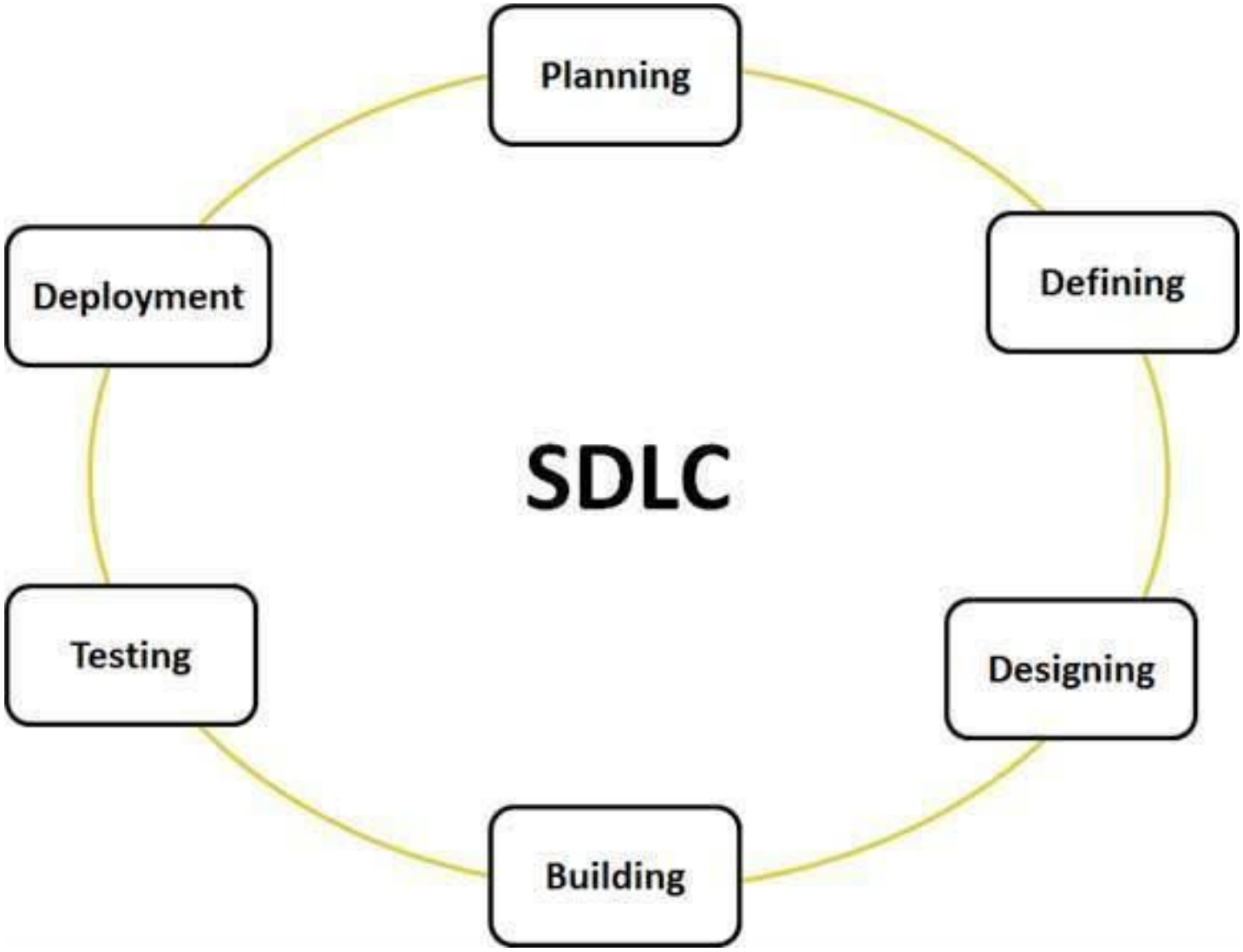
Software Development Life Cycle (SDLC)

What is SDLC?

- SDLC is a process followed for a software project, within a software organization. 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.


Planning
Defining
Designing
Building
Testing
Deployment
SDLC

Stages/Phases of SDLC

Stage 1: Planning and Requirement Analysis

Stage 2: Defining Requirements

Stage 3: Designing the product architecture

Stage 4: Building or Developing the Product

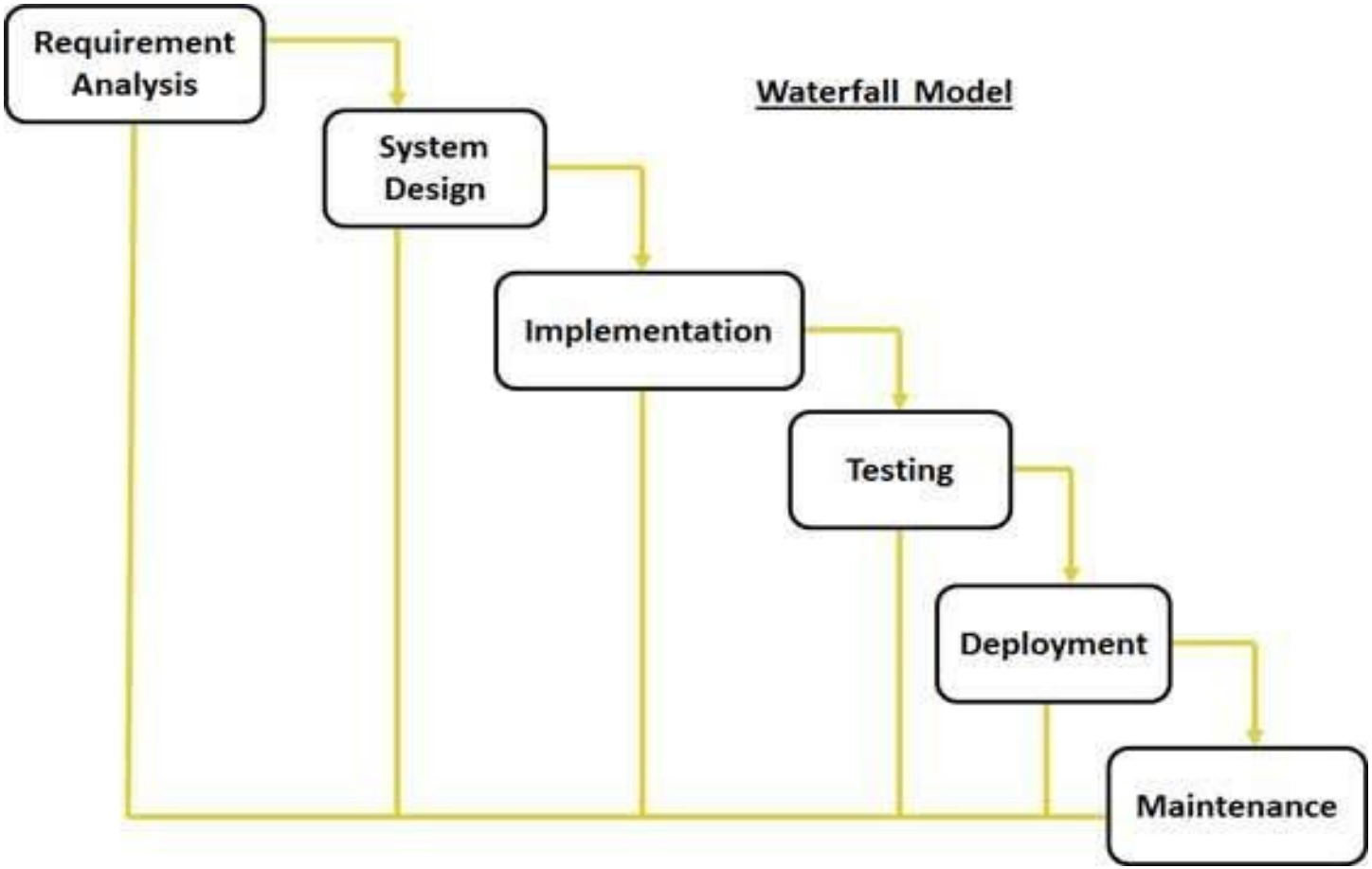
Stage 5: Testing the Product

Stage 6: Deployment in the Market and Maintenance

SDLC Models

- Waterfall Model
- Iterative Model
- Spiral Model
- V-Model
- Big Bang Model
- Agile Model

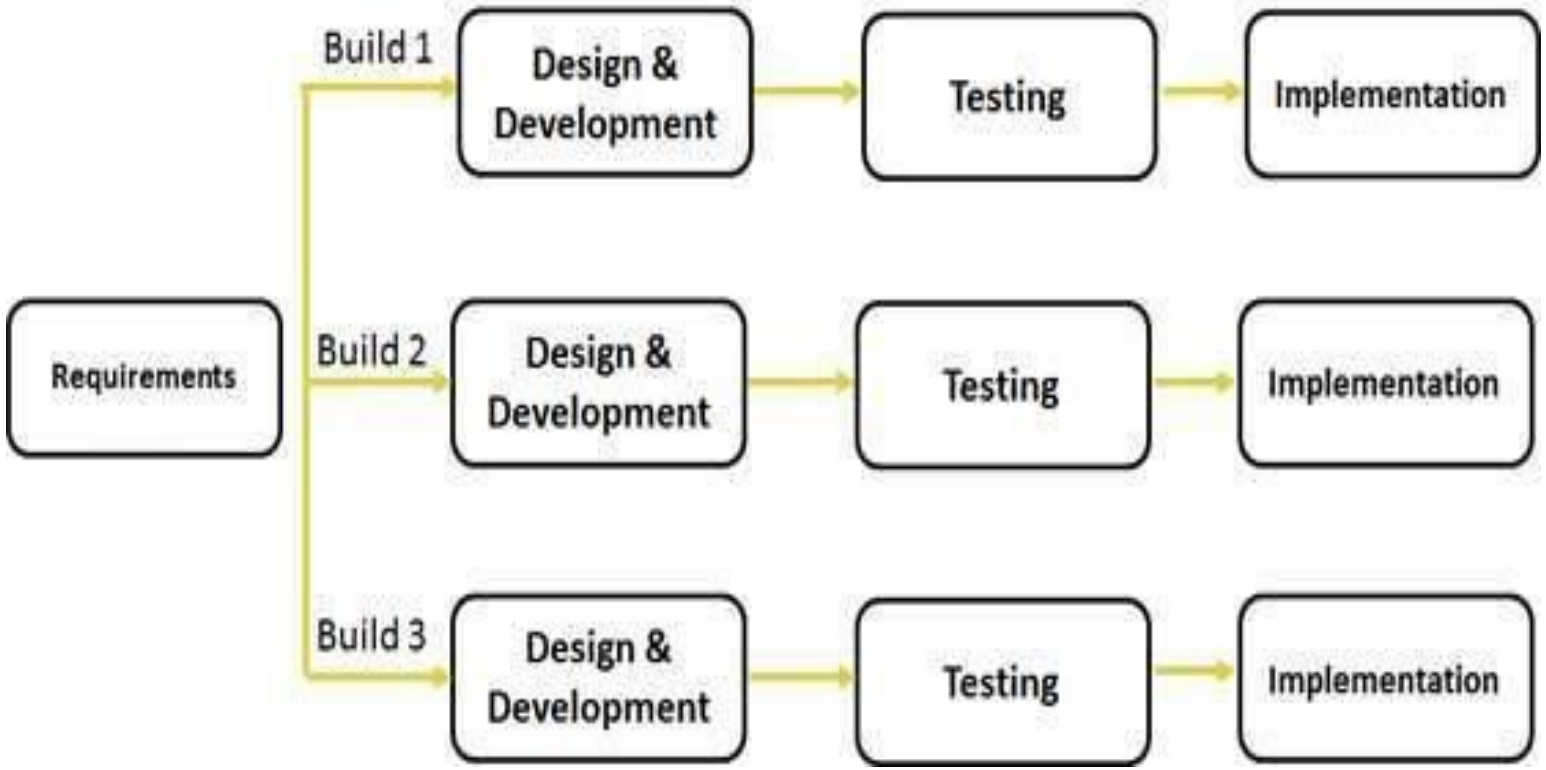
Waterfall Model



Cons of Waterfall Model

- 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. So risk and uncertainty is high with this process model.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- No working software is produced until late in the life cycle.
- Adjusting scope during the life cycle can end a project.
- Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

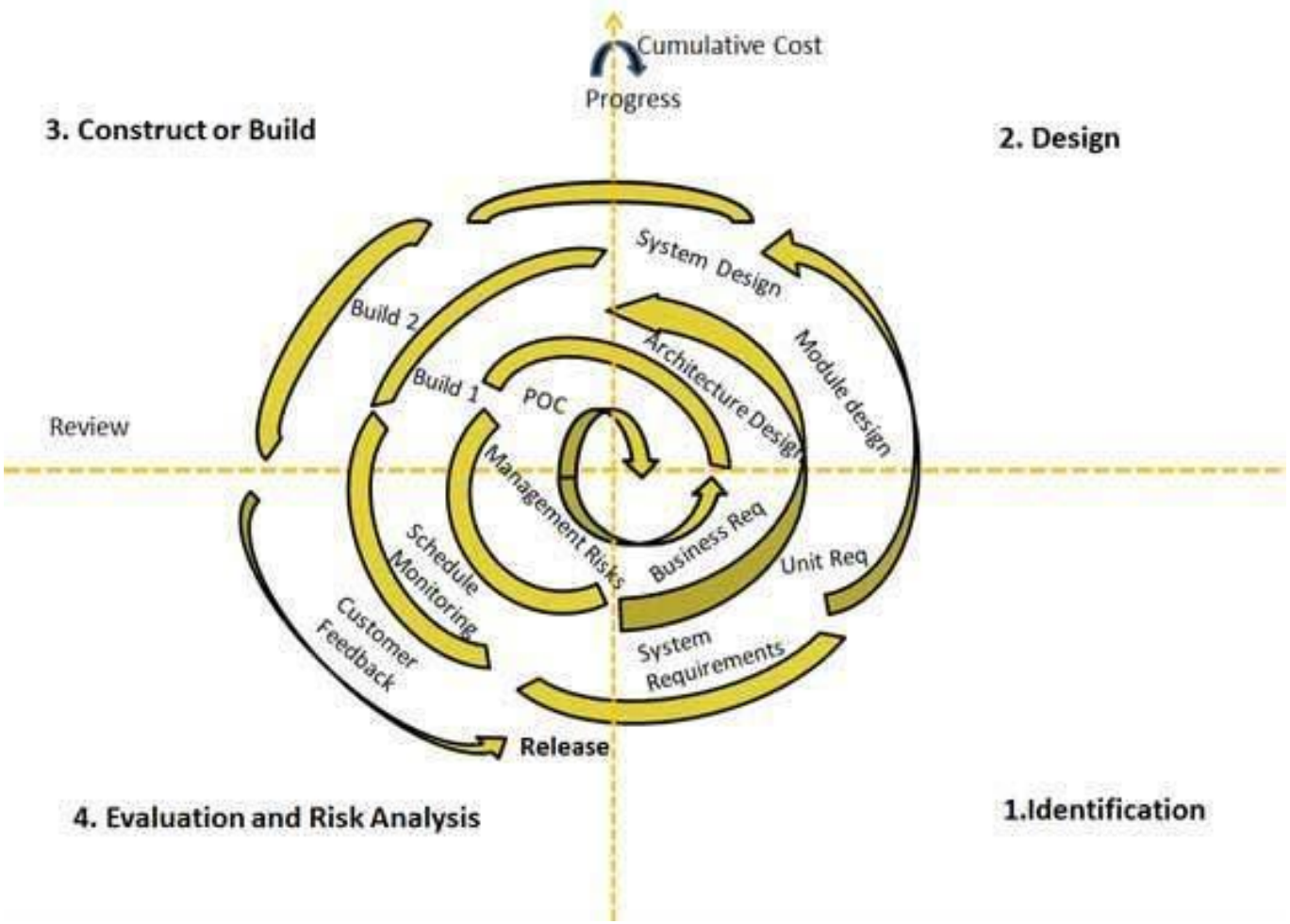
Iterative Model



Cons of Iterative Model

- More resources may be required.
- Although cost of change is lesser but it is not very suitable for changing requirements.
- More management attention is required.
- System architecture or design issues may arise because not all requirements are gathered in the beginning of the entire life cycle.
- Defining increments may require definition of the complete system.
- Not suitable for smaller projects.
- Management complexity is more.
- End of project may not be known which is a risk.
- Highly skilled resources are required for risk analysis.
- Project.s progress is highly dependent upon the risk analysis phase.

Spiral Model



Cons of Spiral Model

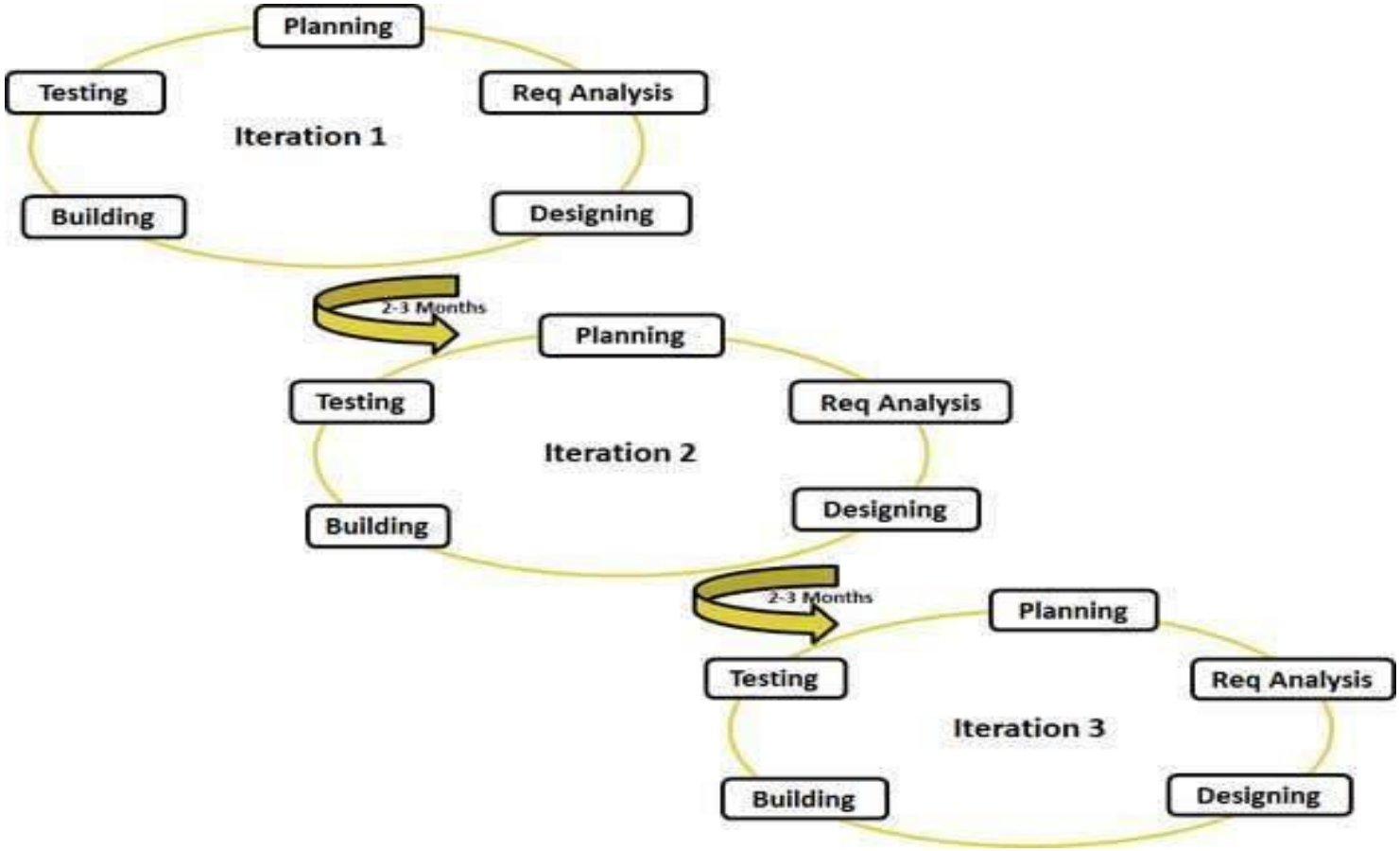
- Management is more complex.
- End of project may not be known early.
- Not suitable for small or low risk projects and could be expensive for small projects.
- Process is complex
- Spiral may go indefinitely.
- Large number of intermediate stages requires excessive documentation.

Agile Model - A Little History

The most popular agile methods include

- 1994- Rational Unified Process
- 1995- Scrum, Adaptive Software Development, Feature Driven Development, and Dynamic Systems Development Method (DSDM)
- 1996- Crystal Clear, Extreme Programming
- 2001- Agile Manifesto was published and these are now collectively referred to as agile methodologies

Agile Model


Planning
Testing
Req Analysis
Iteration 1
Building
Designing
2-3 Months
Planning
Testing
Req Analysis
Iteration 2
Building
Designing
2-3 Months
Planning
Testing
Req Analysis
Iteration 3
Building
Designing

Agile Principles

- Individuals and interactions
- Working software
- Customer collaboration
- Responding to change

Pros of Agile Model

- Is a very realistic approach to software development
- Promotes teamwork and cross training.
- Functionality can be developed rapidly and demonstrated.
- Suitable for fixed or changing requirements
- Delivers early partial working solutions.
- Good model for environments that change steadily.
- Minimal rules, documentation easily employed.
- Enables concurrent development and delivery within an overall planned context.
- Little or no planning required
- Easy to manage
- Gives flexibility to developers

Cons of Agile Model

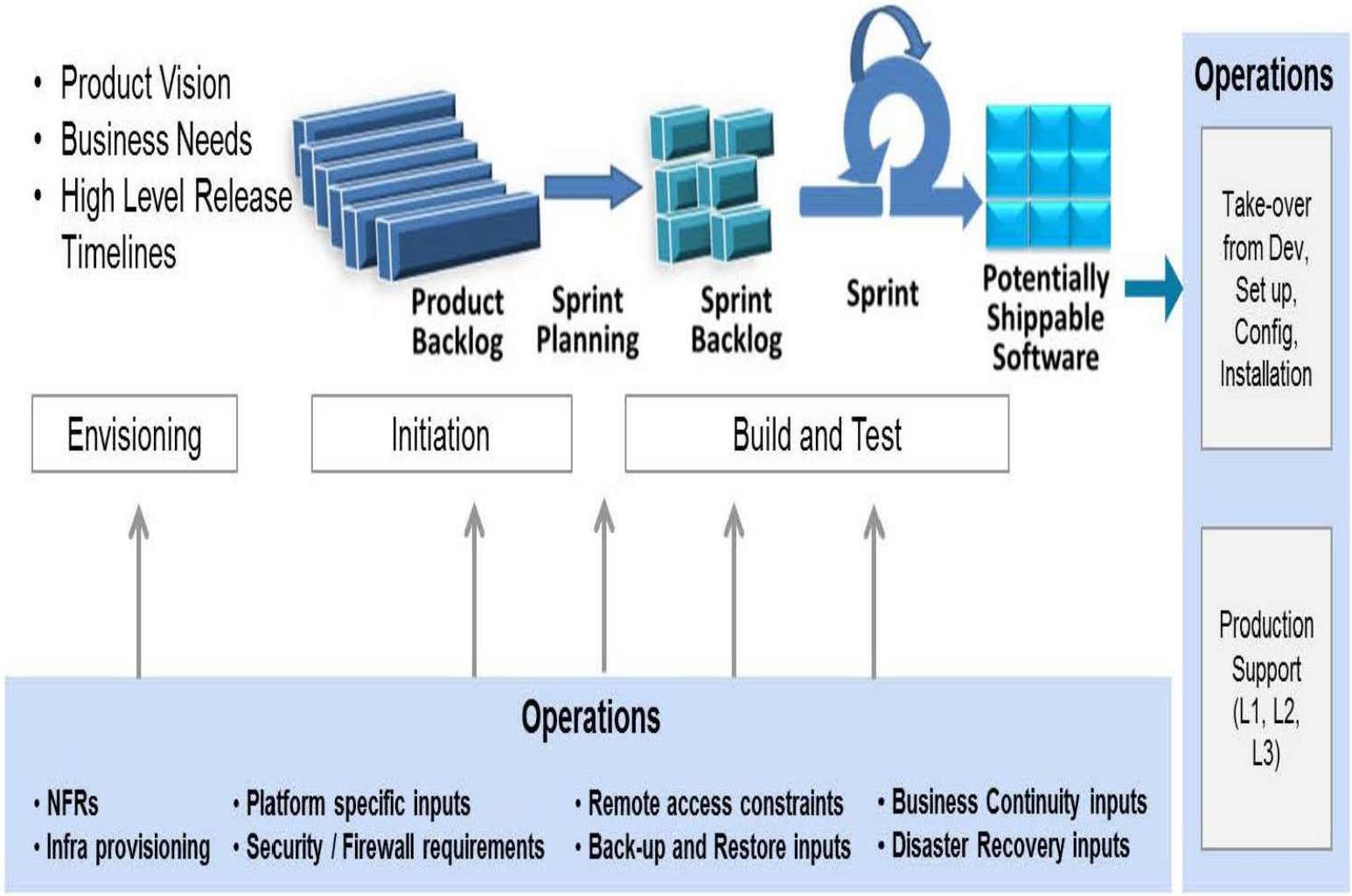
- Not suitable for handling complex dependencies.
- More risk of sustainability, maintainability and extensibility.
- An overall plan, an agile leader and agile PM practice is a must without which it will not work.
- Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.
- Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.
- There is very high individual dependency, since there is minimum documentation generated.

Agile and DevOps

- DevOps enables realization of the benefits of faster delivery of functionality achieved through Agile.
- **Continuous involvement of the Ops team**

A fundamental requirement of DevOps is that the Ops team is continuously engaged with the development team throughout the life cycle of solution development. Ops should participate right from the visioning stage to understand the business vision, the epics, and the release time lines. They should also contribute to determining the solution's technical and schedule feasibility.

From the visioning stage through the development stage, the Ops team should provide the necessary inputs to the development team in order for them to build and validate the Ops-related requirements. An illustrative model of involvement and some examples of the Ops inputs are shown below.


• Product Vision
• Business Needs
• High Level Release Timelines
Product Backlog
Sprint Planning
Sprint Backlog
Sprint
Potentially Shippable Software
Operations
Take-over from Dev, Set up, Config, Installation
Production Support (L1, L2, L3)
Envisioning
Initiation
Build and Test
Operations
• NFRs
• Infra provisioning
• Platform specific inputs
• Security / Firewall requirements
• Remote access constraints
• Back-up and Restore inputs
• Business Continuity inputs
• Disaster Recovery inputs

- **Product owner**

The product owner (PO) is the face of the business for the development team. The PO has a vision for the product and has a complete insight into the functional requirements that must be met. However, if the nonfunctional requirements (NFRs) are not well understood and articulated by the PO, the development team will not be able to take them into account while creating the architecture and building the final solution.

The PO should be equiped with the bellow basic appreciation Deployment and support platforms

- Their availability and limitations
- Dependency on vendor partners on infrastructure maintenance
- Third-party interfaces/applications needed for the final solutions
- The PO is not expected to be an expert in these areas but should be able to foresee such requirements and communicate these to the appropriate stakeholders in IT and the business.

- **Product backlog**
  Typically a product backlog focuses on epics and stories related to the functional requirements. The PO and the Dev team are well trained to brainstorm, break down epics, and document the functional requirements. However, often the NFRs are not well specified in the backlog. In addition to the functional requirements, the backlog should describe items such as the following:
  Performance requirements
- Tech requirements related to deployment and support
- Requirements to develop the guidelines for rapid rollback and roll forward
- Security/firewall requirements

- **Ops representation in the team**
  The Agile development team is cross-functional and self-organizing. Should this team then include an Ops person too? Maybe -- this depends on how cross-skilled the team members are. Typically in a start-up IT organization, some of the developers or testers would also be responsible for deployment and Level-3 (bug fixing) support. In such cases the requirements related to deployment and support would be well discussed in planning and review meetings.

  However, in large organizations, operations would need a large team dedicated to take over completed code from multiple Dev teams and deploy them. In such cases, job rotation could be a useful proposition. That is, some of the developers could play the Ops role for certain period of time, and some of the Ops team members with the right aptitude could be made part of the Dev team for a certain period of time. This way the Ops side would be well represented during the development cycle. - See more at: https://www.scrumalliance.org/community/articles/2014/april/devops-and-agile#sthash.ZgzZH6cO.dpuf

- **Definition of Done**
  Another key lever to involve Ops in the development cycle is to weave in Ops-related aspects in the Definition of Done. Along with the standard coding, testing, and documentation elements, validation of the code in the deployment platform (e.g., a mock production box), specific support instructions as part of the documentation, and a dry run of these instructions should also be included in the Definition of Done. Here again the inputs from the Ops team are crucial.

  Sprint planning and daily stand-up
  Sprint backlog planning and daily stand-ups should pay attention to the Ops needs while prioritizing backlog items and discussing progress. The sprint backlog should include specific line items related to securing the necessary tech platforms for mock deployment and other such coordination activities. It is a good idea to include the Ops team during sprint planning and in selected daily stand-ups where the team would discuss Ops aspects. Any dependency on infrastructure providers and system integrators should be considered at this stage using inputs from the Ops team. - See more at: https://www.scrumalliance.org/community/articles/2014/april/devops-and-agile#sthash.ZgzZH6cO.dpuf

- **Sprint review**
  When the Ops team is continuously involved in the development cycle, it goes without saying that they should be part of the sprint review as well. The Dev team should demonstrate the Ops-related features of the solution. Clearly, all sprint reviews may not include Ops-related features. However, if the Ops team is part of the demos, they have an opportunity to see what is coming up and provide inputs for the subsequent sprints to improve the product and include Ops requirements as well. Here again, if one or more of the Dev team members represent Ops, it is easy to get this alignment. If Ops is a separate team, the Dev team has to make sure to get the Ops team for product demos.

  Scrum of Scrums
  When multiple Scrum teams work on a solution, the integration of the output of each team has to be carefully planned and executed. Each Scrum team should take into account the Ops requirements and build features in alignment with the requirements. The product owners should have a view of the final product, how it will be developed through multiple Scrum teams, and where and how it will be deployed. They should involve the Ops teams to provide specific inputs for each Scrum team. At Scrum of Scrum events, the POs should come together and validate that the Scrum teams in fact include these in their plans and demos. - See more at: https://www.scrumalliance.org/community/articles/2014/april/devops-and-agile#sthash.ZgzZH6cO.dpuf

- **Plan alignment**
  DevOps and Agile complement each other well and help the business and release teams plan the annual release calendar. With continuous engagement and collaboration with the development team, the Ops team gets to know which functionality will be coming out when. With that insight, and using the sprint completion pattern, the Dev and Ops teams should be able to predict with reasonable accuracy the potential release dates. Eventually they should strive to align the release schedule with the sprint plans. And when this alignment happens, the support team would be able to move completed functionality to production faster and at shorter intervals -- and a key benefit of Agile is realized!

  Metrics
  To measure how DevOps would help faster releases, a few leading and lagging indicators such as the following could be used. Industry statistics demonstrate that organizations using DevOps and Agile are able to make multiple releases to production in a single day.
  Release date adherence percentage
- Percentage increase in the number of releases
- Time taken for release to production
- Defects attributable to platform/support requirements
- Percentage of NFRs met
- - See more at: https://www.scrumalliance.org/community/articles/2014/april/devops-and-agile#sthash.ZgzZH6cO.dpuf

Conclusion:

When Agile and DevOps come together, the IT organization can see tangible outcomes. This article has focused on a very specific Agile sprint view to implement DevOps. However, DevOps is much larger and involves multiple dimensions including business, IT, and vendor stakeholders.

Thank You

??Queries??