Prof Dharmaraj Kumbar MCA B.Ed

Shri SANGAMESHWA ARTS AND COMMERCE COLLEGE,CHADCHAN

Mail:raj484u@gmail.com

UNIT- 1

Principles of Testing, Software Development Life Cycle Models (SDLC), Phases of Software Project, Quality, Quality Assurance and Quality Control, Testing, Verification and Validation, Life Cycle Models, White Box Testing: White Box Testing, Static Testing, Structural Testing, Challenges in White Box Testing.

Prof Dharmaraj Kumbar MCA B.Ed -Mail:raj484u@gmail.com1

What is software

- is a collection of data or computer instructions that tell the computer how to work.-entire set of programs, procedures
- **Examples** of applications include office suites, database programs, web browsers, word processors, **software** development tools, image editors

What is software testing

- Testing is the process of checking a system or its component(s) with the intent to find whether it satisfies the specified requirements or not.
- In simple words, testing is executing a system in order to identify any gaps, errors, or missing requirements in against to the actual requirements.
- According to ANSI/IEEE 1059 standard, Testing can be defined as –
- A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item.

Who does Software Testing?

- In the IT industry, large companies have a team.
- These team have responsibilities to check the developed software in context of the given requirements.
- In most cases, the following professionals are involved in testing a system.
- Software Tester
- Software Developer
- Project Lead/Manager
- End User

Who does Software Testing?

- Different companies have different designations for people.
- These people who test the software on the basis of their experience and knowledge
- Such as Software Tester, Software Quality Assurance Engineer, QA Analyst, etc.

What is the purpose of software testing?

- A primary **purpose** of **testing** is to detect **software** failures so that defects may be discovered and corrected.
- **Testing** cannot establish that a product functions properly under all conditions.
- but only that it does not function properly under specific conditions.

When to Start Testing?

- An early start to testing reduces the cost and time to rework and produce error-free software that is delivered to the client.
- However in Software Development Life Cycle (SDLC), testing can be started from the Requirements Gathering phase and continued till the deployment of the software.
- It also depends on the development model that is being used. For example, in the Waterfall model, formal testing is conducted in the testing phase;
- Testing is done in different forms at every phase of SDLC –

Principles of Testing:

The fundamental principles of testing

- The goal of testing is to find defects before customers find them out.
- Testing applies all through the software life cycle and is not an end -of cycle activity.
- Understand the reason behind the test.
- Test the test first.
- Tests develop immunity and have to be revised constantly.
- Defects occur in convoys or clusters, and testing should focus on these convoys.
- Testing is a fine balance of defect prevention and defect detection.
- Testing requires talented, committed people who believe in themselves and work in teams.
- Intelligent and well -planned automation is key to realizing the benefits of testing.
- Program testing can only show the presence of defects, never their absence.

We will go through these Principles by taking simple story from outside the area of information technology

THE INCOMPLETE CAR

- **“Testing should focus on finding defects before customers find them.”**
- If our job is to give a complete car to the customers (and not ask the customers to paint the car) and
- if our intent is to make sure the car works as expected, without any major problems, then we should
- ensure that we catch and correct all defects in the car ourselves.
- This is the fundamental objective of testing.

Car Salesman: "The car is complete—you just need to paint it."

Sales representative / Engineer: "This car has the best possible transmission and brake, and accelerates from 0 to 80 mph in under 20 seconds!"

Customer: "Well, that may be true, but unfortunately it accelerates (even faster) when I press the brake pedal!"

- Whatever a software organization develops should meet the needs of the customers. Everything else is secondary.
- Testing means making sure that the product meets the needs of customers.

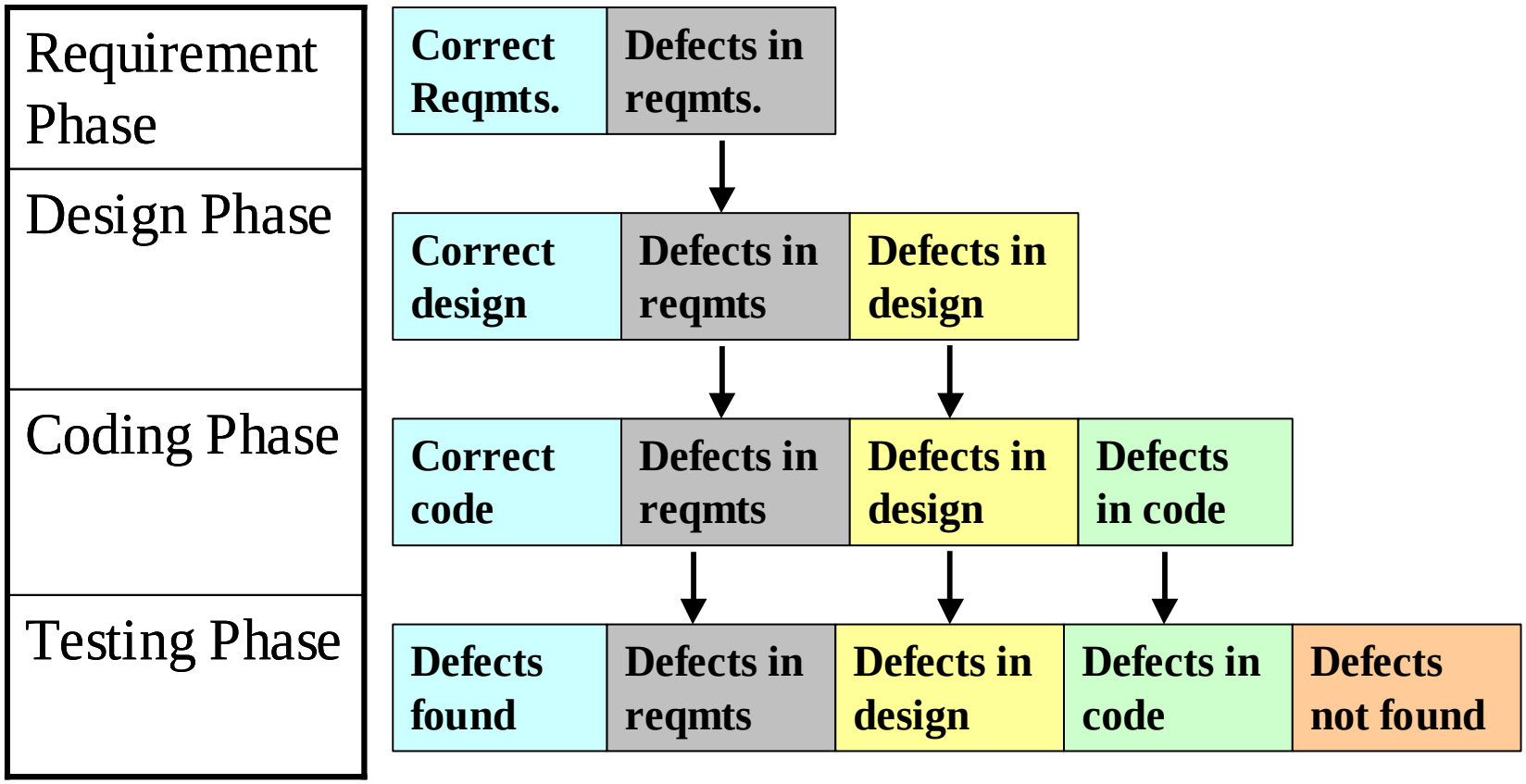
A TEST IN TIME

- “The cost of building a product and the number of defects in it increase steeply with the number of defects allowed to seep in to the later phases”

A TEST IN TIME

- The cost of building a product and the number of defects in it increase steeply .
- with the number of defects allowed to seep in to the later phases.
- Defects in a product can come from any phase.
- If a wrong or incomplete requirement forms the basis for the design and development of a product.
- Then the functionality can never realize correctly.
- Similarly, when a product design which forms the basis for the product development is faculty .
- Then code that realizes the faculty design will also not meet requirement.
- When this erroneous product reaches the customer after the testing phase.
- the customer may incur a potential downtime that can result in loss of productivity or business.

A TEST IN TIME



THE CAT AND THE SAINT

- Knowing why you are testing something is as important as knowing what you are testing.
- why one tests is as important as what to test and how to test
- Testing requires asking about and understanding what you are trying to test.
- knowing what the correct outcome is, and why you are performing any test.
- The story is “we need a cat . only when we get a cat, can we tie it to pillar and only after that can the saint start meditating.

THE CAT AND THE SAINT

- If you carry out tests without understanding why we are running them.
- We will end up in running inappropriate tests that do not address what the product should do.
- Understanding why we are testing certain functionality leads to different types of testing like
- White box–to check various path in the code.
- Black box –to check external functionality.
- Integration test-to check different components fit together
- Regression test-to check that changes work as designed and do not have any side-effects.

TEST THE TESTS FIRST

- A defective test is more dangerous than a defective product.
- From above example it is clear that the audiologist who have a hearing problem , not the patient
- We cannot assume that the tests will be perfect either.
- it is important to make sure that the tests themselves are not faculty before we start using them.

TEST THE TESTS FIRST

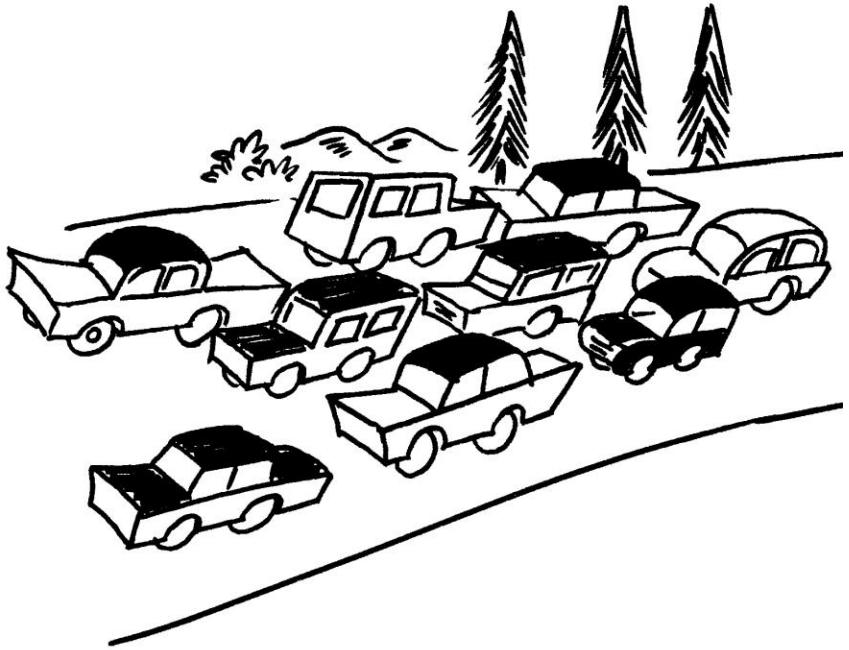
- A test should document the
  - Input data
  - Expected results
  - Test process
- The behaviour has to be externally corroborated.
  - No Turing machine can verify itself!

THE CONVOY (group or line)AND THE RAGS(a waste piece of cloth):

- Testing can only find a part of defects that exist in a cluster;
- fixing a defect may introduce another defect to the cluster
- Defects come in convoys ; Fixing a defect in the convoy is likely to add more defects to it.
- Defects in a program also typically display this convoy phenomenon, they occur in clusters.
- Fixing a tear in one place in a shirt would most likely cause damage in another place.
- A fix from one defect generally introduces some instability and necessitates another fix.
- All these fixes produce side effects that eventually cause the convoy of defects in certain parts of the product

THE CONVOY AND THE RAGS

- We have seen traffic congestions , during these congestions we have seen convoy effect.
- There may be heavy congestions with vehicles looking like they are going in convoy.
- Until encounter next convoy.
- Look for side effects.
- Look for code or “rags.”
- Tie maintenance and testing functions closely.



THE POLICEMEN ON THE BRIDGE:

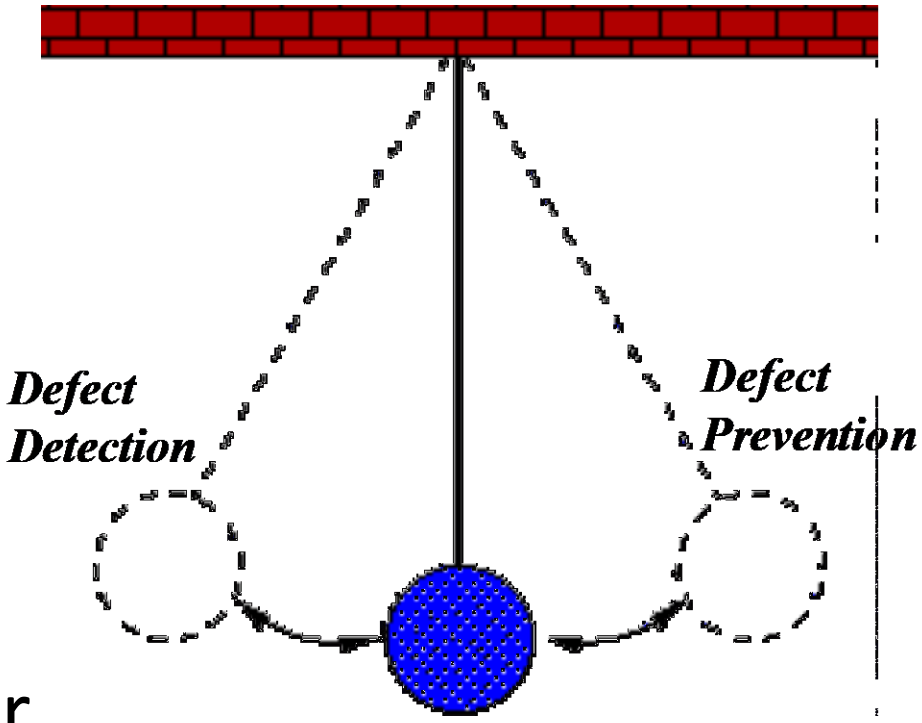
- Prevention is better than cure-you may be able to expand your horizon much farther.

THE POLICEMEN ON THE BRIDGE

- Testers are probably best equipped to know the problems customers may encounter.
- Like the second police officer, they know people fall and they know why people fall. Rather than simply catch people who fall.
- Also they look at the root cause for falling and advise preventive action.
- Testers would work with the development engineers to make sure the root cause of the defects are addressed.
- Defect prevention is a part of a testers job.

Ends of a Pendulum

- Achieve a balance between defect prevention (quality assurance) and defect detection (quality control).
- Distribute QA / QC functions throughout for early detection



The Pesticide Paradox

A substance used for destroying insects

- Bugs are like bacteria – they develop resistance to antibiotics; new antibiotics have to be developed!
- We need to redefine and refine tests as we move forward.
- As we “pass” old tests, new defects will surface.
- Regression tests have to be “retired,” and new regression tests have to be designed

MEN IN BLACK

- Ride in test will take care of the rest.
- People in testing profession should have a customer focus.
- Understanding the implications from customers perspective.
- They should think ahead in terms of defects prevention and yet be able to spot and rectify errors that crop up.
- Technical and inter personal skills required.
- The team was given an identify by black dress and increased their pride in work always.
- Instilling pride in testers
- Establishing an identity for testers
- Showing them a career path

AUTOMATION SYNDROME

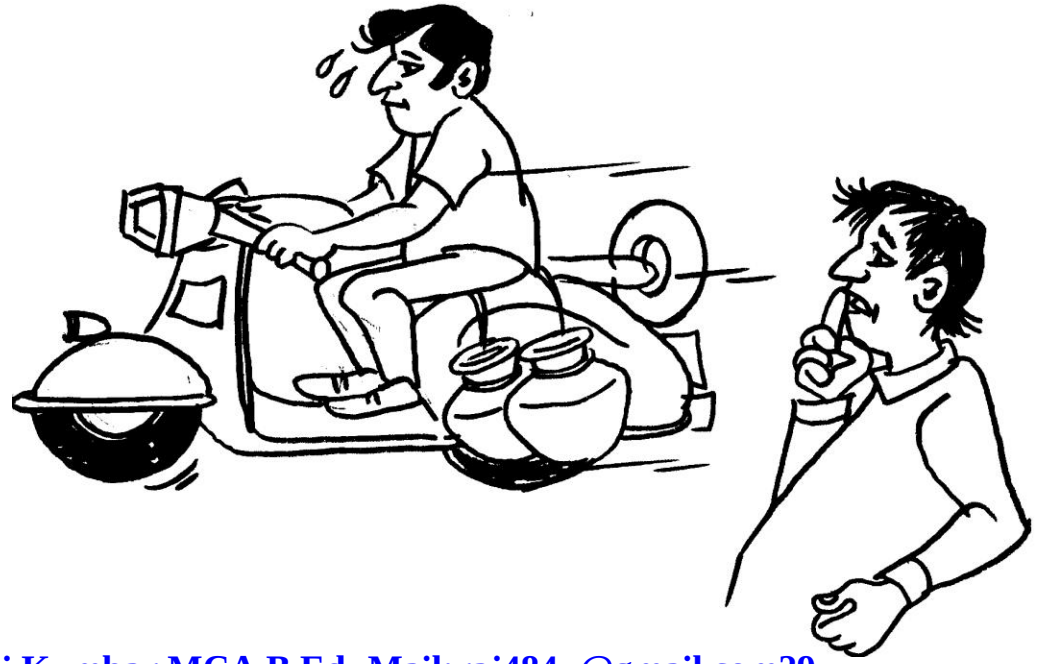
- “Failures outnumber successes in automation, equal skills and focus are needed for automation as in product development”
- A farmer had to use water from a well which was located more than a mile away.
- Therefore, he employed 100 people to draw water from the well and water his fields.
- Each of those employed brought a pot of water a day but this was not sufficient, the crops failed.
- Then next crop cycle, the farmer remembered the failures of previous season.
- He thought about automation as a viable way to increase productivity and avoid such failures.
- So, he had heard about motorcycles as faster means of commuting therefore he got 50 motorcycles.
- laid off 50 of his workers and ask each rider to get two pots of water.

AUTOMATION SYNDROME

- He chooses to use motorcycles just before his crop cycle started.
- Hence for first few weeks, the workers were kept busy learning to use the motorcycle,
- In the process of learning to balance the motorcycles, the number of pots of water they fetch fell.
- And number of workers was also lower, the productivity actually dropped.
- The crops failed again.
- The next crop cycle came,
- The farmer bought a truck this time to fetch water.
- This time he realized the need for training and got his worker to learn driving.

AUTOMATION SYNDROME

- However, the road leading to the farm from the well was narrow and the truck did not help in bringing in the water.
- no portion of the crop could be saved this time also.
- After these experiences the farmer said, “My life was better without automation!”



AUTOMATION SYNDROME

- If you go through the story closely there appear to be several reasons for the crop failures.
- That are not to do with the automation intent at all.
- It is not automation but on the process followed for automation and the inappropriate choices made.
- In the second crop cycle, the reason for failure was lack of skills and in the third cycle it is due to improper tool implementation.
- In the first crop cycle, the farmer laid off his workers immediately after the purchase of motorcycles and expected cost and time to come down.
- He repeated the same mistake for the third crop cycle. Automation does not yield results immediately.

AUTOMATION SYNDROME

- The moral of the above story as it applies to testing is that automation requires careful planning, evaluation, and training.
- Automation may not produce immediate returns.
- An organization that expects immediate returns from automation may end up being disappointed and wrongly blame automation for their failures.

- A large number of organizations fail in their automation initiatives and revert to manual testing. Unfortunately, they conclude—wrongly—that automation will never work.

Software Development Life Cycle Models (SDLC),

- What is SDLC?
- Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality softwares.

- It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software.
- The life cycle defines a methodology for improving the quality of software and the overall development process.

Prof Dharmaraj Kumbar MCA B.Ed -Mail:raj484u@gmail.com32

The following figure is a graphical representation of the various stages of a typical SDLC.

Phases of Software Project

- A software project is made up of a series of phases.
- Broadly, most software projects comprise the following phases.
- Requirements gathering and analysis
- Planning
- Design
- Development or coding
- Testing
- Deployment and maintenance

Requirements Gathering and Analysis:

- Specific requirements of the software to be built are gathered and-documented.
- The requirements get documented in the form of a System Requirements Specification (SRS) document.
- This document acts as a bridge between the customer and the designers chartered to build the product.
- If the software is bespoke software, then there is a single customer who can give these requirements.
- If the product is a general-purpose software, then a product marketing team with inthe software product organization specifies the requirements by aggregating the requirements of multiple potential customers.

Planning

- The purpose of the planning phase is to come up with a schedule the scope, and resource requirements for a release.
- What will be met and what will not be met—for the current release to decide on the scope for the project.
- look at resource availability, and to come out with set of milestones and release date for the project.
- The planning phase is applicable for both development and testing activities.
- At the end of this phase, both project plan and test plan of documents are delivered.

Design

- The design phase produces a representation that will be used by the following phase, the development phase.
- This representation should serve two purposes.
- First, from this representation, it should be possible to verify that all the requirements are satisfied.
- Second, this representation should give sufficient information for the development phase to proceed with the coding and implementation of the system.
- Design is usually split into two levels—high-level design and low-level or a detailed design.
- The design step produces the system design description (SDD) document that will be used by development teams to produce the programs that realize the design

Development or Coding

- Design acts as a blueprint for the actual coding in proceed.
- This development or coding phase comprises coding the programs in the chosen programming language.
- It produces the software that meets the requirements the design was meant to satisfy.
- In addition to programming, this phase also involves the creation of product documentation

Testing

- As the programs are coded (in the chosen programming language), they are also tested.
- In addition, after the coding is (deemed) complete, the product is subjected to testing.
- Testing is the process of exercising the software product in pre-defined ways to check if the behavior is the same as expected behavior.
- By testing the product, an organization identifies and removes as many defects as possible before shipping it out.

Deployment and Maintenance:

- Once a product is tested, it is given to the customers who deploy it in their environments.
- As the users start using the product in their environments.
- They may observe difference between the actual behavior of the product and what they were given to expect.
- Such difference could end up as product defects, which need to be corrected.
- Where in the product is maintained or changed to satisfy the changes that arise from customer expectations, environmental changes etc.
- Maintenance is made up of corrective maintenance(for example, fixing customer-reported problems),.
- adaptive maintenance(for example, making the software run on a new version of an operating system or database),.
- And preventive maintenance(for example, changing the application program code to avoid a potential security hole in an operating system code

Quality

- quality is meeting the requirements expected of the software.
- quality: The degree to which a component, system or process meets specified requirements and/or user/customer needs and expectations.
- software quality: The totality of functionality and features of a software product.
- that bear on its ability to satisfy stated or implied needs.

Quality Control

- Quality control attempts to build a product.
- test it for expected behavior after it is built.
- and if the expected behavior is not the same as the actual behavior of the product.
- fixes the product as is necessary and rebuilds the product.
- This iteration is repeated till the expected behavior of the product matches the actual behavior for the scenarios tested.
- Thus, quality control is defect-detection and defect-correction oriented.
- And works on the product rather than on the process.

Software Quality Control (SQC)

- SQC is a set of activities like

Reviews

- Requirement Review
- Design Review
- Code Review
- Deployment Plan Review
- Test Plan Review
- Test Cases Review

Testing

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing

Quality Assurance

- Quality assurance, on the other hand, attempts defect prevention.
- It concentrating on the process of producing the product .
- It is not working on defect detection/correction after the product is built.
- A quality assurance approach would be to first review the design. THEN
- The product is built and correct the design errors.
- Similarly, to ensure the production of a better code, a quality assurance process may mandate
- coding standard to be followed by all programmers.
- quality assurance normally tends to apply to all the products that use a process. Also, since quality
- assurance continues throughout the life of the product it is everybody's responsibility.

Software Quality Assurance (SQA)

- Is a set of activities for ensuring quality in software engineering processes.

It includes the following activities:

- Process definition and implementation
- Auditing
- Training

Processes could be:

- Software Development Methodology
- Project Management
- Configuration Management
- Requirements Development/Management
- Estimation
- Software Design
- Testing
- *etc*

Difference between quality assurance and quality control.

Quality Assurance	Quality Control
Concentrates on the process of producing the product's	Concentrates on specific products
Defect-prevention oriented	Defect-detection and correction oriented
Usually done throughout the lifecycle	Usually done after the product is built
This is usually a staff function	This is usually a line function
Examples: reviews and audits	Examples, software testing at various levels

Testing

- Software testing is a process of executing a program or application with the intent of finding the software bugs.
- It can also be stated as the process of validating and verifying that a software program or application or product.
- Meets the business and technical requirements that guided its design and development
- Works as expected
- take place throughout the Software (SDLC).
- Timely testing increases the chances of a product or service meeting the customer's requirements.
- Testing is done by a set of people within a software product

Verification

- Verification is the process of evaluating a system or component.
- It determine whether the products of a given phase satisfy the conditions imposed at the start of that phase.
- Verification, takes care of activities to focus on the question *"Are we building the product, right?“*
- Requirements review, design review, and code review are some examples of verification activities.

Validation

- We can define validation as the process of evaluating software during or at the end of the development process after verification.
- It determine whether it satisfies specified requirements.
- validation takes care of a set of activities to address the question *"Are we building the right product?“*
- Validation include unit testing performed to verify if the code logic works.
- Integration testing performed to verify the design, and system testing performed to verify that the requirements are met.

- Quality assurance = verification and Quality Control =validation or testing
- Verification and quality assurance to be one and the same.
- Quality control, validation, testing mean the same

PROCESS MODEL TO REPRESENT DIFFERENT PHASES

Entry Task verification eXit or ETVX model

- A process model is a way to represent any given phase of software development.
- It builds on verification and validation .
- Helps to prevent and minimize the delay between defect injection and defect detection.
- In this model each phase of software project is characterized as

- Entry criteria- which specify when that phase can be started.
- Tasks- or steps that need to be carried out in that phase.
- Verification - which specifies methods of checking that the tasks have been carried out correctly.

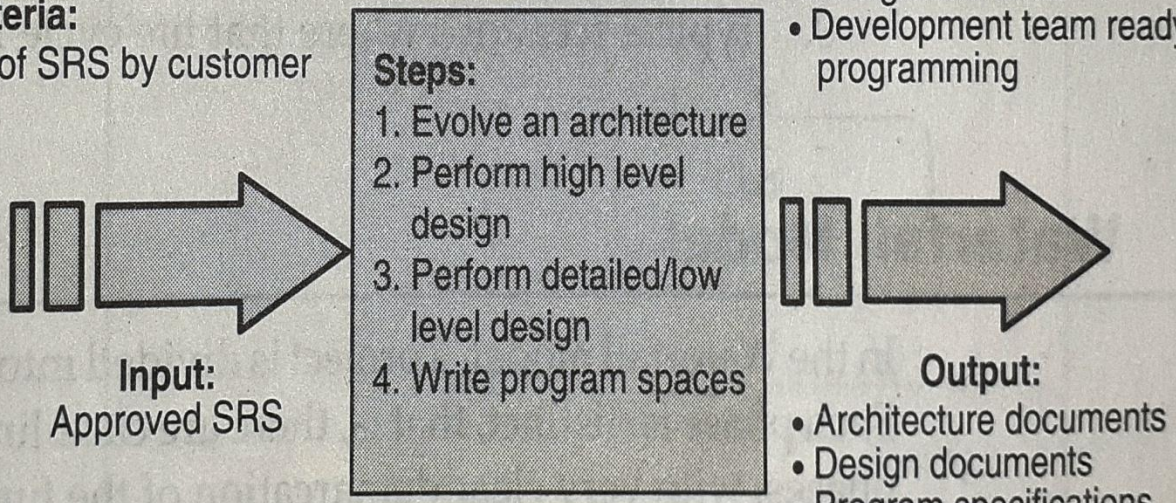
- Exit criteria - conditions under which one can consider the phase as done ,specified requirement are completed

- This model know as Entry Task verification eXit or ETVX model

An example of applying the ETVX model to the design phase is presented in Figure 2.1.

Figure 2.1

ETVX model applied to design.



Life Cycle Models

Life Cycle Models

Prof Dharmaraj Kumbar MCA B.Ed -Mail:raj484u@gmail.com54

Life Cycle Models

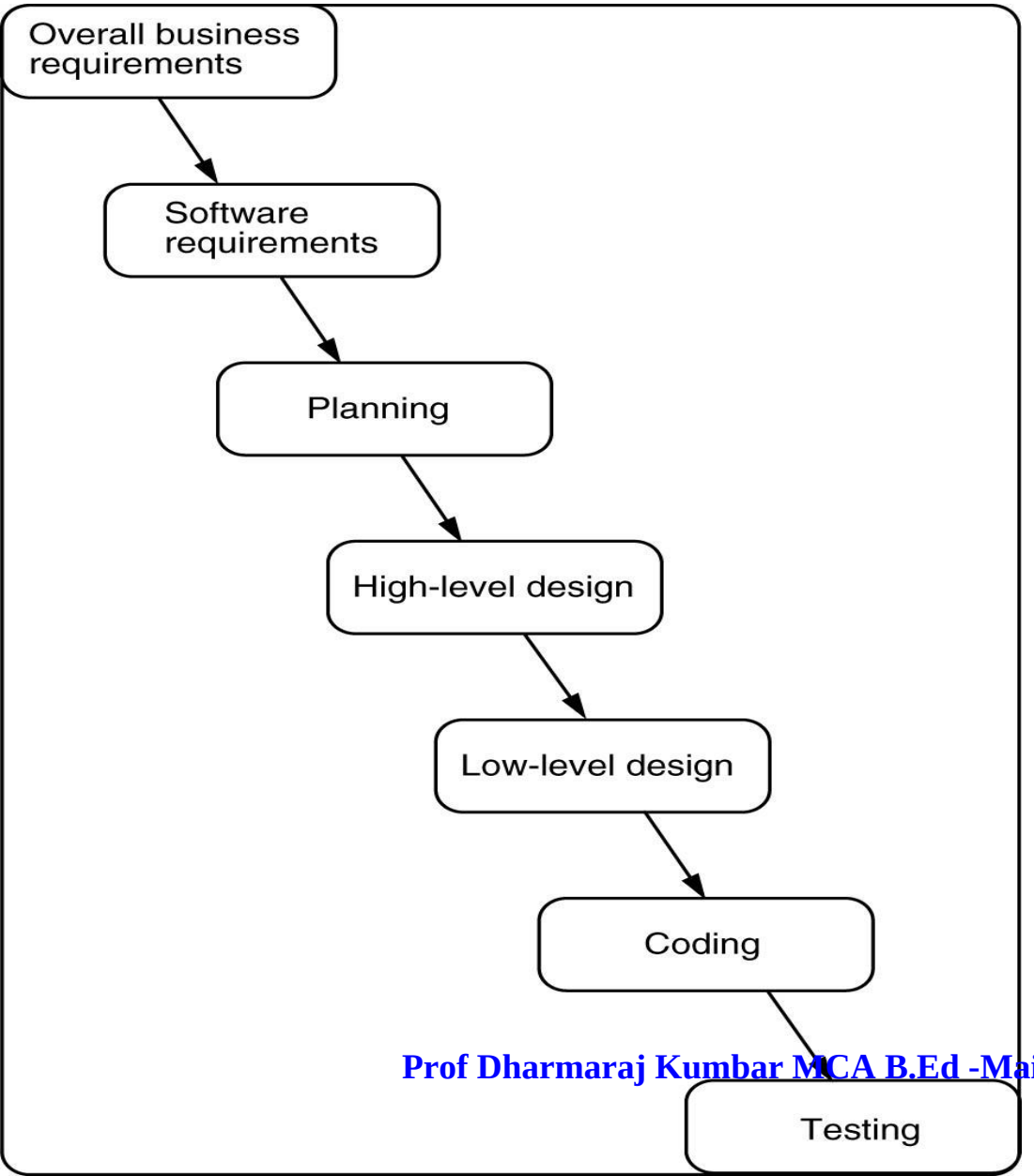
- A Life Cycle model describes.
- How the phases combine together to form a complete project or life cycle.
- It has following attributes.
- **The activities performed-** technical activities and non-technical.
- **The deliverables from each activity-** For example.
- The requirements gathering phase..
- produces the SRS document
- The design phase produces the SDD document.
- Each activity produces a set of deliverables

Life Cycle Models

- **Methods of validation of the deliverables-** it is necessary to have proper validation criteria for each output.
- **The sequence of activities-**
- For example, the process of requirements gathering may involve steps such as..
- Interviews with customers, documentation of requirements and freezing of requirements.

- **Methods of verification of each activity, including the mechanism of communication amongst the activities –**
- The different activities interact with one another by means of communication methods.
- For example, when a defect is found in one activity and is traced back to the causes in an earlier activity.
- proper verification methods are needed to retrace
- steps from the point of defect to the cause of the defect.

Water Fall Model



Waterfall Model

- In the Waterfall model, a project is divided into a set of activities.
- A project starts with an initial phase,
- Upon completion of the phase, moves on to the next phase.
- The project moves to the subsequent phase and so on.
- Thus the phases are strictly time sequenced.

- As shown in Figure- The project goes through a phase of requirements gathering.
- At the end of requirements gathering.
- A System Requirements Specification document is produced.
- This becomes the input to the design phase.

- During the design phase.
- A detailed design is produced in the form of a System Design Description.

- With the SDD as inputs the project proceeds to the development or coding phase.
- Where in programmers develop the programs required to satisfy the design.

- Once the programmers complete their coding tasks.
- They hand the product to the testing team.
- Testing team test the product before it is released.

- If there is no problem in a given phase.
- Then this method can work, going in one direction (like a waterfall).
- This model is very useful when a project can actually be divided into watertight section.
- Major drawback - delay in feedback thus lead to ineffectiveness of verification and validation activities.
- An error in one phase is not detected till at least the next phase

Advantages of waterfall model

- This model is simple and easy to understand and use.
- In this model phases/activities are processed and completed one at a time. Phases do not overlap.

- It is easy to manage due to the easily adjusting of the model
- each phase has specific deliverables and a review process.

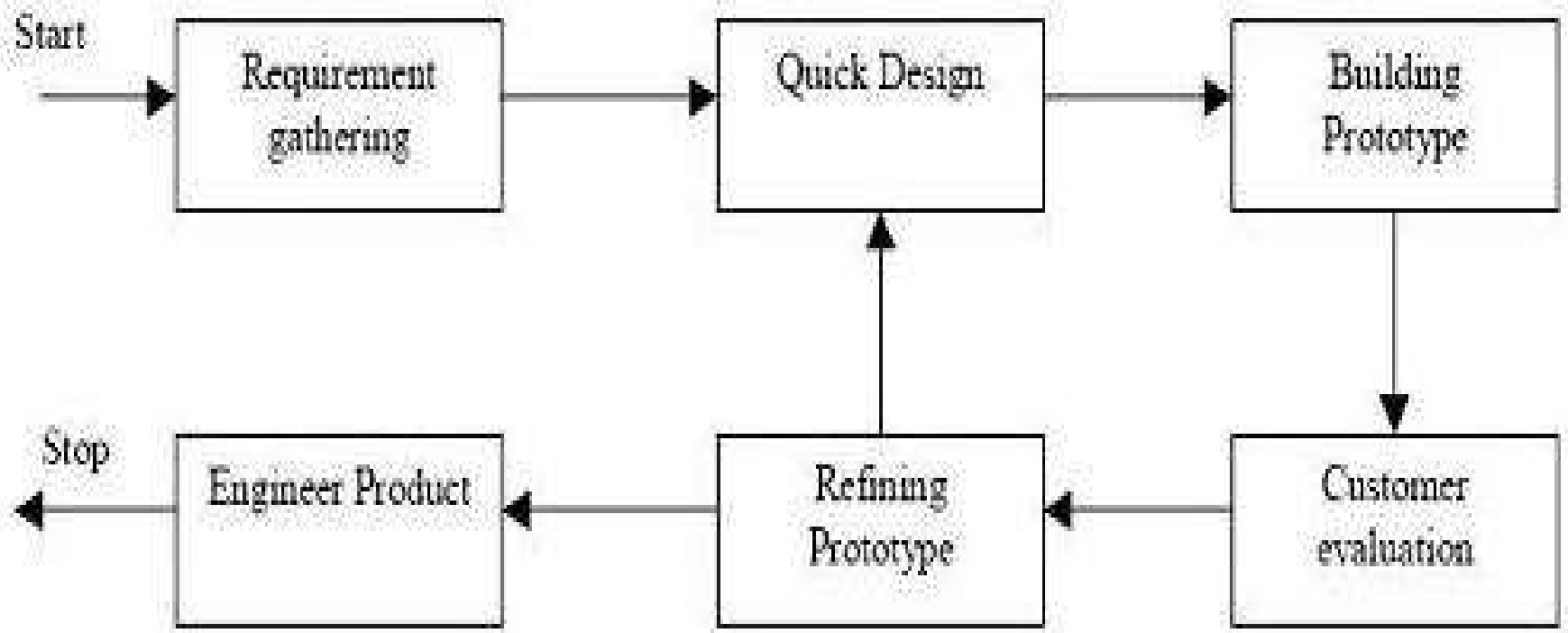
Disadvantages of waterfall model:

- It is very difficult to go back and change something that was not implementation stage.
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.

Prototyping model

- A prototyping model uses constant user interaction. Early in the requirements gathering stage, to produce a prototype.
- The software development team interacts with customers to understand their requirements.
- Then Produces a prototype to show how the software system would look like.
- This prototype would have the models of how the input screens and output reports would look like.
- Also include workflow and processing logic.

- The customer and the development team review the prototype frequently.
- Customer's feedback is taken very early in the cycle (that is, during the requirements gathering phase).
- Based on the feedback and the prototype that is produced.
- the software development team produces the System Requirements Specification document.
- Once the SRS document is produced, the prototype can be discarded.
- The SRS document is used as the basis for further design and development.



Prototyping Model

Prof Dharmaraj Kumbar MCA B.Ed -Mail:raj484u@gmail.com66

- Users are actively involved in the development
- Methodology of working model is provided so users gets better understand of a software.
- Errors can be detected much earlier.
- Quicker user feedback is available leading to better solutions.
- Missing functionality can be identified easily

Disadvantages of Prototype model

- Leads to implementing and then repairing way of building systems.
- May increase the complexity of the system.
- As scope of the system may expand beyond original plans.
- Incomplete application always we cannot use full system difficult in problem analysis.

Advantages of Prototype model

- Users are actively involved in the development.
- Methodology of working model is provided so users gets better understand of a software.
- Errors can be detected much earlier.
- Quicker user feedback is available leading to better solutions.
- Missing functionality can be identified easily

Disadvantages of Prototype model

- Leads to implementing and then repairing way of building systems.
- May increase the complexity of the system.
- As scope of the system may expand beyond original plans.
- Incomplete application always we cannot use full system difficult in problem analysis.

Rapid Application Development Model

- The Rapid Application Development model is a similarity of the Prototyping Model.
- It differs from the RAD Model on two counts.
- First, in the RAD Model, it is not a prototype that is built but the actual product itself.
- That is the built application is not discarded.
- Hence, it is named Rapid Application Development model
- Second, A computer Aided Software Engineering (CASE) tool is used throughout the life cycle, right from requirements gathering.

RAD Model

- A CASE tool can provide inbuilt means of verification and validation.
- For example, the tool may be able to automatically detect and resolve conflicts in data types or dependencies.
- In RAD model
- the components or functions are developed in parallel as if they were mini projects and then assembled.
- The phases in the rapid application development (RAD) model are:
- **Business modeling:**
- **Data modeling:**
- **Process modeling:**
- **Application generation:**
- **Testing and turnover:**

RAD Model

- **Business modeling:** The information flow is identified between various business functions.
- **Data modeling:** Information gathered from business modeling is used to define data objects that are needed for the business.
- **Process modeling:** Data objects defined in data modeling are converted to CRUD of data objects.
- **Application generation:** Automated tools are used to convert process models into code and the actual system.
- **Testing and turnover:** Test new components and all the interfaces.

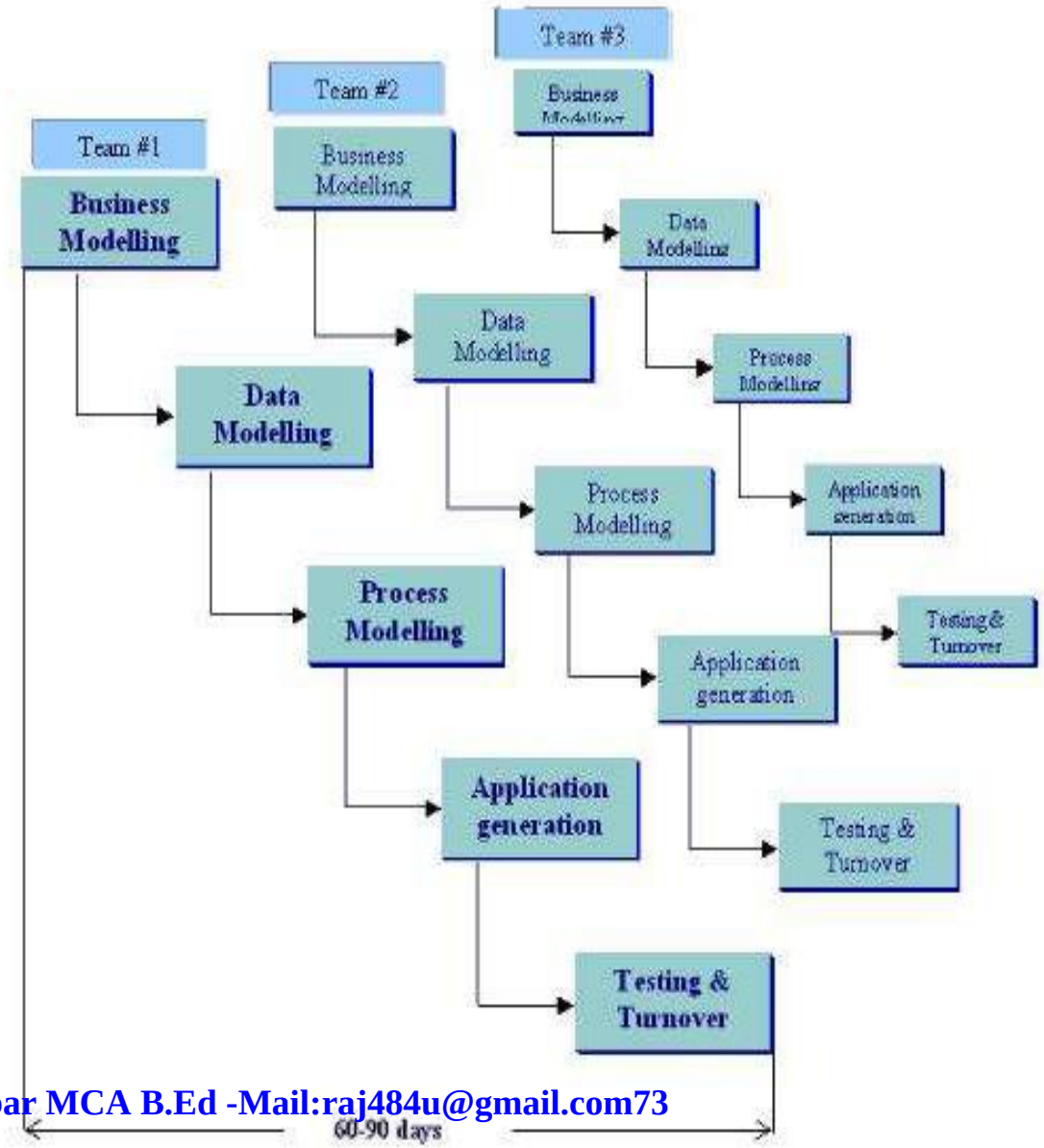


Figure 1.5 – RAD Model

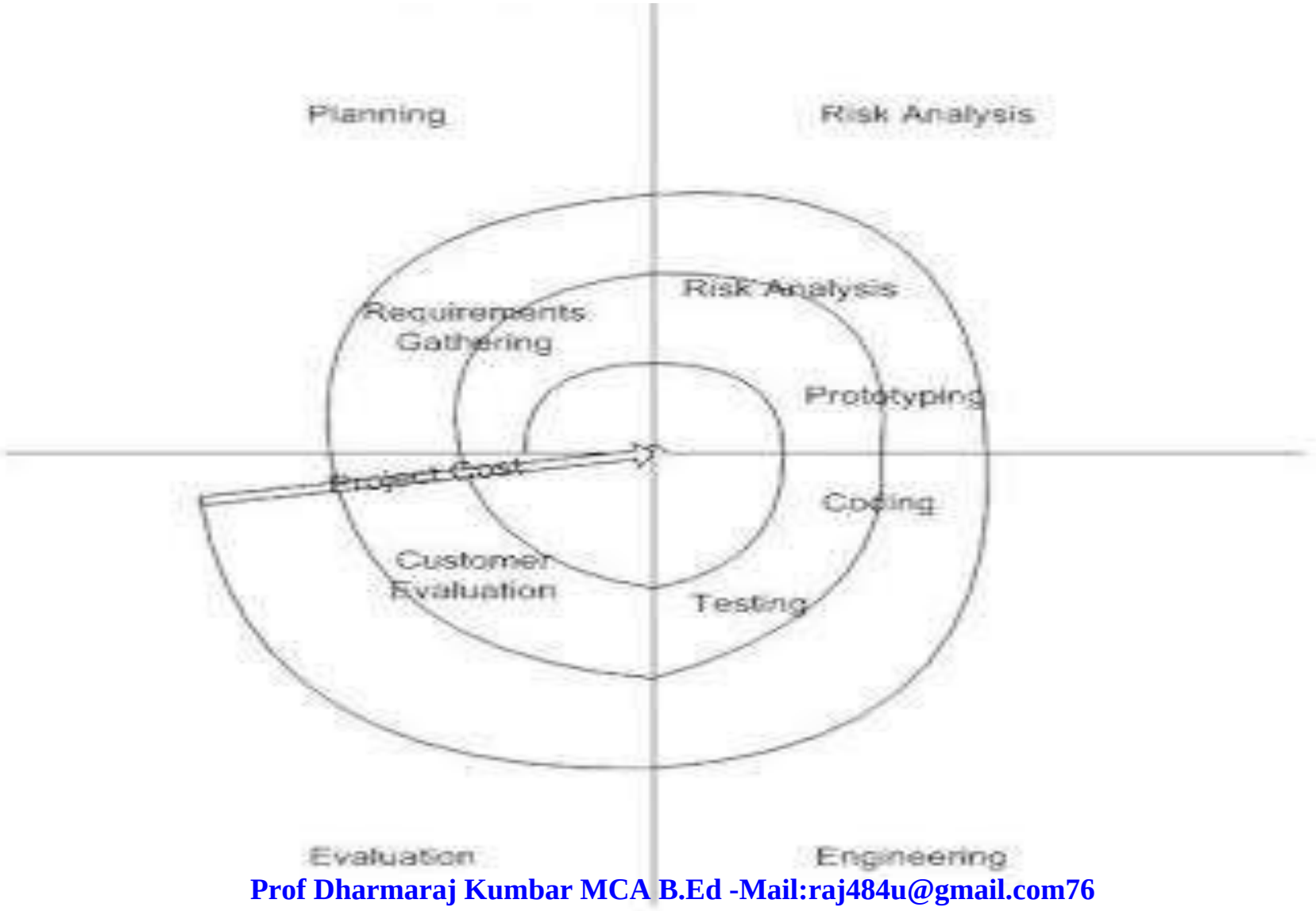
Advantages of the RAD model

- Reduced development time.
- Increases reusability of components.
- Quick initial reviews occur.
- Encourages customer feedback.
- Integration from very beginning solves a lot of integration issues.

Disadvantages of RAD model:

- Depends on strong team and individual performances for identifying business requirements.
- Only system that can be modularized can be built using RAD.
- Requires highly skilled developers/designers.
- High dependency on modeling skills.
- Inapplicable to cheaper projects as cost of modeling and automated code generation is very high.

Spiral or Iterative Model:



- **Planning Phase**: Requirements are gathered during the planning phase. Requirements like BRS and SRS.
- **Risk Analysis**: In the risk analysis phase, a process is undertaken to identify risk and alternate solutions.
- A prototype is produced at the end of the risk analysis phase.
- If any risk is found during the risk analysis then alternate solutions are suggested and implemented.
- **Engineering Phase**: In this phase software is developed, along with testing at the end of the phase.
- Hence in this phase the development and testing is done.
- **Evaluation phase:** This phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.

Activities which are performed in the spiral model are shown below:

Phase Name	Activities performed	Deliverables / Output
Planning	-Requirements are studied and gathered. - Feasibility study - Reviews and walkthroughs to streamline the requirements	Requirements understanding document Finalized list of requirements.
Risk Analysis	Requirements are studied and brain storming sessions are done to identify the potential risks Once the risks are identified, risk mitigation strategy is planned and finalized	Document which highlights all the risks and its mitigation plans.
Engineering	Actual development and testing if the software takes place in this phase	Code Test cases and test results Test summary report and defect report.
Evaluation	Customers evaluate the software and provide their feedback and approval	Features implemented document

Advantages of Spiral model:

- High amount of risk analysis hence, we can avoid risk.
- Good for large and mission-critical projects.
- Strong approval and documentation control.
- Additional Functionality can be added at a later date.
- Software is produced early in the software life cycle.

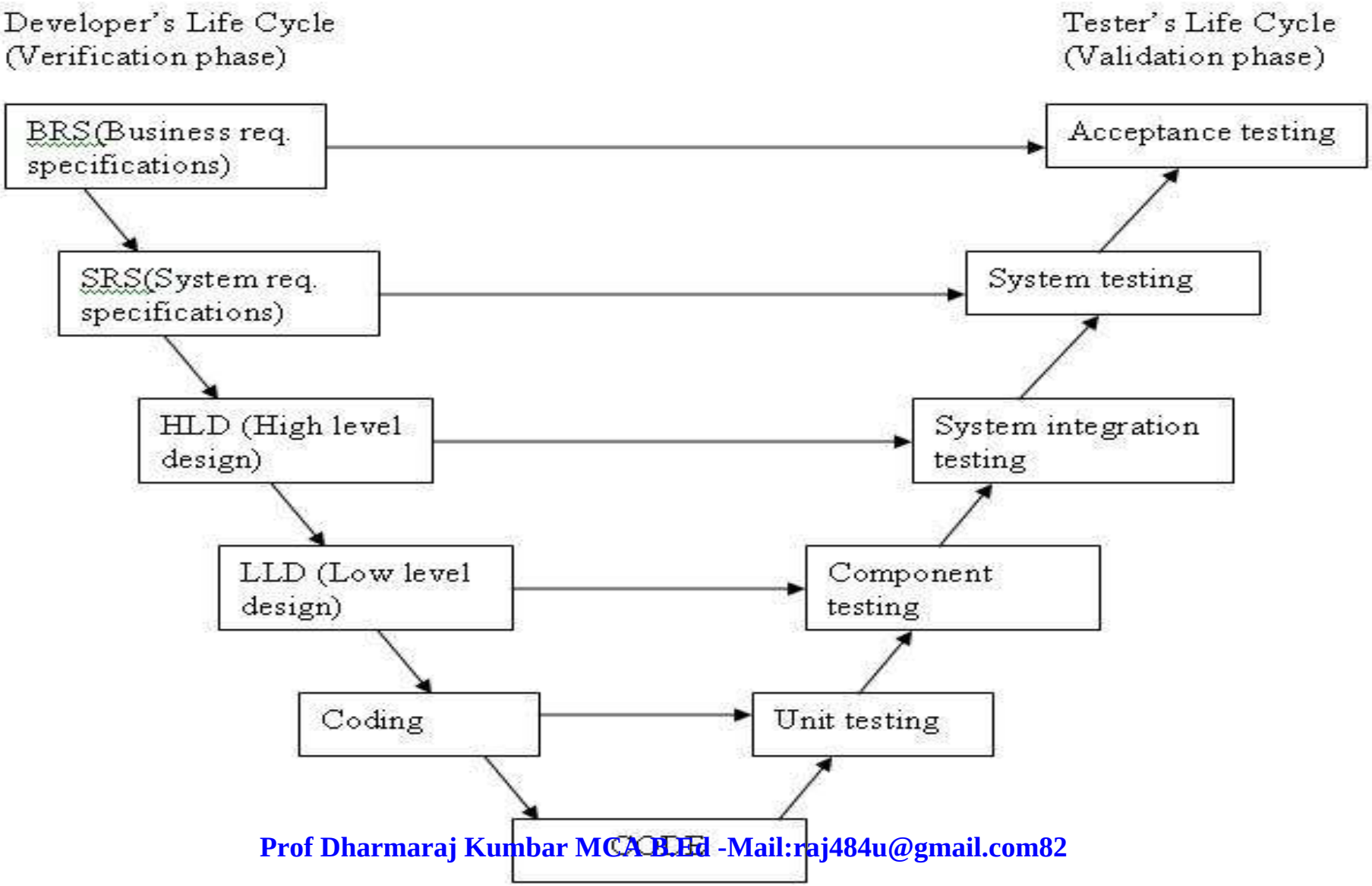
Disadvantages of Spiral model

- Can be a costly model to use.
- Risk analysis requires highly specific expertise.
- Project's success is highly dependent on the risk analysis phase.
- Does not work well for smaller projects.

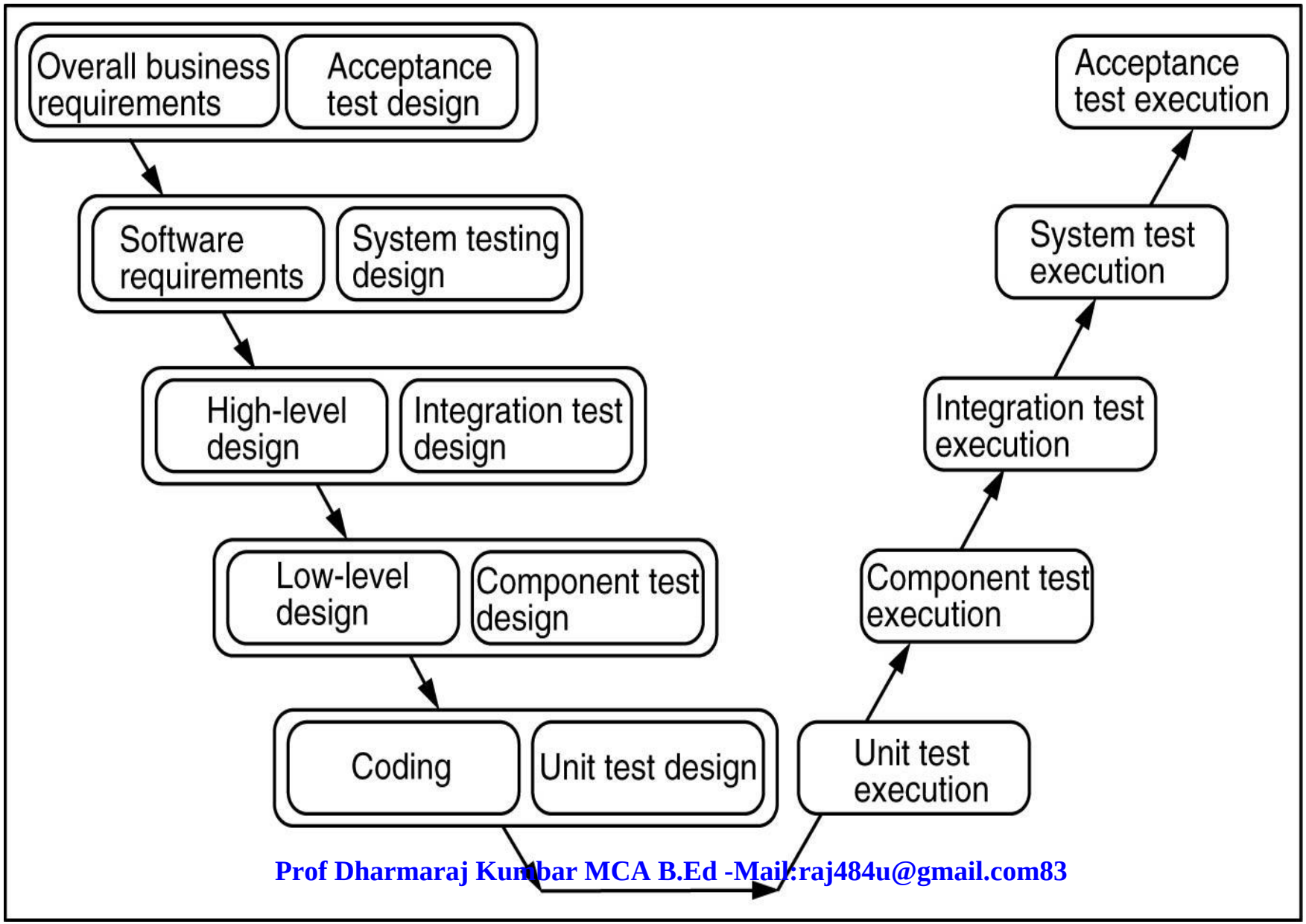
The V-Model:

- The V-Model splits testing in to two parts design and execution.
- Test design is done early, while test execution is done in the end.
- There are different types of tests for each phase of life cycle.
- V- model means Verification and Validation model.
- Each phase must be completed before the next phase begins.
- Testing of the product is planned in parallel with a corresponding phase of development in V-model.

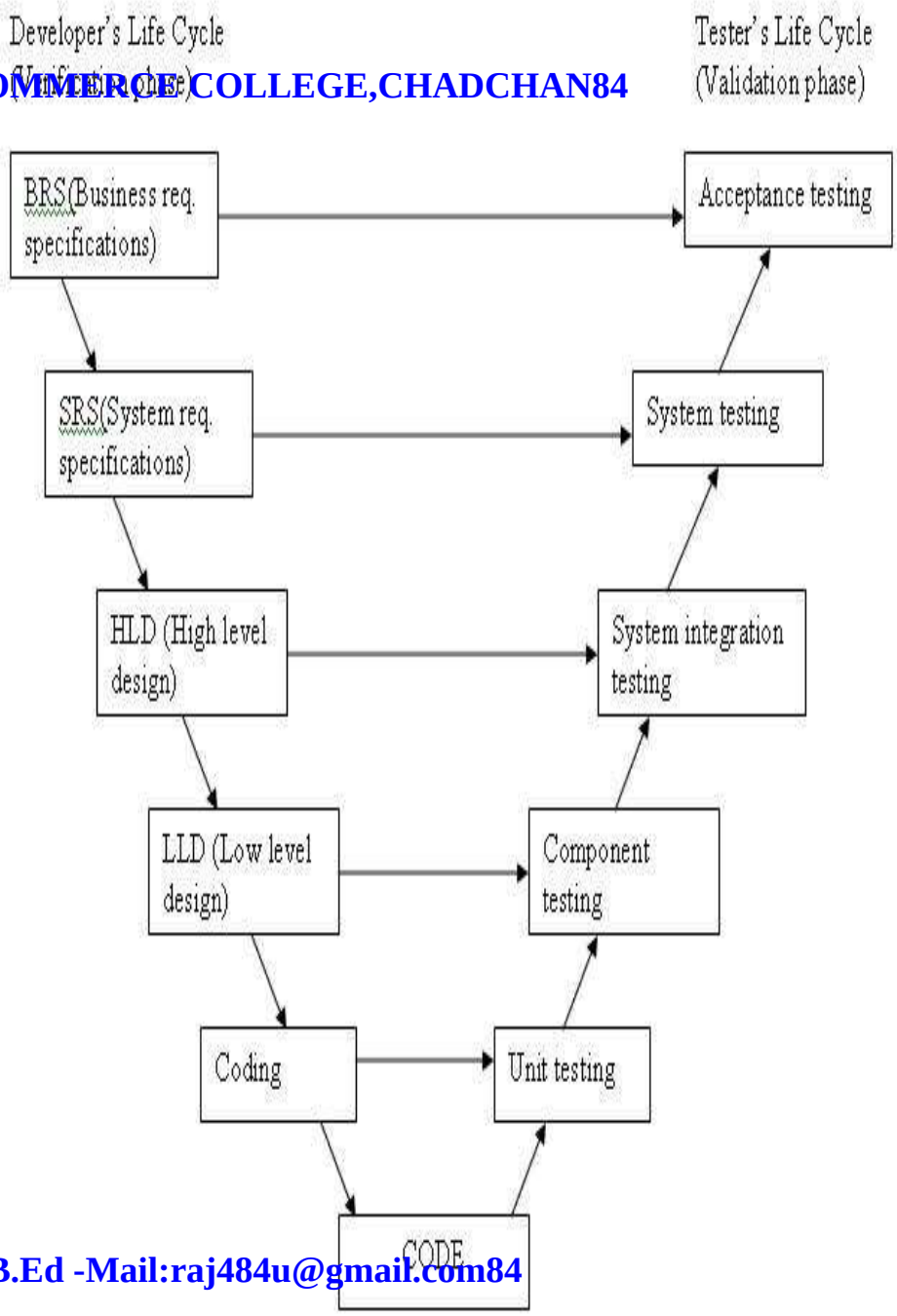
V-Model


Developer's Life Cycle
(Verification phase)
Tester's Life Cycle
(Validation phase)
BRS(Business req. specifications)
Acceptance testing
SRS(System req. specifications)
System testing
HLD (High level design)
System integration testing
LLD (Low level design)
Component testing
Coding
Unit testing
CODE

V-Model


Overall business requirements
Acceptance test design
Software requirements
System testing design
High-level design
Integration test design
Low-level design
Component test design
Coding
Unit test design
Unit test execution
Component test execution
Integration test execution
System test execution
Acceptance test execution

- **Requirements:** Started with BRS and SRS but before development is started, a system test plan is created.
- The test plan contain functionality specified in the requirements gathering.
- **The high-level design (HLD):** It focus in system design.
- It provide overview of solution, platform, system, product and service/process.
- An integration test plan is created in this phase.
- In order to test the pieces of the software systems ability to work together.
- **The low-level design (LLD):** here actual software components are designed.
- LLD defines the actual logic for each and every component of the system.



V-Model

- For example ,Class diagram with all the methods and relation between classes comes under LLD.
- Component tests are created in this phase as well.
- **The implementation:** Here all coding takes place.
- Once coding is complete, the path of execution continues up the right side of the V.
- Where the test plans developed earlier are now put to use.
- **Coding:** This is at the bottom of the V-Shape model.
- Module design is converted into code by developers.
- Unit Testing is performed by the developers on the code written by them.

Advantages of V-model:

- Simple and easy to use.
- Testing activities like planning, test designing happens well before coding.
- This saves a lot of time. higher chance of success over the waterfall model.
- Defects are found at early stage and track defects.
- Avoids the downward flow of the defects.
- Works well for small projects where requirements are easily understood.

Disadvantages of V-model:

- High risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.
- Once an application is in the testing stage, it is difficult to go back and change a functionality.
- No working software is produced until late during the life cycle.

V-Model – Application

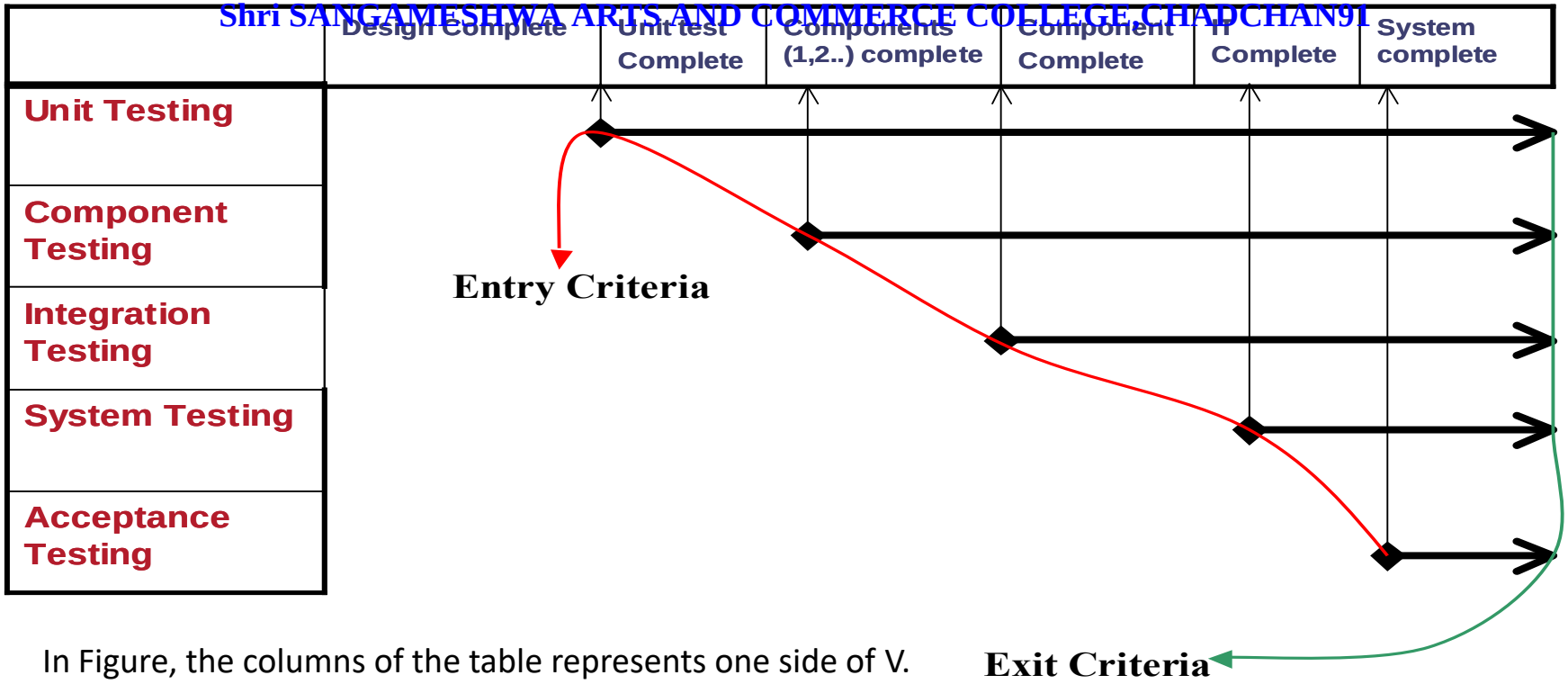
- This model is used in the medical development field, as it is strictly a disciplined domain.
- Requirements are well defined, clearly documented and fixed.
- Product definition is stable.
- Technology is not dynamic and is well understood by the project team.
- There are no undefined requirements. The project is short.
- If above these scenarios are suitable then apply V-model

The Modified V Model:

- V Model introduced various types of testing, the modified V model introduces various phases of testing.
- A phase of testing has a one-to-one mapping to the types of testing.
- That is, there is a unit-testing phase, component-testing phase, and so on.
- Once a unit has completed the unit testing phase, it becomes part of a component and enters the component testing phase.
- It then moves to integration-testing phase and so on.

- Rather than view the product as going through different types of tests as the V model does.
- The modified V Model views each part of the product to go through different phases of testing.

Modified V- Model



- In Figure, the columns of the table represents one side of V.
- And rows which are test phases represent the other side of V.
- Notice that different phases of testing are done in parallel.
- While starting a phase of testing it is important to look at whether the product is ready for testing.
- It is determined by a set of entry criteria.
- The testing phases are also associated with a set of exit criteria to complete the test activities for each phase.

- They are determined by exit criteria.

- The entry and exit criteria for each of the phases ensure that right quality of product delivered.
- for starting the test and right amount of testing is completed for the release.

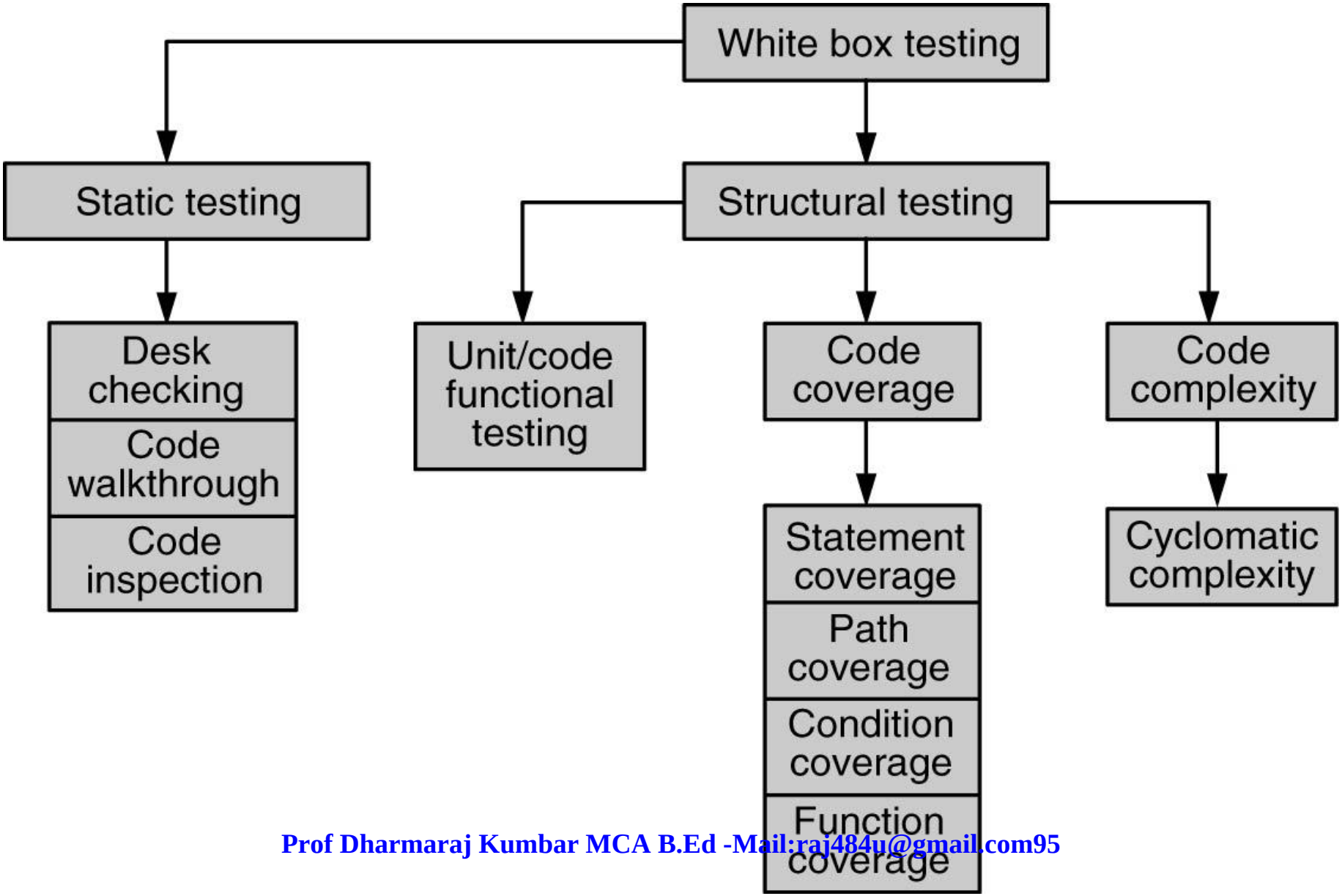
White Box Testing

- White Box Testing: White Box Testing, Static Testing, Structural Testing, Challenges in White Box Testing.

White Box Testing

- White Box Testing: White Box Testing, Static Testing, Structural Testing, Challenges in White Box Testing.

White Box Testing



What is White Box Testing?

- White box testing is a way of testing the external functionality of the code.
- In White box testing we examine and test the program code that fulfill the external functionality.
- This is also known as clear box, or glass box or open box testing.
- White box testing takes into account the program code, code structure, and internal design flow.
- As shown in Figure above, white box testing is classified into "static" and “structural“ testing.

Static testing

- Static testing is a type of testing..
- Which requires only the source code of the product, not the binaries or executables.
- Static testing does not involve executing the programs on computers.
- But rather humans going through it or the use of specialized tools.
- Some of the things tested by static testing:
- Whether the code works according to the functional requirement
- Whether the code has been written in accordance with the design developed earlier in the project life cycle
- Whether the code for any functionality has been missed out
- Whether the code handles errors properly
- Whether the code follows all applicable standards

Static Testing by Humans

- Humans can find errors that computers can’t.
- Multiple humans can provide multiple viewpoint.
- A human evaluation can compare the code against the specifications more thoroughly.
- It can detect multiple defects at one go.
- It reduces downstream, inline pressure.

Static Testing Types

- Different types
  - Desk checking of the code
  - Code walkthrough
  - Code review
  - Code inspection

Desk Checking

- Done manually by the author of the code.
- Desk checking is a method to verify the portions of the code for correctness
- Code Verification is done by comparing the code with the design or specifications. THEN check
- Make sure that the code does what it is supposed to do and effectively.
- Desk checking that most programmers do before compiling and executing the code.
- Whenever errors are found, the author applies the corrections for errors on the spot.

Desk Checking

- There is no process or structure that guarantees or verifies the effectiveness of desk checking.
- If errors are easily seen or understood then this method is effective for correcting coding errors.
- but will not be effective..
- In detecting errors that arise due to incorrect understanding of requirements or incomplete requirements.
- No log or checklist is maintained.

Advantages and Disadvantages of Desk Checking

- **Advantages**
  - The programmer knows the code and programming language well and hence is best suited to read the program
  - Less scheduling and Planning overheads
  - Reduces delay in defect detection and correction
- **Disadvantages**
  - Person dependent, not scalable or reproducible
  - A developer is not the best person to detect problems in his or her own code.
  - Developers prefer writing new code and do not like testing

Code Walkthrough

- This is Group-oriented methods.
- The advantage that walkthrough has over desk checking is that it brings multiple viewpoint.
- In walkthroughs, a set- of people Took at the program code and raise questions for the author.
- The author explains the logic of the code, and answers the questions.
- If the author is unable to answer some questions, he or she then takes those questions and finds their answers.
- Multiple specific roles (author, moderator, inspector, etc.),

Code Inspection-formal Inspection

- Code inspection is a method, normally with a high degree of formal and structured with planned.
- Requires thorough preparation.
- The focus of this method is to detect all faults, break in rule, and other side effects.
- A formal inspection should take place only when the author has made sure the code is ready for inspection.
- But he first perform some basic desk checking and walkthroughs.
- When the code is in such a reasonable state of readiness an inspection meeting is arranged.

There are four roles in inspection.

- Author
  - Author of the work product- author of the code
  - Makes available the required material to the reviewers
  - Fixes defects that are reported
- Moderator
  - Controls the meeting(s)- Run the inspection according to the process.
- Inspectors (reviewers)
  - Prepare by reading the required documents.
  - Take part in the meeting(s) and report defects
  - These are the people who actually provides, review comments for the code.
  - There are typically multiple inspectors.
- Scribe
  - Takes down notes during the meeting
  - Who takes detailed notes during the inspection meeting.
  - Circulates them to the inspection team after the meeting.
  - Can participate to review to the extent possible

Process in a Formal Inspection

- Typical documents circulated:
  - Program code
  - Design / program specifications
  - SRS (if needed)
  - Any applicable standards (e.g., coding standards)
  - Any necessary checklists (e.g., code review checklist)

Meetings in a Formal Inspection

- **Preliminary meeting (optional)**
  - Author explains his / her perspective
  - Makes available the necessary documents
  - Highlights concern areas, if any, for which review comments are sought
- **Defect Logging Meeting**
  - All come prepared!
  - Moderator goes through the code sequentially
  - Each reviewer comes up with comments
  - Comments / defects categorized as “defect” / “observation,” “major” / “minor,” “systemic” / “mis-execution”
  - Scribe documents all the findings and circulates them
- **Follow-up meeting (optional)**
  - Author fixes defects
  - If required, a follow-up meeting is called to verify completeness of fixes

Challenges in conducting formal inspections

- These are time consuming.
- Since the process calls for preparation as well as formal meetings, these can take time.
- The planning and scheduling can become an issue since multiple people are involved.
- It is not always possible to go through every line of code.
- We cannot apply several parameters and their combinations for correctness of the logic, side-effects
- We cannot check appropriate error handling.
- It may also not be necessary to subject the entire code to formal inspection.

Advantages and Disadvantages of Formal Inspection

- Advantages
  - Thorough, when prepared well
  - Brings in multiple perspectives
  - Has been found to be very effective
- Disadvantages
  - Logistically difficult
  - Time consuming
  - May not be possible to exhaustively go through the entire code

Static Analysis Tools:

- There are several static analysis tools available in the IT market.
- These tools can reduce the manual work and perform analysis of the code to find out errors such as those listed below.
- Whether there are unreachable codes (usage of GOTO statements sometimes creates this situation
- Variables declared but not used
- Mismatch in definition and assignment of values to variables.
- illegal or error prone typecasting of variables.
- Use of non-portable programming constructs.
- Memory allocated but not having corresponding statements for freeing them up memory

STRUCTURAL TESTING

Shri SANGAMESHWA ARTS AND COMMERCE COLLEGE,CHADCHAN111

Prof Dharmaraj Kumbar MCA B.Ed -Mail:raj484u@gmail.com111

Prof Dharmaraj Kumbar MCA B.Ed -Mail:raj484u@gmail.com112

Use of a Code Review Checklist

- “Never leave home without it!”
- The code review checklist aids in organizational learning by indicating what to look for.
- It should be kept current as we learn.

Structural Testing

- Done by running the executable on the machine.
- Structural testing takes into account the code, code structure, internal design, and how they are coded.
- In structural testing tests are actually run by the computer on the built product.
- It involve running the product against some predefined test cases and comparing the results against the expected results.
- Designing test cases and test data to ***exercise*** certain portions of code

Types of structural testing

- Types of structural testing
  - Unit / code functional testing
  - Code coverage testing
  - Code complexity testing

Unit / Code Functional Testing

- Initial quick checks by developer.
- Done before more expensive checks.
- The developer can perform certain tests, knowing the input variables and the corresponding expected output variables.
- By repeating these tests for multiple values of input variables, the confidence level of the developer to go to the next level increases.
- Removes “noticeable” errors.
- For modules with complex logic or conditions, the developer can build a "debug version" of the product. Then run on IDE.

Unit / Code Functional Testing

- The debug version can run by putting intermediate print statements.
- And checking the program is passing through the right loops and iterations the right number of times.
- It is important to remove the intermediate print statements after the defects are fixed.
- Another approach to do the initial test is to run the product under a debugger or an Integrated Development Environment (IDE).
- These tools allow single stepping of instructions.
- Allowing the developer to stop at the end of each instruction.
- view or modify the contents of variables, and so on.
- setting break points at any function or instruction.
- And viewing the various system parameters or program variable values.

Code Coverage Testing

- Code coverage testing involves,
- Designing and executing test cases,
- And finding out the percentage of code that is covered by testing.
- The percentage of code covered by a test is found by adopting a technique called “instrumentation” of code.
- Instrumentation rebuilds the product, linking the product with a set of libraries provided by the tool vendors.
- Instrumented code can monitor what parts of code is covered
- The tools also allow reporting on the portions of the code that are covered frequently.
- So that the critical or most-often portions of code can be identified.

Types of Code Coverage

- Statement coverage
- Path coverage
- Condition coverage
- Function coverage

Statement coverage

- In this the test case is executed in such a way that every statement of the code is executed at least once.
- Statement coverage is a white box testing technique, which involves the execution of all the statements at least once in the source code.
- It is a metric, which is used to calculate and measure the number of statements in the source code which have been executed.

Statement coverage

- The statement coverage is also known as line coverage or segment coverage.
- The statement coverage **covers only the true conditions.**
- Through statement coverage we can identify the statements executed and where the code is not executed because of blockage.
- In this process each and every line of code needs to be checked and executed.

- The statement coverage can be calculated as shown below:

$$\text{Statement coverage} = \frac{\text{Number of statements exercised}}{\text{Total number of statements}} \times 100\%$$

- let us take an example

Example of Statement Coverage

- Now, let us take example where we will measure the coverage first.
- In order to simplify the example, we will regard each line as a statement.
- A statement may be on a single line, or it may be spread over several lines.
- One line may contain more than one statement, just one statement, or only part of a statement.
- Some statements can contain other statements inside them.
- In code sample 4.2, we have two read statements, one assignment statement, and then one IF statement on three lines,
- but the IF statement contains another statement (print) as part of it.

Code sample 4.2

- 1 READ X
- 2 READ Y
- 3 Z =X + 2*Y
- 4 IF Z> 50 THEN
- 5 PRINT large Z
- 6 ENDIF

- TEST SET 1
- Test 1_1: X= 2, Y = 3
- Test 1_2: X =0, Y = 25
- Test 1_3: X =47, Y = 1

We have numbered each line and will regard each line as a statement. Let's analyze the coverage of a set of tests on our six-statement program:

Which statements have we covered?

• In Test 1_1, the value of Z will be 8, so we will cover the statements on lines 1 to 4 and line 6.

• In Test 1_2, the value of Z will be 50, so we will cover exactly the same statements as Test 1_1.

• In Test 1_3, the value of Z will be 49, so again we will cover the same statements.

Since we have covered five out of six statements, we have 83% statement coverage (with three tests).

Statement coverage =5/6*100 = 83.00 %

- What test would we need in order to cover statement 5, the one statement that we have not exercised yet? How about this one:

- Test 1_4: X = 20, Y = 25
- This time the value of Z is 70, so we will print ‘Large Z’ and we will have exercised all six of the statements,
- so now statement coverage = 100%.
- Notice that we measured coverage first, and
- Then designed a test to cover the statement that we had not yet covered.
- Note that Test 1_4 on its own is more effective which helps in achieving 100% statement coverage, than the first three tests together.
- Just taking Test 1_4 on its own is also more efficient than the set of four tests, since it has used only one test instead of four.
- Being more effective and more efficient is the mark of a good test technique.

Advantage of statement coverage:

- It verifies what the written code is expected to do and not to do.
- It measures the quality of code written.
- It checks the flow of different paths in the program and it also ensure that whether those path are tested or not.

Disadvantage of statement coverage:

- It cannot test the false conditions.
- It does not report that whether the loop reaches its termination condition.
- It does not understand the logical operators.

Path Coverage

- In this the test case is executed in such a way that every path is executed at least once.

Prof Dharmaraj Kumbar MCA B.Ed -Mail:raj484u@gmail.com130

Prof Dharmaraj Kumbar MCA B.Ed -Mail:raj484u@gmail.com131

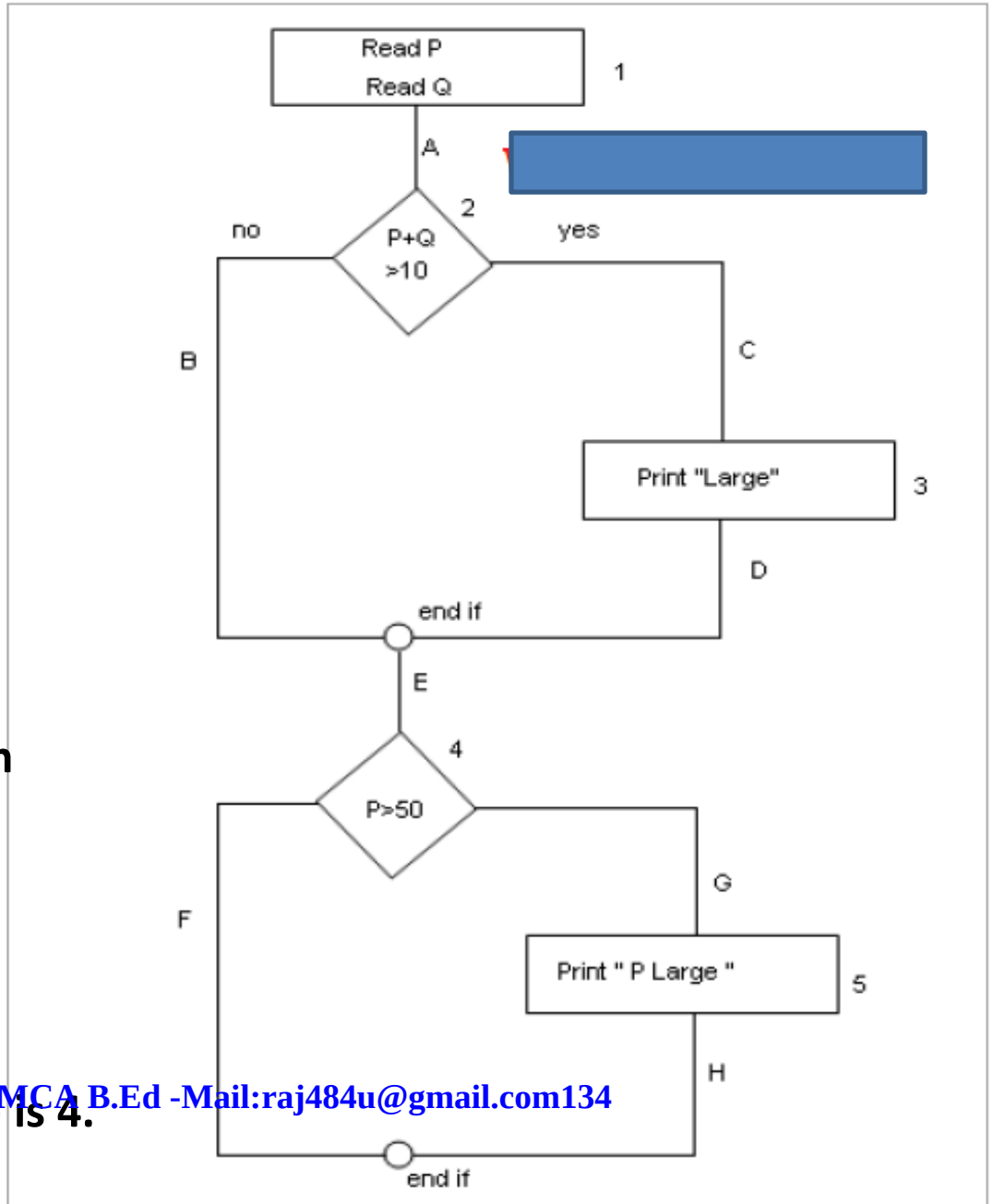
Path Coverage

- In this the test case is executed in such a way that every path is executed at least once.
- All possible control paths taken..
- The objective of Path coverage is to define the number of independent paths, so the number of test cases needed can be defined to maximize test coverage.
- Including all loop paths taken zero, once.. and multiple (ideally, maximum) items in path coverage technique.
- The test cases are prepared based on the logical complexity measure of a procedural design.
- In this type of testing every statement in the program is guaranteed to be executed at least one time.
- Flow Graph, Cyclomatic Complexity and Graph Metrics are used to arrive at basis path.

- In path coverage, we split a program into a number (Total paths exercised/ of distinct paths.
- A program (or a part of a program) can start from the beginning and take any of the paths to its completion.
- Path coverage = (Total paths exercised / Total number of paths in program) * 100

Example of Path Coverage

- Read P
- Read Q
- IF P+Q > 100 THEN
- Print “Large”
- ENDIF
- If P > 50 THEN
- Print “P Large”
- ENDIF



Path Coverage (PC): Path Coverage ensures covering of all the paths from start to end. All possible paths are-

1A-2B-E-4F
1A-2B-E-4G-5H
1A-2C-3D-E-4G-5H
1A-2C-3D-E-4F

So path coverage is 4.

Condition coverage

- The condition coverage, as defined by the formula gives an indication of the percentage of conditions covered by a set of test cases.
- Condition coverage is a much stronger criterion than path coverage, which in turn is some much stronger criteria than statement coverage.
- With Condition coverage the possible outcomes of (—true or —false) for each condition are tested at least once.
- This means that each individual condition is one time true and false.
- In other words, we cover all conditions, hence condition coverage.
- **Condition coverage = (Total decisions exercised / Total number of decisions in program) * 100**

Example of Conditional Coverage

- Result:
- In order to guarantee complete Condition coverage criteria for the above example,
- A, B and C should be evaluated at least once against "true" and "false“
- So, in our example,
- the 3 following tests would be sufficient for 100% Condition coverage testing.

```
A = true  | B = not eval | C = false
A = false | B = true     | C = true
A = false | B = false    | C = not eval
```

```
if ((A || B) && C)
{
<< Few Statements >>
}
else
{
<< Few Statements >>
}
```

Function coverage

- This is a new addition to structural testing to identify how many program functions (similar to functions in "C" language) are covered by test cases.
- The requirements of a product are mapped into functions during the design phase and each of the functions form a logical unit.
- For example, in a database software, "inserting a row into the database" could be a function.
- Or, in a payroll application, "calculate tax" could be a function.
- Each function could, in turn, be implemented using other functions.

- While providing function coverage..
- Test cases can be written so as to exercise each of the different functions in the code.
- **Function coverage = Total functions exercised / total number of functions in program) * 100**

- Advantages of function coverage

Advantages of function coverage

- Functions are easier to identify in a program and hence it is easier to write test cases to provide function coverage.
- Since functions are at a much higher level of abstraction than code, it is easier to achieve 100 percent function coverage than 100 percent coverage in any of the earlier methods.
- We can also measure how many times a given function is called.
- Function coverage provides a natural transition to black box testing.

Code complexity testing

- Cyclomatic complexity is a software metric (measurement), used to indicate the complexity of a program.
- It is a quantitative measure of the number of linearly independent paths through a program's source code.
- Cyclomatic complexity is a source code complexity measurement that is being correlated to a number of coding errors.
- It is calculated by developing a Control Flow Graph of the code that measures the number of linearly-independent paths through a program module.

- Lower the Program's cyclomatic complexity, lower the risk to modify and easier to understand.
- It can be represented using the below formula:

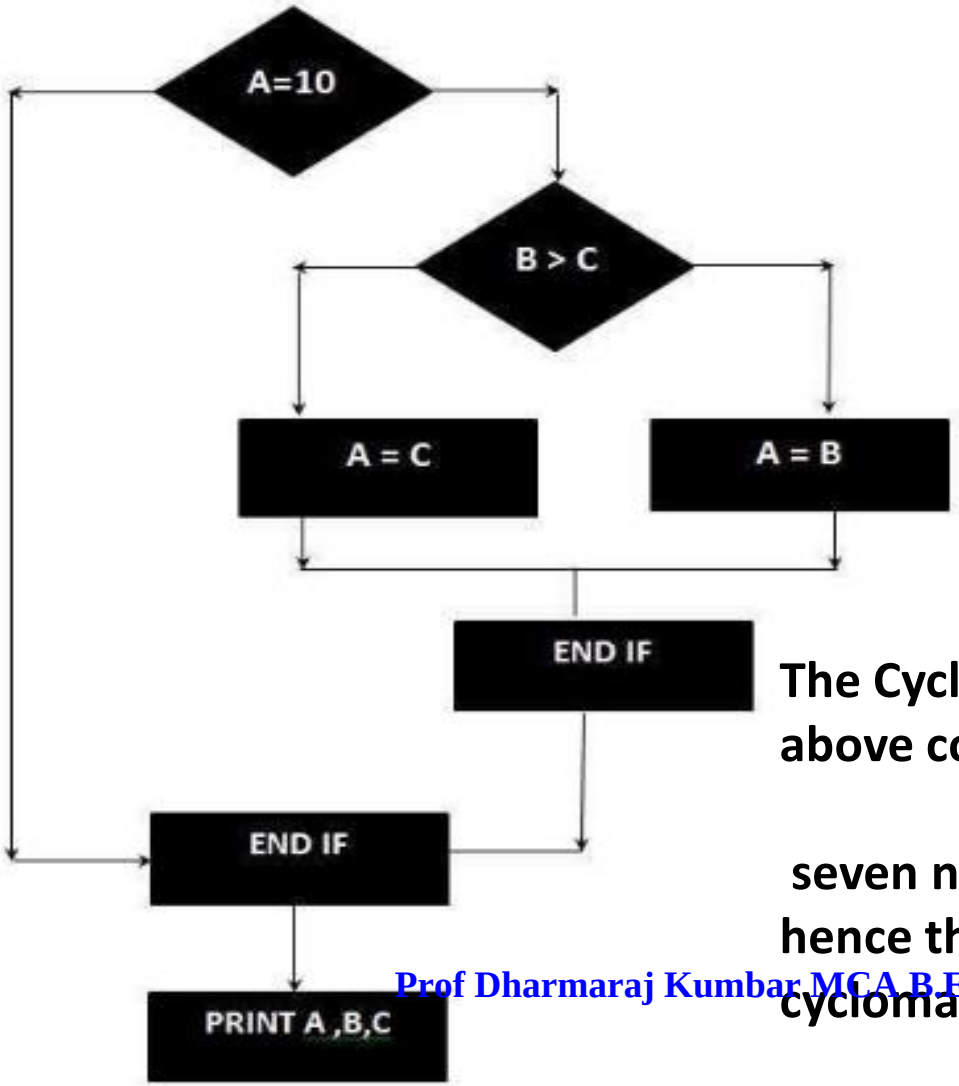
- Cyclomatic complexity = E - N +2*P

where, E = number of edges in the flow graph.

- N = number of nodes in the flow graph.
- P = number of nodes that have exit points

Example of Cyclomatic complexity

- Flow Graph




```
IF A =10 THEN
  IF B > C THEN
     A = B
  ELSE
     A = C
  ENDIF
ENDIF
Print A
Print B
Print C
```

The Cyclomatic complexity is calculated using the above control flow diagram that shows

seven nodes(shapes) and eight edges (lines), hence the
cyclomatic complexity is 8 - 7 + 2 = 3

Challenges in White Box Testing

- White box testing requires a sound knowledge of the program code and the programming language.
- Developers should get involved in white box testing, but developers In general, do not like to perform testing functions.

- for example in static and structural testing methods such as reviews. because of the timeline pressures, the programmers may not "find time" for reviews .
- Human tendency of a developer being unable to find the defects in his or her code - - most of us have blind spots in detecting error in our own products.

- Fully tested code may not correspond to realistic scenarios- Programmers generally do not have a full appreciation of the external (customer) perspective or the domain knowledge to visualize how a product will be deployed in realistic scenarios. This may mean that even after extensive testing, some of the common user scenarios may get left out and defects may creep (move slowly) in.

End of UNIT-1

Prof Dharmaraj Kumbar MCA B.Ed -Mail:raj484u@gmail.com145