* Write code for people not for the computer – comments should be why its done now how
* Write your comments, test and exception handling before you write functional code
* Break long methods into manageable pieces – google says no more than 40 lines
* Make each class represent single intuitive concept
* Methods focussed on a single task. Preferably without side effects
* Writing code for a second time? Extract into a method
* Optimize only when you’re sure it’s necessary- code profiler

Implementation skills are training well elsewhere

These tips will help you understand the difference between coder and developer

People, not computers are the primary focus

Let the computer handle optimization for computers

These tips make the developers using them more effective

**Deployment**

Deployment stage occurs at the end

Deployment means production deployment release to users

Occurs after testing/ QA and others have signed off on the release

**Planning**

Largely determined by project scope

Includes planned steps, problem areas and plans to recover

**Deployment Plan**

* 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 occurs in the end stage of active development.- TRUE

Product deployment should encapsulate three main focus areas. What are they? – problem areas, plan for recover in case of failure, planned steps

**Software Deployment 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 with 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 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

**Cutover Strategies**

Cold backup (cold storage) – needs to be started up 24 hours delay

Warm standby – ready to be up and running in a short amount of time

Hot failover – virtual machine all machines ready but don’t receive the info

BCP – Business continuity plan

Software cutover is a trade-off between cost and preparedness

* Speed of recover is directly proportional to cost of recover
* Hot failover allows you to redirect live data with minimal
* Warm standby has services ready for initialization
* Gold backup is a replacement machine need full setup

**Software Testing: Introduction**

Testing Definitions

* The process of executing a program (or part of a program) with the intention of finding errors( Myers vs Humphrey)
* The purpose of testing is to find erros
  + Testing is the process of trying to discover every conceivable fault or weakness in a work product (Myers via Kit)
* The process of searching for errors (Kaner)

What is a Test

Test Cases

Test Data Output - Correct Result? Expected Output

\*Software under Test \*

Oracle

Test Data and test cases

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 is a Bug?

Failure

Error

Latent

Effective

Fault

‘dependable computing and fault tolerance: concepts and terminology – Jean Clause Laprie 1985

Programmers mistake is fault, leads to error, stays latent until it becomes effective, then when the outcome is not as expected it causes failure

Axiom of Testing

Program testing can be used to show the presence of bugs but never their absence – Dikkstra 1969

When to stop testing? No right answer, testing is to prove that it is not good

Software Testing is a process to find (and fix) defects in a systems implementation

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

**Software Testing: Definitions**

**Verification**: the process of evaluation a system or component to determine whether the products satisfy the conditions imposed

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

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

Confirm that the software performs and conforms to its specification ‘Are we building the right thing’

**Validation**: The process of evaluating a system or component to determine whether it satisfies specified requirements

Checking the programme against the published user or system requirements

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

Confirm that the software performs to the users satisfaction. Assure that the software system meets the users needs ‘Are we building the right thing’

**Dynamic vs Static Verification**

**Dynamic V & V**

* Concerned with exercising and observing product behaviour
* Testing in all forms

**Static V&V**

* Concerned with analysis of the static system representations to discover problems
* Proofs and inspections – formal testing

Verification and Validation

* Building the thing right and build the right thing
* Regardless of definition an important aspect of testing
* Dynamic and static V&V aren’t exclusive
* Verification is cheaper/easier than validation

**Software Testing: Strategies**

Incremental Testing – regression testing

Top down testing – stubs are for levels not created yet, single or few lines of code that return a set value. Mock is similar but not returned value, just line works

Bottom up testing – Driver is the reverse to a stub, level above current

Back to back testing – using previous versions for testing

Test Scaffolding

Goal - setup the environment for executing test

* Driver
* Stub
* Program Unit
* Oracle

Tradeoff – Effort in testing and regression testing v effort in developing drivers/stubs

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’

Axioms of testing

* As the number of detected defects in a piece of software increases the probability of the existence of more undetected defects also increases
* Assign your best programmers to test
* Exhaustive testing is impossible
* You cannot test a program completely
* Even if you do find the last bug you’ll never know it
* It takes more time than you have to test less than you’d like
* You will run out of time before you run out of test cases

Strategies of tesing 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**

**Blackbox Testing**

* Designed without knowledge of programs 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
* Discover of defects in a system
* Assessment of whether or not the system is usable

Stages of testing

Unit Testing – single unit

Module Testing – collection of units

Sub system testing – collections of modules

System testing – testing complete system, made up of sub systems

Acceptance testing – testing by users, alpha / beta testing

Many software testing perspective exist

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

**Software Development Models**

**Predictive vs Adaptive**

Predicitive – understand what the requirements are from the outset – Design, implementation, testing – product delivered

Adaptive – Has an idea but not fully – initial version built, feedback gathered, version updated etc etc

**Incremental vs Iterative**

Incremental – understand what is required, however each part is built part at a time

Iterative – Idea is known, version is built, next version is built upon & enhanced and end may be totally different to original idea

**Incremental vs Iterative vs Both vs None**

All models are going to have characteristics, use cases, pros & cons, where it fits adaptive vs predictive through the normal flow: requirements,design,implementation,testing,deployment,maintenance

**Waterfall Model**

+ Simplest model. Very predictive. Simple and easy to understand. Efficient

<- Not fexible for change. First release takes a long time

Use for predictable repeatable work

requirements, design, Implementation, testing, deployment, maintenance

Feedback loop between next stage and previous exists should any problem occur

Later a bug is found the more costly it becomes, up x50

#1 We know requirements very well, they wont change

#2 Team has experience building similar software

#3 Translation from requirements to product is going to be perfect

**V Model**

Very similar to waterfall but as a V

Concept Operations & Maintenance

Requirements Verification & Validation

Design Integration Tests

Implementation

-------------------------------------------------------------------------------------------------------------------

Left side is project definition

Right side is project test & validation

Along the bottom is project completion time

Emphasis on validation earlier in the process

Very predictive – doesnt allow feedback or changes

+ Earlier detection of potential defects/issues

<- More upfront work

Ambiguity in requirements and early validation would be useful

**Sashimi Model**

Phases of the lifecycle are allowed to overlap

Requirements

Design

Implementation

Testing

Deployment

Maintenance

3months 6months 9 months

A phase can start before previous phase ends

Less predictive that previous model but still fairly predictive

+ Shorten the development time. People with different skill can start without waiting. Can do learning spike early

<- May result in rework

Use to shorten the time. For resource utilization

**Incremental Models**

Built in parts, multiple mini waterfalls. May overlap. Can use different model for each increment.

Predictive with some movement towards adaptive depending on model associated alongside

+ Deliver value early. Get feedback and make necessary changes between increments

<- May result in rework. May cost more

Use. If organisation may benefit from early delivery of part of product. If building one increment will help define future increments

Incremental Model Variations

Each increment follows all steps of the process

**Unified Process and its Variants**

Inception Elaboration Construction Transition

Business modelling

Requirements

Analysis and design

Implementation

Test

Deployment

Time

Each top section are broken down into iterations and there can be various iterations per phase

**Inception**

* Short phase
* What happens?
  + Establish business case
  + Feasibility
  + Build vs Buy
  + Preliminary schedule and cost

Milestone: lifecycle objective

**Elaboration**

* What happens
  + Capture requirements
  + Address known risks
  + Validate the system architecture ->
  + Executable architecture baseline ->

Milestone: lifecycle architecture

**Construction**

* Largest Phase
* What happens?
  + Software is built
  + Multiple iterations – each results in release
  + Iterative and incremental
* Milestone: lnitial operational capability

**Transition**

* What happens
  + Deployment
  + Get feedback and refine
* System conversion and user training

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

Unified process: much more adaptive than predicitive

+ Adaptive. Quality and reuse. Focus on risk mitigation increases chances of success. Flexible to incorporate other software development models

<- Complicated. Need more resources. Too much overheard for smaller project

Uses – Bigger and riskier projects. All requirements not known early in the project. Desire to deliver value earlier

Rational unified process – 9 disciplines. 6 best practices + tools

Enterprise unified process

Open UP – lighter version

Agile UP – Lighter version, Agile focussed

**Spiral Model**

Cyclic model

Four basic steps in each cycle:

* Determine objectives
* Identify risks
* Double up
* Plan for next iteration

Not every step needs to be performed

Determine Objectives

* Objectives
* Constraints
* Alternatives

Identify and Resolve Risks

* Identify Risks
* Resolve what you can

Dev and Tests

* Work done to meet objectives

Plan for next Iteration

* Review work done and commitment for next iteration

**Then starts the next cycle**

How to track process

* Life cycle objective – sufficient definition of a technical and management approach
* Life Cycle architecture – sufficient definition of the preferred approach and significant risks eliminated or mitigated
* Initial Operational Capability – sufficient preparation of the software, site, users, operators and maintainers
* Risk Drive Model
* Effort and detail driven by risk
* Process model generator. Incorporates other models

Summary

* First described 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

More adaptive focussed

+ Adaptive. Risk focus increases chances of success. Flexible for using any model. Minimizes waste. Options for go / no go

<- Complicated. Coss more to manage. Need stakeholder engagement

Use – Very large high risk projects

**Phase Gates / Stage Gates**

Requirements Design Implementation Testing Deploy

Gates or checkpoints that exist between the various phase

Gate Requirements gate Design gate Implementation gate Testing gate Deploy

At any gate if it doesn’t come up to scratch the idea gets binned

What do you check at a gate;

* Check quality of execution of previous step
* Business case still makes sense
* If yes does the action plan for the next step reasonable and sound?

Gate structure and elements

Deliverables of previous phase

Criteria: questions or metrics on which the project is judged

Results of the gate review – a decision, go yes or no. Approved action plan for the next gate. List of deliverables and date for the next gate

Sumary:

Technique to provide opportunity to reflect on curreny progress and decide if we need to change/continue or terminate. Organization customise it to their needs

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

<- Potential for structural organization to interfere with creativity and innovation

**Applying Software Development Models**

Example:

Company need to install a well known HR Management System at a big retailers HQ

Company has done the same instsall with other companys

Known problem, known solution

Waterfall model – simple

Sashimi model – perhaps, testing various parts as go along

V model – if there is any additional aspect regarding needing testing

Example 2

Very large hospital with locations all over want to automate processes

Hired a company that is expert in this area and has done similar automation but not on this scale

Hospital management want consistency across the globe but not sure about the nuances in different parts

Few of the place can benefit greatly from the immediate automation

Fairly known problem

Fairly known solution except at scale

Benefit from a phased delivery

May tweak a bit based on early iterations

Want consistency

Incremental Model

**Assignment Topic:**

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 a software development approach suited to the situation.

Here is the situation:

Zenith Healthcare is a new company in the market and has launched its product two years ago. The product is loved by clients and is growing in popularity. The level of product demand was not anticipated, and the current system architecture cannot support the rising demand. To support the anticipated demand, the company needs to re-architect the system and provide the exact same functionality. Thus, the requirements from the client perspective are very well known and do not need to change. What needs to be changed in the system to support the growing demand is also well understood. The product has 4, fairly independent components. All 4 components need to be re-architected. Out of the 4, one of them has caused the most pain and the organization could benefit greatly if that component could be replaced first with a new, highly scalable architecture. The work of migrating to a new platform is a tedious job and the deployment of a new system will involve a lot of external communication, managing customer expectations, etc.

The technical architect and one project manager will be working from the corporate headquarters in Germany, but most of the team who will be doing the coding for the migration will be offshore in Belarus. The testing team will also be in Belarus.

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.

Ans: Did the learner identify "**Known User Needs**" or "**Known Requirements**" (or something similar) as one of the characteristics and specified the correct logic?



0 pts

Didn't identify this characteristic



1 pt

Identified the characteristic but the logic/reference statement used to support the characteristic was incorrect. The correct logic/reference statement to support this characteristic is "**...with the exact same functionality. Thus, the requirements from client perspective are very well known and do not need to change**"



2 pts

Identified the characteristic and specified the correct logic

Did the learner identify "**Known Solution**" (or something similar) as one of the characteristics?



0 pts

Didn't identify this characteristic



1 pt

Identified the characteristic but the logic / reference statement to support this characteristic was incorrect. The correct logic / reference statement to support this characteristic is "**What needs to be changed in the system to support the growing demand is also well understood**"



2 pts

Identified the characteristic and specified the correct logic

Did the learner identify "**Benefit in deploying part of the product**" (or something similar) as one of the characteristics?



0 pts

Didn't identify this characteristic



1 pt

Identified the characteristic but the logic/reference statement to support this characteristic was incorrect. The correct logic/reference statement to support this characteristic is "**Out of the 4, one of them has caused the most pain and organization could benefit greatly if that component could be replaced first with a new, highly scalable architecture**."



2 pts

Identified the characteristic and specified the correct logic

Did the learner select the right model for the scenario and provide the correct logic?



0 pts

Learner selected a model that is ill-suited to this situation like the Spiral Model, the V-Model, Sashimi, or the Waterfall method



1 pt

Learner selected a model that will work but is not the preferred model (e.g. "Unified Process")



2 pts

Learner selected the right model: the Incremental Model



3 pts

Learner selected the right model and the right variation of it: the most basic incremental model -- all phases are completed in each increment. This allows us to replace the most pain-inducing component as fast as we can.



4 pts

Learner selects the right model and specifies the right logic behind the selection: "Out of the 4, one of them has caused the most pain and the organization could benefit greatly if that component could be replaced first with a new, highly scalable architecture."

What is the overall quality and detail of the response and the facts supporting the response.



0 pts

Little detail



1 pt

Enough detail



2 pts

Enough detail with additional, out-of-the-box/creative thinking

-------------------------

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" Model Selection" to get an idea. The video stays at high level, but 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.

Does the story supports the model selected by the learner?



0 pts

The story does not supports the model selected by the learner



1 pt

The story supports the model selected but story was shallow with not much details



2 pts

The story was very detailed and complete (covers all artifacts and ceremonies of the model selected)



4 pts

The story was very creative and covers things that weren't taught in the course but applicable to this scenario.

---------------

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 following videos to learn about various types of testing methods: <https://www.coursera.org/learn/software-processes/lecture/G30EZ/software-testing-perspectives>

Are the types of testing suggested by the submission appropriate for the example?



0 pts

There are no suggested types of testing.



5 pts

Some types of testing are listed, but no attempt is made to justify them, right or wrong.



6 pts

Some types of testing are listed, some are right but most are wrong. The justification does not do a good job of explaining why these types of testing are needed.

7 pts

Types of testing listed make sense for the project (with at most one exception), but the justification does not do a good job of explaining why they are necessary.



8 pts

Types of testing listed make sense for the project (with at most one exception), but the justification only does a mediocre job of explaining why they are necessary.



9 pts

Types of testing listed make sense for the project (with at most one exception), and the justification provided makes sense (with at most one exception).



10 pts

The types of testing listed are perfectly applicable to the project, and these types of testing are fully justified.

------------------

Continuing your role as a quality lead or technical lead for the project. Write a few example 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.

Do the test cases or narrative provided make sense relative to the project at hand?



0 pts

There are no test cases or narrative



5 pts

Test cases or a narrative exist, but is either not at all clear or completely misses the point of the assignment.



6 pts

Some test cases or a brief narrative appears, but only applies to the project in a tangential way (is mostly off-topic, hardly related, etc.)



7 pts

Test cases provided are not fully described (e.g. they are missing expected output); the narrative merely lists ways of testing which are generic or the definition of the type does not apply them to this specific project.



8 pts

Test cases exist but are not considered comprehensive; Narrative only applies to the project partially, or is incorrect in some major way



9 pts

Test cases exist and are nearly comprehensively descriptive; Narrative applies but has mistakes which affect applicability/understandability



10 pts

Test cases included are excellent; Narrative clearly explains the how the testing should be approached.

---------------

**Software Development Models: Agile and Lean**

Why Agile?

Verification phase a lot of errors would occure in the waterfall method. Expensive to fix as found late in the process. Errors in what delivered vs what was needed / required

Predicting requirements / needs is hard

Market shift may have changed from concept through to delivery during the time line

Necessity is the mother of invention

Crystal, SCRUM, FDD, DSDM, XP – other methodologies

Agile is 4 values, 12 principles. Defines a mindset

Whether it's scrum, kanban, xp, lean, dsdm, or another kind of agile methodology, they all rely on the 4 Agile Value Statements.

While both the items on the right and the items on the left are valued, the items on the left are valued more:

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

**Agile Manifesto and Principles**

Agile is 4 values, 12 principles

Values:

Principles

1. Our highest priority is to satisfy the customer through early and continuous devliery of valuable software
2. Welcome changing requirements even late in development. Agile processes harness change for the customers competitive advantage
3. Delivery working software frequently from a couple of weeks to a couple of months with a preference to the shorter timescale
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 tryst them to get the job done
6. The most efficient and effective method of conveying information to and within a 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 adjusts its behaviour accordingly

How does it solve problems of waterfall model?

Adaptive

* Deliver working software frequently
* Welcome chang
* Technical excellence and good designs
* Continuous improvement

Detect translation issues early. Validate user needs early. Detect integration issues early

People and Interaction

* Business and developer work together
* Face to face conversations
* Self organizing teams
* Promote sustainable development
* Motivated individuals

Detect translation issues early

What new problems does it bring?

Adaptive

* Deliver working software frequently
* Welcome change
* Technical excellence and good designs
* Continuous improvement

Architecture, design, database modelling is a challenging. Lack of control, unpredictable journey, very uncomfortable for leaders/ organizations

People and Interaction

* Business and developer work together
* Face to face conversations
* Self organizing teams
* Promote sustainable development
* Motivated individuals

Requires participation from customers throughout the development process

Practices

Principles

Values

**Agile Frameworks**

Agile is a mindset. Frameworks should be customised

Scrum – 1 to 4 week cycle – most popular framework

Kanban – continuous flow model

Scrumban – continuous flow in a cycle

XP – Extreme Programming

Scrum XP hybrid

Lean startup

Feature Driven Development – FDD

Crystal

Custom Hybrid

Dynamic Systems Development Methodology – DSDM

**Scrum**

1 to 4 week sprint

Define, design, build, test

Framework contains: Product owner, Scrum master and The team

Product owner – Inputs from executives, team, stakeholders, customer users to create what is to be done, list of what is required, definitions etc.

Product backlog – Sprint Planning meeting, work out what order from the product backlog things will be done.

Sprint backlog, what exactly needs to be done for each task, during each sprint.

Daily scrum meeting (stand-up) to review what was done yesterday and for today. Sprints are 1-4 weeks, and the sprint end date and team deliverable do not change. – Finished work.

Sprint review – full team gets together with all involved, customers, stakeholders etc to go through what was done and if any changes are needed

Sprint retrospective – how can it be better next time, what went well, badly, how it can be improved.

Burndown / Burnup charts – breaksdown what daily work is needed and to be finalised for that sprint

**Kanban**

Set or properties and principals so long as there is a continuous flow.

Backlog – Software development pipeline – end result

Backlog is a list of work to be done

Backlog, Analyze, Develop, Rest, Release – all steps are put into columns, with Doing, Done breakdown within each

From the backlog each task with the breakdown is moved from Doing to Done as it progresses.

1 – Visualize your workflow

2 – Limit WIP (work in progress) to a set number so to prevent log jam in testing for example, once work is done if there are too many items in a WIP then it should be looked as to why, code is wrong, testing taking too long etc.

3 – Manage the flow, are there issues, is the code wrong, does it require more people.

4 – Make process policies explicit – When is something done to move to testing

Kanban Principles

* Start with what you do know
* Agree to pursue incremental evolutionary change
* Respect the current process, roles, responsibilities and titles

Kanban Properties

* Visualize the workflow
* Limit WIP – work in progress
* Manage flow
* Make process policies explicit
* Improve collaboratively

**Agile and Lean Summary**

Mindset not process / model

Incremental and iterative

Very short dev cycle

Very adaptive vs predictive

+ flexible to change, increases the chances of building the right product. Speed to market

<- may result in rework. Requires close collaboration with clients and users

Use – where requirements may change. Team using unproven technology

**Lean Startup**

Unique approach, how to learn faster about market or user need. Validated learning.

Assumption Build

Learn Experiment

Measure

Metric

Cycles should be completed as quickly as possible

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

Incremental and iterative

Very short dev cycles

Very adaptive vs predictive

+ helps you learn faster and build the right project. Speed to market

<- may result in rework, requires you to experiment iteratively with clients and users

Use – doubtful business case / user need. Lots of high probability risks

**Model Selection: When to use which model:-**

Example – big retailer wants to sell online, understand they need to but not sure what to offer, need to offer something asap, ecommerce is new, company stakeholders of the company will be driving it.

Analysis and recommendation

* Company fairly confident of the idea
* Need to iterate for the following reasons
  + Speed to market – get something out quick
  + Learn about what customers like and want
  + New stuff for the company so need to learn about technology as well
* Business stakeholders are dedicated and hence have bandwidth to collaborate with dev team regularly

AGILE – allow them to be flexible, tweak plans, design – work alongside stakeholders

Lean Statup

Example

Unknown expected use

Unexpected user need

Example

Existing successful company has a new disruptive idea. Like to create a new company for it

Untested new idea

Unexpected user need

Learn Startup

**Assignment Topic:**

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.

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.