UNIT 1

Introduction to Software Engineering

Topics Covered

The story of Software development and issues faced, Need for Systematic process for addressing issues, Products, custom solutions, services, domains, Technologies, Software life cycle, software development lifecycle, software release process, source control, versioning, maintenance of software. DevOps.

Software Development Processes: Waterfall, Iterative, Spiral.

Software definition

Software can defined as:

- ❖ Collection of programs & related documents that provide desire features, function & performance.
- ❖ Software products developed for a particular customer can be either Generic/ customised
- ❖ **Generic -** developed to be sold to a range of different customers/group

  e.g. PC software such as Excel or Word.
- ❖ (custom) - developed for a single customer according to their specification

Software Engineering definition

- **It is Defined as a** Discipline in which theories, method & tools are applied to develop quality software Based on problem constraint tools & technologies are applied to develop quality software
- Software engineering is the most important technology with rapid development

❖ National economy is dependent

❖ More systems are controlled by software

Legacy software

- They are old and larger computer based systems which may use “obsolete technology”
- These systems may include older hardware, older software old process, old procedures
- Legacy systems are used for long period because it is too risky to replace them
- Air traffic control system
- **Legacy software has important characteristics**
- Poor quality
- Poor documentation
- Poor managed history
- Poor code

Examples of Software Development

- Web Development
- Mobile Development
- Software Tool Development
- Application Development
- Data Management Software Development
- Security Software Development
- Embedded Systems Development
- Cloud Computing Software Development

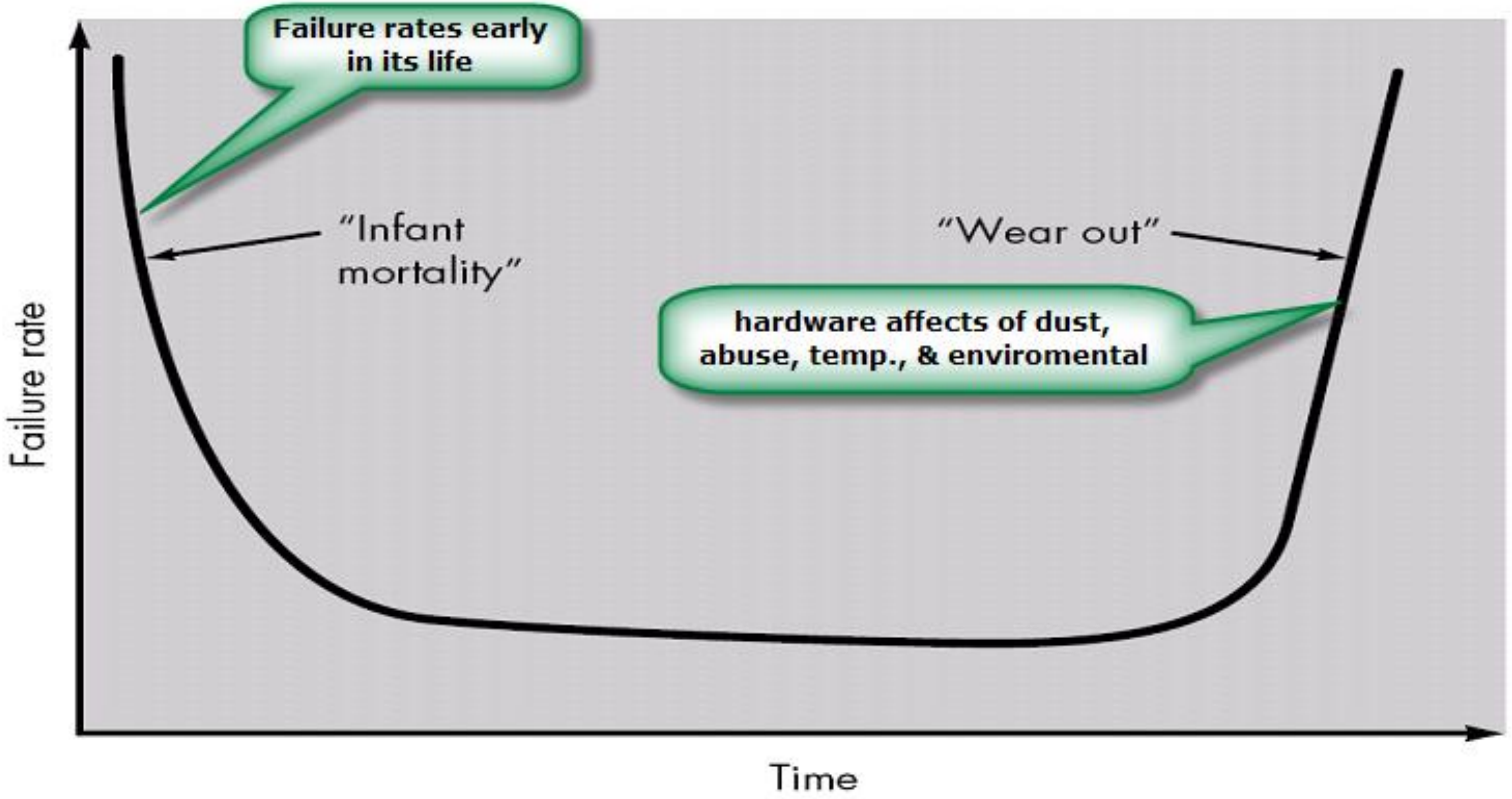
Work product of SDLC

Requirement gathering	SRS
Design	ER,DFD,Design document
Coding	Source code
Testing	Test plan
Maintanence	User manual

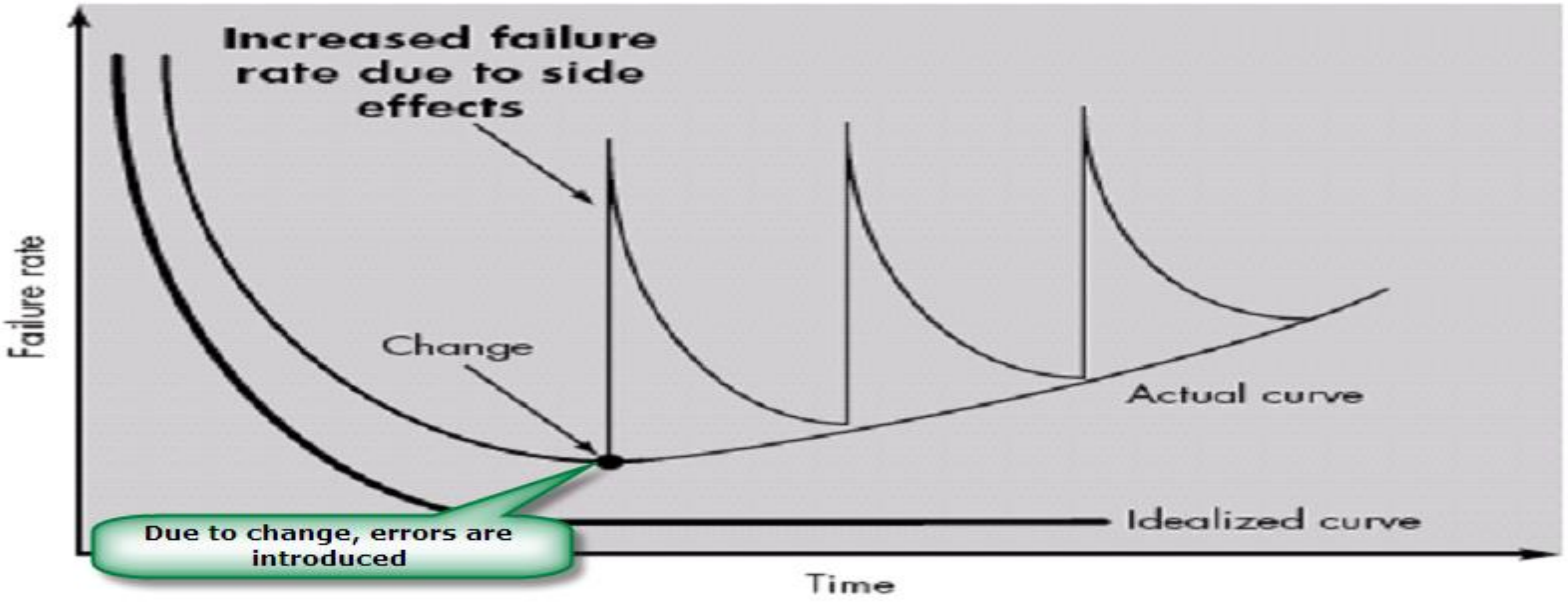
Software Characteristics

Hardware curve	Software curve
In early stages of hardware development process failure rate is high because of manufacturing defects. After correcting failure rate decreases Failure rate remains constant for period of time Again increases because of environment. Hardware components wears out can be replaced	It doesn’t effected with environment. Hence it is a idealized curve. Due to undiscovered error rate is high. During life of software as defects get introduced failure rate is high Curve looks like spike Frequent changes in software causes software to detoriate.

Failure curve for Hardware



Failure curve for Software



When a hardware component wears out, it is replaced by a spare part.

The Story Of Software Development: Evolution Of Software Development

"In today's digital age, software development is at the heart of technological advancements, influencing how we live, work, and communicate."

GITAM
DEEMED TO BE UNIVERSITY

Evolution of Software Development

- 1950s - Assembly Language
- 1970s - High-level Programming languages like C and Pascal
- 1990s - Object-oriented Programming (OOP) brought modularity and code reusability.
- 2000s - Agile and Iterative Development emphasizing collaboration, flexibility, and iterative development.

Changes in software over a span of 50 years

Era of computing	Software
Early years	Batch processing, customized software
Second era	Multi users, real time systems
Third era	Distributed systems
Fourth era	Object oriented systems, expert systems,parallel computing
Fifth era	Mobile computing

Need for Systematic process for addressing issues: Managing Complexity

Problem: Software systems are inherently complex due to their size, functionality, and integration with other systems.

Systematic Process:

- Divide the system into smaller, manageable components (modular design).
- Use structured methodologies like the Waterfall or Spiral model to approach development in stages.

Need for Systematic process for addressing issues: Handling Changing Requirements

Problem: Requirements often evolve, leading to scope and impacting timelines and budgets.

Systematic Process:

- Implement Requirements Engineering to gather, analyze, and document needs systematically.
- Use iterative models (e.g., Agile) that accommodate changes through incremental updates.

Need for Systematic process for addressing issues: Risk Management

Problem: Unanticipated risks like technology failures or resource unavailability can derail projects.

Systematic Process:

- Identify and assess risks during project planning.
- Use risk mitigation frameworks to develop contingency plans.

Need for Systematic process for addressing issues: Better Resource Management

Problem: Projects often face resource constraints, including time, budget, and skilled personnel.

Systematic Process:

- Use tools like Gantt charts and resource allocation matrices for effective planning.
- Employ methodologies like Critical Path Analysis (CPA) to identify and focus on key activities.

Need for Systematic process for addressing issues: Collaboration and Communication

Problem: Miscommunication between stakeholders and developers leads to mismatched expectations.

Systematic Process:

Use clear documentation standards for requirement specifications, design models, and test cases.

Conduct regular stakeholder meetings and progress reviews.

Need for Systematic process for addressing issues: Long-Term Maintenance and Sustainability

Problem: Software often needs updates, bug fixes, and integration with newer systems post-deployment.

Systematic Process:

Follow coding and design standards that prioritize maintainability.

Use Configuration Management tools to track changes and version history

Need for Systematic process for addressing issues: Security and Reliability

Problem: Increasing cybersecurity threats demand robust and secure software systems.

Systematic Process:

- Embed security requirements during the design phase.
- Conduct systematic penetration testing and threat modeling to identify vulnerabilities.

Need for Systematic process for addressing issues: Meeting Deadlines and Budgets

Problem: Without structured planning, projects often exceed time and cost estimates.

Systematic Process:

- Use project management tools and frameworks (e.g., SCRUM in Agile).
- Set realistic milestones and monitor progress regularly.

Products

There are two kinds of software products:

1.Generic products These are stand-alone systems that are produced by a development organization and sold on the open market to any customer who is able to buy them. Examples of this type of product include software for PCs such as databases, word processors, drawing packages, and project-management tools. It also includes so-called vertical applications designed for some specific purpose such as library information systems, accounting systems, or systems for maintaining dental records.

2. **Customized (or bespoke) products** These are systems that are commissioned by a particular customer. A software contractor develops the software especially for that customer. Examples of this type of software include control systems for electronic devices, systems written to support a particular business process, and air traffic control systems.

Custom Solutions

- **Codebase/Source Code:** The actual program written in a programming language. This is the core product of software development.
- **Executable/Binaries:** Compiled versions of the source code that can be run on specific hardware and operating systems.
- **Documentation:** Technical documentation, user manuals, and other written materials that provide information about the software's design, functionality, and usage.
- **Requirements Specification:** A document outlining the functional and non-functional requirements of the software, including features, constraints, and user expectations.
- **Design Artifacts:** Diagrams, flowcharts, and other design documents that illustrate the architecture and structure of the software.
- **Prototypes:** Early versions of the software designed to demonstrate key features and gather feedback from stakeholders.
- **Test Cases and Test Plans:** Documents outlining the testing strategy, including test cases and plans for unit testing, integration testing, system testing, and acceptance testing.
- **Bug Reports:** Documents reporting issues and defects found during testing or in the production environment.

- **Release Versions:** Different versions of the software released to users, each containing new features, improvements, and bug fixes.
- **User Interfaces (UI) and User Experience (UX) Designs:** Designs and mockups for the graphical user interface and the overall user experience of the software.
- **Database Schemas:** Definitions of the structure and organization of databases used by the software.
- **Configuration Files:** Files that specify the configuration settings for the software, allowing customization and adaptation to different environments.
- **Installation Packages:** Software installers or packages that facilitate the installation of the software on users' machines.
- **Maintenance and Support Documentation:** Information on how to maintain, troubleshoot, and support the software after it has been deployed.
- **Training Materials:** Materials created to train users or administrators on how to use and manage the software effectively.
- **Legal and Licensing Documents:** Documents specifying the terms of use, licensing agreements, and any legal considerations related to the software.

What is Utility Software?

- Utility Software is a type of software which is used to analyse and maintain a computer.
- This software is focused on how OS works on that basis it performs tasks to enable the smooth functioning of the computer.

Types of Utility Software

- Antivirus software
- Disk cleaners
- Backup and recovery software
- System optimizers
- Disk defragmenter
- File compression software
- Disk encryption software

Services:

Application services

provide specific applications such as email, conferencing, photo sharing

Access to specific educational content such as scientific

films or historical resources.

They are external services that are either specifically purchased for the system or

are available freely over the Internet.

Configuration services

- used to manage and customize technical systems and products to meet customer preferences:
- It define how services are shared between students, teachers, and their parents.
- Wi-Fi CSP contains the settings to create a Wi-Fi profile.
- CSPs are behind many of the management tasks and policies

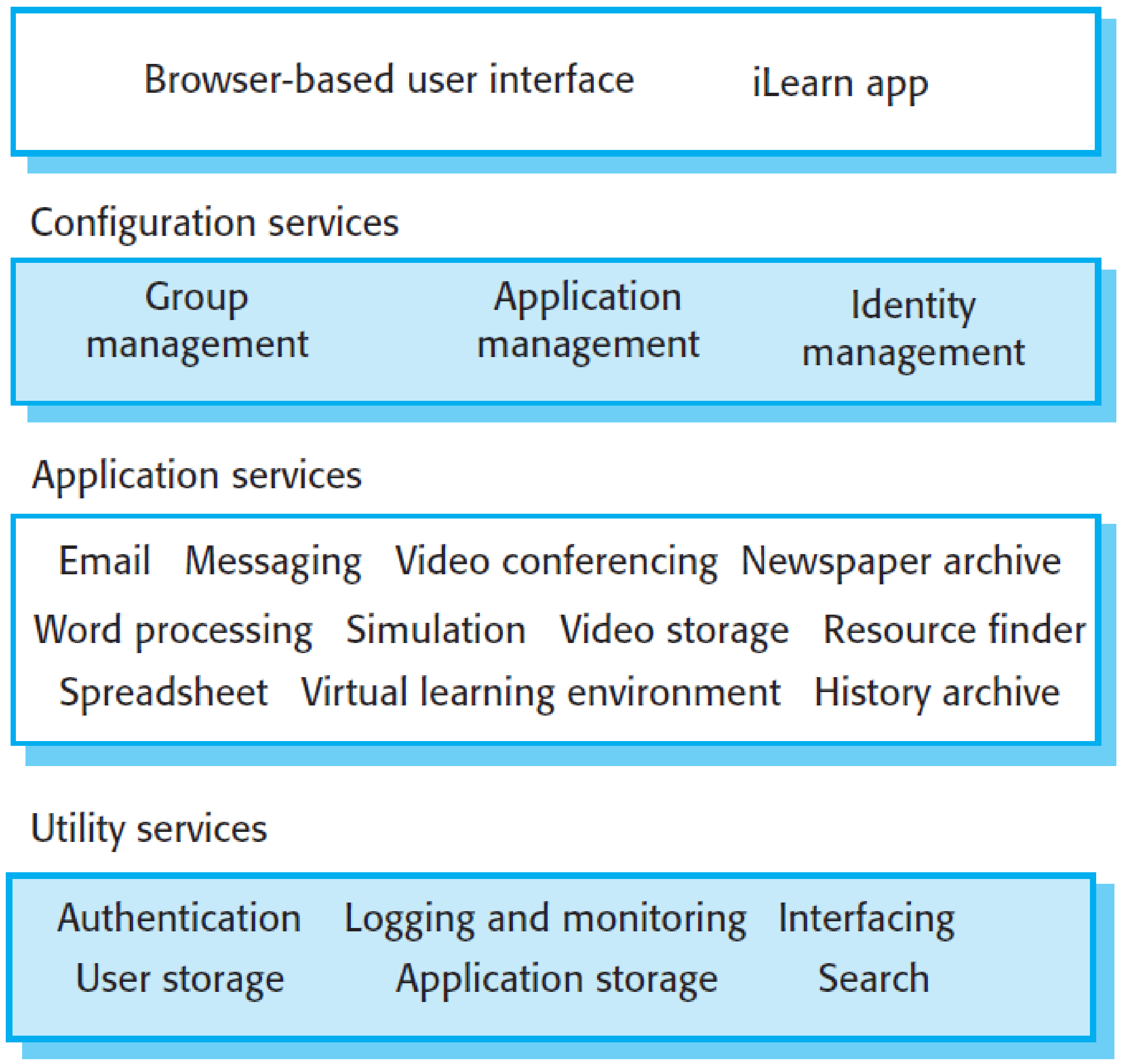


Figure 1.8 The architecture of a digital learning environment (iLearn)

Integrated services

- They are services that offer an API (application programming interface)
- They can be accessed by other services through that API.
- An authentication service is an example of an integrated service.
- Rather than use their own authentication mechanisms
- Authentication service may be called on by other services to authenticate users.
- If users are already authenticated, then the authentication service may pass authentication information directly to another service.

Independent services

- ***They*** are services that are simply accessed through a browser
- interface and that operate independently of other services.
- reauthentication may be required for each independent service.

Domains

In software engineering, the term "domains" can refer to different contexts or areas of application where software development is applied.

- **Web Development:** Designing and building applications that are accessed through web browsers. This includes both front-end development (user interface) and back-end development (server-side logic).
- **Mobile App Development:** Creating applications specifically designed for mobile devices, such as smartphones and tablets. This includes both Android and iOS platforms.
- **Desktop Application Development:** Developing software applications that run on desktop computers, typically with a graphical user interface (GUI).
- **Embedded Systems:** Designing software for embedded systems, which are specialized computing systems embedded within other devices or systems. Examples include firmware for IoT devices or control systems in appliances.
- **Database Development:** Focusing on designing, implementing, and maintaining databases. This involves creating database structures, optimizing queries, and ensuring data integrity.

- **Game Development:** Creating interactive and immersive software for entertainment purposes. Game development involves graphics programming, physics simulation, and user interaction design.
- **Cloud Computing:** Developing applications that leverage cloud services, including Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS).
- **Artificial Intelligence (AI) and Machine Learning (ML):** Building software that uses AI and ML techniques to analyze data, make predictions, and automate tasks.
- **Cybersecurity:** Developing software to secure systems and data from unauthorized access, attacks, and vulnerabilities.
- **Networking:** Creating software for network management, communication protocols, and data transmission.

- **Healthcare Informatics:** Developing software solutions for healthcare systems, including electronic health records (EHR), medical imaging, and healthcare data analysis.
- **Financial Software:** Creating applications for financial institutions, including banking software, accounting systems, and financial analysis tools.
- **E-commerce:** Building software for online retail and electronic commerce platforms, including shopping cart systems and payment processing.
- **Educational Software:** Developing software for educational purposes, such as learning management systems (LMS), educational games, and interactive learning applications.
- **Human-Computer Interaction (HCI):** Focusing on designing software with a strong emphasis on user experience and interface design.

Technologies

Software engineering encompasses a wide range of technologies that are used to design, develop, test, deploy, and maintain software applications. The choice of technologies depends on various factors, including the type of application, project requirements, scalability needs, and the preferences of the development team.

Programming Languages: Java, Python, JavaScript, C#, C++, Ruby, PHP, Swift, Kotlin, Go, Rust: Different programming languages are chosen based on factors such as application type, performance requirements, and developer preferences.

Web Development: HTML, CSS, JavaScript, TypeScript, React, Angular, Vue.js: These technologies are used for building interactive and dynamic web applications.

Mobile App Development: Swift, Objective-C, Kotlin, Java, Flutter, React Native: For developing mobile applications on platforms like iOS and Android.

Server-Side Development: Node.js, Django, Flask, Ruby on Rails, ASP.NET, Spring: Frameworks and technologies for building server-side logic and APIs.

Database Management Systems (DBMS): MySQL, PostgreSQL, MongoDB, Oracle, Microsoft SQL Server: Used for managing and organizing data in databases.

Cloud Computing: Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP): Platforms providing cloud infrastructure, storage, and services for deploying and scaling applications.

Containerization and Orchestration: Docker, Kubernetes: Technologies for packaging, deploying, and managing applications in containers for improved scalability and portability.

Version Control: Git, GitHub, GitLab, Bitbucket: Tools for version control to track changes in the source code and facilitate collaboration among developers.

DevOps Tools: Jenkins, Travis CI, CircleCI: Continuous Integration (CI) tools that automate the testing and integration of code changes.

Code Editors and Integrated Development Environments (IDEs): Visual Studio Code, IntelliJ IDEA, Eclipse: Software for writing, testing, and debugging code efficiently.

Testing Frameworks: JUnit, Selenium, Jest, pytest: Tools for automating and performing tests to ensure the quality of the software.

Front-End Libraries and Frameworks: React, Angular, Vue.js: Libraries and frameworks for building interactive user interfaces in web applications.

Backend Frameworks: Express (Node.js), Django, Flask, Ruby on Rails, Spring: Frameworks for building the server-side logic and handling HTTP requests.

Microservices Architecture: Spring Boot, Flask, Express, .NET Core: Technologies for developing microservices, which are small, independent, and scalable services that work together.

Artificial Intelligence and Machine Learning: TensorFlow, PyTorch, scikit-learn: Libraries and frameworks for developing AI and machine learning applications.

Cybersecurity Tools: Wireshark, Metasploit, Nmap: Tools for analyzing and securing network communication and identifying vulnerabilities.

Collaboration and Communication Tools: Slack, Microsoft Teams, Jira, Confluence: Tools for team collaboration, communication, and project management.

Monitoring and Logging Tools: ELK Stack (Elasticsearch, Logstash, Kibana), Prometheus, Grafana: Tools for monitoring and logging application performance and health.

The Story Of Software Development: Software Lifecycle

Utilization: The period during which the software is actively used by end-users.

Maintenance: Ongoing support, updates, and modifications during the software's active use.

Retirement/Disposal: The phase when the software is phased out, replaced, or discontinued.

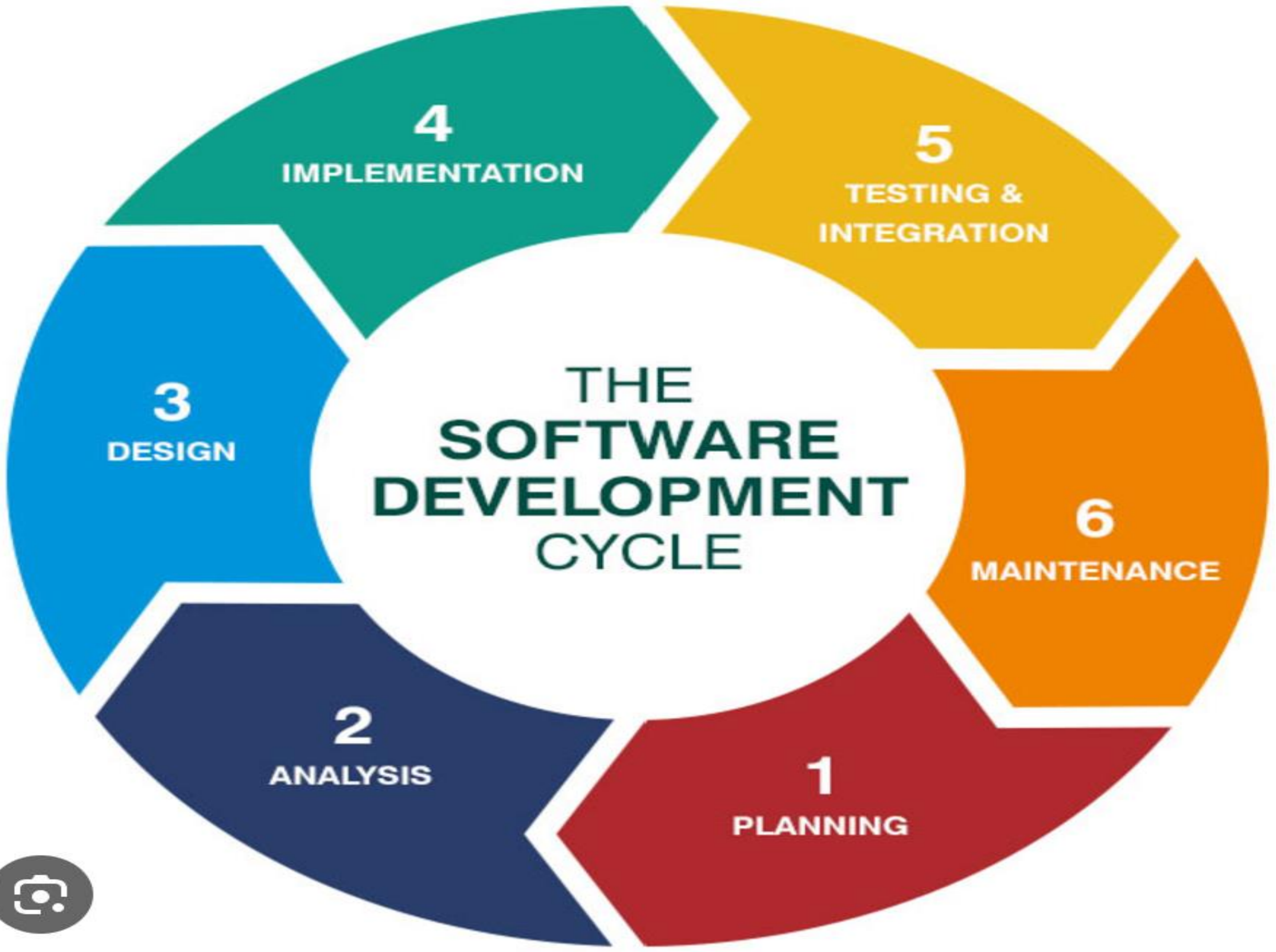
The Story Of Software Development: Software Development Lifecycle

- SDLC is a systematic process used by software developers to design, develop, test, and deploy high-quality software.\

Key Stages:

1. **Requirements:** Gathering and defining what the software is expected to do.
2. **Planning** how the software will meet the requirements.
3. **Design:** Structuring the system architecture.
4. **Implementation:** Writing code that fulfills the design.
5. **Testing:** Verifying that the software works as intended. Ensuring the software meets quality standards
6. **Deployment:** Releasing the software for use.
7. **Maintenance:** Ongoing support, bug fixes, and updates. Updating and refining software post-deployment.

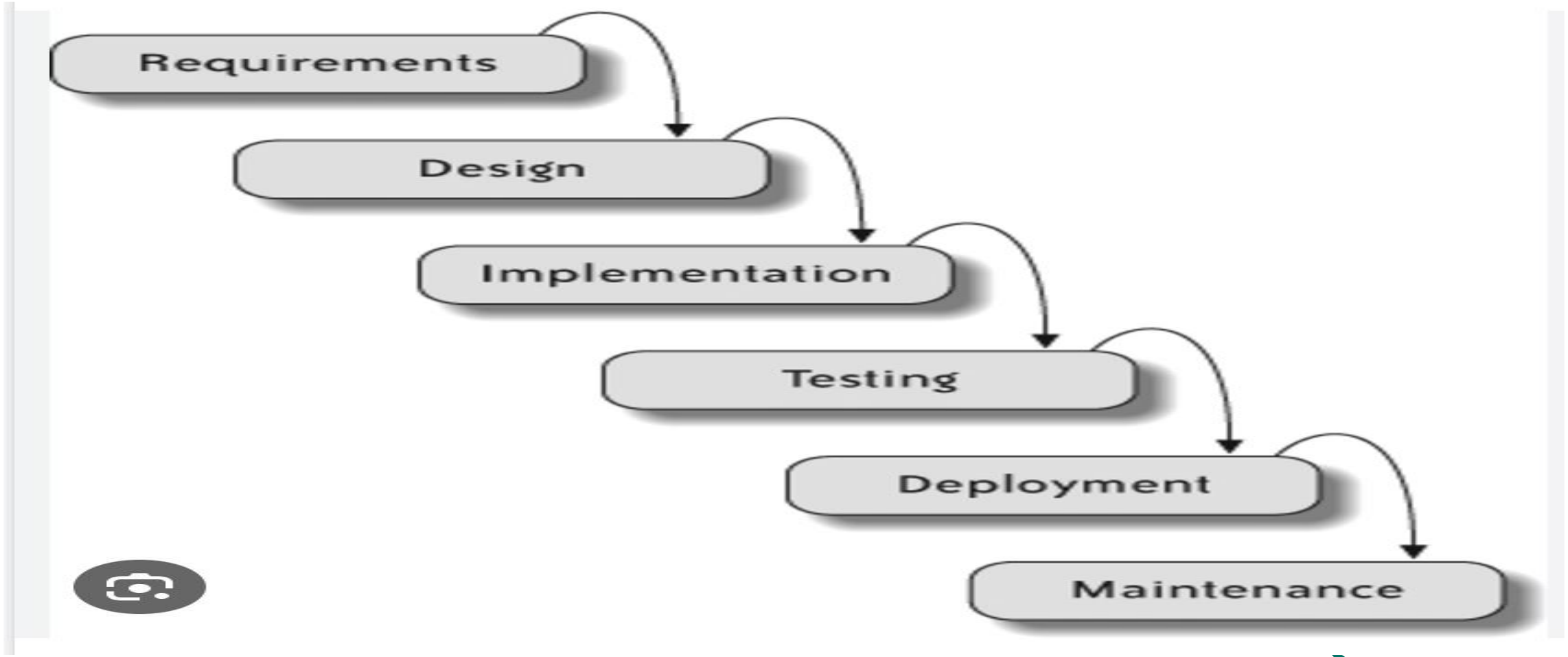
The Story Of Software Development: Software Development Lifecycle

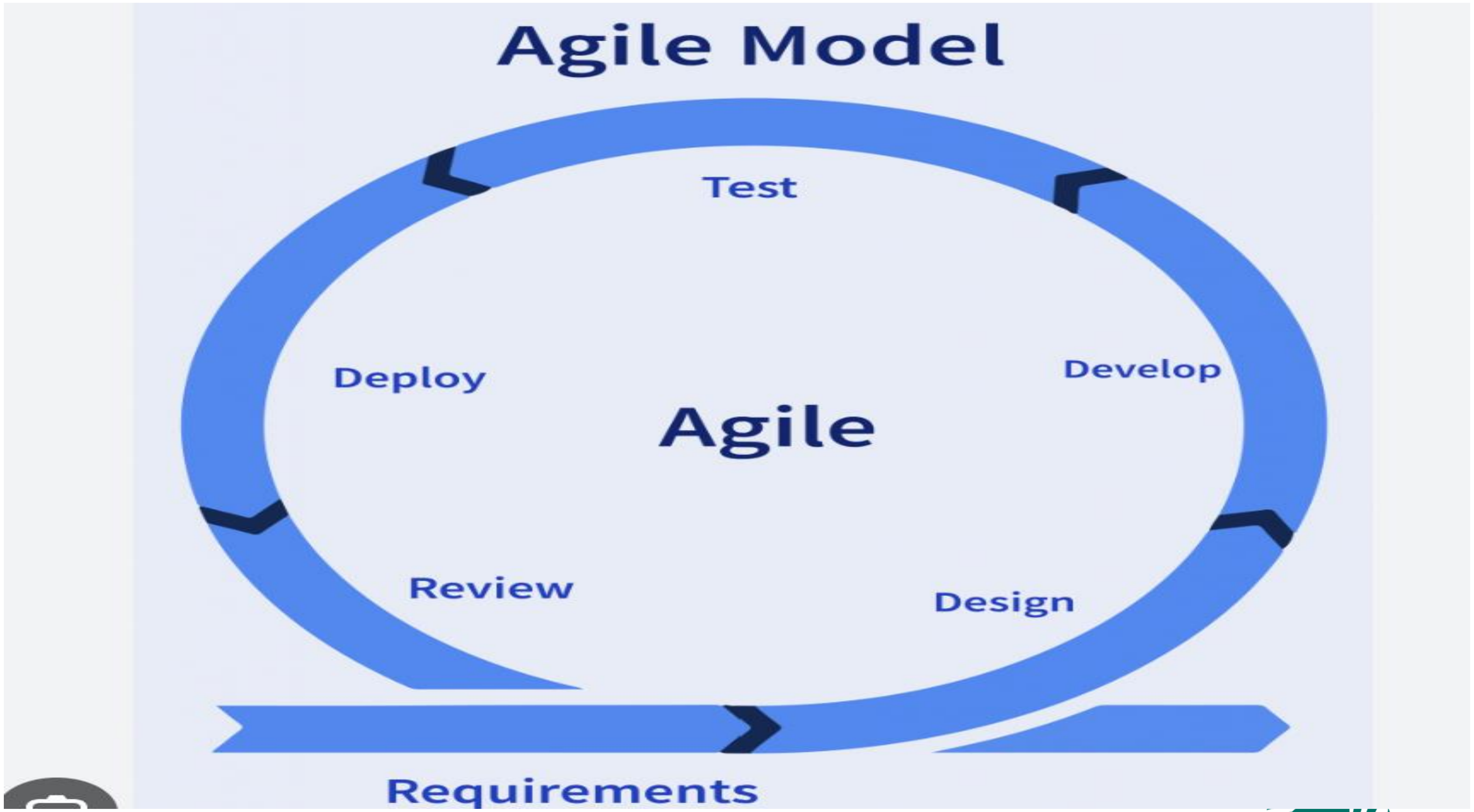


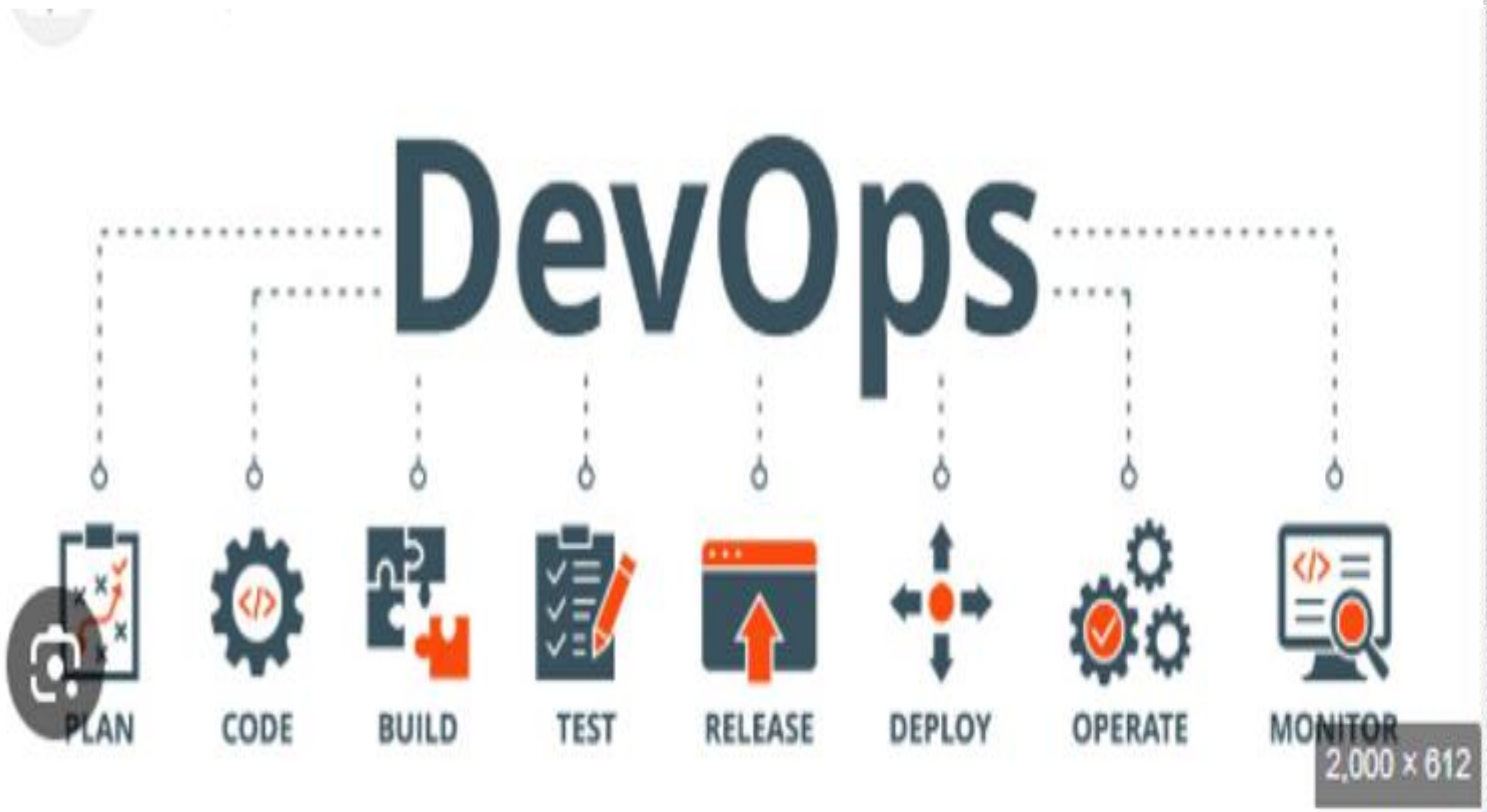
The Story Of Software Development: Approaches to Development

- **Waterfall Model**: Sequential development with well-defined stages.
- **Agile Methods**: Iterative and incremental development with a focus on adaptability.
- **DevOps**: Integration of development and operations for continuous delivery

Waterfall Model




Agile Model
Test
Develop
Deploy
Agile
Review
Design
Requirements



Devops Flowchar t

Challenges in Software Engineering

- **Collaborating** with other team members, such as project managers, designers, and other developers

- **"Complexity of Projects - Managing intricate software projects with numerous components and dependencies."**

- **"Changing Requirements - Adapting to evolving client needs throughout the development process.“**

- **Meeting Deadlines** to ensure the quality of the final product.

- **Debugging** software can be time-consuming and complex, particularly for large and complex systems.

Common challenges faced in software development

- **Security concerns**
- **Quality assurance**
- **Keeping up with new technologies**
- **Managing changing requirements**
- **Collaboration issues**
- **Time Delays**
- **Dealing with legacy code:** Developers often have to work with legacy code.
- **Balancing short-term and long-term goals:** we need to ensure that the software is maintainable and scalable in the long-term.

Common issues in software development

- "Budget Overruns - Overspending due to unforeseen challenges or scope changes."
- "Code Complexity - Difficulty in managing codebases, leading to potential errors.“

Software Release Process

- It is also known as Release Management Cycle

It performs a cycle for

Developing

testing

deploying

supporting new versions of software.

Phases of a Software Release Process

1. **Define specific requirements for the release**

2. **Specify your acceptance criteria**

2. **Test your software in production**

4. **Iterate and refine your product**

5. **Release your product to end-users**

How Does The Software Release Process Typically Work?

The software release process is a series of steps that teams follow to get new software ready.

1. **Planning:** First, developers figure out what changes or new features need to be added to the software.

2. **Building:** Next, they write the code to create those features. This is where the actual work of making the software happens.

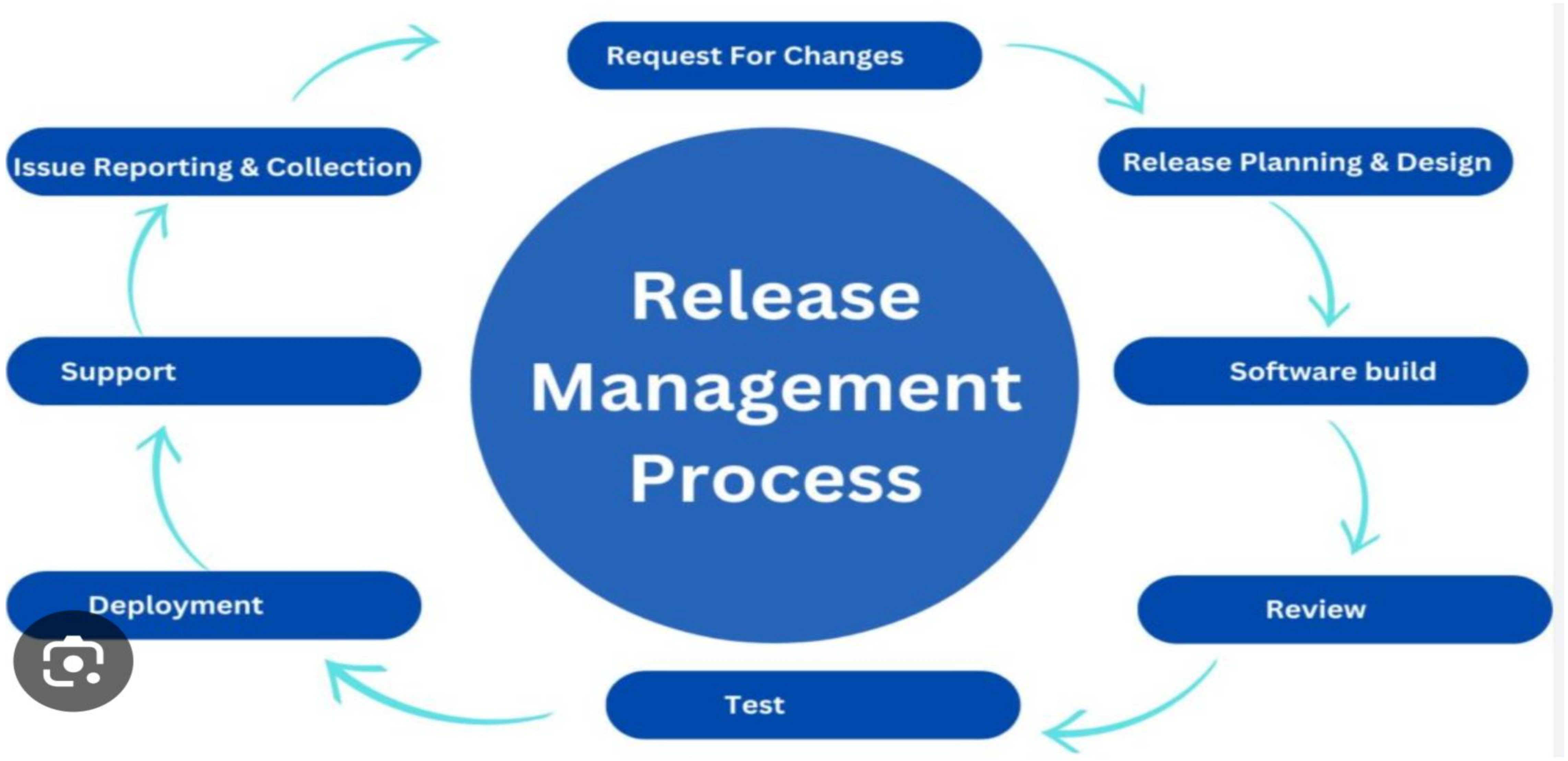
How Does The Software Release Process Typically Work?

3. **Testing:** Before the software is released, it goes through testing to catch any problems.

4. **Launching:** Once everything is tested and working well, the software is made available to users.

5. **Updating:** After the launch, the team keeps an eye on the software, fixing any issues that come up an making improvements based on user feedback.

GITAM
DEEMED TO BE UNIVERSITY


Request For Changes
Release Planning & Design
Software build
Review
Test
Deployment
Support
Issue Reporting & Collection
Release Management Process

Source control (or version control)

- The practice of tracking and managing changes to code.
- Source control management (SCM) systems provide a running history of code development
- It help to resolve conflicts when merging contributions from multiple sources.

Version control

- They are category of software tools that helps in recording changes made to files by keeping a track of modifications done in the code.
- They keeps track on changes made on a particular software and take a snapshot of every modification.
- Helps in recovery in case of any disaster or contingent situation,
- Informs us about Who, What, When, Why changes have been made.
- Reduce possibilities of errors and conflicts

GITAM
DEEMED TO BE UNIVERSITY

Types of Version Control Systems

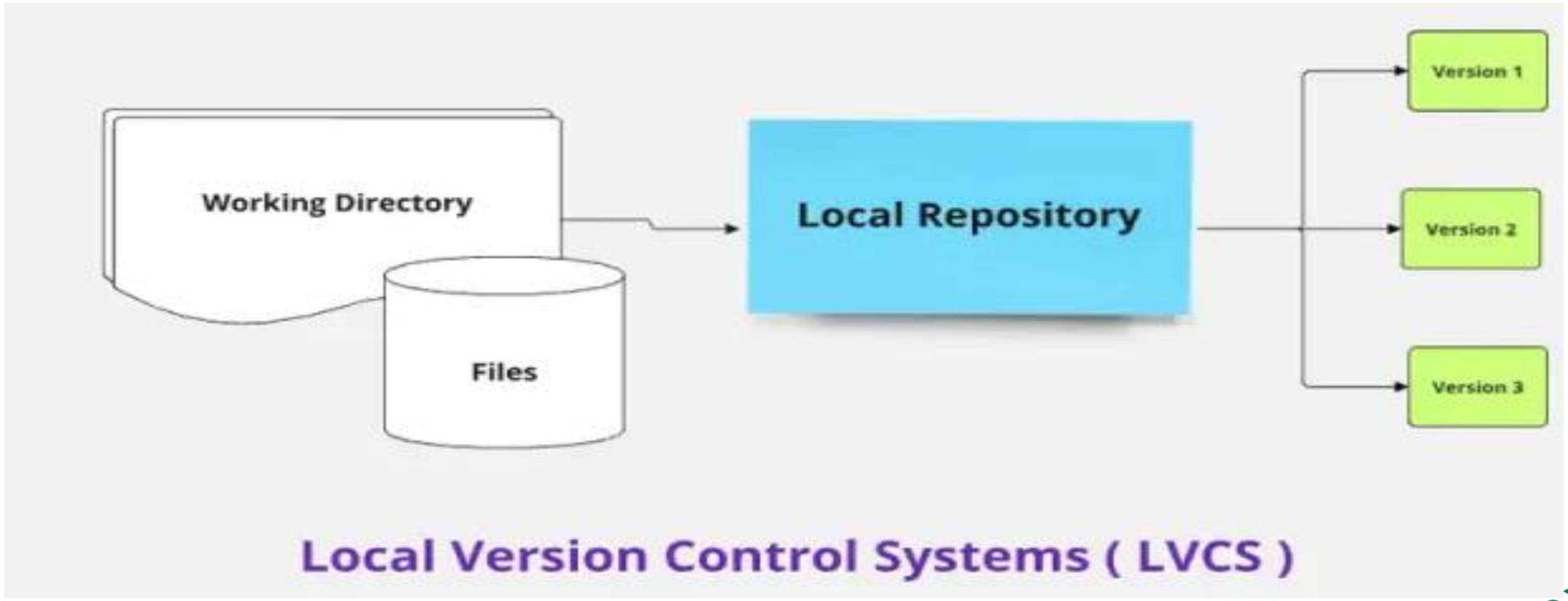
- ❖ Local Version Control Systems
- ❖ Centralized Version Control Systems
- ❖ Distributed Version Control Systems

Two things are required to make your changes visible to others

- ❖ You commit
- ❖ They update

Local Version Control Systems

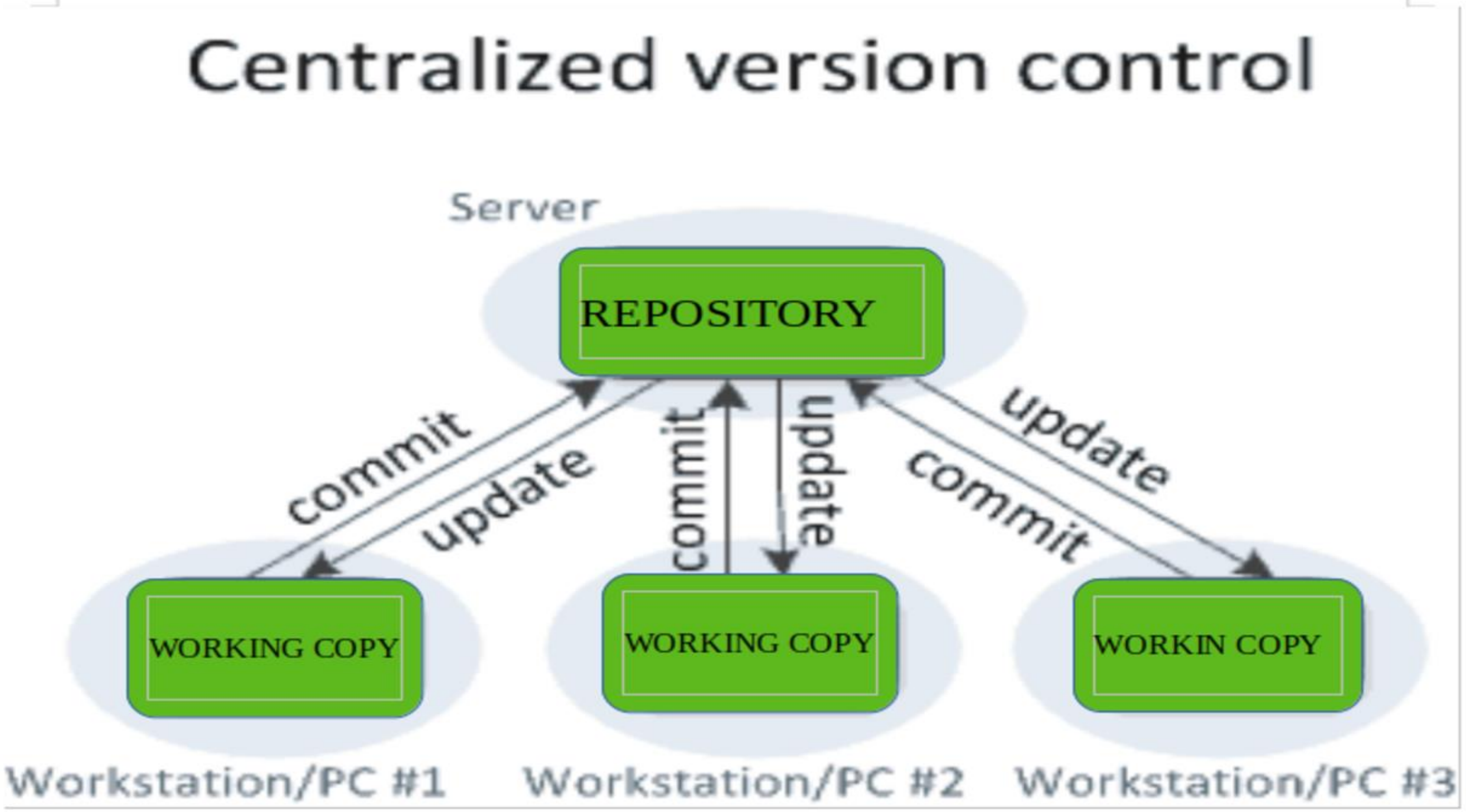
- It is one of the simplest forms
- It has a database
- It keeps all the changes to files under revision control



Centralized Version Control Systems

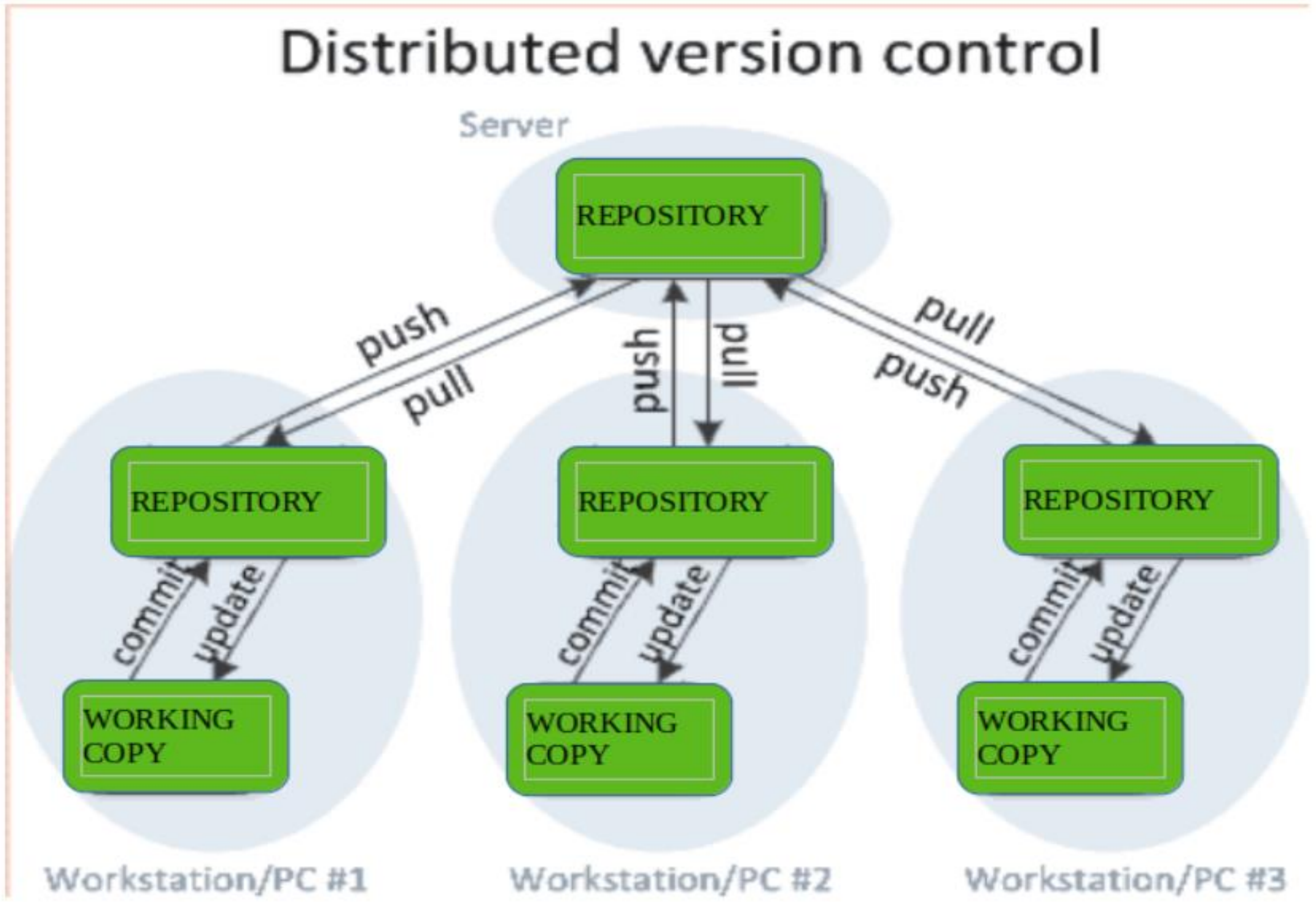
Contain Just One Repository Globally

Every User Need To Commit For Reflecting One's Changes In The Repository



Distributed Version Control Systems

- It contain multiple repositories.
- Each user has their own repository and working copy.
- Just committing your changes will not give others access to your changes.
- This is because commit will reflect those changes in your local repository
- you need to push them in order to make them visible on the central repository


Distributed version control
Server
REPOSITORY
push
pull
push
pull
pull
push
REPOSITORY
REPOSITORY
REPOSITORY
commit
update
commit
update
commit
update
WORKING COPY
WORKING COPY
WORKING COPY
Workstation/PC #1
Workstation/PC #2
Workstation/PC #3

Software Maintenance

Software maintenance is a crucial phase in the software engineering life cycle that involves activities to keep a software system operational, enhance its capabilities, and address issues that may arise during its use. The goal of software maintenance is to ensure that the software remains effective and efficient throughout its lifecycle.

Corrective Maintenance: This type of maintenance involves fixing defects, bugs, or errors discovered during or after the software deployment. Corrective maintenance aims to restore the software to a working state and improve its reliability.

Adaptive Maintenance: Adaptive maintenance is performed to adapt the software to changes in its environment, such as operating system updates, hardware changes, or other external factors. The goal is to ensure that the software remains compatible with evolving technologies.

Perfective Maintenance: Perfective maintenance focuses on improving the functionality and performance of the software. It includes enhancements, optimizations, and additions to existing features, often based on user feedback or changing business requirements.

Preventive Maintenance: Preventive maintenance aims to anticipate and address potential issues before they become critical. This involves activities such as code refactoring, performance tuning, and security updates to proactively enhance the software's maintainability.

Key Practices in Software Maintenance

Bug Tracking and Resolution: Maintain a systematic approach for identifying, tracking, and resolving software defects. Use bug tracking systems to prioritize and manage reported issues.

Version Control and Configuration Management: Utilize version control systems to manage changes to the codebase. Configuration management helps maintain consistency across different environments and configurations.

Documentation: Keep comprehensive documentation for the software, including design documents, user manuals, and API documentation. This helps new developers understand the system and facilitates easier maintenance.

Testing: Regularly conduct testing, including regression testing, to ensure that new changes do not introduce new defects and that the existing functionality remains intact.

Code Review: Perform code reviews to ensure code quality, adherence to coding standards, and to catch potential issues early in the development process.

Monitoring and Logging: Implement monitoring and logging mechanisms to detect and diagnose issues in real-time. This helps identify performance bottlenecks, errors, and other issues that may arise during runtime.

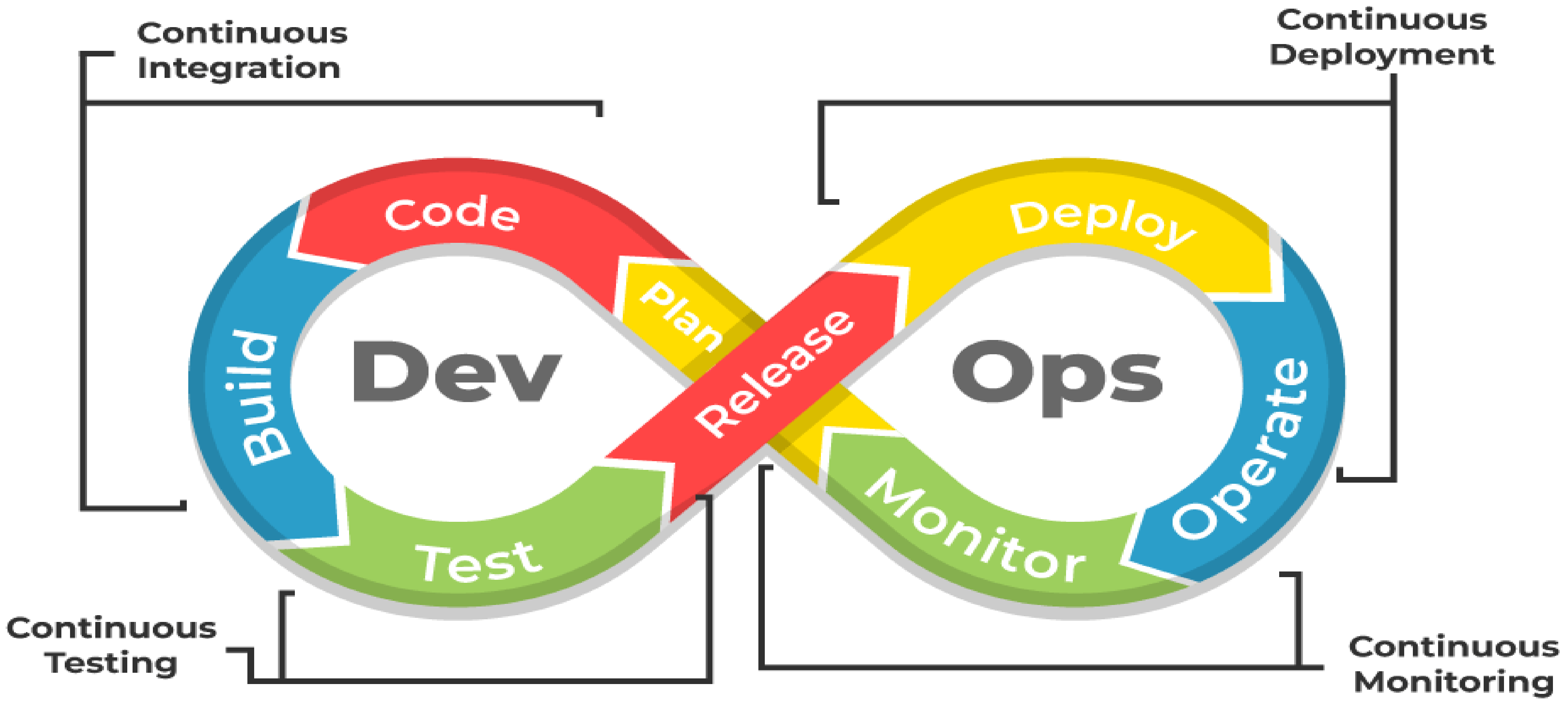
Security Updates: Stay vigilant about security vulnerabilities and apply timely updates to address them. Regularly review and update dependencies to ensure a secure software environment.

User Feedback and Communication: Encourage users to provide feedback and promptly address their concerns. Effective communication channels with users can help in understanding their needs and expectations.

Legacy System Management: If dealing with legacy systems, create a plan for their maintenance and eventual migration or replacement. Legacy systems may require special attention due to outdated technologies and potential security risks.

DevOps:

- DevOps is a software development approach emphasizing collaboration, automation, and continuous delivery to provide high-quality products to customers quickly and efficiently.
- DevOps breaks down silos between development and operations teams to enable seamless communication, faster time-to-market, and improved customer satisfaction.
- It allows a team to handle the complete application lifecycle, from development to testing, operations, and deployment. It shows cooperation between Development and Operations groups to deploy code to production quickly in an automated and repeatable manner.
- Every phase of the software development lifecycle, including planning, coding, testing, deployment, and monitoring, is heavily automated in DevOps.
- This improves productivity, ensures consistency, and lowers error rates in the development process.

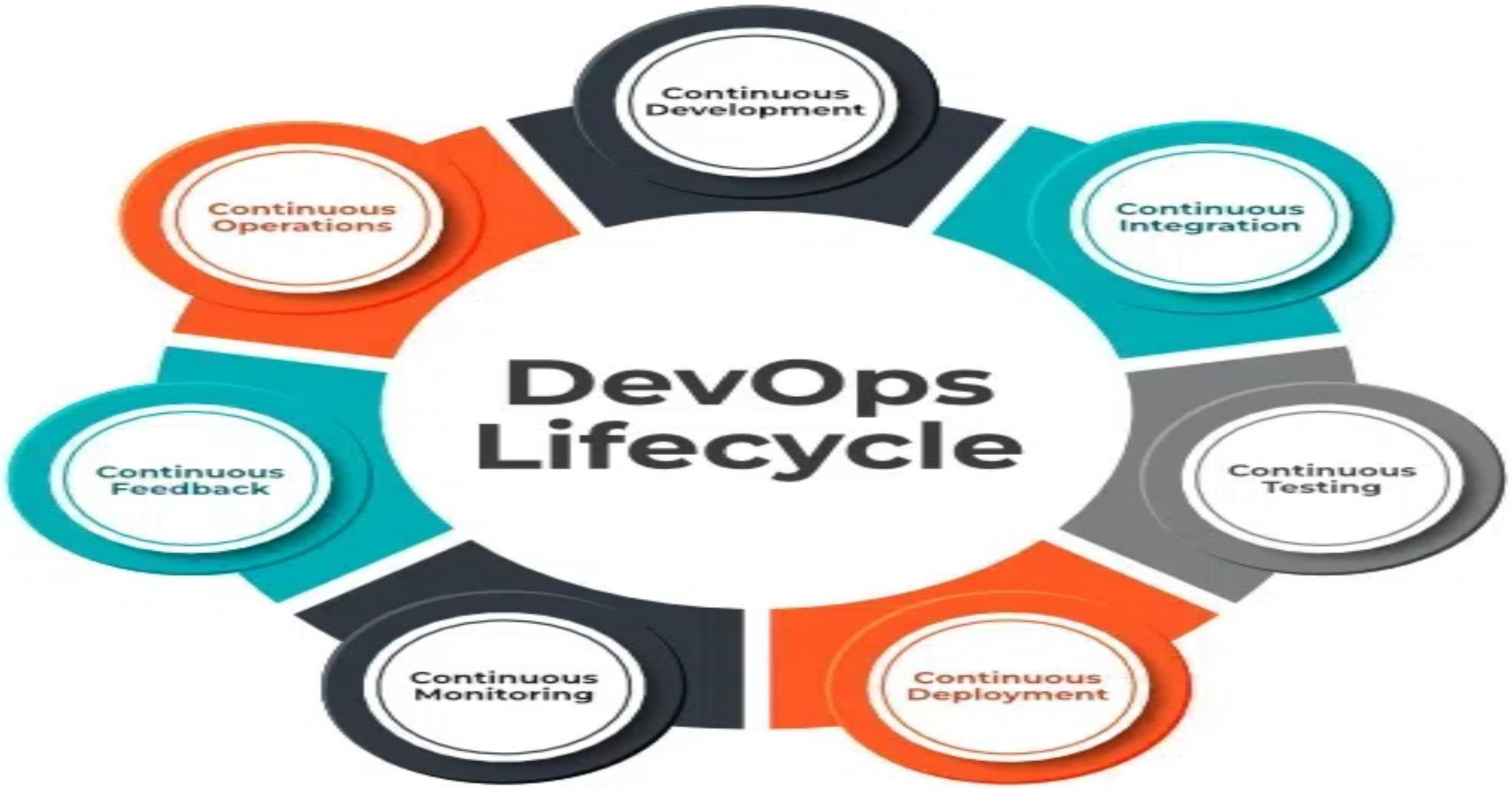

Continuous Integration
Continuous Deployment
Code
Plan
Release
Deploy
Build
Dev
Ops
Operate
Test
Monitor
Continuous Testing
Continuous Monitoring

DevOps: Delivery Pipeline

- The pipeline represents the different stages that software goes through before it is released to production.

These stages might typically include:

- **Build:** software code is compiled and packaged into a deployable unit.
- **Test:** software is rigorously tested to ensure it functions as expected and identifies any bugs.
- **Release:** software is deployed to production for end users.


Continuous Development
Continuous Integration
Continuous Testing
Continuous Deployment
Continuous Monitoring
Continuous Feedback
Continuous Operations
DevOps Lifecycle

C’s of DevOps

1. Continuous Development
2. Continuous Integration
3. Continuous Testing
4. Continuous Deployment/Continuous Delivery
5. Continuous Monitoring
6. Continuous Feedback
7. Continuous Operations

Benefits of DevOps

1. **Faster Delivery:** DevOps enables organizations to release new products and updates faster and more frequently, which can lead to a competitive advantage.

2. **Improved Collaboration:** DevOps promotes collaboration between development and operations teams, resulting in better communication, increased efficiency, and reduced friction.

3. **Improved Quality:** DevOps emphasizes automated testing and continuous integration, which helps to catch bugs early in the development process and improve the overall quality of software.

4. **Increased Automation:** DevOps enables organizations to automate many manual processes, freeing up time for more strategic work and reducing the risk of human error.

Benefits of DevOps

6. **Increased Customer Satisfaction:** DevOps helps organizations to deliver new features and updates more quickly, which can result in increased customer satisfaction and loyalty.

7. **Improved Security:** DevOps promotes security best practices, such as continuous testing and monitoring, which can help to reduce the risk of security breaches and improve the overall security of an organization’s systems.

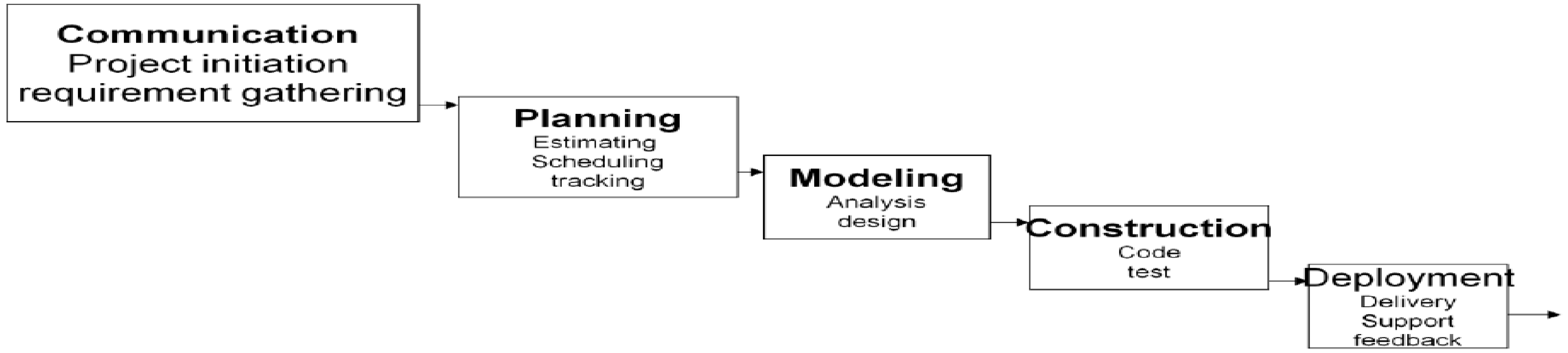
8. **Better Resource Utilization:** DevOps enables organizations to optimize their use of resources, including hardware, software, and personnel, which can result in cost savings and improved efficiency.

Waterfall Model

The Classic Model / The Linear Model / the Waterfall Model

- Used when requirements are well understood in the beginning.
- Also called the classic life cycle.
- A systematic, sequential approach to software development.
- Begins with customer specification of Requirements and progresses through planning, modeling, construction, and deployment.
- This Model suggests a systematic, sequential approach to software development that begins at the system level and progresses through analysis, design, code, and testing.

Conti..



OR

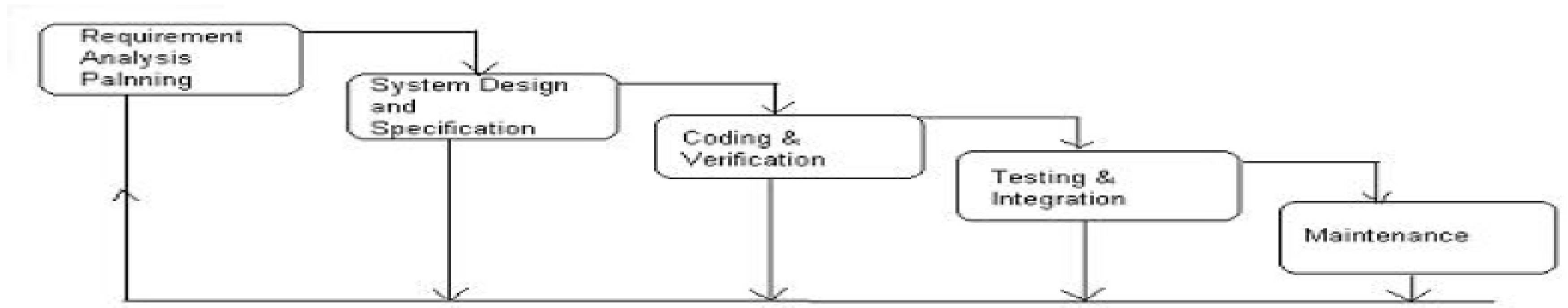


Figure: The Waterfall Model

GITAM
DEEMED TO BE UNIVERSITY

Advantages:

- Simple goal.
- Simple to understand and use.
- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.
- Easy to manage.
- Each phase has a specific deliverable and a review.
- Works well for projects where requirements are well understood.
- Works well when quality is more important than cost/schedule.
- Customers/End users already know about it.

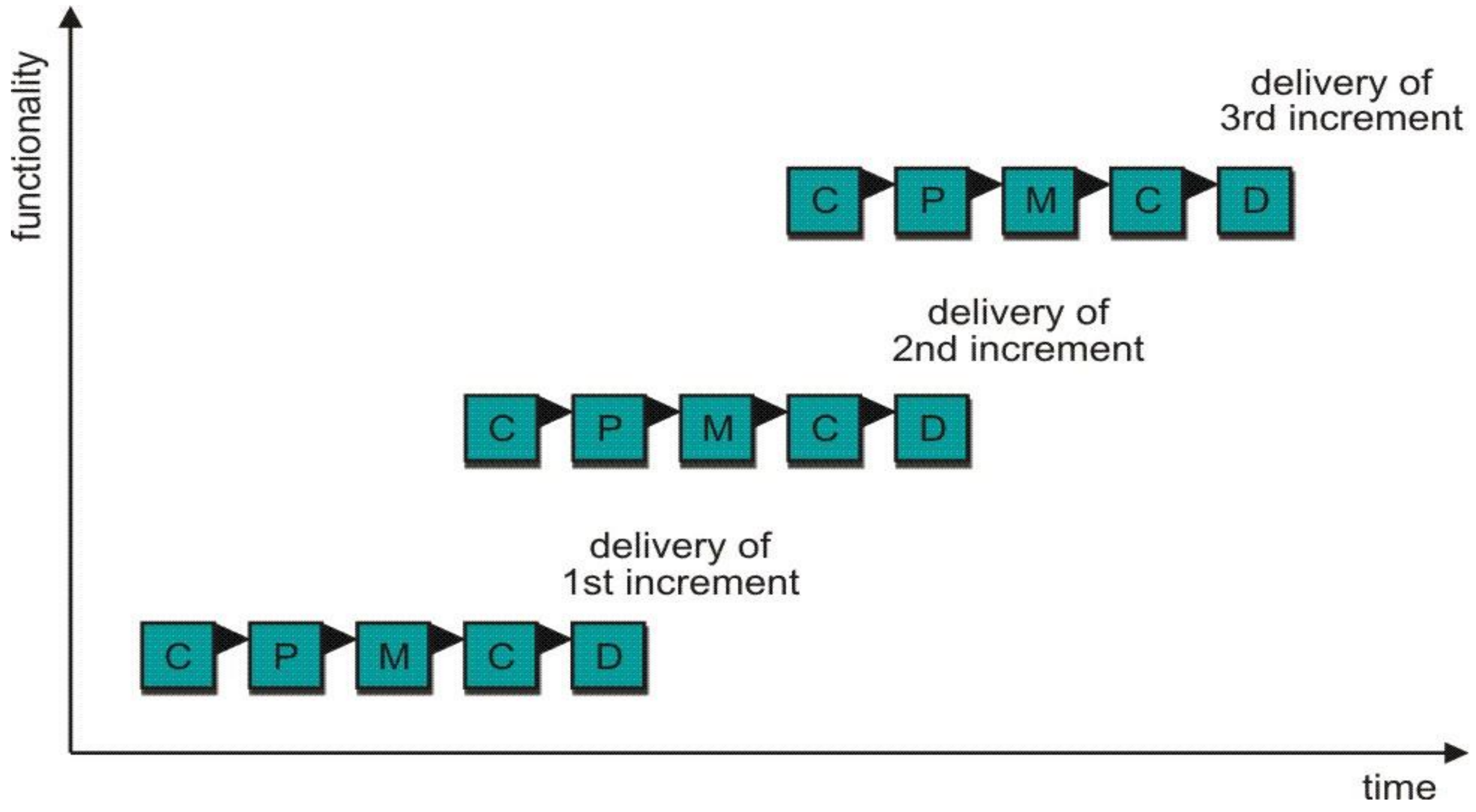
Disadvantages:

- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- No working software is produced until late in the life cycle.
- Risk and uncertainty are high with this process model.
- Adjusting scope during the life cycle can end a project
- Not suitable for complex projects
- Not suitable for projects of long duration because in long-running projects requirements are likely to change.
- Integration is done as a "big-bang” at the very end [Single shot], which doesn't allow identifying any technological or business bottleneck or challenges early.
- Users can only judge quality at the end.
- Attempting to go back two or more phases is very costly.
- Percentage completion of functionality cannot be determined in the middle of the project development because functionality will be undergoing some phase.
- Very risky, since one process can not start before finishing the other.

Problems

- Real projects rarely follow the sequential flow since they are always iterative
- The model requires requirements to be explicitly spelled out at the beginning, which is often difficult.
- A working model is not available until late in the project time plan.
- The customer must have patience.

Incremental Process Model


functionality
delivery of 3rd increment
C P M C D
delivery of 2nd increment
C P M C D
delivery of 1st increment
C P M C D
time

C- Communication
P - Planning
M – Modeling
C - Construction
D - Deployment

Delivers software in small but usable pieces, each piece builds on pieces already delivered

The Incremental Model

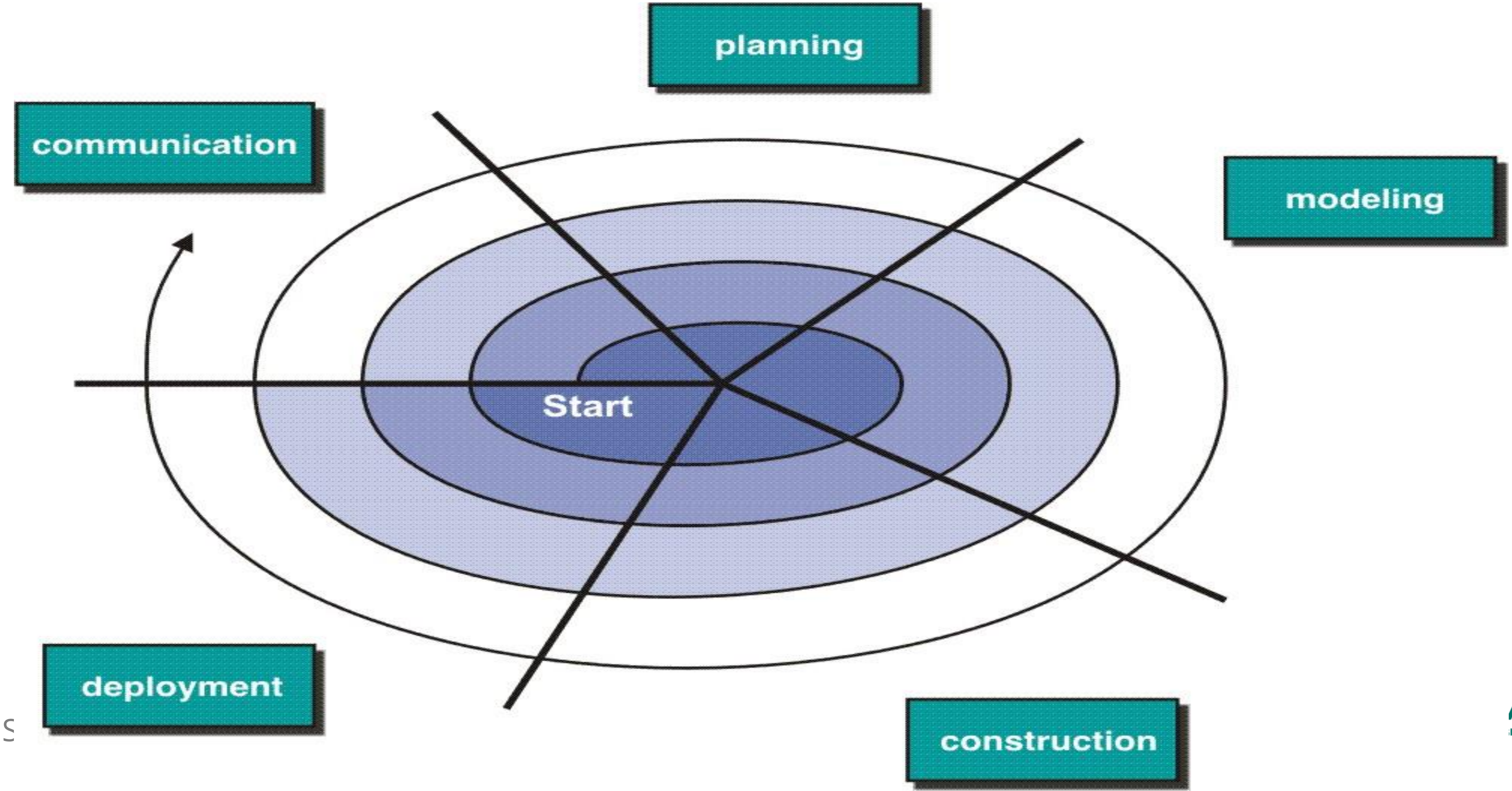
- Linear model in staggered fashion
- It follows iterative approach
- Same phases as in waterfall model
- Increment: Series of releases delivered to customer
- More functionality is added to each release
- First release is basic core product
- New requirements are added to each release
- We choose this model when overall scope is limited & provides limited functionality

The Incremental Model

Merits:

- Less people are involved
- Risks are managed
- Small span project can be developed
- **Eg:** Word processing package:
- 1st increment: Document processing facilities
- 2nd increment: file processing capabilities are available
- 3rd increment: Spelling facilities are available

Evolutionary Model: Spiral Model



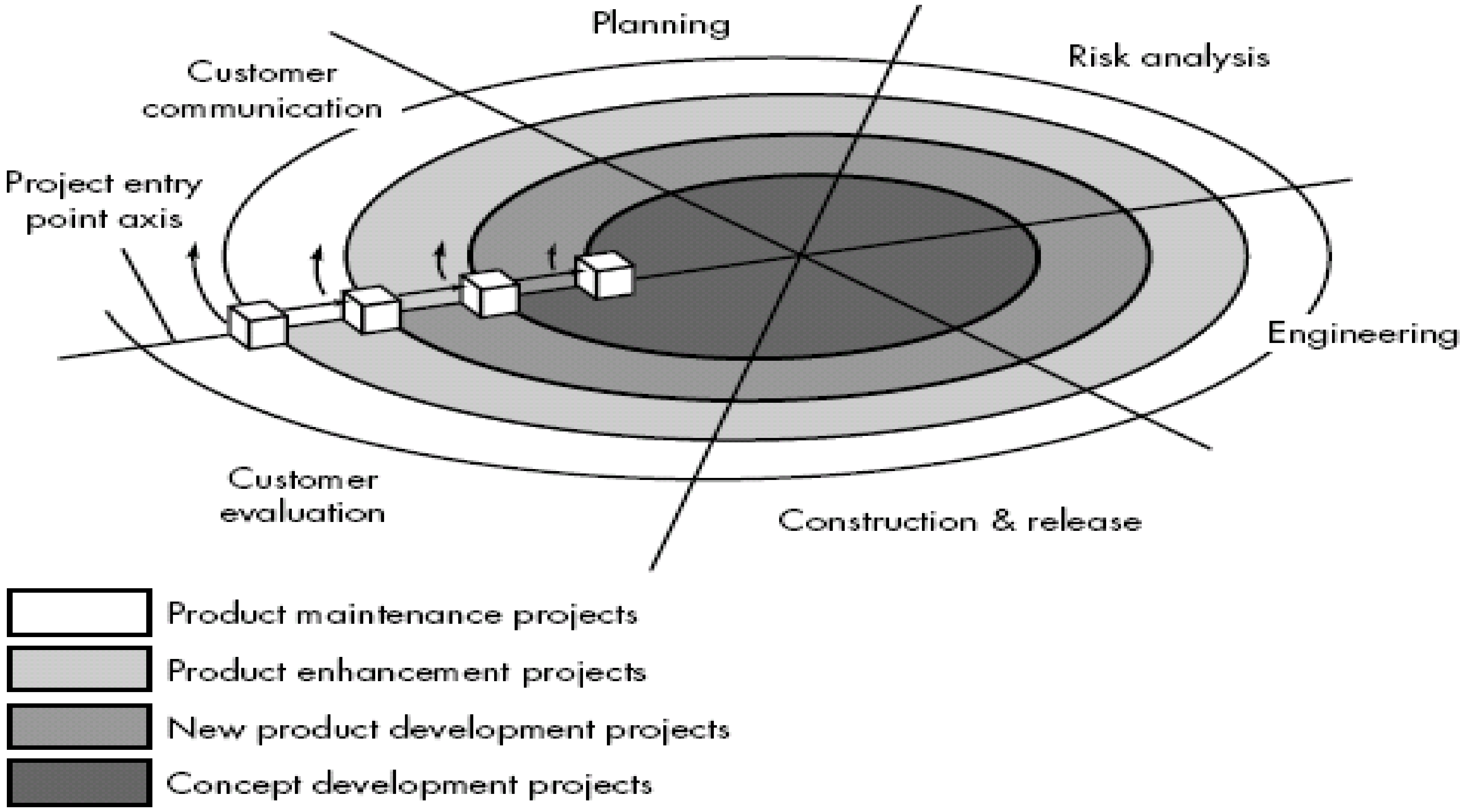
Spiral Model

- This has prototype & waterfall model
- Using spiral, software developed in as series of evolutionary release.
  - Early iteration, release might be prototype.
  - Later iteration, more complete version of software.
- Divided into framework activities (C,P,M,C,D). Each activity represent one segment.
- Evolutionary process begins in a clockwise direction, beginning at the center risk.

Spiral Model

- Early stage is a prototype
- Last stage is a complete software
- Moves in clockwise direction
- 1ST Pass: Product Specification
- 2nd pass: prototype
- 3rd pass: more sophisticated software

Spiral Model



Spiral Model

Customer communication: it is established

Planning: planning is done, resource time ,schedule

Risk analysis: technical & management risks are calculated

Engineering: tasks required to build one /more applications are carried out

Construction & release: tasks required to construct ,test & install the application

Customer evaluation: customer feedback is obtained based on customer evaluation

Advantages of spiral model:

- ✖ Requirements can be identified at every stage
- ✖ Risks can be identified & rectified
- ✖ **Disadvantages**

- Management is more complex.
- End of project may not be known early.
- Not suitable for small or low risk projects and could be expensive for small projects.
- Process is complex
- Spiral may go indefinitely.
- Large number of intermediate stages requires excessive documentation.

Thank You

GITAM
DEEMED TO BE UNIVERSITY