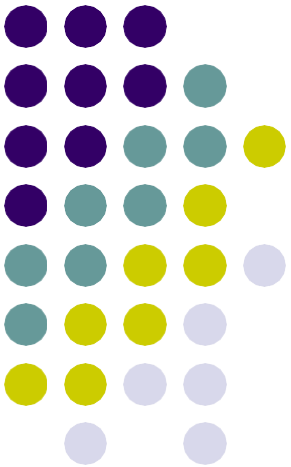
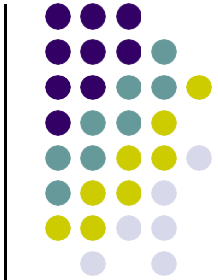
System Development Life Cycle Approaches

Anurag Srivastava

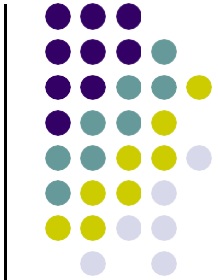


Why Life Cycle Approach for Software ?



- ***'Life cycle'*** is a sequence of events or patterns that are displayed in the lifetime of an organism.
- Software products are seen to display such a sequence of pattern during their development.
- Recognition of life cycle in software development holds the key to its successful development.

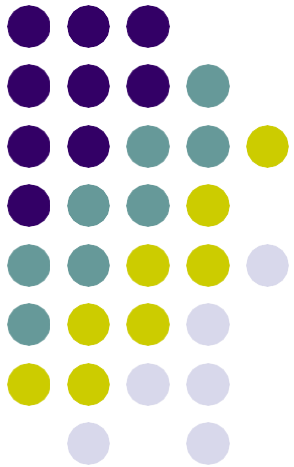
And the Models Are…



- The quality of developed software increases substantially when the development process is thoroughly managed.
- The commonly used models are—
  - *The code-and-fix model*
  - *The waterfall model*
  - *The evolutionary model*
  - *The spiral model*

SDLC(Systems Development Life Cycle)

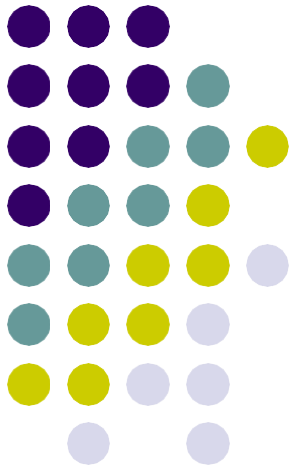
- Nearly 70% Projects fail
- Major cost sink…….hard to plan the project…
- Quality & Safety are major issues…….
- Historical knowledge/experience is minimal…and really not in shareable form…
- Professional expertise…only about 30-40 years……
- Many skills called for – business, computers, programming……



Why Bother?

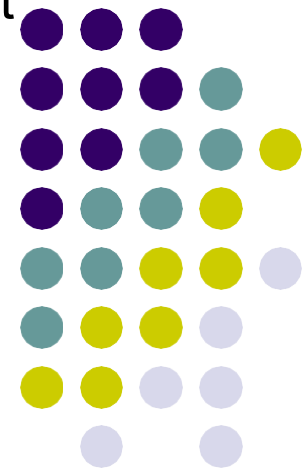
SDLC(Systems Development Life Cycle)

- Reduced risk of project failure
- Consideration of system and data requirements throughout the entire life of the system
- Early identification of technical and management issues
- Disclosure of all life cycle costs to guide business decisions
- Fostering realistic expectations of what the systems will and will not provide



SDLC(Systems Development Life Cycle)

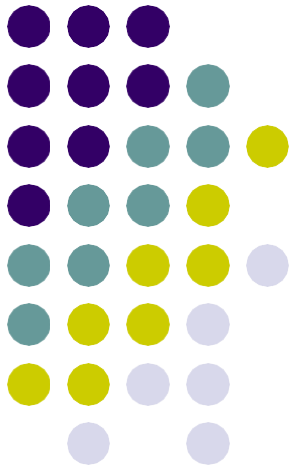
- Information to better balance programmatic, technical, management, and cost aspects of proposed system development or modification
- Encouragement of periodic evaluations to identify systems that are no longer effective
- Measurements of progress and status to enable effective corrective action
- Information that supports effective resource management and budget planning
- Consideration of meeting current and future business requirements



SDLC(Systems Development Life Cycle)

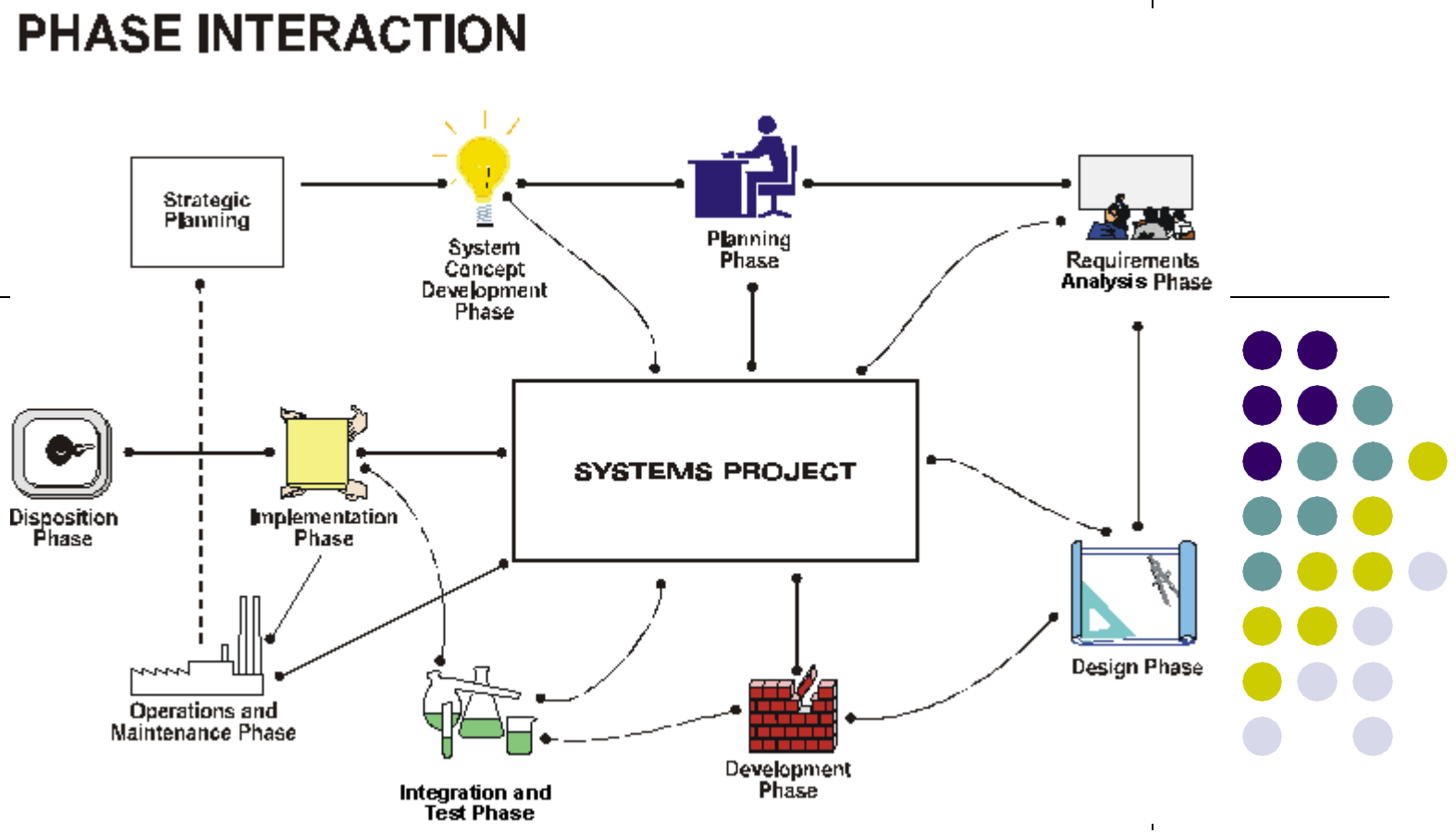
Five Steps of the SDLC

1. Investigation
2. Analysis
3. Design
4. Implementation
5. Maintenance



Most Common SDLC topology

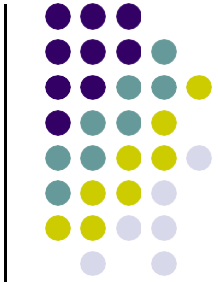
SDLC(Systems Development Life Cycle)



Multiplicity of Representations

SDLC(Systems Development Life Cycle)

Multiplicity of Representations // Private Consultant

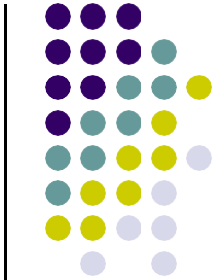


The Code-and-Fix Model

The Code-and-Fix Model (Bellingon, 1956)

- A single person develops it before using the same.
- Used during the ***early years*** of software development (the fifties and sixties)
- It was a science or an engineering application?
- The developer was also the user of the software.
- The requirements were fully known.
- The development of a software product primarily involved coding and fixing bugs, if any.

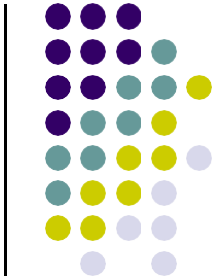
Limitations of Code-and-Fix Model



- Over the years as the scope and complexity of projects increased, this type of model was found to be highly inadequate

Code and Fix Model Example:

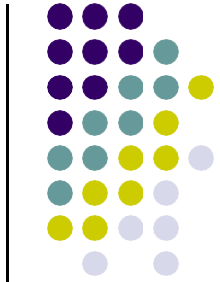
- **Scenario:** A student creates a simple "To-Do List" mobile app for a weekend hackathon.
- **Code Phase:** The developer immediately writes the UI and data storage code without formal requirements or design.
- **Fix Phase:** The developer runs the app, finds that data doesn't save after closing (a bug), and immediately adds code to fix it.
- **Iterate:** They continue testing and fixing bugs (e.g., UI crashes on rotation) until the demo is ready.



Key Characteristics:

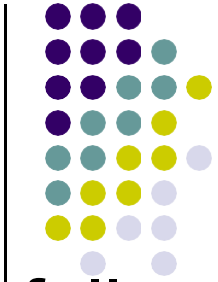
- **No Overhead:** Zero time spent on planning, documentation, or design, allowing immediate, visible progress.
- **High Risk:** Dangerous for large projects due to lack of quality control, high technical debt, and difficulty in managing complexity.
- **Low Expertise:** Requires little formal training; suitable for beginners.

This model works well for small tasks, throwaway prototypes, or R&D activities, but is generally discouraged for professional software engineering.



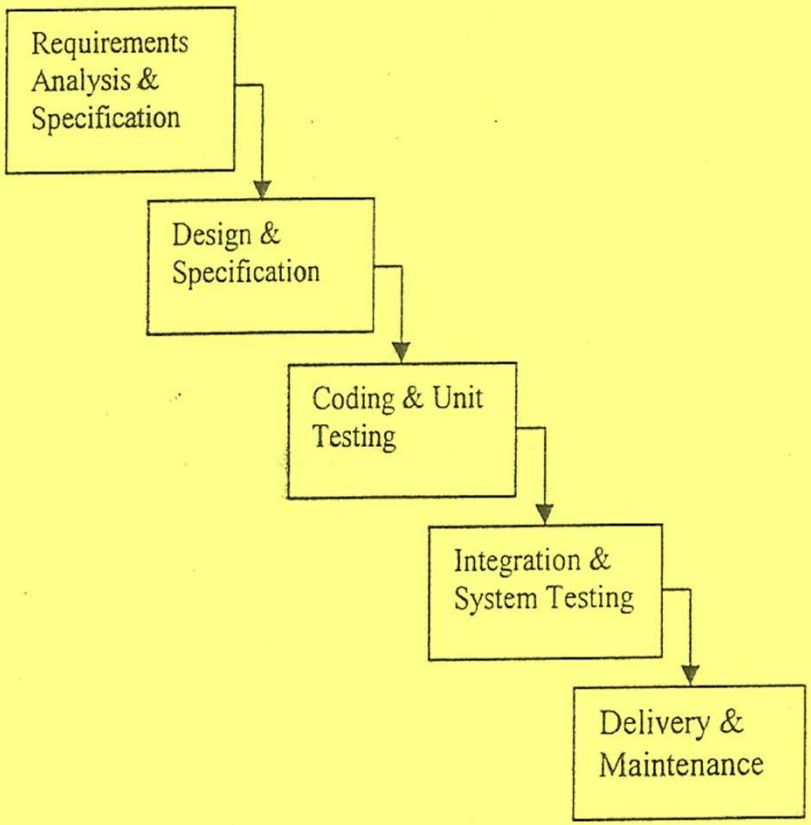
Water fall Model

Water fall Model

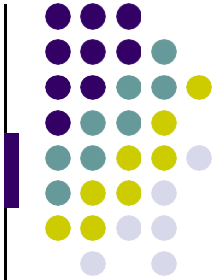


- Royce (1970) proposed the famous 'Waterfall Model' of the software development process.
- Boehm provided an economic rationale behind this model (Boehm 1976) and proposed various refinements (Boehm 1981).
- model drives its name from the structural (geometric) similarity of a software development process with a waterfall.

The Waterfall Model of Royce (1970)

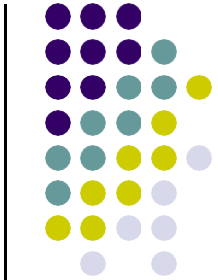


Assumptions of Waterfall Model



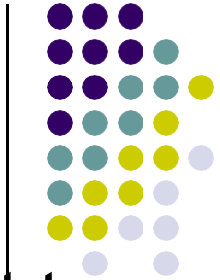
- The software development process consists of a number of ***phases*** which are in sequence.
- Only after a phase is complete, work on the next phase can start. It, thus, presupposes a ***unidirectional flow of control*** between phases.
- From the first phase (analysis) to the last phase (delivery), there is a ***downward flow*** of primary information and development effort (Sage 1995).

Assumptions…



- Work can be divided, according to phases, among different classes of ***specialists***.
- It is possible to associate a ***goal for each phase*** and accordingly to plan the deliverables (the exit condition or the output) of each phase.
- The output of one phase becomes the input (i.e. the starting point) to the next phase.

Assumptions…

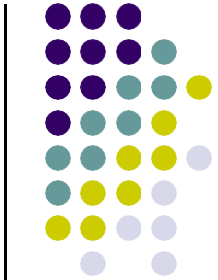


- Before the output of one phase is used as input to the next phase, it is subjected to various types of review and ***verification*** and ***validation*** testing.

verification- Checking the accuracy of the deliverable against the local goal set up by the person.

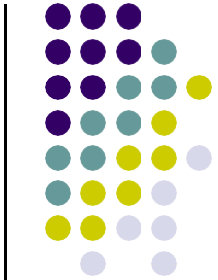
validation- Checking the accuracy of the delivered product against the bigger goal (user requirement)

Assumptions…



- Normally, the output is ***frozen***, and the output documents are signed off by the staff of the producing phase, and these become the essential documents with the help of which the work in the receiver phase starts.

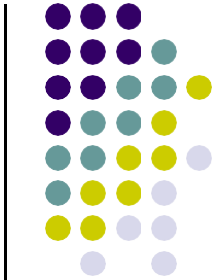
- It is possible to develop different development tools suitable to the requirements of each phase.



- The phases provide a basis for ***management and control*** because they define segments of the flow of work, which can be identified for managerial purposes, and specify the documents or other deliverables to be produced in each phase.

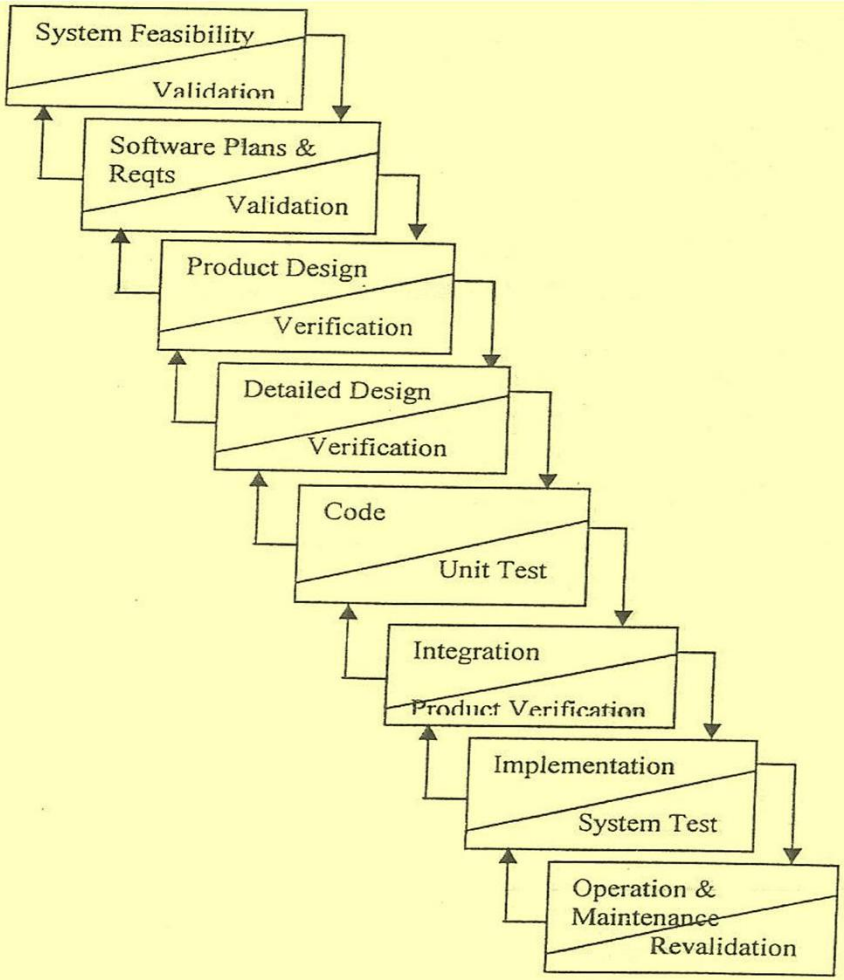
- The model thus provides a practical, ***disciplined*** approach to software development.

Strength and weakness

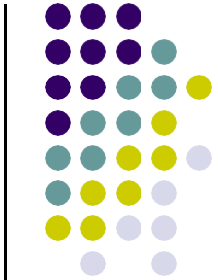


- Capability to accommodate changes as per user requirements
- Less cost involved in in scope change,
- Early delivery of working product
- Easy testing and debugging
- Each phase is rigid and does not overlap with other phase
- In absence of availability of system requirement in early stages, deciding system architecture is difficult.

Modified waterfall model by Boehm



Difference Between Royce and Boehm Model



- original model by Royce was a ***feed-forward*** model without any feedback

- whereas the Boehm's model provided a ***feedback*** to the immediately preceding phase.

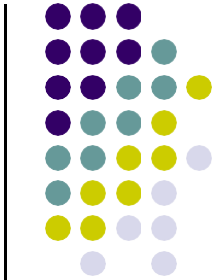
- Further, the Boehm's model required ***verification*** and ***validation*** before a phase's output was frozen.

Boehm (1976, 1981) Economic Rationale

- All the phases and their associated goals are necessary.

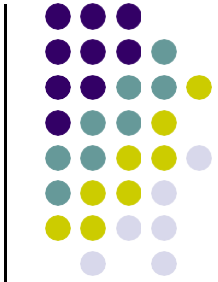
- Any different ordering of the phases will produce a less successful software product.

Usefulness of Waterfall Model



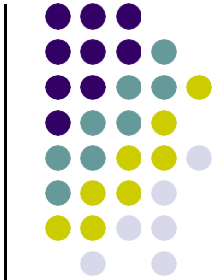
- The model encourages one to specify what the system is supposed to do (i.e., define the requirements) before building the system (i.e., designing).
- It encourages one to plan how components are going to interact (i.e. designing before building the components-coding).
- It enables project managers to track progress more accurately and do uncover possible slippages early.

Usefulness…



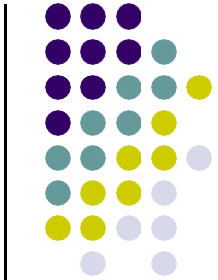
- It demands that the development process generates a series of documents that can be utilized later to test and maintain the system.
- It reduces development and maintenance costs due to all of the above-mentioned reasons.
- It enables the organization that will develop the system to be more structured and manageable.

Limitations of Waterfall Model (Royce Jr. , 2000)



- Protracted integration and Late Design Breakage.
- Late risk Resolution.
- Requirements-Driven Functional Decomposition
- Adversarial Stakeholder Relationships.

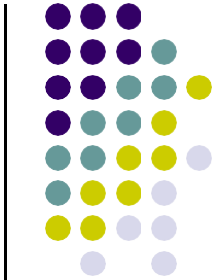
Limitations of Waterfall Model



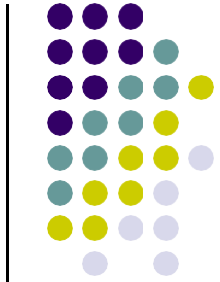
- The waterfall model is ***rigid***. The phase rigidity, that the results of each phase are to be frozen before the next phase can begin, is very strong.

- In the waterfall model, ***errors*** found at a later stage of the development cost more and take much longer to fix.

Limitations…

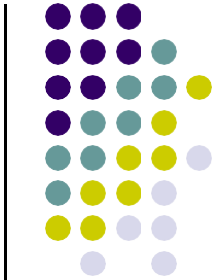


- It is ***monolithic***. The planning is oriented to a single delivery date.
- Not knowing if the software being developed meets the customer expectations till almost implementation stage. There is little time to make the alterations.
- The model is heavily ***document driven*** to the point of being bureaucratic.



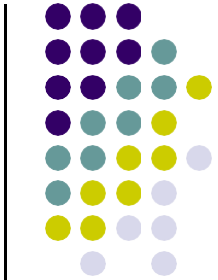
The Evolutionary Model

The Evolutionary Model



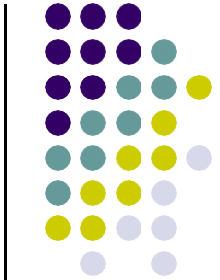
- This model has distinct ***custom orientation***.
- In an evolutionary model, working models of software are developed and presented to the customers for his feedback, for consideration (read incorporation) in the final version of the software.

Types of Evolutionary Models

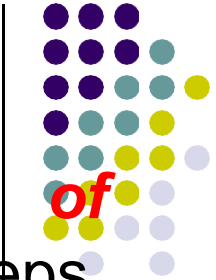


- Incremental implementation model
- Prototyping Model
  - --***Rapid/Throwaway Prototype***
  - ***--Evolutionary Prototype***

The incremental implementation (Boehm 1981, Gilb 1988)



- Develop small ***basic part*** of the software to begin with and add on other features (or functionality).
- This approach reduces the costs incurred before an initial capability is achieved.
- This also produces an operational system more ***quickly*** & thus reduces the possibility that the user needs will change during development process



- Here the software is developed in ***increments of functional capability***; i.e. the development is in steps, with parts of some stages postponed in order to produce useful working functions earlier in the development of the project.

- Other functions are slowly added later as increments.

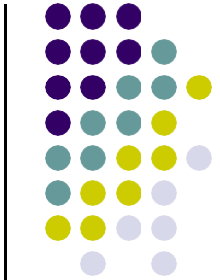
- Thus, while analysis and design are done following the waterfall process model, coding, integration, and testing are done in an incremental manner.

Advantages of Incremental approach

- Users can give suggestions on the parts to be delivered at later points of time.

- The developers engage themselves in developing the most fundamental functional features to the software's in its first increment. Thus, these features get the maximum, and the most concentrated attention from the developers. Therefore, there is likelihood that the programs are error free.

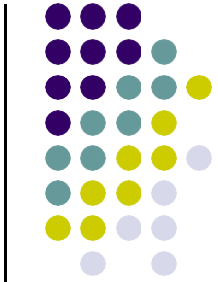
Advantages…



- The time to show some results to the users is considerably reduced. User reactions, if any, can therefore be incorporated in the software with great ease.

- Testing, error detection and error correction become relatively easy tasks.

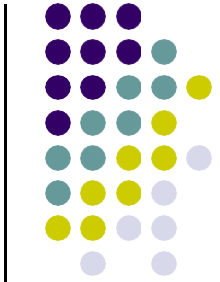
Limitations of Incremental approach



- The overall architectural framework of the product must be established in the beginning and all increments must fit into this framework.

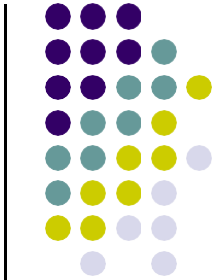
- A customer-developer contract oriented towards incremental development is not very usual.

Prototyping Model



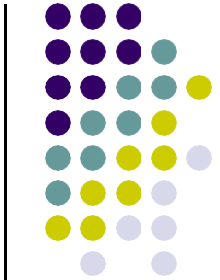
- This method is based on an experimental procedure whereby a working prototype of the software is given to the user for comments and feedback. It helps the user to express his requirements in more definitive and concrete terms. Prototyping can be of 2 types:
- The ***rapid throwaway*** prototyping and
- ***Evolutionary*** prototyping

Rapid Throwaway prototyping



- It follows the ***'do it twice'*** principle advocated by Brooks(1975).
- Here the ***working prototype*** produced quickly is subject to large changes from customer and hence it becomes very costly to work on.
- So the working prototype is actually ***thrown*** and the software is developed once again from the beginning.

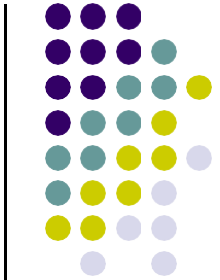
Evolutionary Prototyping



- In case of evolutionary prototyping, the initial prototype is not thrown away. Instead, it is progressively transformed into the final application.

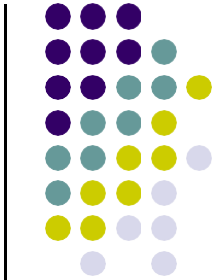
- Here the initial prototype is prepared with care and time and hence subject to minor changes from the user side

Evolutionary Vs Throwaway Prototyping



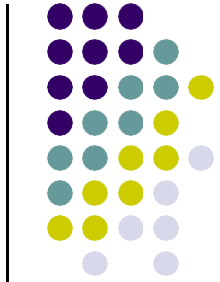
- Both assume some incomplete set of requirement to be identified.
- Both allows user feedback.
- An evolutionary prototyping is continuously modified as user gives his feedback and is satisfied. A throwaway prototype allows users to give feedback only once. Such feedback provides the basis for specifying a complete set of requirement specifications.

Limitations of Prototyping



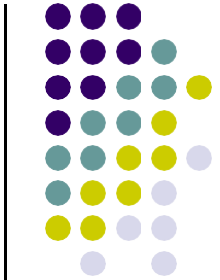
- Various revisions carried on evolutionary prototyping usually result in a bad program structure and make it quite bad from maintainability point of view.

- A through away prototype is usually unsuitable for testing non functional requirements.



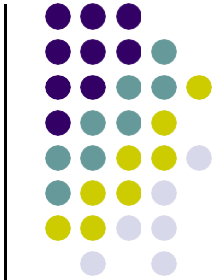
The Spiral Model

The Spiral Model (Boehm, 1988)



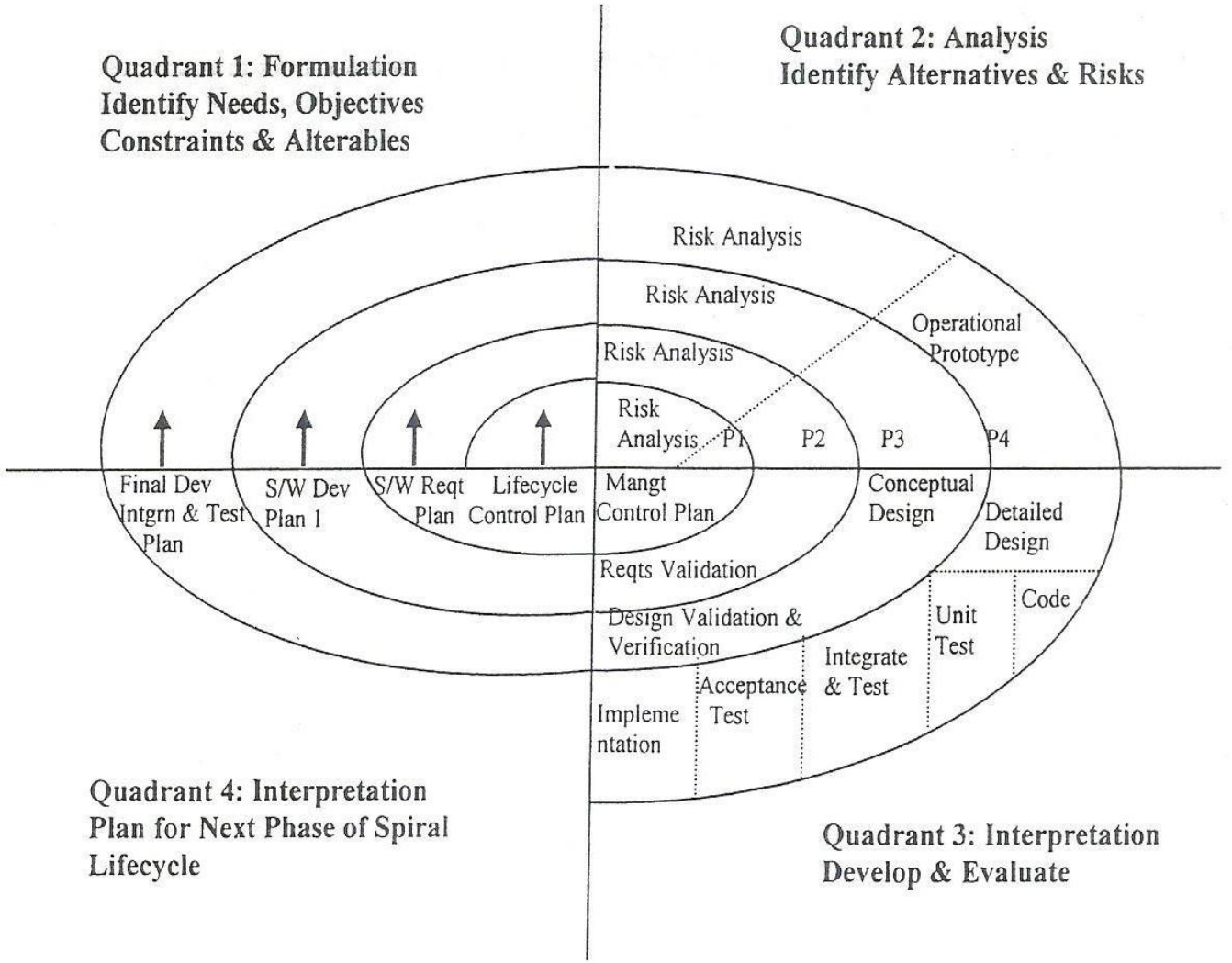
- It integrates the characteristics of the waterfall model, the incremental implementation and the evolutionary prototyping approach.
- In this sense, it is a *metamodel* (Ghezzi et al.1994).

Features of Spiral Model



- The process of the software development can be depicted in the form of a spiral that moves in a clockwise fashion.
- Each cycle of the spiral depicts a particular phase of software development life cycle.

Spiral Model

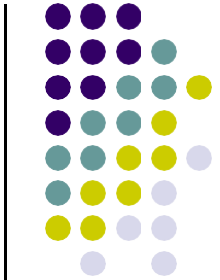


- Each quadrant of the spiral corresponds to a particular set of activities for all phases. The four sets of activities are the following:

1. *Determine objectives, alternatives and constraints*
2. *Evaluate alternatives, identify and resolve risks with the help of prototypes.*
3. *Develop and verify next-level product, and evaluate.*
4. *Plan next phases.*

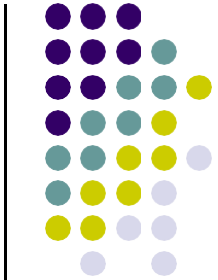
- The radius of the spiral represents the cumulative cost of development;
- the angular dimension represents the progress;
- the number of cycles represents the phase of software development at a particular point of time.
- An important feature of the spiral model is the explicit consideration (identification and elimination) of risks.
- The number of cycles that is required to develop a piece of software is of course dependent upon the risks involved.

Two Other Software Development Models (Davis et al., 1988)



- ***Reusable software***, whereby previously proven designs and code are reused in new software products.
- ***Automated software synthesis***, whereby user requirements or high level design specifications are automatically transformed into operational code by either algorithmic or knowledge-based techniques using very-high-level languages.

Software Reuse Model



- Many software firms develop, in course of time, libraries of generic reusable software components that can be reassembled to give shape to new software products.

  Not only codes, but designs and specifications can also be reused.

  Components that can be reused may be

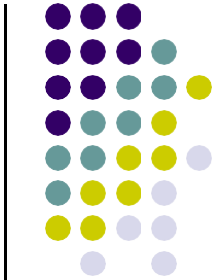
  (1) a whole application system,

  (2) major subsystems of an application,

  (3) modules or objects, or

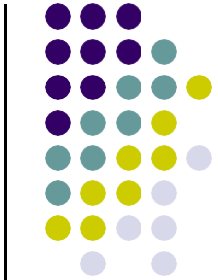
  (4) functions.

Advantages of Reuse



- Reduced development effort and time
- Reduced defect density and high reliability
- Reduced overall project risk due to less uncertainty in cost estimates
- Effective use of specialists in developing generic components than in a wide array of variety of products.
- Standardising reusable components
- Even though costly to begin with, reuse is cost effective in the long run

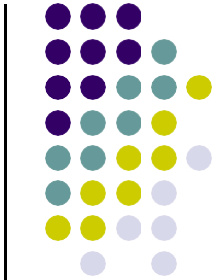
Automatic Software Synthesis



- Program generators for stereotypical functions and code generators in CASE tools are examples of automatic software synthesis.

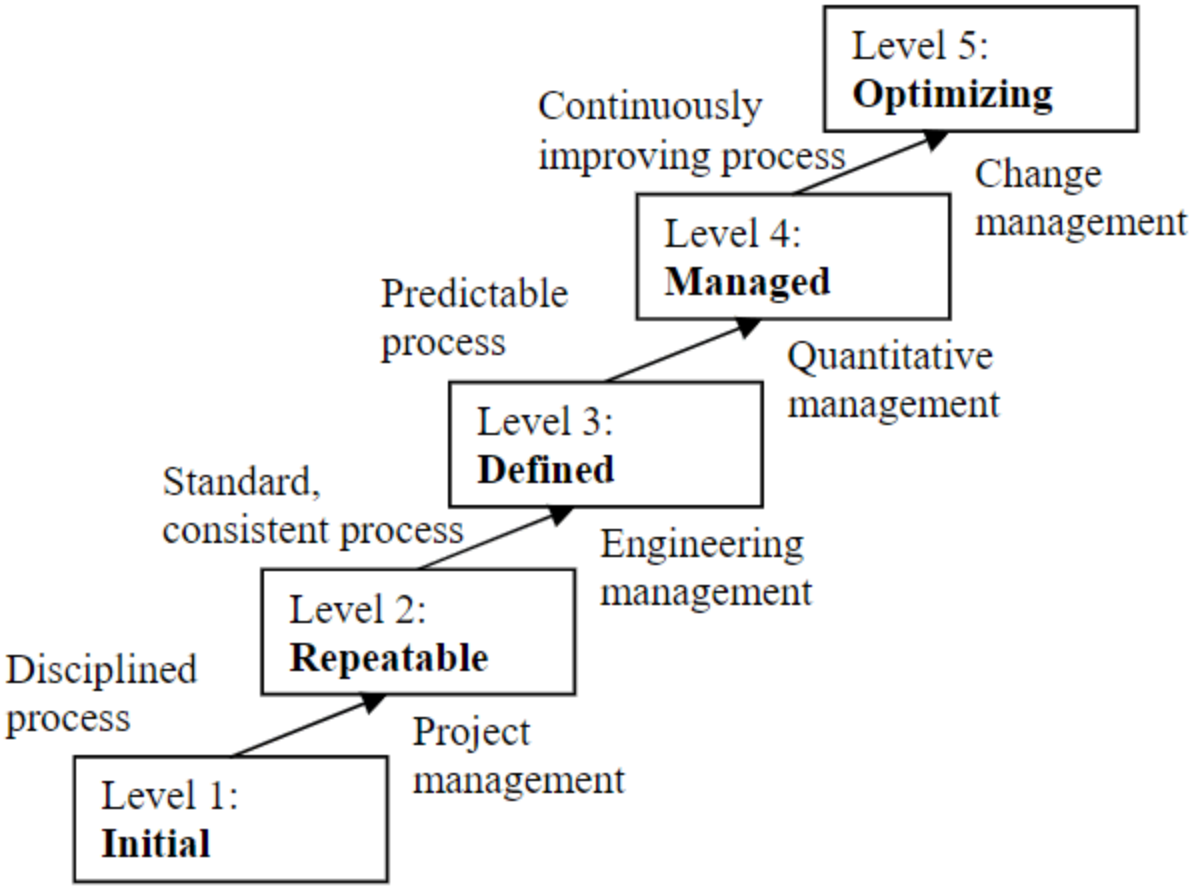
- They are very useful in generating codes for such functions as Creating screens, Editing input data, Preparing reports, Processing transactions, and Updating database.

Other Models

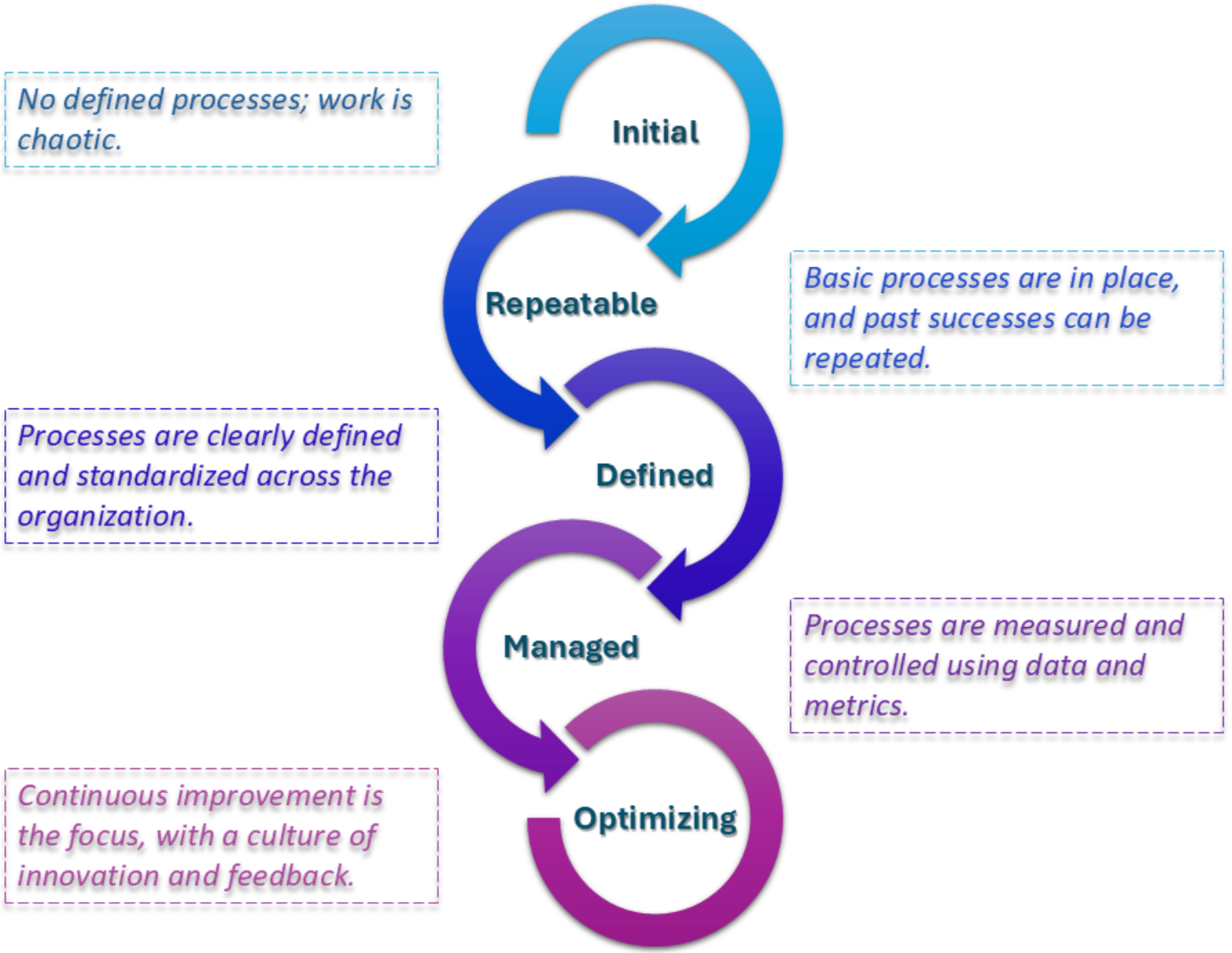


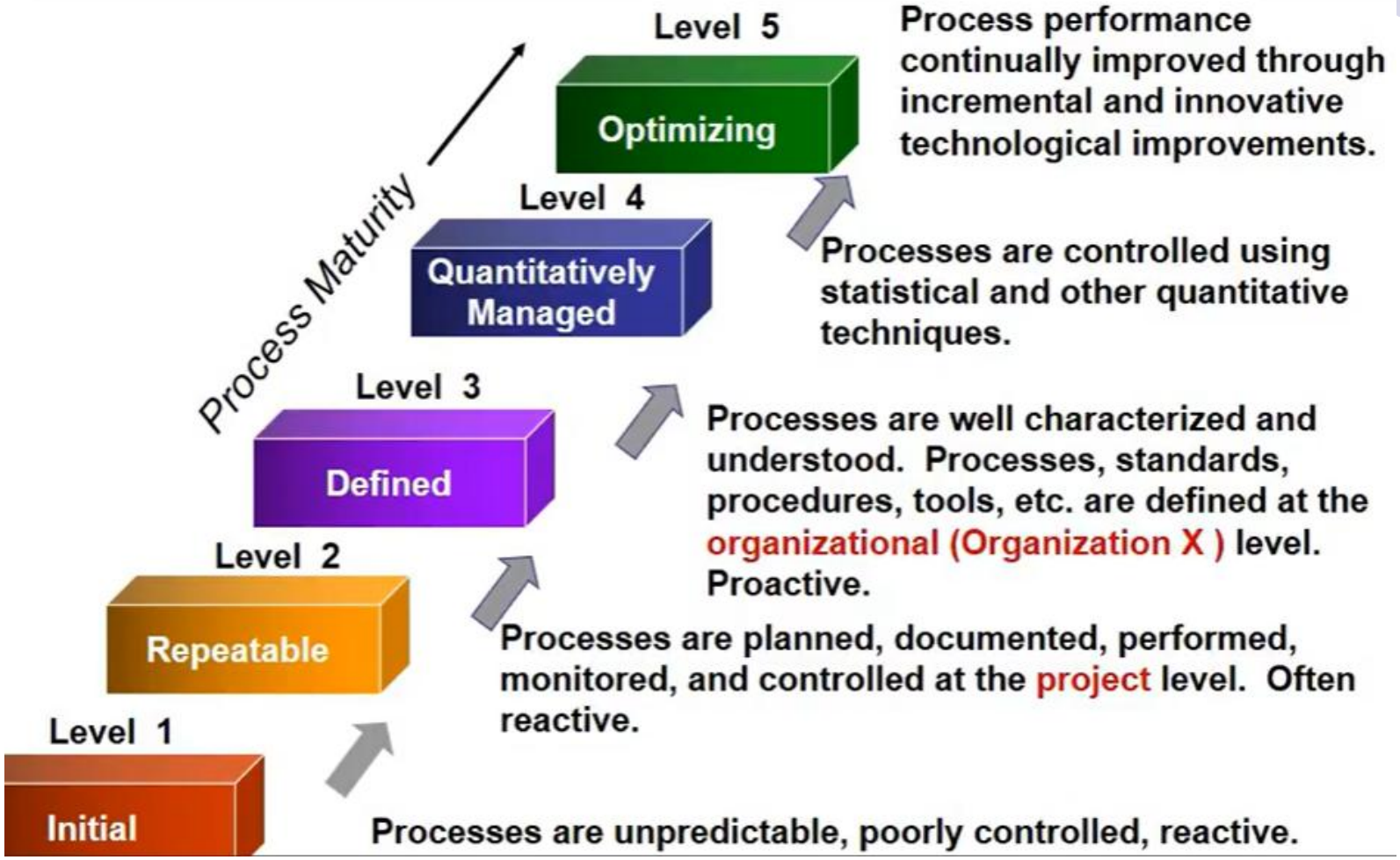
- Capability Maturity Model
  - Project and process management
- The Six Sigma

Capability Maturity Model

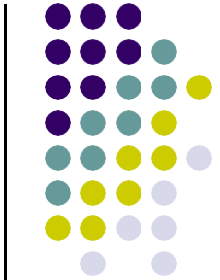


Capability Maturity Model (CMM), which provides a clear roadmap for organizations to improve their processes over time. As companies move through different maturity levels, their processes become more controlled, efficient, and optimized, ultimately leading to better customer satisfaction and stronger business results.




Level 5
Optimizing
Process performance continually improved through incremental and innovative technological improvements.
Level 4
Quantitatively Managed
Processes are controlled using statistical and other quantitative techniques.
Process Maturity
Level 3
Defined
Processes are well characterized and understood. Processes, standards, procedures, tools, etc. are defined at the organizational (Organization X) level. Proactive.
Level 2
Repeatable
Processes are planned, documented, performed, monitored, and controlled at the project level. Often reactive.
Level 1
Initial
Processes are unpredictable, poorly controlled, reactive.

Six Sigma Model

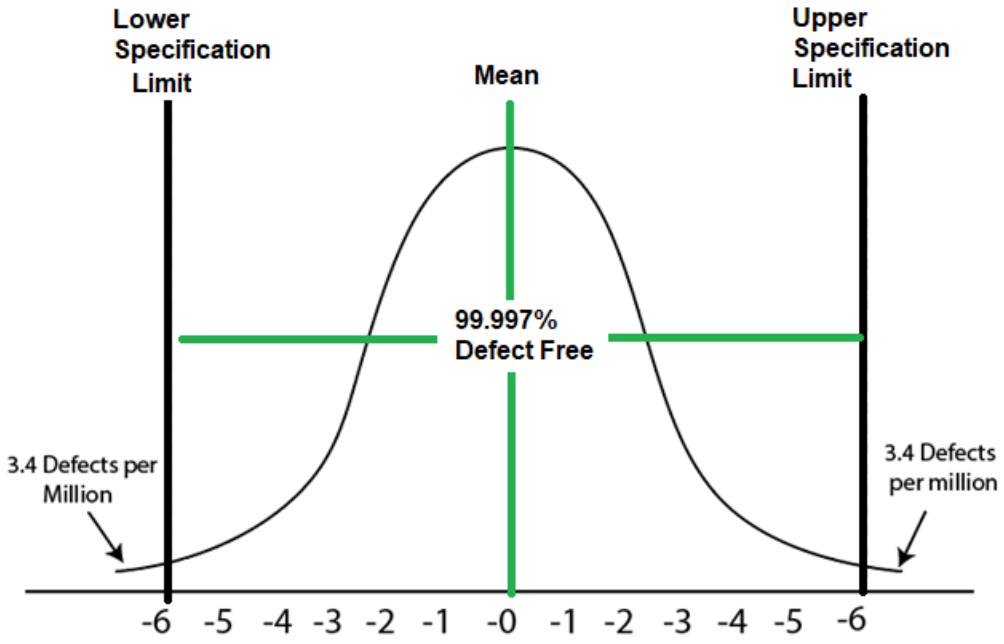


Six Sigma is a methodology that helps organizations in making their process better and more efficient by identifying and removing errors and variations. Variations in processes can lead to errors, these errors can lead to product defects and product defects can lead to poor customer satisfaction. By reducing variation and errors Six Sigma can reduce process costs and increase customer satisfaction. Six Sigma was introduced in 1986 by an American Engineer Bill Smith. He introduced this term while working at Motorola. Industries like manufacturing, service industry, government agencies, aerospace, and e-commerce use Six Sigma to improve their processes and product quality.

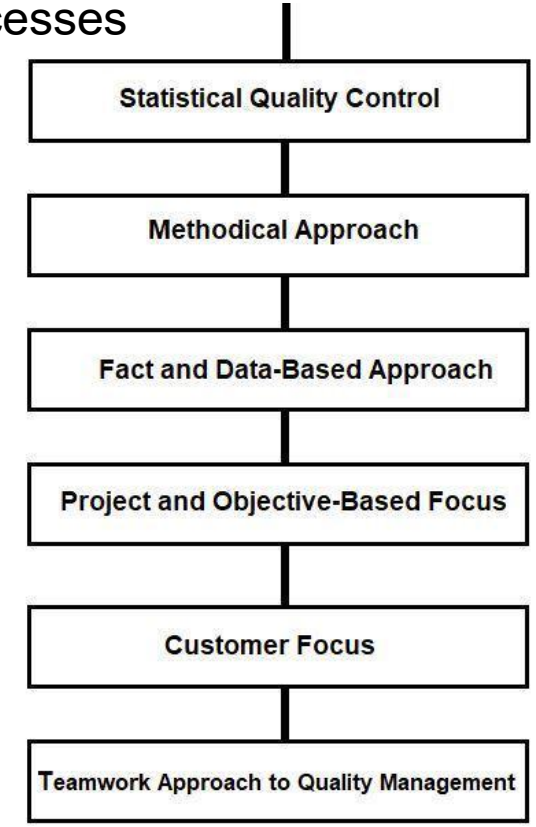
Six Sigma Model

What is Six Sigma?

Six Sigma is a methodology used by most organizations for process improvement, and It is a statistical concept that aims to define the variation found in any process. Six Sigma is a process of producing high and improved quality output. This can be done in two phases - identification and elimination. The cause of defects is identified and appropriate elimination is done, which reduces variation in whole processes. Six Sigma processes have a failure rate of only 3.4 per million opportunities i.e. 99.99966 percent of Six Sigma products are free from defect, while Five Sigma processes have a failure rate of only 233 errors per million opportunities.

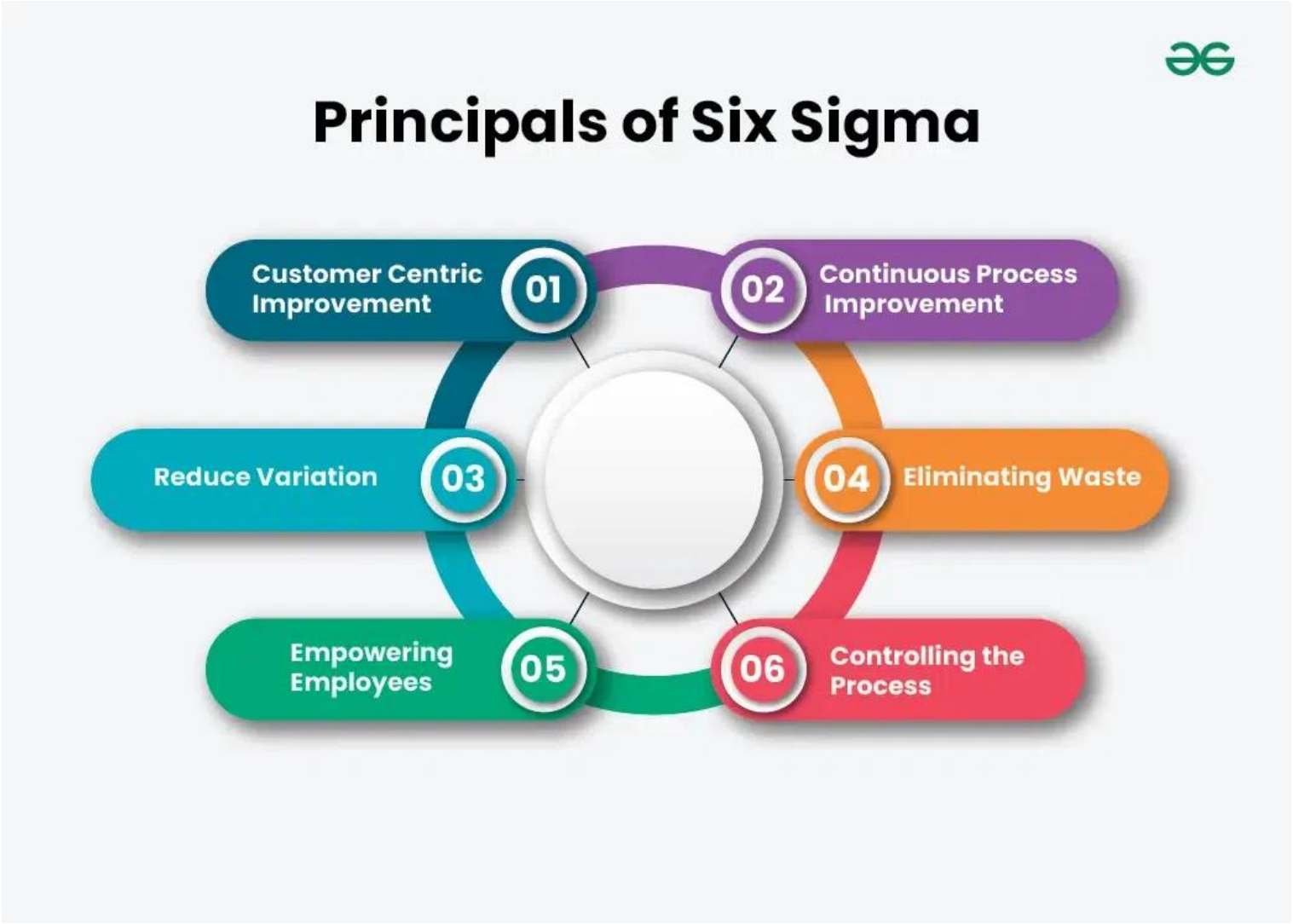


Six Sigma Curve

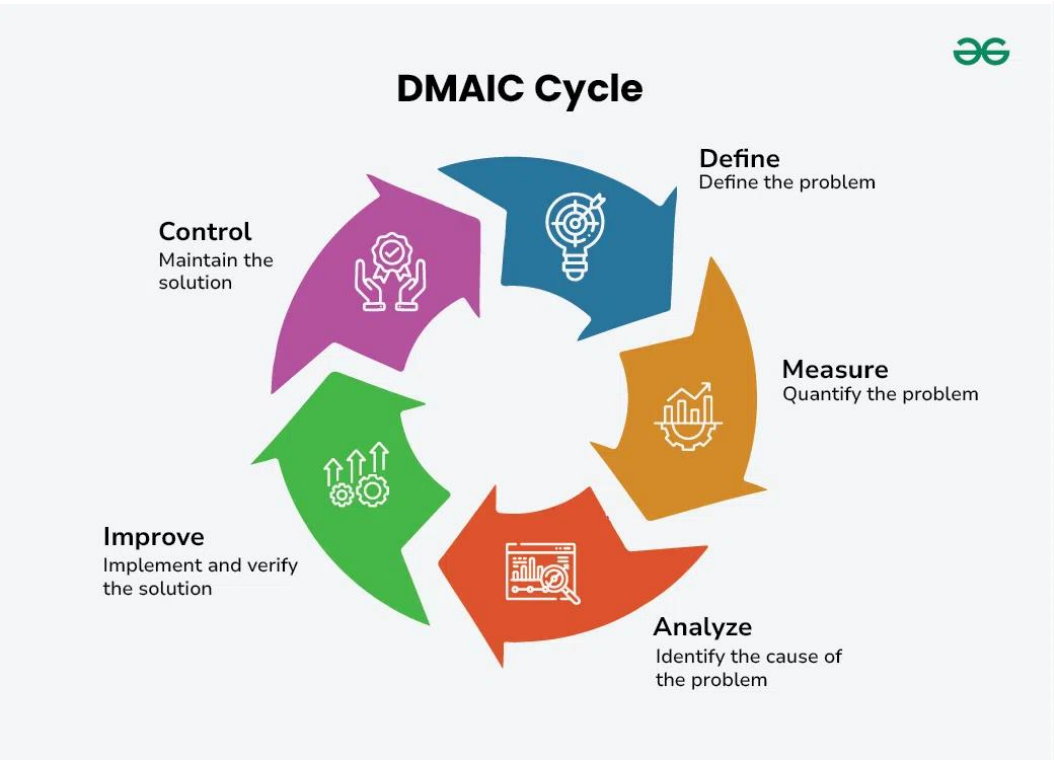


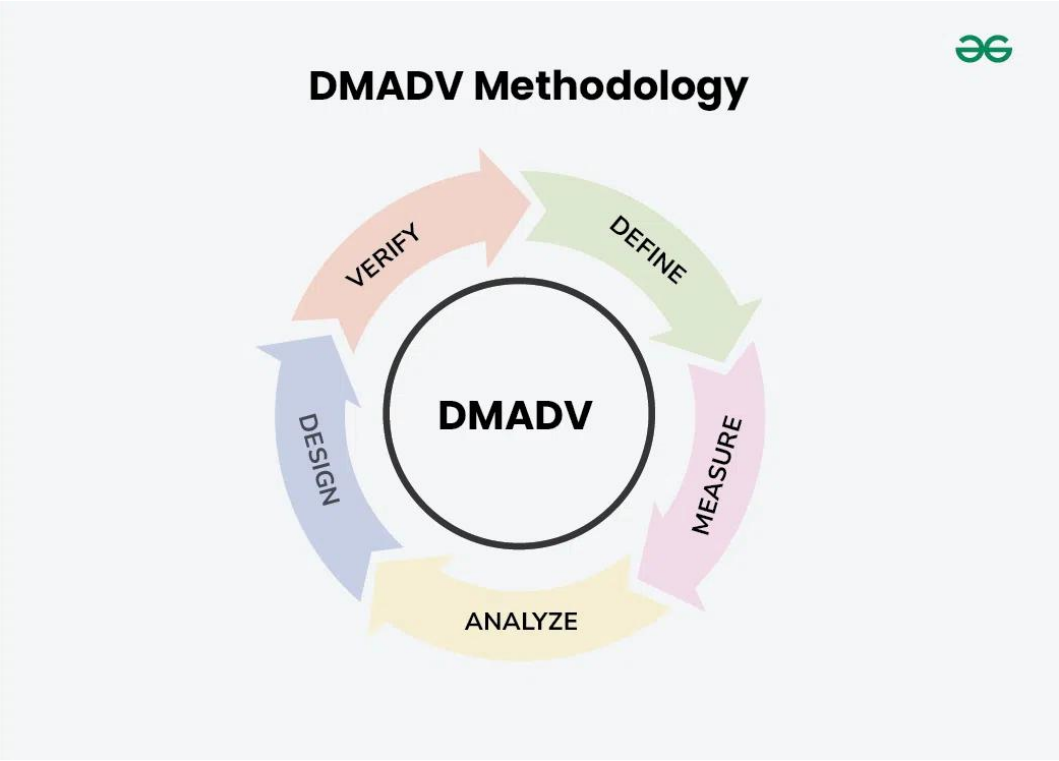
Characteristics of Six Sigma

Six Sigma Model



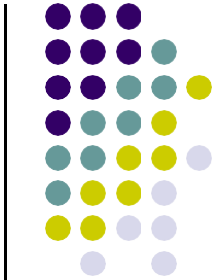
Six Sigma Model





Levels of Six Sigma Certification

Comparing SDLC models

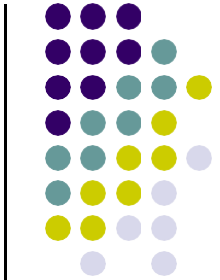


- Waterfall model has ***sequence*** of phases with limited feedback and iterations between phases, prototype works on number ***iterations*** between developer and user while incremental is ***additive*** in nature.
- Waterfall model is ***document*** based, Evolutionary approach is ***user*** based, Spiral approach is ***risk*** based.

How Models fare?

Risk Item	Waterfall	Incremental	Prototyping
System too large for one time build	High	Medium	Low
User Requirements Not Understood Enough to Specify	Medium	Medium	Low
Rapid Changes Expected in Technology	High	Medium	Low
Limited Staff or Budget	Medium	High	Very High
Volatile System Requirements	Very High	High	Medium

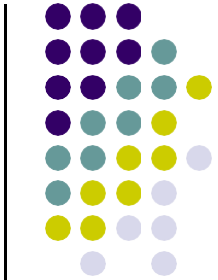
Phase wise Distribution of Effort



40-20-40 rule:

- Analysis and Design: 40% of the total effort
- Coding and Debugging: 20% of the total effort
- Testing and checkouts: 40% of the total effort

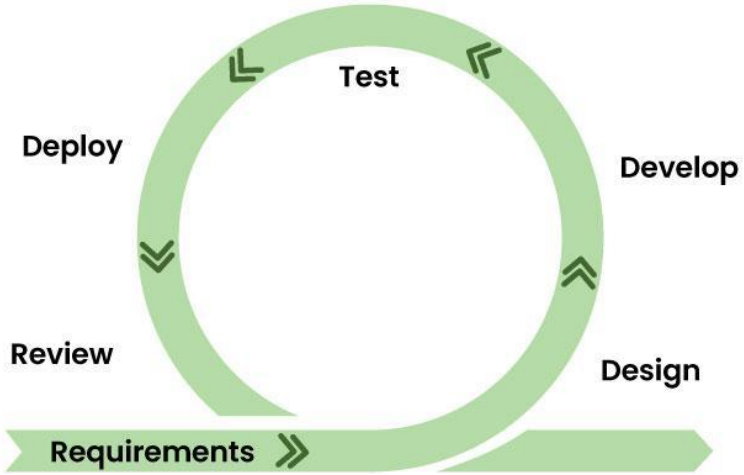
Iterative SDLC models vs Waterfall SDLC Models



Aspect	Iterative SDLC Models	Waterfall SDLC Models
Development Approach	Cyclical and Incremental	Sequential and Linear
Phases	Planning, Design, Coding, Testing, Evaluation (Repeated Iteratively)	Requirements, Design, Implementation, Testing, Deployment
Flexibility	High - Embraces Changes and Enhancements Throughout Development	Low - Changes are Difficult to Accommodate After Initial Phases
Risk Management	Proactive - Risks Addressed Throughout Iterations	Reactive - Risks Addressed in a Linear Manner
Time-to-Market	Gradual Releases, Quicker Time-to-Market for Incremental Features	Single Release at the End, Potentially Longer Time-to-Market
User Involvement	Continuous User Feedback and Involvement Throughout	Limited User Involvement until the Testing Phase
Testing	Continuous Testing Throughout Iterations	Testing Conducted After the Completion of the Implementation
Adaptability	Highly Adaptable to Changing Requirements	Less Adaptable, Changes May Be Costly and Time-Consuming
Complexity Management	Easier to Manage and Control Complexity	Complexity Management Challenging Due to Sequential Approach

Agile SDLC Models

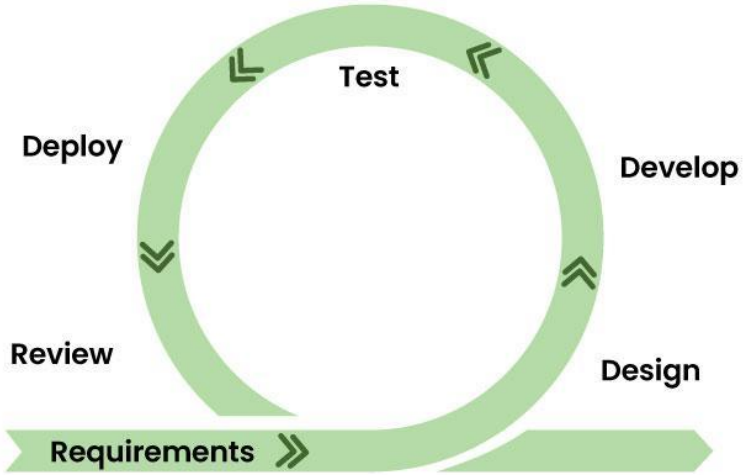
Agile is not a specific methodology but rather a set of principles and values outlined in the Agile Manifesto. The Agile Manifesto prioritizes individuals and interactions, working solutions, customer collaboration, and responding to change over rigid processes and documentation. Several Agile methodologies, including Scrum, Kanban, and Extreme Programming (XP), have been developed to implement these principles.



Agile Model

Agile SDLC Models

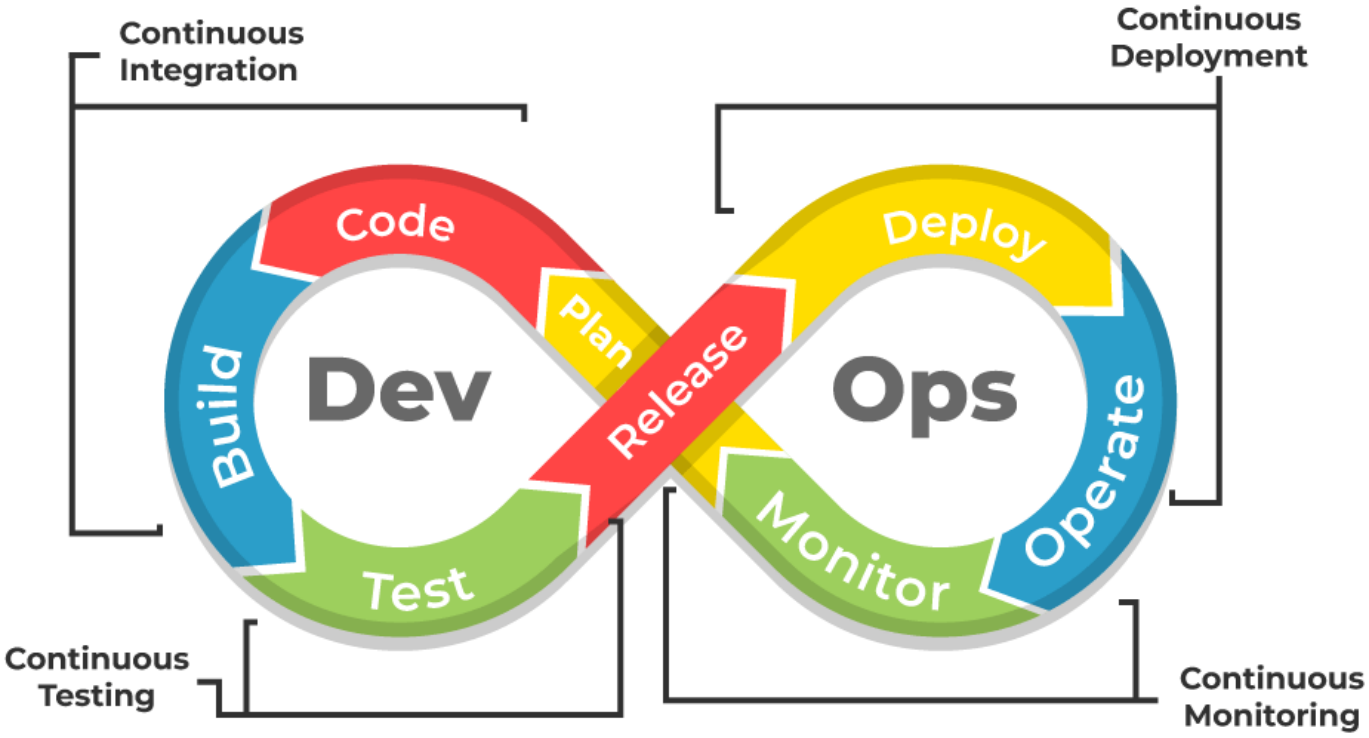
Agile is not a specific methodology but rather a set of principles and values outlined in the Agile Manifesto. The Agile Manifesto prioritizes individuals and interactions, working solutions, customer collaboration, and responding to change over rigid processes and documentation. Several Agile methodologies, including Scrum, Kanban, and Extreme Programming (XP), have been developed to implement these principles.



Agile Model

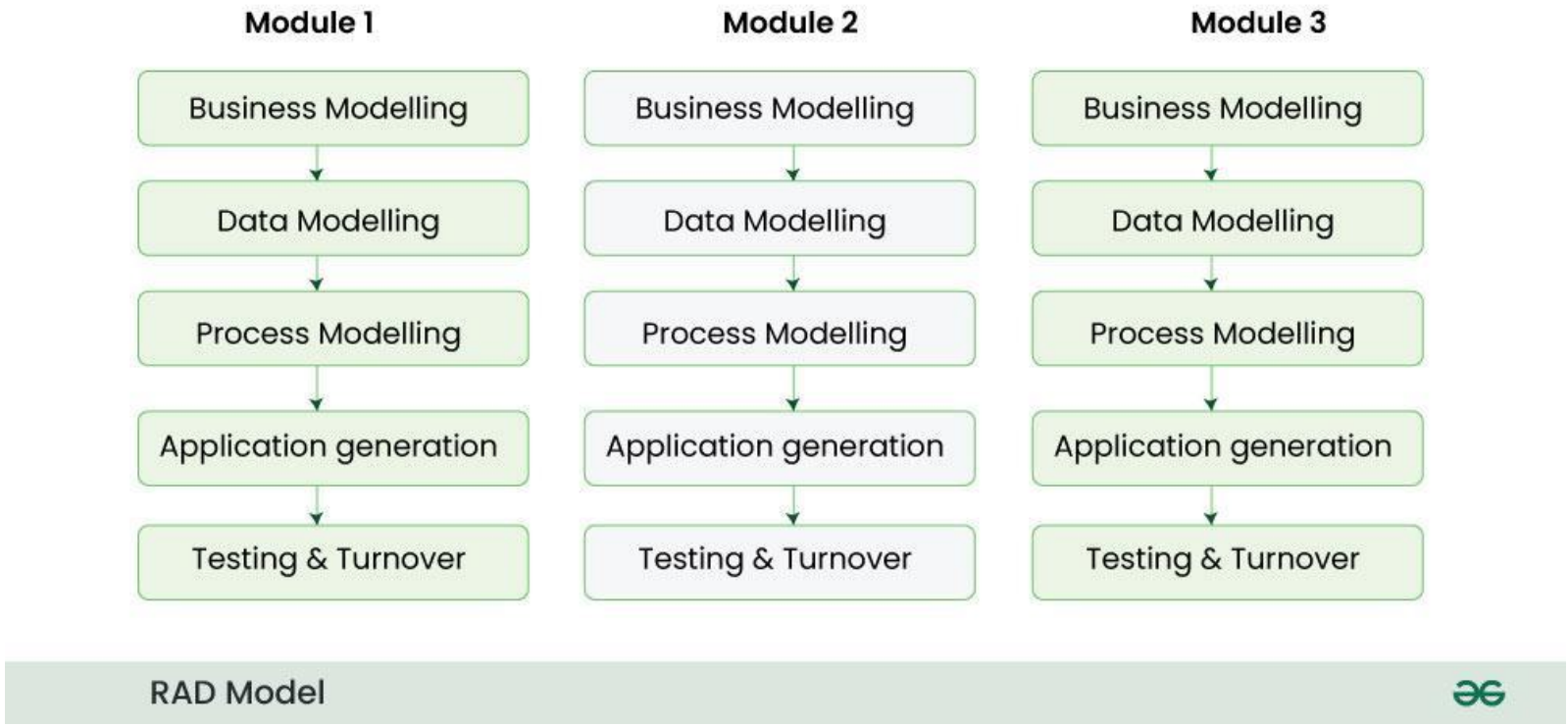
DevOps SDLC Models

DevOps, comprised of "development" and "operations," represents a cultural and organizational shift in how software is developed, tested, and deployed. It emphasizes collaboration and communication between software developers and IT operations, promoting automation and continuous delivery. DevOps is not just a set of practices; it is a cultural mindset that seeks to improve collaboration and efficiency across the entire software development lifecycle.



Rapid Application Development (RAD) SDLC Models

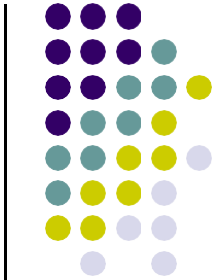
Rapid Application Development is an iterative and incremental model that prioritizes quick development and iteration cycles. It places a strong emphasis on user feedback and involvement throughout the development process. RAD aims to deliver functional prototypes rapidly, allowing stakeholders to provide feedback and guide ongoing development.



Comparison between various SDLC Models

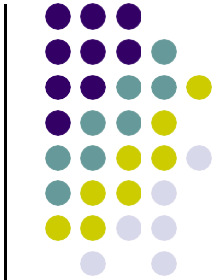
Aspect	Waterfall	Iterative	Spiral	Agile	V-Models	Incremental
Development Approach	Sequential	Iterative	Iterative	Iterative	Iterative	Iterative
Phases	Linear	Planning, Design, Coding, Testing, Evaluation (Repeated Iteratively)	Planning, Risk Analysis, Engineering, Testing (Cyclical)	Planning, Sprint, Review, Retrospective (Iterative Cycles)	Planning, Design, Implementation, Testing, Deployment (Parallel)	Divided into increments, each with Planning, Implementation, Testing
Flexibility	Low	High	High	High	Moderate	High
Risk Management	Late mitigation, Limited adaptability	Proactive risk management, Adaptability to changes	Continuous risk assessment, Proactive mitigation	Continuous risk assessment, Adaptability to changes	Risk management aligned with phases, Moderate adaptability	Proactive risk management, Adaptability to changes
Time-to-Market	Longer	Faster	Variable	Faster	Moderate	Faster
User Involvement	Limited	Continuous	Periodic	Continuous	Periodic	Continuous
Testing	After Implementation	Continuous throughout iterations	Integrated throughout the spiral	Continuous and c	After Implementation	Continuous throughout increments
Adaptability	Low	High	High	High	Moderate	High
Complexity Management	Linear approach, Limited adaptability	Easier to manage, Adaptability to changes	Cyclical approach, Risk-d	Adaptive approach to changes	Traceability helps manage c	Adaptive approach to changes

When to use which SDLC models?



Consideration	Waterfall	Iterative	Spiral	Agile	V-Models	Incremental
Project Size	Small to Medium	Medium to Large	Large	Small to Medium	Medium to Large	Small to Large
Project Complexity	Low to Medium	Medium to High	High	Low to High	Medium to High	Medium to High
Requirements Stability	Stable	Can evolve	Can evolve	Likely to change frequently	Moderate stability	Stable to Moderate Stability
Client Involvement	Limited	Continuous	Periodic	High and Continuous	Periodic	Continuous
Budget Constraints	High Predictability, Fixed Budget	Moderate Predictability, Some Flexibility	Some Flexibility	Variable, Suitable for Changing Budgets	Moderate Predictability, Fixed Budget	Moderate Predictability, Some Flexibility
Risk Tolerance	Low	Moderate	High	Moderate	Moderate	Moderate to High
Time-to-Market	Moderate	Faster	Variable	Faster	Moderate	Faster
Documentation Emphasis	Extensive Documentation	Moderate Documentation	Detailed Documentation	Minimal Documentation	Moderate Documentation	Moderate Documentation
Testing Approach	Sequential Testing after Development Phases	Continuous Testing Throughout Iterations	Continuous Testing Throughout the Spiral	Continuous and Collaborative Testing	Testing Conducted After the Completion of Phases	Continuous Testing Throughout Increments
Change Management	Limited Flexibility	High Flexibility	Adaptive to Changes	Highly Adaptive to Changes	Moderate Flexibility	High Flexibility

How to Choose an SDLC Model?



Selecting the right Software Development Life Cycle (SDLC) model is crucial for project success. The SDLC defines how software is planned, built, tested, and maintained, and the right model aligns with project goals, team skills, and constrain.

1. Project Requirements:

- Clear and Stable Requirements: Use Waterfall or V-Model when requirements are well-defined and unlikely to change.
- Evolving or Unclear Requirements: Use Agile or Iterative models when requirements are likely to change.

2. Project Size and Complexity:

- Small and Simple Projects: Use Waterfall or RAD for straightforward projects with limited scope.
- Large and Complex Projects: Use Agile, Spiral, or DevOps for projects requiring flexibility and scalability.

3. Team Expertise:

- Experienced Teams: Use Agile or Scrum when teams are skilled in iterative development.
- Less Experienced Teams: Use Waterfall or V-Model for teams needing clear guidance.

4. Client Involvement:

- Frequent Client Feedback: Use Agile, Scrum, or RAD when regular client interaction is required.
- Minimal Client Involvement: Use Waterfall or V-Model when client input is limited to initial stages.

5. Time and Budget Constraints:

- Fixed Time and Budget: Use Waterfall or V-Model for strict schedules and budgets.
- Flexible Time and Budget: Use Agile or Spiral when adjustments are feasible.

6. Risk Management:

- High-Risk Projects: Use Spiral for projects with significant uncertainties.
- Low-Risk Projects: Use Waterfall for projects with minimal risks