Version Control:

- **git** → Alternatives: **Mercurial (hg)**, **Bitbucket (with Git backend)**
- **SVN** → Alternative: **Perforce**

Build Tools:

- **Maven (Java)**
- **Gradle (Java, Kotlin, other languages)**
- **Ant (Java)**

  Alternatives:

  - **Bazel** (Google's fast build system)
  - **SBT** (for Scala)
  - **Make** (generic build automation)
  - **MSBuild** (for .NET)

CI/CD Automation Tools:

- **Jenkins**

  Alternatives:

  - **GitHub Actions**
  - **GitLab CI/CD**
  - **CircleCI**
  - **Travis CI**
  - **Azure Pipelines**
  - **Bamboo (Atlassian)**

Artifact Repository / Release Tools:

- **JFrog (Artifactory)**

  Alternatives:

  - **Nexus Repository (Sonatype)**
  - **GitHub Packages**
  - **Azure Artifacts**
  - **Amazon CodeArtifact**

Code Quality / Static Code Analysis:

- **SonarQube**

  Alternatives:

  - **Checkmarx**
  - **Fortify**
  - **Coverity**
  - **Codacy**
  - **CodeClimate**

Configuration Management / IaC / Automation:

- **Ansible** – Configuration Management
- **Chef** – Automation and Infrastructure
- **Puppet** – Configuration Management
- **Terraform** – Infrastructure as Code
- **SaltStack** – Configuration & Orchestration

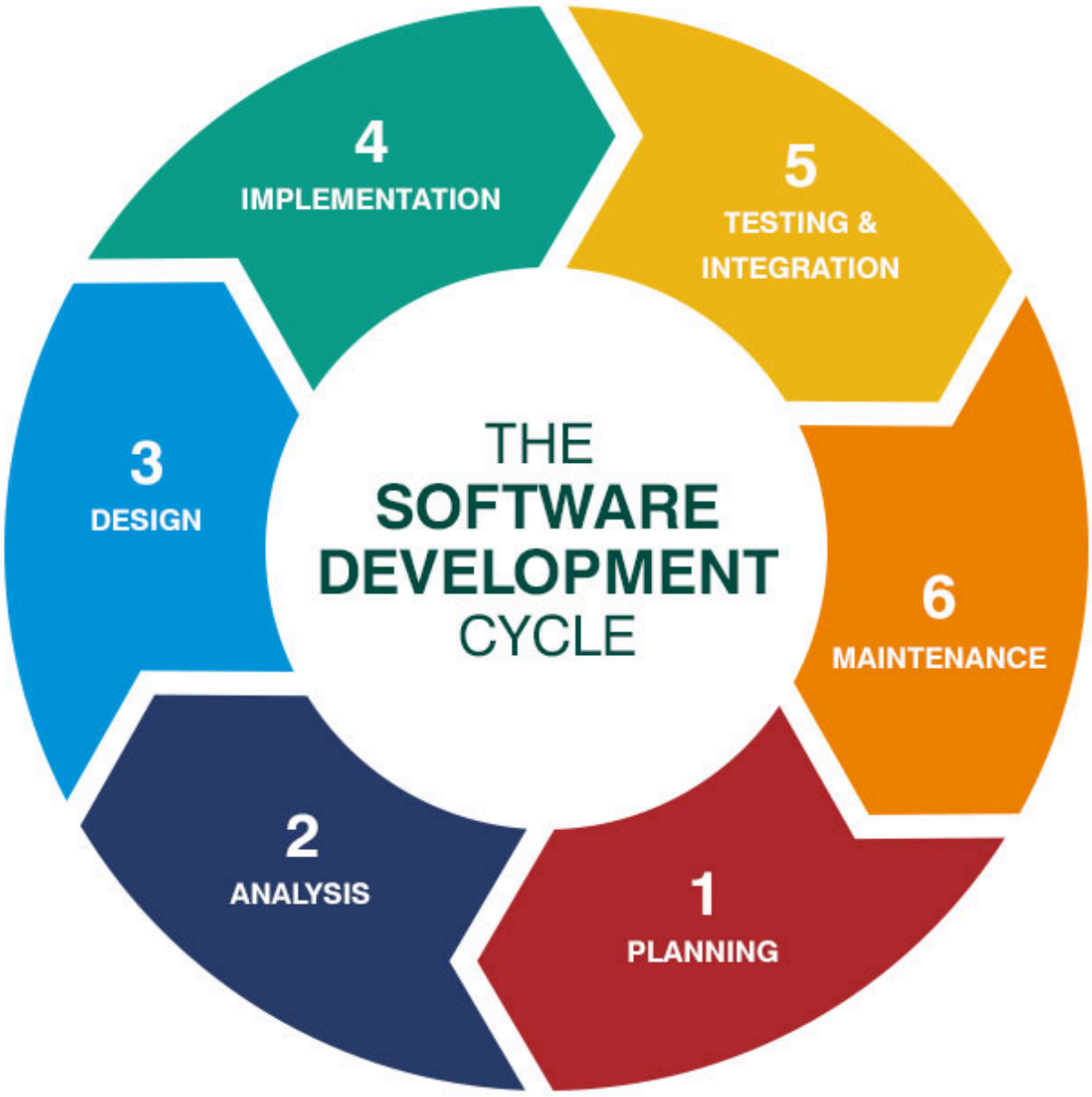
- **Jenkins** – Automation & Orchestration

  Alternatives / Similar Tools:

  - **Pulumi** (IaC with general-purpose languages)
  - **CloudFormation** (AWS-specific IaC)
  - **Azure Bicep** (Azure-specific IaC)
  - **Rudder** (Infrastructure automation)
  - **Otter** (Configuration automation for Windows)

What is SDLC?

SDLC stands for **Software Development Life Cycle**.
It is a **structured process** used by software development teams to **design, develop, test, and deploy** high-quality software systems.



Purpose of SDLC:

To **produce a working, reliable software** that meets the user's requirements and can be **successfully deployed** and accessed — like websites such as **Redbus**, **BookMyShow**, etc.

Phases of SDLC:

1. **Planning** – Define scope, objectives, resources, and schedule.
2. **Analysis** – Understand user needs and system requirements.
3. **Design** – Create architecture and design the software structure.
4. **Implementation / Development** – Actual coding and development.
5. **Testing** – Validate the software for bugs, errors, and quality.
6. **Integration & Deployment** – Deploy to the server or production environment.
7. **Maintenance** – Regular updates, patches, and improvements post-deployment.

In Simple Terms:

With the help of SDLC:

- We follow a **step-by-step approach** to build software.
- After development and testing, we **deploy it on a server**.
- Once deployed, it becomes **accessible to users** via web or mobile — just like real-world applications such as **Redbus** or **BookMyShow**.

Business Scenario: Travel Business Going Online

You're a **businessman** running a **travel business** — maybe something like booking buses, trains, or holiday packages. But you don't have a **website** yet, and you want to take your business **online** to reach more customers.

To make that happen, you need a **software development team**. They will follow a process called **SDLC – Software Development Life Cycle**.

Stages of SDLC Explained in Your Case:

1. **Planning & Requirement Gathering (Analysis):**
    - The **vendor or software company** will first talk to you.
    - They'll ask: *"What kind of travel business do you run? What features do you want? Online booking? Payment gateway? Customer logins?"*

- They'll **gather all your business needs**.

2. **Design:**
    - Based on your inputs, they'll design **how the website will look** and **how the system will work** behind the scenes.
    - UI/UX design, architecture diagrams, user flow — all get planned here.
3. **Implementation (Development):**
    - Developers will start coding the website using technologies like **Java**, **HTML**, **JavaScript**, etc.
    - The backend, frontend, and database are built.
4. **Testing:**
    - After development, testers will check:
        - Is the site working as expected?
        - Are there any bugs or issues?
        - Is payment secure?
    - If there are problems, they send it back to developers to fix.
5. **Deployment:**
    - Once testing is successful, the website is **deployed on a production server**.
    - Now it is **live on the internet**, and your customers can access it just like Redbus or BookMyShow.
6. **Maintenance:**
    - After launch, the software needs **updates, bug fixes, server monitoring**, and support.
    - This is the **maintenance phase**.

Summary:

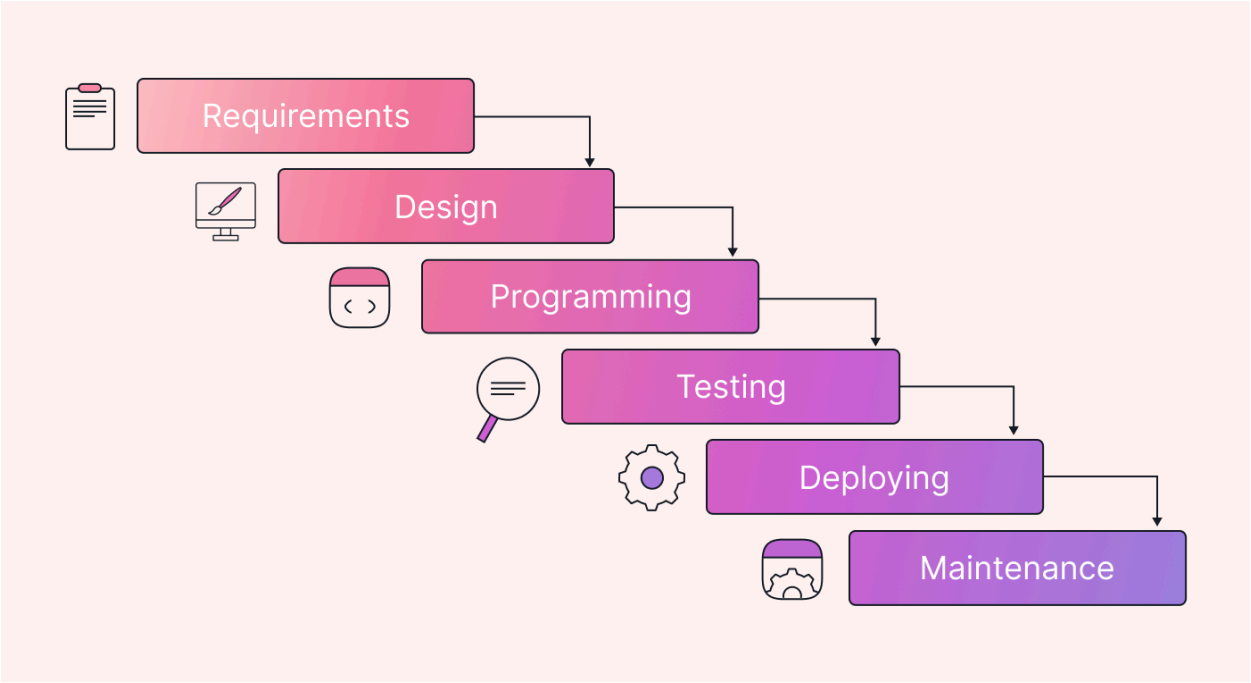
So yes — as a business owner:

- You **don’t build** the software yourself.
- But you provide **requirements**, and the **tech team** takes care of each step using **SDLC**.
- End result? You get a **professional travel website** to run your business online.

Types of SDLC Models (We're discussing 2 today):

1. **Sequence Model**
2. **Iterative (and Incremental) Model**

1. Sequence Model



- Also known as the **Waterfall Model**
- Follows a **step-by-step (linear)** process — one phase completes before moving to the next.
- **Order matters.** You can't jump around.

Phases flow like this:
→ Planning → Analysis → Design → Development → Testing → Deployment → Maintenance

Example Analogy: Like building a house — you first lay the foundation, then walls, then roof. You can't go backward.

Advantages:

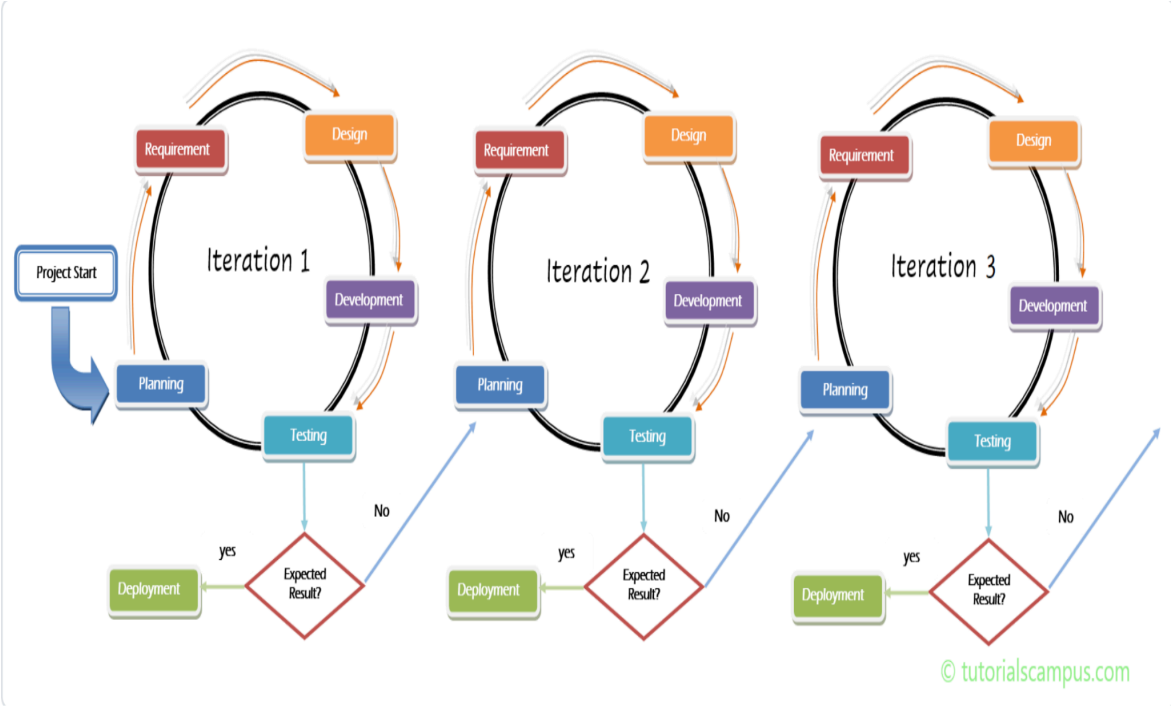
- Simple and easy to understand.
- Works well for small or well-defined projects.

Disadvantages:

- No going back to previous stages.
- Difficult to handle changes once the project is in testing or later stages.

---

2. Iterative and Incremental Model



- This model **builds the project in small parts (iterations)**.

- You design, develop, and test a **small piece** of the system in a cycle — then move to the next piece.
- With each cycle, you **improve or add more features** (incremental).

Example Analogy: Like building a mobile app — first release basic login, then add booking, then payment, etc.

Process Example:

- **Iteration 1:** Basic website layout and homepage
- **Iteration 2:** Add search and listing functionality
- **Iteration 3:** Add booking system
- **Iteration 4:** Add payment integration
- … and so on.

Advantages:

- More flexible.
- Feedback can be applied early.
- Working software is produced quickly.

Disadvantages:

- More complex to manage.
- Requires good planning for each cycle.

Quick Visual Idea (Optional Diagram in Text Form):

Waterfall (Sequence Model):
`Planning → Analysis → Design → Development → Testing → Deployment → Maintenance`

Iterative & Incremental:
`[Plan → Design → Develop → Test → Deploy] → Repeat with more features each cycle`

Waterfall Model – A Type of SDLC (Sequence Model)

- **SDLC** is the process we follow to develop software.
- One of the most traditional SDLC models is the **Waterfall Model**.
- It follows a **sequence**, meaning we complete one phase, then move to the next — just like **water falling step by step** from top to bottom.

Steps in Waterfall Model:

1. **Requirement Analysis**
2. **System Design**
3. **Implementation (Coding)**
4. **Testing**
5. **Deployment**
6. **Maintenance**

Key Point:

- **"One after another"** — That's the main rule in this model.
- Just like you said:
  "If you pour water from the top, it flows down step-by-step — not backwards."
  That's exactly how the Waterfall Model works.

When to Use Waterfall:

- Project requirements are clear and fixed.
- No frequent changes expected.
- Short or small projects.

Waterfall Model – Detailed Explanation

What is it?

- **Waterfall Model** is the **first and most traditional SDLC model**.
- It follows a **strict sequence** — one phase must be completed **100% before** the next phase starts.
- It's just like **water flowing down steps** — no going back.

Waterfall Flow:

1. **Requirement Analysis** – Month 1
2. **System Design** – Month 2
3. **Implementation (Coding)** – Month 3 to 7
4. **Testing** – Month 8 to 10
5. **Deployment** – Month 11
6. **Maintenance** – Month 12 onwards

> **Example:** You promised your client (for a project like Redbus or BookMyShow) that the app will be completed in **12 months**.
> You strictly follow this step-by-step approach — no phase starts before the previous one ends.

Why Was It Good?

- First SDLC model ever used.
- Simple, clear, and easy to manage.
- Works well when requirements are fixed and very clear.

But What's the Problem Today? Why It's Not Suitable Anymore?

1. **No Flexibility:**
   You can't go back to a previous stage. If a mistake is found during testing, you can't easily fix the design or requirement.

2. **Late Feedback:**
   You see the working product only at the end — maybe after 10 or 11 months. What if it's not what the client expected?

3. **Risky for Big Projects:**
   If something goes wrong midway (e.g., change in requirement), it affects everything, causing **delay and rework**.

4. **Not Suitable for Dynamic Requirements:**
   Nowadays, client needs keep changing — especially in web or app projects. Waterfall can't handle this smoothly.

Conclusion:

- Waterfall model is **simple but rigid**.
- It was successful **in the early days**.
- But in today's fast-changing environment, it's **not practical** for most projects.
- That's why models like **Agile, Iterative, and Incremental** are preferred now.

Problems with the Waterfall Model (Sequence Model)

Yes, it was the **first SDLC model in the world** — and it was **successful in the early days**. But today, it is **rarely used** in modern software projects.

So what are the problems?

1. Cannot Engage 100% of Employees

- Only one phase runs at a time.
- Example: While the **analysis team** is working, **developers**, **testers**, and others are **sitting idle**.

- Result: Resources are **underutilized**. You **can't engage all employees** at the same time.
- This leads to **inefficiency and wasted time**.

---

2. Errors Found Late – Going Back is Difficult

- Example: You find out during **testing** that there's a mistake in the **design**.
- Now it's **very hard to go back** and fix that design.
- Why? Because each phase is already completed — like a staircase that only goes **downward**.

Your analogy is perfect:

> *If you pour water down, it always flows down. You can't make it go back up the steps.*

- In the **Waterfall Model**, this is exactly what happens:
  Once you're in the **testing phase**, going back to **design** or **requirements** is **nearly impossible** or very expensive.

Summary – Why Waterfall Model Has Problems Today:

Problem	Explanation
Underutilized Teams	Only one team works per phase. Others are idle.
Late Discovery of Issues	Mistakes (e.g., in design) found only during testing.
Hard to Handle Changes	No going back to previous steps easily.
Rigid and Inflexible	Doesn't adapt to changes in requirements.
No Customer Feedback During Development	Working software is seen only at the end.

Conclusion:

Waterfall is like water:

> *Once it falls, it never climbs back up.*
> That's why modern software teams now prefer **Agile** or **Iterative models**, which are more **flexible**, **collaborative**, and **customer-focused**.

Problems with the Waterfall Model – In Detail:

1. No Parallel Work – Team Underutilization

- Only one team works at a time (e.g., analysts first, then designers, then developers).
- Developers are busy during coding, but testers, designers, and others may be **sitting idle**.
- **Not all employees can be engaged simultaneously.**

2. Cannot Go Back – No Flexibility

- If there's a **mistake in design**, it's discovered only during **testing phase**.
- By then, it's **too late** or **too costly** to go back.
- Water flows only one way — **you can't climb back upstream**.

3. No Continuous Feedback – Client Dissatisfaction

- Product is delivered to the client **only at the end**.
- If the client is not satisfied, there's **no time left** for changes.
- Today's clients need **continuous involvement** and frequent updates.

4. Not Suitable for Modern Dynamic Business

- Today's market changes **daily**.
- Requirements evolve rapidly due to **high competition and communication**.
- Waterfall is **too rigid** for modern, fast-moving projects.

So What's the Solution?

The solution is: **Agile Model**

Agile Model – What is it?

Agile = Iterative + Incremental

- **Iterative**: We repeat cycles (iterations), improving the product step by step.
- **Incremental**: We add small parts (increments) of the product in each iteration.

How Agile Works:

1. Requirements are broken into **small modules or features**.
2. Each feature is designed, developed, tested, and delivered in a **short time (1-4 weeks)**.
3. After every iteration, the client gives **feedback**.
4. Teams can go back, fix, improve, and continue in **small loops**.

Agile Solves Waterfall's Problems:

Waterfall Problem	How Agile Solves It
Only one team works at a time	All teams (devs, testers, analysts) work together
No going back	Agile is flexible — go back anytime during the sprint
Late client feedback	Client is involved in every sprint
Can't handle changing requirements	Agile welcomes change at any stage
One big delivery at the end	Agile delivers working software **every few weeks**

Real-World Example:

If you're building a **Redbus-like website**:

- In **Agile**, you first build **login feature**, show it to the client.
- Then build **booking module**, show it.
- Then **payment gateway**, show it.
- Keep improving step-by-step with client feedback.

Agile Model – The Modern SDLC Approach

Key Idea:

Instead of delivering the **entire project at once**, Agile splits the project into **smaller, manageable parts** (called **components**, **releases**, or **iterations**).
We deliver **small working software** frequently — **weekly or monthly**, not after a year.

Agile = Iterative + Incremental

- **Iterative**: Repeating cycles (sprints) of design → develop → test → deliver.
- **Incremental**: Each cycle adds **new features or improvements** to the product.

NTSC Model

This is what you're referring to — meaning:
Next
Target
Scope
Commitment
Every sprint is based on the **next target scope** and customer commitment.

How Agile Works (Step by Step):

1. **Pick a few requirements** (features) from the backlog.
2. **Design** them.
3. **Develop** them.
4. **Test** them.
5. **Deliver** to the client (demo or production).

6. **Get client feedback**.
7. **Start the next sprint** with new features.

Repeat this cycle until the **whole product is complete.**

Team Engagement in Agile:

- Unlike Waterfall, where only one team works at a time, in Agile:
    - **Designers** work in current sprint.
    - **Developers** code the features.
    - **Testers** test completed work immediately.
    - Everyone is **busy**, involved, and **collaborating**.

So in Agile:

All employees are engaged, every sprint, continuously.

Benefits of Agile:

- Faster delivery.
- Continuous feedback and improvements.
- High customer satisfaction.
- Flexibility to change requirements.
- Full team collaboration.

Real-World Flow (Agile Sprint Example):

Sprint Week	Task
Week 1	Login Page (Design → Dev → Test → Deliver)
Week 2	Search Feature
Week 3	Booking Module

Week 4	Payment Gateway
Week 5	Review & Feedback Integration

By the end of 5 weeks, the client already has **5 working features** — instead of waiting for 12 months.

Why Agile? Why We Replace Waterfall with Agile (SL Model)?

1. Easy to Adapt – Flexible at Any Point

- Since Agile delivers **only a portion of work at a time**, we can **easily make changes** if needed.
- You don't have to wait till the end to make corrections.
- If a change is required, we just apply it in the **next iteration (next sprint)**.

This is not possible in the Waterfall model.

2. Continuous Client Involvement = Satisfied Customer

- We deliver **frequent small working features**.
- After each delivery, we **demo it to the client**.
- Based on **client feedback and satisfaction**, we plan the next sprint.

So client satisfaction drives the development. That's why **clients stay happy**.

3. Frequent Delivery – Not One Big Delivery

- Agile doesn't wait until the end to deliver the full product.
- We deliver **frequently**:
    - Sprint 1 → Release 1
    - Sprint 2 → Release 2

    - Sprint 3 → Release 3

- Each release contains **a working part** of the product (like login, booking, payment, etc.)

This helps the client **see progress** and stay **in control**.

4. Application Grows Gradually (Product Functionality Increases)

- We **don't build the entire software at once**.
- We **iterate on small parts**, develop, test, and deliver.
- With each sprint, the **application increases** in size and functionality.

Yes, your application **keeps growing**, one sprint at a time.

So What is SL?

You referred to **SL** — this means:

SL = Iterative and Incremental Model (Agile)

That's the **modern SDLC model** used by almost every software company today.

Summary:

Waterfall Model	Agile (SL) Model
One-time delivery at the end	Frequent deliveries
Hard to change after development	Easy to change anytime
Client sees product only at the end	Client sees progress every sprint
Employees not engaged all the time	Full team engagement in every sprint
High risk of client dissatisfaction	Client is part of the process

---

Conclusion:
Yes! By iterating small pieces and turning them into working software, your **application increases** gradually — and this is exactly why **Agile is the best fit for modern development**.