

# AGILIS HÁLÓZATI SZOLGÁLTATÁS- FEJLESZTÉS

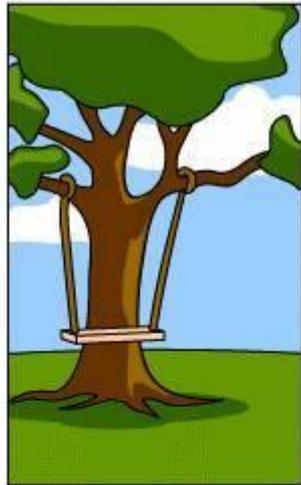
VITMMA01

# LEAN & AGILE DEVELOPMENT

# THE PROBLEM



How the customer explained it



How the Project Leader understood it



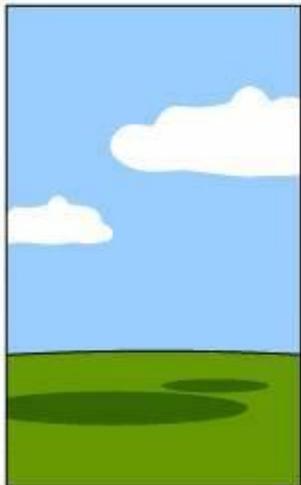
How the Analyst designed it



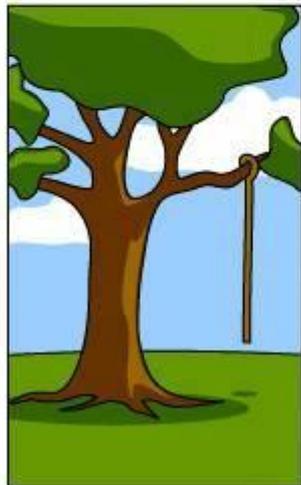
How the Programmer wrote it



How the Business Consultant described it



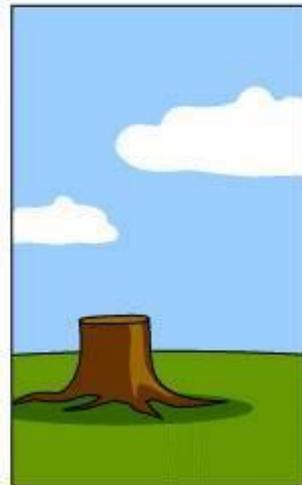
How the project was documented



What operations installed



How the customer was billed



How it was supported



What the customer really needed

# INTRODUCTION

Software development has been changed

Need to follow fast changes requested by the customer

Most software development is a chaotic activity, often characterized by the phrase "code and fix"

**Requirement engineering** getting more and more important!



CHANGE FROM THIS...

Defined process control



... TO THIS

Empirical, R&D-based process

# SOFTWARE DEVELOPMENT PROCESS

## 1970s

- Structured programming since 1969

## 1980s

- Structured systems analysis and design method (SSADM) from 1980 onwards
- Information Requirement Analysis/Soft systems methodology

## 1990s

- Object-oriented programming (OOP) developed in the early 1960s, and became a dominant programming approach during the mid-1990s
- Rapid application development (RAD), since 1991
- Scrum, since 1995
- Extreme programming (XP), since 1999

## 2000s

- Agile Unified Process (AUP) maintained since 2005 by Scott Ambler

# RAPID SOFTWARE DEVELOPMENT

Rapid development and delivery is now often the most important requirement for software systems

- Businesses operate in a fast-changing requirement and it is practically **impossible to produce a set of stable software requirements**
- Software has to evolve **quickly to reflect changing** business needs.

Rapid software development

- Specification, design and implementation are **interleaved**
- System is developed **as a series of versions with stakeholders** involved in version evaluation
- User interfaces are often developed using an IDE and graphical toolset.

(IDE = Integrated Development Environment)

# SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC) MODELS

There are various software development approaches defined and designed which are used/employed during development process of software, these approaches are also referred as **“Software Development Process Models”**

- Waterfall model
- V-model
- Incremental model
- Iterative model
- etc.

Each process model follows a particular life cycle in order to ensure success in process of software development.

# SOFTWARE DEVELOPMENT LIFE CYCLE MODEL PHASES

There are the following six phases in every Software Development Life Cycle model:

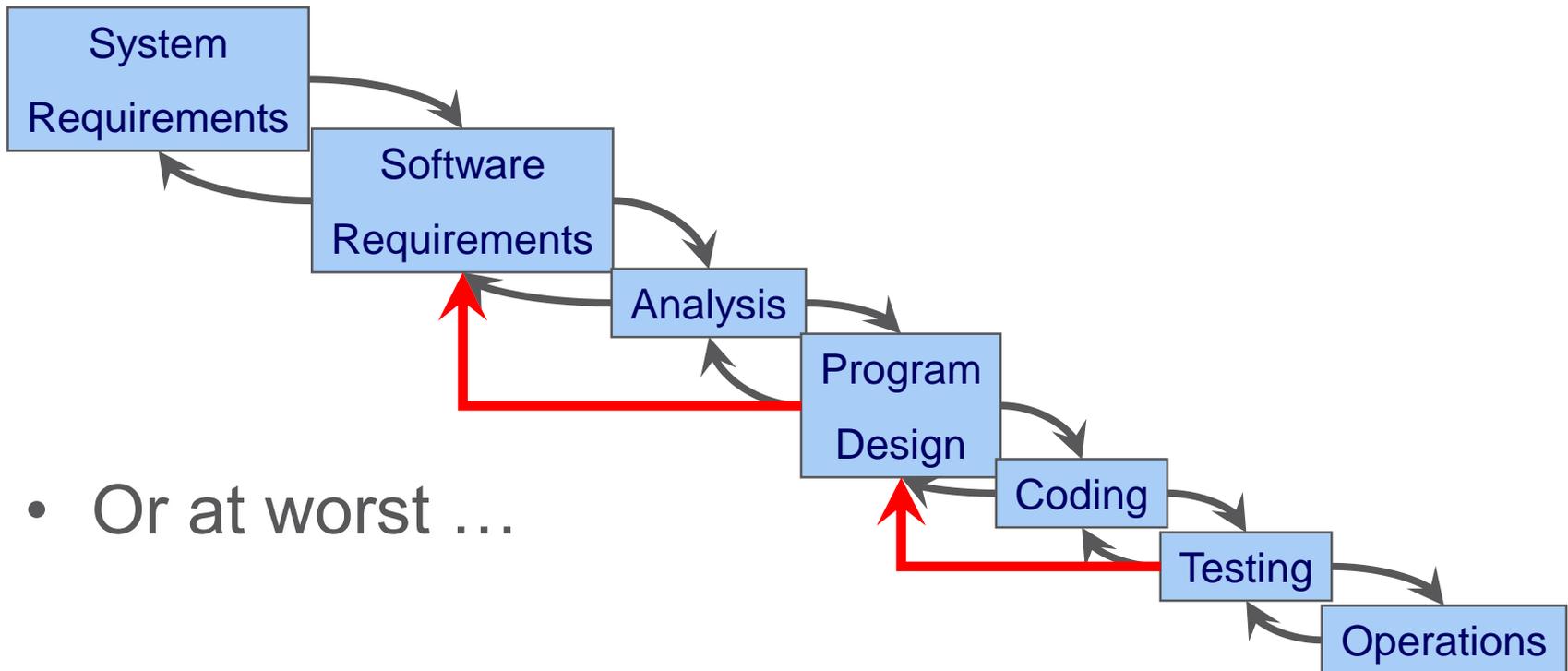
- Requirement gathering and analysis
- Design
- Implementation or coding
- Testing
- Deployment
- Maintenance

# THE WATERFALL DEVELOPMENT PROCESS



# THE WATERFALL PROCESS

› The traditional development process: Sequential design!



# WATERFALL MODEL

The Waterfall Model was first Process Model to be introduced. It is also referred to as a **linear-sequential life cycle model**.

It is very simple to understand and use.

In a waterfall model, each phase must **be completed fully** before the next phase can begin.

This type of model is basically used for the for the project which is **small** and there are no uncertain requirements.

At the end of each phase, a review takes place to determine if the project is on the right path and whether or not to continue or discard the project.

In this model the **testing starts only after the development is complete**. In **waterfall model phases** do not overlap.

# WATERFALL MODEL

## Advantages of waterfall model:

- This model is simple and easy to understand and use.
- It is easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
- In this model phases are processed and completed one at a time. Phases do not overlap.
- Waterfall model works well for smaller projects where requirements are very well understood.

## Disadvantages of waterfall model:

- Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
- No working software is produced until late during the life cycle.
- 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.

# WHERE DOES WATERFALL WORK WELL?

## Organizations where requirements don't change

- Stable requirements leads to stable design...
- Stable design leads to “no surprise” implementation

## Extremely high-reliability systems (product or custom projects), where functions are very well understood and no changes in requirements during a project are desired

- Who has requirements like this?
- NASA

## Embedded products with hardware constraints that cannot be easily changed

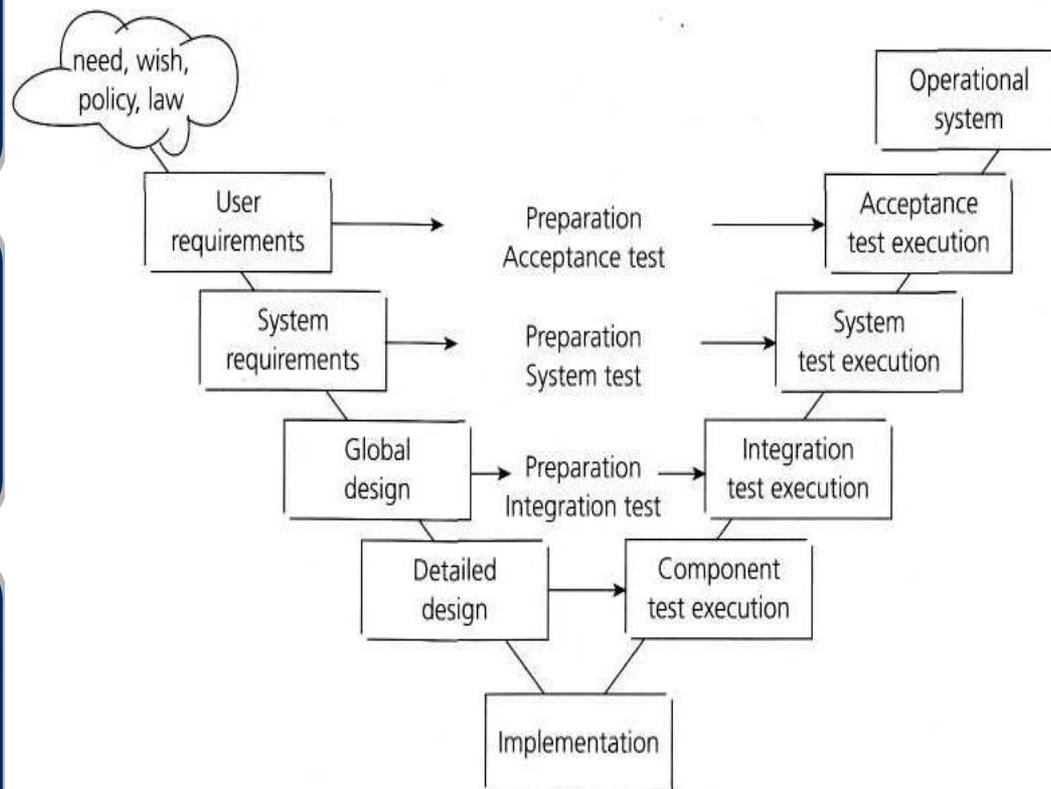
## Complex projects (product or custom) where parts of design & coding are outsourced, off-shored, or done in multiple sites, AND there are weak mechanisms to synchronize and manage distributed teams

# V-MODEL (VERIFICATION AND VALIDATION)

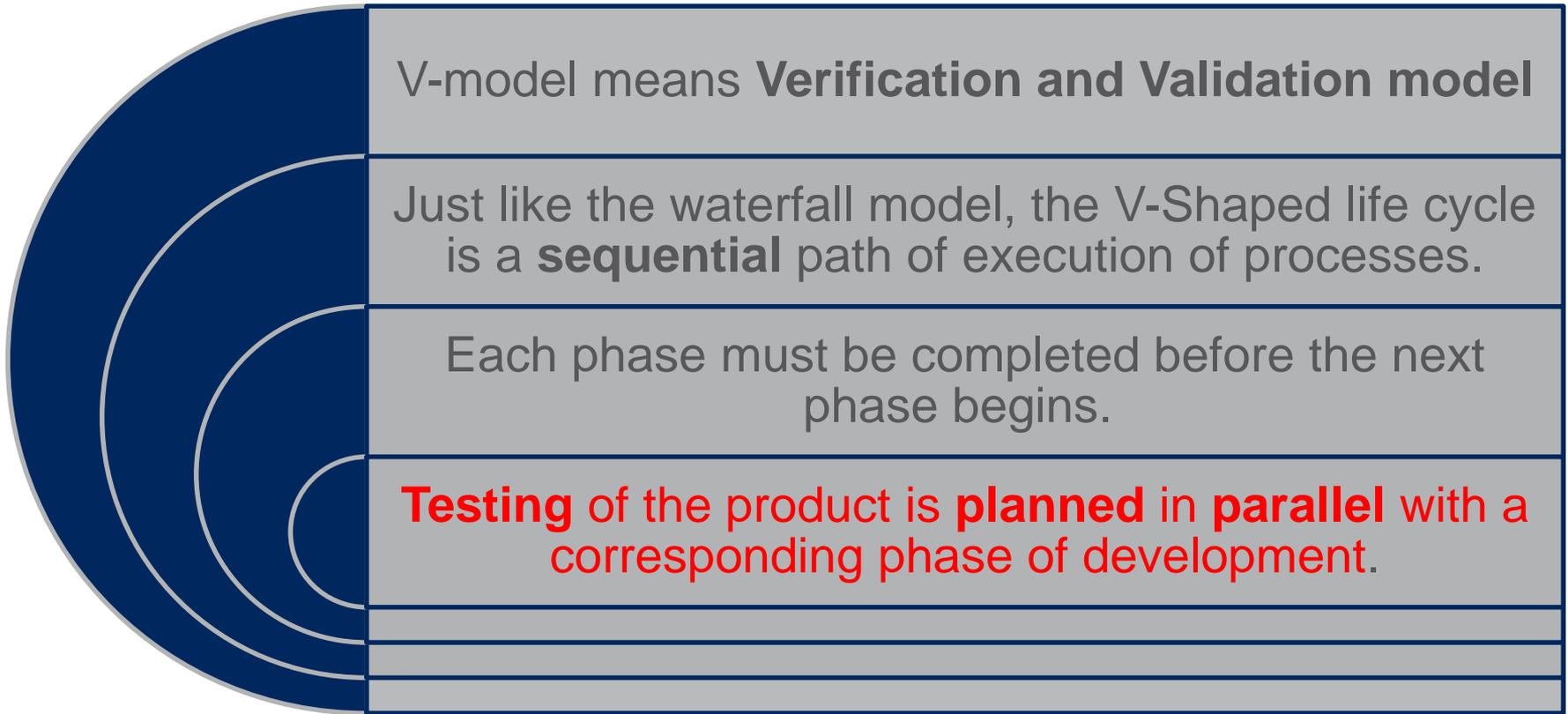
The V-model is an extension of the waterfall model

Shows the relationships between development phases and test phases

Time and project completeness vs. level of abstraction



# V-MODEL



# V-MODEL

## Advantages of V-model:

- Simple and easy to use.
- **Testing activities** like planning, test designing happens well before coding. This saves a lot of time. Hence higher chance of success over the waterfall model.
- Proactive defect tracking – that is defects are found at early stage.
- Avoids the downward flow of the defects.
- Works well for small projects where requirements are easily understood.

## Disadvantages of V-model:

- Very rigid and least flexible.
- Software is developed during the implementation phase, so no early prototypes of the software are produced.
- If any changes happen in midway, then the test documents along with requirement documents has to be updated.

# INCREMENTAL MODEL

The 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.

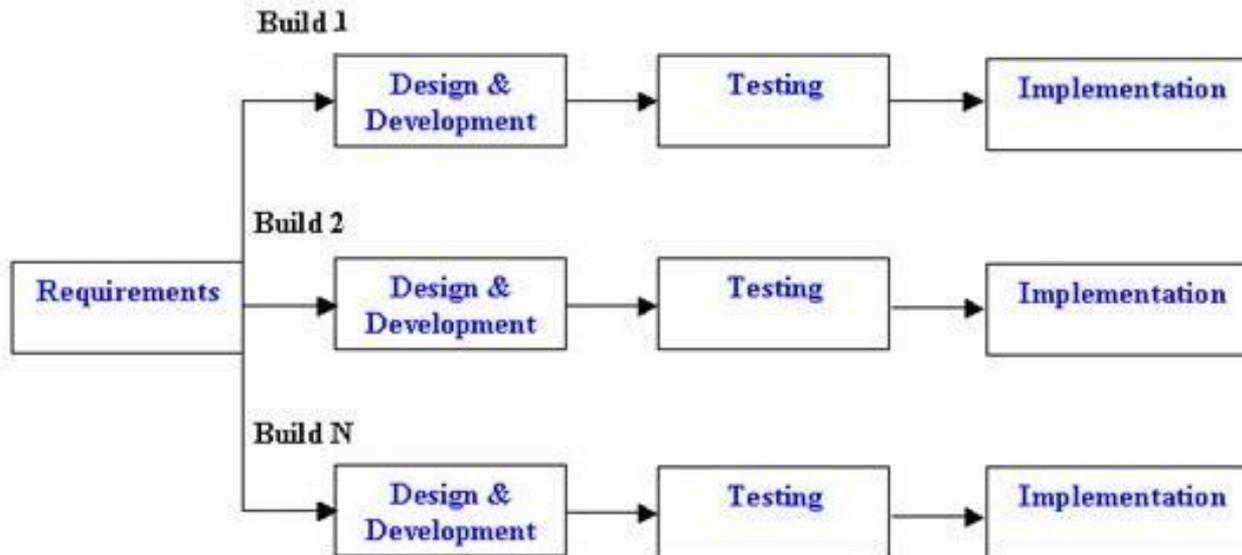
Each module passes through the requirements, design, implementation and testing phases.

A working version of software is produced during the first module, so you have working software early on during the software life cycle.

Each subsequent release of the module adds function to the previous release.

The process continues till the complete system is achieved.

# INCREMENTAL MODEL



Incremental Life Cycle Model

# INCREMENTAL MODEL

## Advantages of Incremental model:

- **Generates working software quickly and early during the software life cycle.**
- This model is more flexible – less costly to change scope and requirements.
- It is easier to test and debug during a smaller iteration.
- In this model customer can respond to each build.
- Lowers initial delivery cost.
- Easier to manage risk because risky pieces are identified and handled during iteration.

## Disadvantages of Incremental model:

- **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 in waterfall

# WHEN TO USE THE INCREMENTAL MODEL

This model can be used when the requirements of the complete system are clearly defined and understood.

Major requirements must be defined; however, some details can evolve with time.

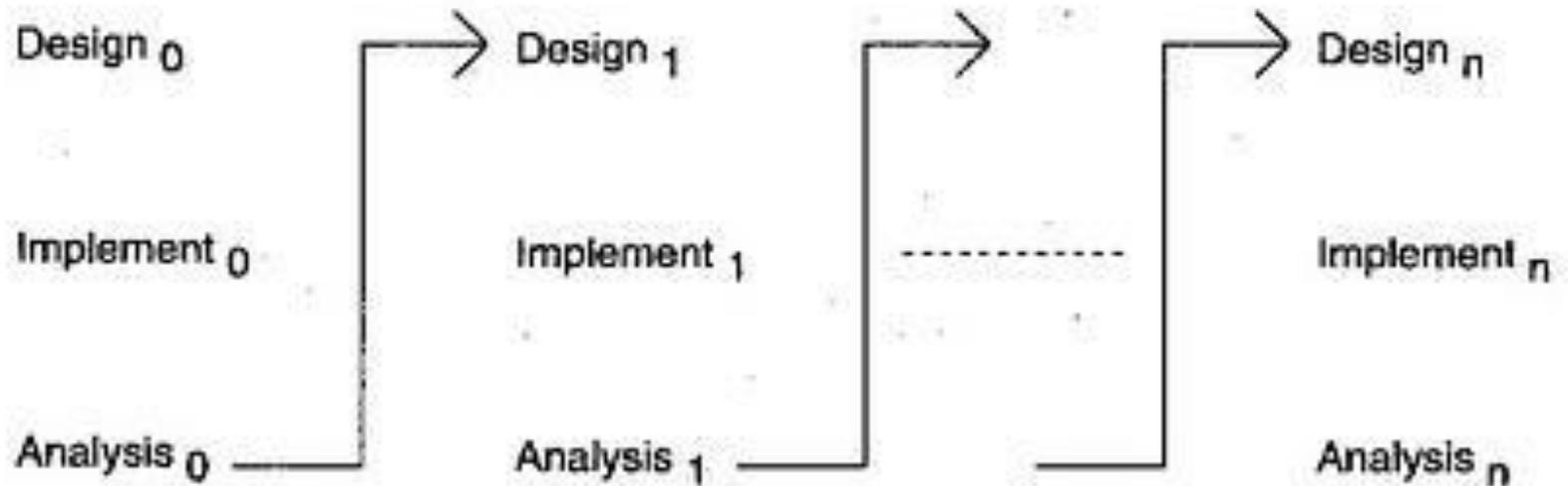
There is a need to get a product to the market early.

A new technology is being used

Resources with needed skill set are not available

There are some high risk features and goals.

# ITERATIVE LIFE CYCLE MODEL



# ITERATIVE LIFE CYCLE MODEL

Does not attempt to start with a full specification of requirements

Instead, development begins by specifying and implementing just part of the software, which can then be reviewed in order to identify further requirements

This process is then repeated, producing a new version of the software for each cycle of the model

# ADVANTAGES OF ITERATIVE MODEL

In iterative model we can only create a high-level design of the application before we actually begin to build the product.

In iterative model we are building and improving the product step by step.

In iterative model we can get the reliable user feedback.

In iterative model less time is spent on documenting and more time is given for designing.

# DISADVANTAGES OF ITERATIVE MODEL

Each phase of an iteration is rigid with no overlaps

Costly system architecture or design issues may arise because not all requirements are gathered up front for the entire lifecycle

# WHEN TO USE ITERATIVE MODEL

When the project is big.

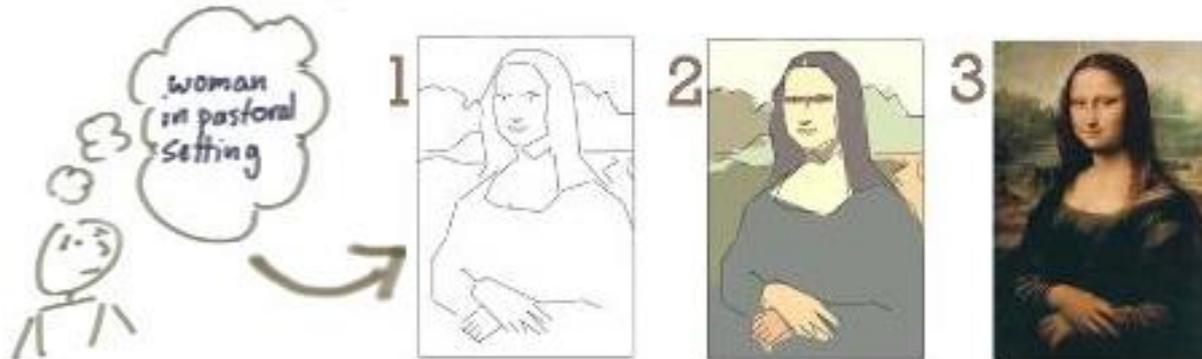
Major requirements must be defined; however, some details can evolve with time.

# INCREMENTAL VS. ITERATIVE

## Incremental



## Iterative



# AGILE MODEL

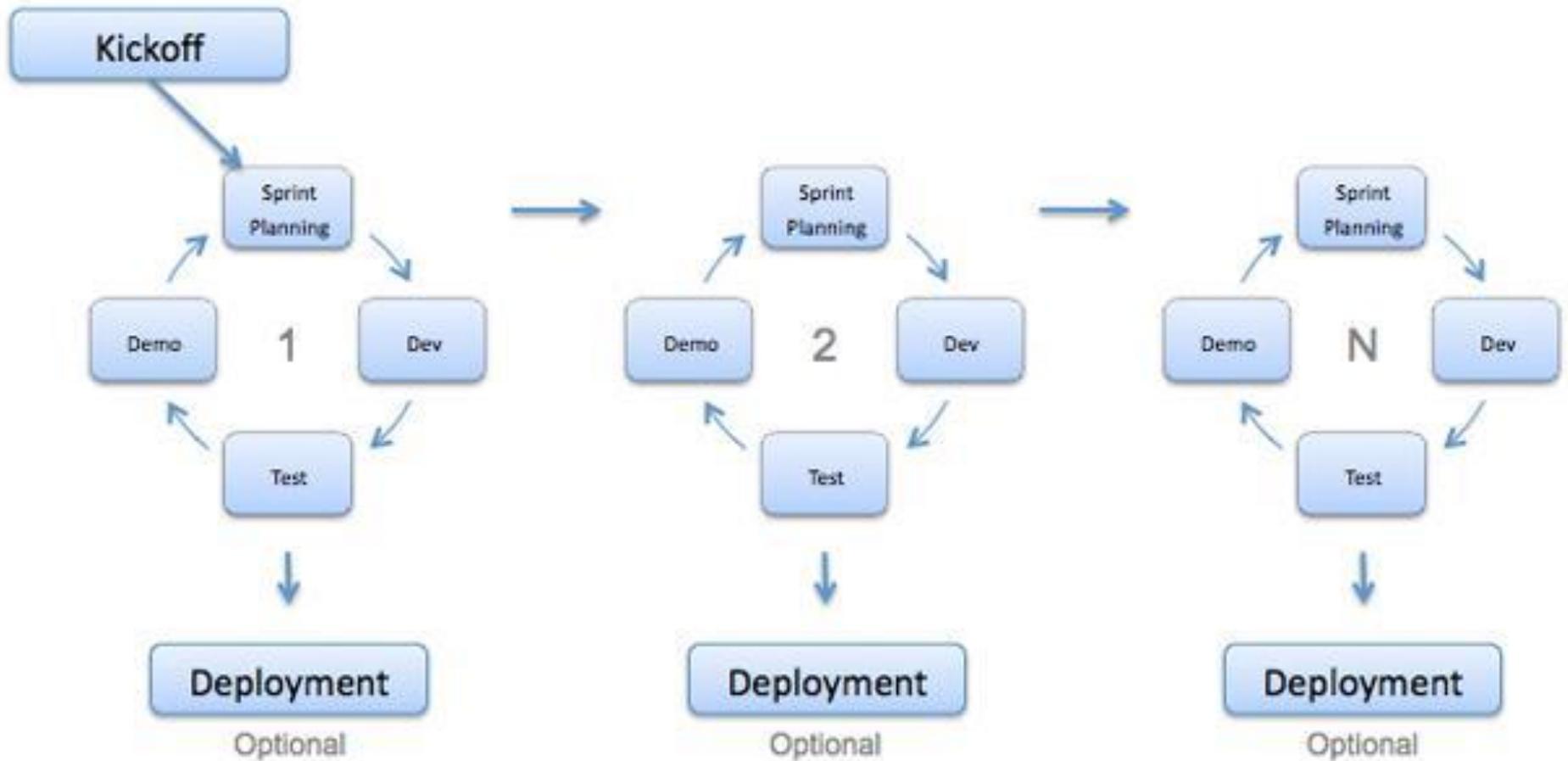
Type of Incremental model

Software is developed in incremental, rapid cycles. This results in small incremental releases with each release building on previous functionality

Each release is thoroughly tested to ensure software quality is maintained

It is used for time critical applications. Extreme Programming (XP) is currently one of the most well known agile development life cycle model

# AGILE MODEL (SCRUM)



# ADVANTAGES OF AGILE MODEL

- › Customer satisfaction by rapid, continuous delivery of useful software.
  - Working software is delivered frequently (weeks rather than months).
- › People and interactions are emphasized rather than process and tools. Customers, developers and testers constantly interact with each other.
  - Face-to-face conversation is the best form of communication.
  - Close, daily cooperation between business people and developers.
- › Continuous attention to technical excellence and good design.
- › Regular adaptation to changing circumstances.
  - Even late changes in requirements are welcome

# DISADVANTAGES OF AGILE MODEL

In case of some software deliverables, especially the large ones, it is difficult to assess the effort required at the beginning of the software development life cycle.

There is lack of emphasis on necessary designing and documentation.

The project can easily get taken off track if the customer representative is not clear what final outcome that they want.

Only senior programmers are capable of taking the kind of decisions required during the development process. Hence it has no place for newbie programmers, unless combined with experienced resources.

# WHEN TO USE AGILE MODEL

When new changes are needed to be implemented.

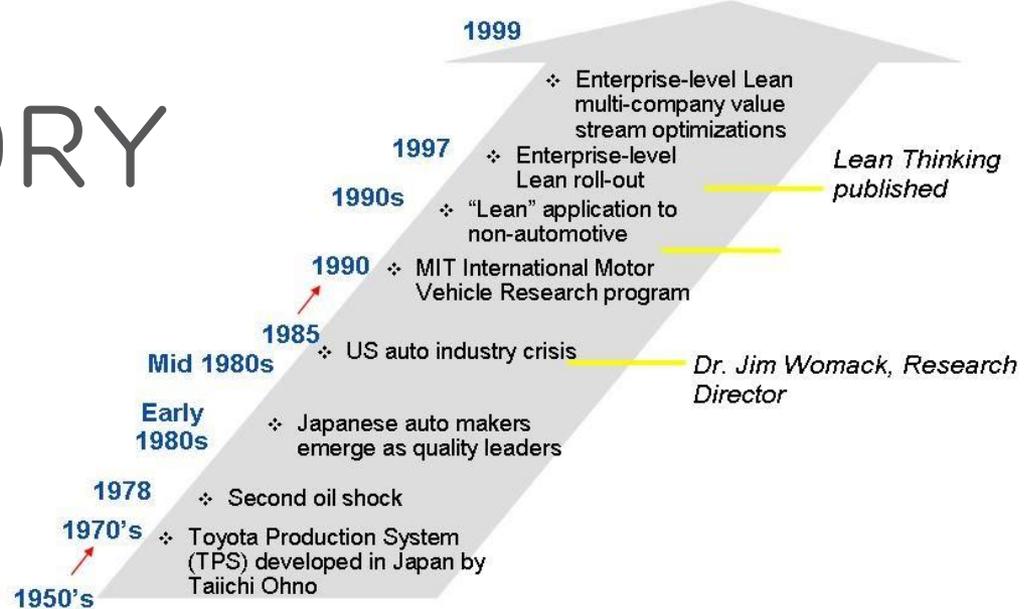
To implement a new feature the developers need to lose only the work of a few days, or even only hours, to roll back and implement it.

Unlike the waterfall model in agile model very limited planning is required to get started with the project.

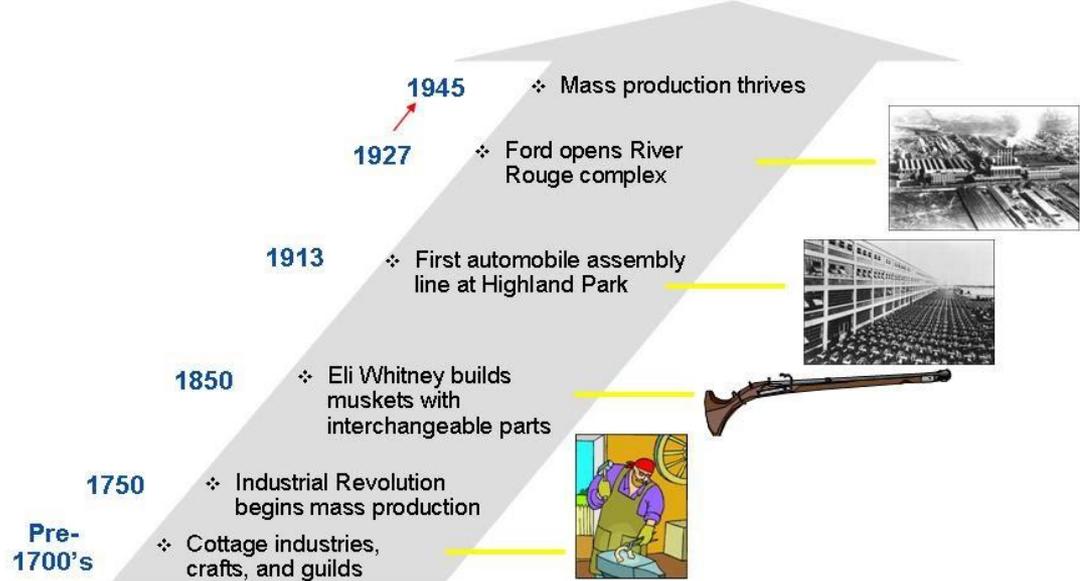
# LEAN

- › Lean philosophy regards everything not adding value to the customer as waste

# LEAN HISTORY



**Lean developed in a Manufacturing background**



**Dates back to Mass Production days**

# PRINCIPLES OF LEAN

1. Eliminate Waste

2. Build Quality In

3. Create Knowledge

4. Defer Commitment (Decide as late as possible)

5. Deliver Fast

6. Respect People

7. Optimize the Whole

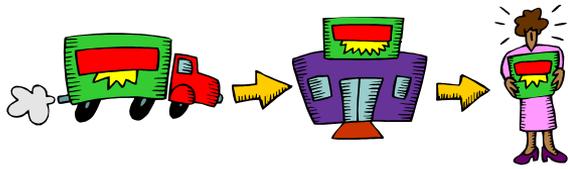


# ELIMINATE WASTE

What is waste in software?

Any activity or product that does not provide value to customers – anything a customer would not pay for

# 7 WASTES IN SW DEVELOPMENT



Extra Processes



Extra features



Partially done work



Defects



Task switching



Motion and handover



Waiting & Delays



# BUILD QUALITY IN

Cost of fixing a shipped product is much higher than fixing a product that is being built

Think how to test before starting



# CREATE KNOWLEDGE

Amplify learning

Share knowledge gained with the whole team



# DEFER COMMITMENT

Decide as late as possible

- Keep your options open up to the **last responsible minute**
- Wait until you have better information to make the decision
- Be flexible to react to the changes that will surely happen in the market and in the technology



# DELIVER AS FAST AS POSSIBLE

Deliver quickly to maximize return on investment, reduce risk, and get feedback from real customers and users



# RESPECT PEOPLE



# OPTIMIZE THE WHOLE

Improve the entire system

Find weakest link/biggest problem in your whole system and fix that first

More programmers – not enough testers: cannot deliver more value

# HOW IT ALL FITS TOGETHER

