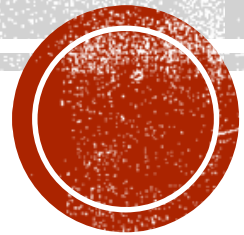
SOFTWARE ENGINEERING: INTRODUCTION



PREFACE

- What is Engineering?
- What is Software?
- What is software engineering?
- What is a software process?
- Different types of process models?
- Different Models with Strengths and Weaknesses
- Agile Software Development

WHAT IS ENGINEERING?

- **Engineering** is the application of **scientific** and **practical** knowledge in order to **invent, design**, **build**, **maintain**, and **improve systems**, **processes,** etc.

WHAT IS SOFTWARE?

- The software is collection of Integrated programs
- Software consists of carefully-organized instructions and code written by programmers in any of various special computer languages.
- Computer programs and associated documentation such as requirements, design models and user manuals.

WHAT IS SOFTWARE ENGINEERING?

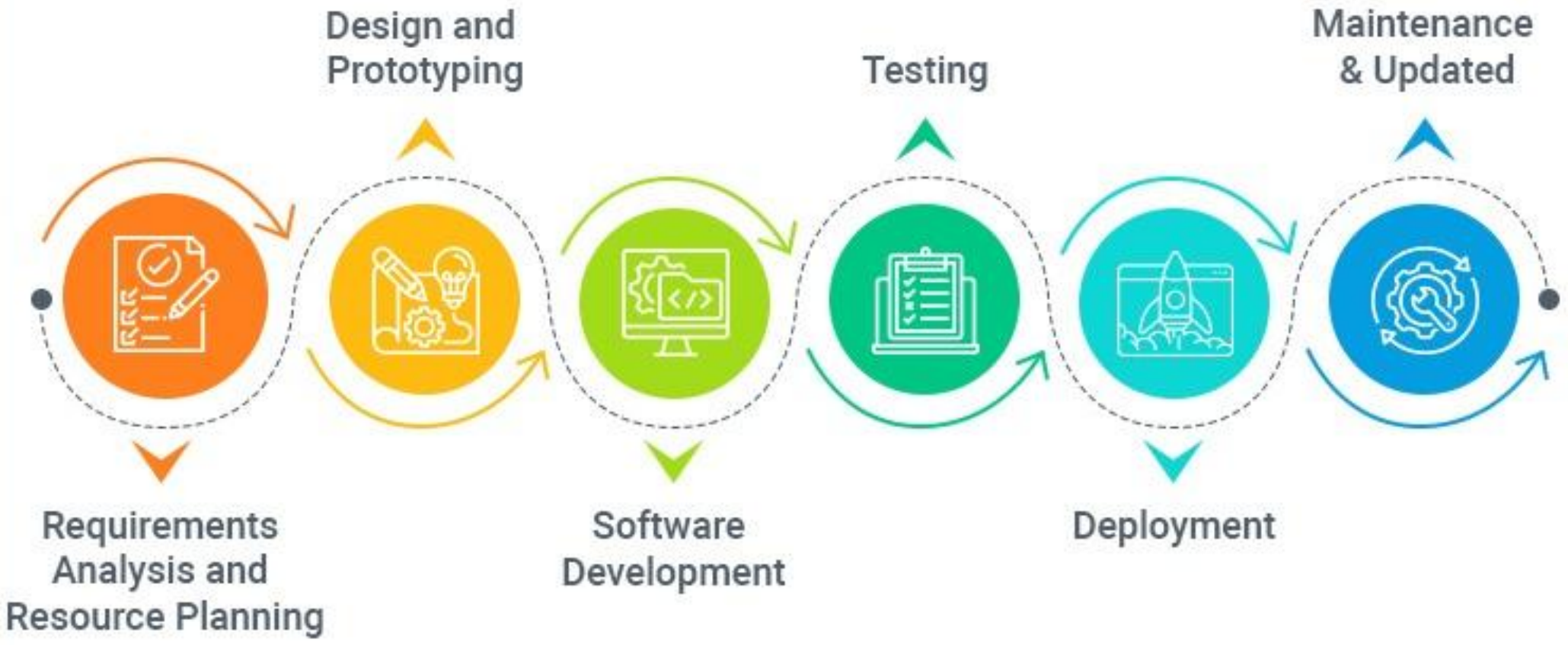
- Software engineering is an engineering discipline that is concerned with all aspects of software production.

- According to **IEEE's definition software engineering** can be defined as the application of a **systematic**, disciplined, **quantifiable** approach to the **development**, **operation**, and **maintenance** of **software**, and the study of these approaches; that is, the application of **engineering** to **software**.

WHAT IS A SOFTWARE PROCESS?

- A **framework** that describes the activities performed at each stage of a software development project.
- A set of activities whose goal is the development or evolution of software.
- Generic activities in all software processes are:
  - **Specification** - what the system should do and its development constraints
  - **Development** - production of the software system
  - **Validation** - checking that the software is what the customer wants
  - **Evolution** - changing the software in response to changing demands.

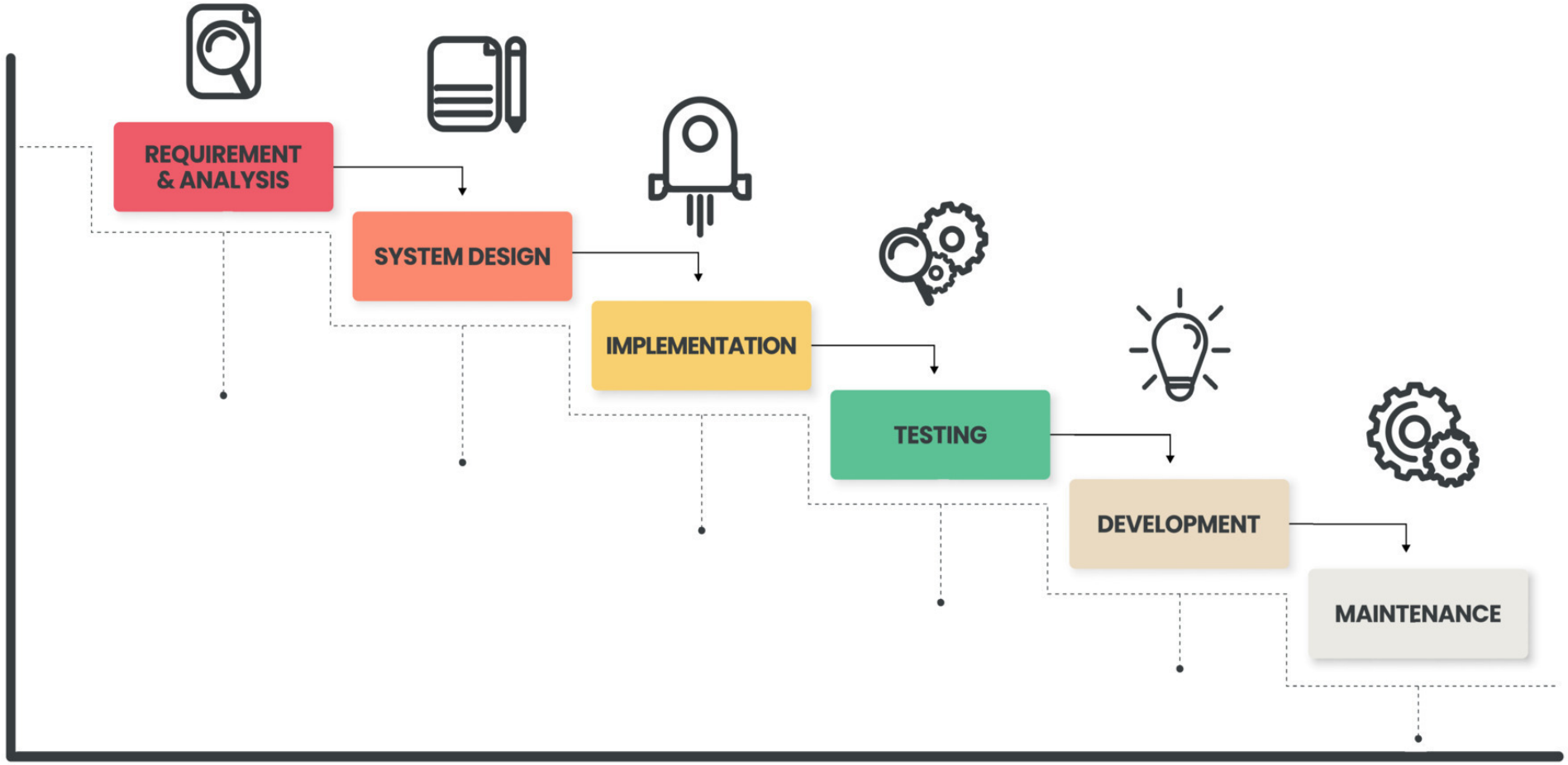
Stages of Software Development Processes



TYPES OF SDLC MODELS:

- Waterfall Model
- Incremental Model
- RAD Model
- Spiral Model
- Scrum Model
- Lean Development Methodology
- Extreme Programming
- Etc

WATERFALL MODEL



WATERFALL MODEL: STRENGTHS

- Easy to understand, easy to use
- Provides structure to inexperienced staff
- Milestones are well understood
- Good for management control (plan, staff, track)
- Works well when quality is more important than cost or schedule

WATERFALL MODEL: DEFICIENCIES

- All requirements must be known upfront
- Deliverables created for each phase are considered frozen inhibits flexibility
- Can give a false impression of progress
- Does not reflect problem-solving nature of software development – iterations of phases
- Integration is one big bang at the end
- Little opportunity for customer to preview the system (until it may be too late)

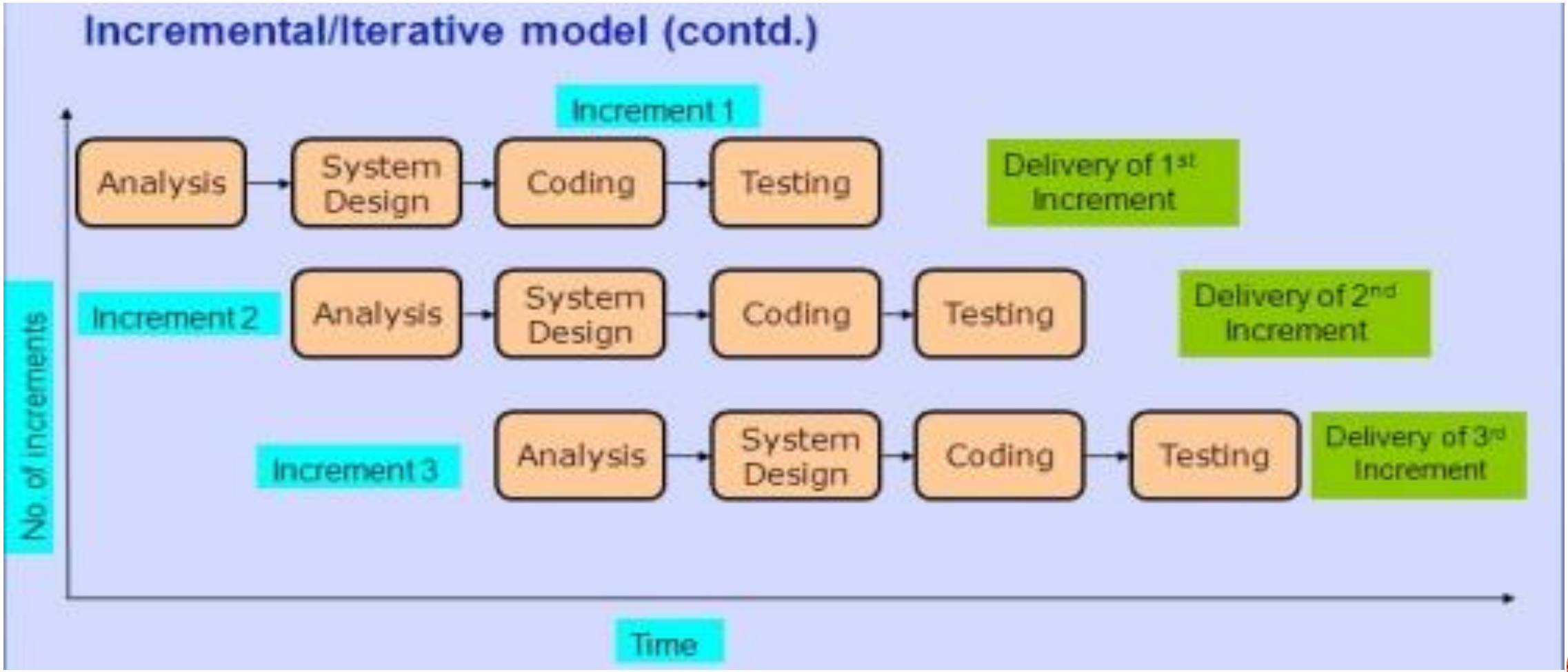
WATERFALL MODEL: WHEN TO USE

- Requirements are very well known
- Product definition is stable
- Technology is understood
- New version of an existing product
- Porting an existing product to a new platform.
- High risk for new systems because of specification and design problems

INCREMENTAL MODEL

- Whole requirement is divided into various builds
- Multiple development cycles take place here, making the life cycle a "multi-waterfall" cycle.
- Cycles are divided up into smaller, more easily managed modules.
- Each module passes through the requirements, design, implementation and testing phases.
- A working version of software is produced during the first module, so you have working software early on during the software life cycle.

INCREMENTAL MODEL



INCREMENTAL MODEL: ADVANTAGES

- Generates working software quickly and early during the software life cycle.
- More flexible – less costly to change scope and requirements.
- Easier to test and debug during a smaller iteration.
- Customer can respond to each built.
- Lowers initial delivery cost.

INCREMENTAL MODEL: DISADVANTAGES

- Needs good planning and design.
- Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
- Total cost is higher than waterfall.

INCREMENTAL MODEL: WHEN TO USE

- Requirements of the complete system are clearly defined and understood.
- Major requirements must be defined; however, some details can evolve with time.
- There is a need to get a product to the market early.
- Large project

WHAT IS AGILE?

- Traditional approach to managing software development projects was failing far too often, and there had to be a better way

- Agile development is a different way of managing IT development teams and projects

- Came up with the agile **manifesto**, which describes 4 important values that are as relevant today

AGILE MANIFESTO:

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

10 KEY PRINCIPLES OF AGILE:

1. Active user involvement is imperative
2. The team must be empowered to make decisions
3. Requirements evolve but the timescale is fixed
4. Capture requirements at a high level
5. Develop small, incremental releases and iterate

10 KEY PRINCIPLES OF AGILE:

6. Focus on frequent delivery of products

7. Complete each feature before moving on to the next

8. Apply the 80/20 rule

9. Testing is integrated throughout the project lifecycle – test early and often

10. A collaborative & cooperative approach between all stakeholders is essential

AGILE FRAMEWORKS

- Scrum
- Kanban
- Extreme Programming (XP)
- SAFe (Scaled Agile Framework)
- Etc…

SCRUM METHODOLOGY:

- Is also an agile development method, which concentrates particularly on how to manage tasks within a team-based development environment.
- Scrum is the most popular and widely adopted agile method
- Relatively simple to implement and addresses many of the management issues that have plagued IT development teams for decades

SCRUM METHODOLOGY

•A product owner creates a prioritized wish list called a product backlog.
•During sprint planning, the team pulls a small chunk from the top of that wish list, a sprint backlog, and decides how to implement those pieces.
•The team has a certain amount of time — a sprint (usually two to four weeks) — to complete its work, but it meets each day to assess its progress (daily Scrum).
•Along the way, the Scrum Master keeps the team focused on its goal.
•At the end of the sprint, the work should be potentially shippable: ready to hand to a customer, put on a store shelf, or show to a stakeholder.
•The sprint ends with a sprint review.
•As the next sprint begins, the team chooses another chunk of the product backlog and begins working again-


Development
Groom Product Backlog
Daily Scrum
Sprint
Project Initiation
Sprint Planning
Sprint Review
Product Increment
Sprint Retrospective

COMPONENTS OF THE SCRUM PROCESS

- **Project Initiation**
  - This is the starting point of the project.
  - The vision, high-level requirements, and product backlog are defined.
  - Key roles (Product Owner, Scrum Master, Development Team) are assigned.
- **Sprint Planning**
  - A meeting to **plan the upcoming sprint** (usually 1–4 weeks).
  - The team selects items from the **Product Backlog** to work on in the sprint.
  - A **Sprint Goal** is established.
- **Sprint**
  - The core iteration cycle in Scrum.
  - A time-boxed period (e.g., 2 weeks) where the team develops a **potentially shippable product increment**.
  - Involves **designing, coding, testing**, and integrating features.

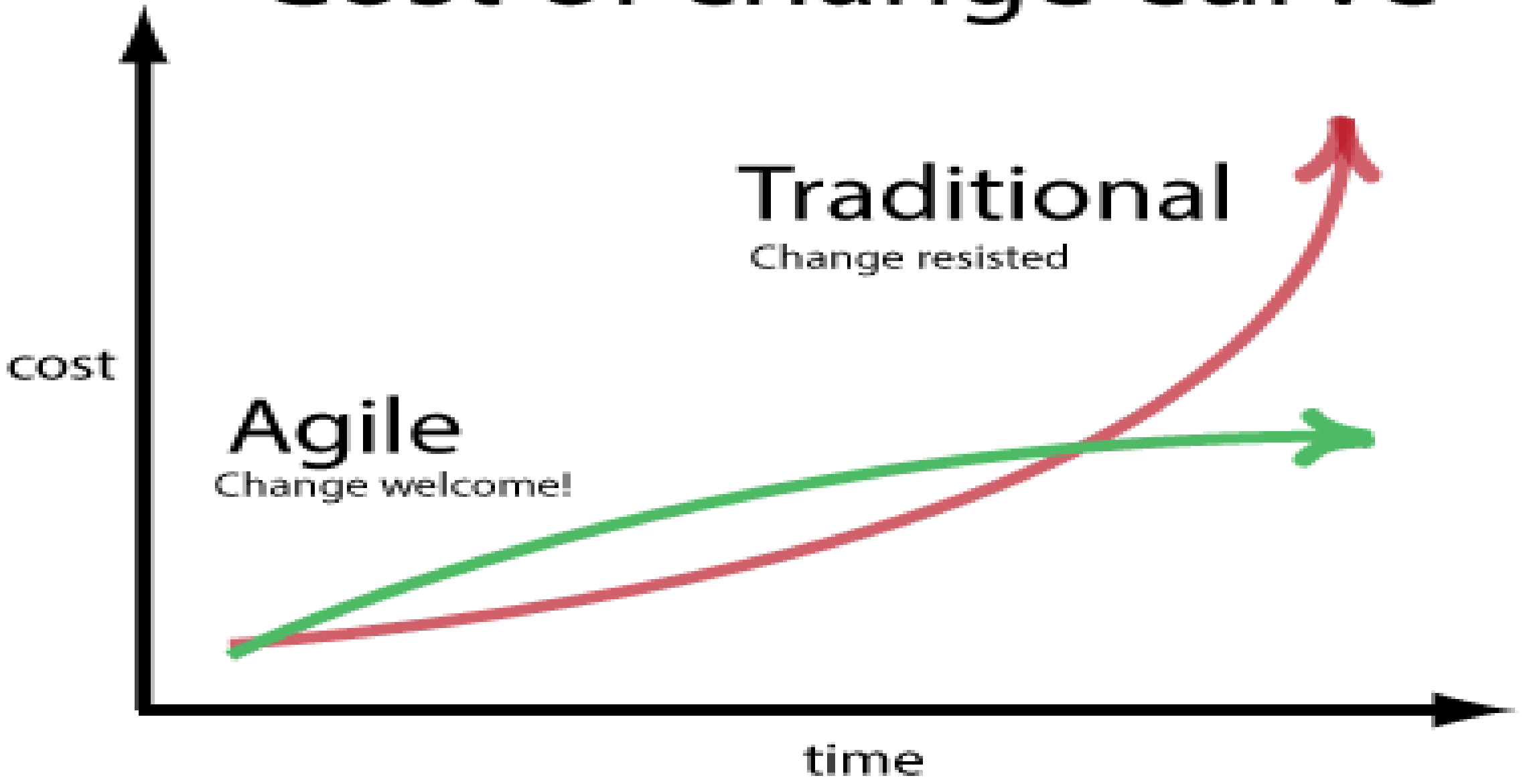
- **Daily Scrum**
  - A short 15-minute **daily stand-up meeting**.
  - Team members share:
    - What they did yesterday
    - What they'll do today
    - Any blockers they're facing
- **Development**
  - This is the execution phase of the sprint where actual product development happens.
  - Continuous collaboration between developers, testers, and product owners.
- **Groom Product Backlog (also called Backlog Refinement)**
  - Ongoing process of updating and refining the **Product Backlog**.
  - Involves:
    - Adding new user stories
    - Re-prioritizing
    - Estimating effort
    - Clarifying requirements

- **Sprint Review**
  - Held at the end of the sprint.
  - The team **demonstrates** the working product increment to stakeholders.
  - Feedback is gathered and potentially added to the product backlog.
- **Sprint Retrospective**
  - Internal team meeting to **reflect** on the sprint.
  - Discusses:
    - What went well
    - What didn't
    - How to improve in the next sprint
- **Product Increment**
  - The final outcome of the sprint.
  - A working, tested, and potentially deployable piece of software.
  - Added to the existing product for cumulative growth.

PROS & CONS

- PROS
  - Freedom and adaptive
  - High quality, low risk product
  - Reduce the development time up to 40%
  - Scrum customer satisfaction is very important
  - Reviewing the current sprint before moving to new one
- CONS:
  - More efficient for small team size
  - No changes

Cost of change curve



DEVELOPERS VS IT OPERATIONS: THE CONFLICT

Aspect	Developers	IT Operations
Primary Goal	Rapid innovation, writing and releasing new code/features	System stability, performance, and availability
Focus	Speed, new features, short development cycles	Risk minimization, uptime, long-term reliability
Mindset	"Move fast and break things"	"If it works, don’t touch it"
Tooling	CI/CD pipelines, source control, build tools	Monitoring, alerting, backups, infrastructure as code
Success Metric	Number of features delivered, time to market	System uptime, low incident count, SLA compliance

HOW DEVOPS SOLVES THIS

- **DevOps** bridges the gap by encouraging:
- **Collaboration** between Dev and Ops
- **Automation** of testing, deployment, and infrastructure
- **Shared responsibility** for both development and operations
- **Continuous Integration/Continuous Deployment (CI/CD)** pipelines

SOFTWARE ENGINEERING: RECOMMENDED BOOKS