SEQUENTIAL VERSUS INCREMENTAL AND ITERATIVE DEVELOPMENT MODELS
# MODELS, FRAMEWORKS AND METHODOLOGIES

## Model

“...all models are wrong, but some are useful.”

"truth ... is much too complicated to allow anything but approximations“

- Abstract representation of an entire software process.
- Does not specify how to do things
  Rather: Outlines the types of things that are done
e.g. Waterfall defines phases, but not artefacts.
  Agile defines values, but not methods

---

## Methodology

- Can be thought of as a “recipe”, or the application of related processes, methods, and tools
to a class of problems that all have something in common.
- Demonstrate a well thought out, defined and repeatable approach

---

## Framework

- A combination of set protocols, rules, standards and guidelines that can be incorporated or
followed as a whole so as to leverage the benefits of the scaffolding provided by the
Framework.

---

Robustness in Statistics, Academic Press

John von Neumann, 1947

“Often deciding on a methodology has more in common to joining a cult than making a technical decision”

Capers Jones & Associates:2011
LIFE CYCLE PROCESS MODELS

Every man made system has a life cycle, even if it is not formally defined.

By it’s nature Software is intangible and malleable
• Permits a wide variety of software life cycles.

Generally grouped into
• Linear (Sequential) – Requirements up front to the extent possible
• Adaptive (Iterative and Incremental) – Evolution of requirements

INCOSE Handbook
Swebok
SEQUENTIAL DEVELOPMENT

Sequential Development

• Traditional way

• Systematic approach that adheres to specified processes as the system moves through a series of representations

• Attention to completeness of documentation, traceability and verification of each representation after the fact

• Strengths: predictability, stability, repeatability, assurance

• Process improvement: standardization, measurement, and control

• Weakness: unprecedented projects, projects with a high rate of unforeseeable change, predictability, and stability
Incremental and Iterative Development (IID)

- Initial capability followed by successive deliveries
- Goal: rapid value and responsiveness
- Useful when the requirements are unclear and cannot be further clarified, or possibilities of inserting new technology
- Velocity and adaptability
- Customer incorporated into working level teams for fast and continuous feedback
- Risk: Unstable project due to frequently changing requirements, too much focus on short term or localized solutions
- May also be plan driven
LIFECYCLE DRIVERS

The best choice of lifecycle is determined by:

- Nature of requirements
- Requirement Stability
- Scope Flexibility
- Minimal Viable product
- Budget Flexibility
- Time Flexibility
- Technology Risk
- Criticality
- Solution Complexity
NEXT UP

• Waterfall

• Incremental

• Iterative and Incremental

• Prototyping

• Agile

• Spiral
BENEFIT (AND RISK) OF SEQUENTIAL DEVELOPMENT

Cost

Influence over final project cost

Cumulative Project Cost Over Time

Management Involvement

Time

Cumulative Cost ($$$)
Cumulative Cost ($$)

THE WATERFALL MODEL

Formally described in 1970 (Sort of)

Phases are sequential
• Entire system behaviour is described first
• Then designed
• Then Implemented, etc.

- Emphasis is on planning, schedules, target date and budget.
- Implementation of entire system at one time
WHEN TO USE WATERFALL

Can be applied successfully in:

- Typically applicable in “Simple” projects
- Stable business environment where the requirements *can* be well understood
- All functionality is required on first release
- Technology is well understood
6 UNDERLYING ASSUMPTIONS REQUIRED BY WATERFALL

1. The requirements are knowable in advance of implementation.
2. The requirements have no unresolved, high-risk implications, such as risks due to COTS choices, cost, schedule, performance, safety, security, user interfaces, and organizational impacts.
3. The nature of the requirements will not change very much either during development or evolution.
4. The requirements are compatible with all the key system stakeholders’ expectations, including users, customer, developers, maintainers, investors.
5. The right architecture for implementing the requirements is well understood.
6. There is enough calendar time to proceed sequentially. These assumptions must be met by a project if the waterfall model is to succeed.

Spiral Development: Experience, Principles, and Refinements, Feb 2000, Barry Boehm
WATERFALL ADVANTAGES

ONLY GIVEN THE PREVIOUS CONDITIONS

• Model is simple and easy to understand and use

• Easy to manage due to the model’s rigidity
  • Provides a simple method for partitioning the work

• Progress is clearly visible
  • Project progress is easily visible to management

• Encourages correctness by design
  • Review of intermediate deliverables are easily managed
WATERFALL DISADVANTAGES

Requirements are assumed to be correct on an ongoing basis
- Any risk of requirement change can have detrimental impact on project

Error recovery is expensive
- Any problems discovered in subsequent phases are difficult and expensive to recover from. E.g. Design problems picked up in coding may require extensive unplanned rework

Limited customer involvement in the design phase
- Customer has little insight into design decisions in terms of for instance usability until later phases (testing and operations), and may require expensive redesign or rework
WATERFALL DISADVANTAGES

Influence over final project cost

Total Project Cost

Cost

Time
INCREMENTAL MODEL

- **Requirements**
  - Build 1
    - Design & Develop
    - Testing
    - Implementation
  - Build 2
    - Design & Develop
    - Testing
    - Implementation
  - Build 3
    - Design & Develop
    - Testing
    - Implementation
INCREMENTAL CHARACTERISTICS

Incremental development model follows a linear path of progression, similar to waterfall.

- All Requirements are gathered, and broken down into logical increments.
- Architecture is completed for the system as a whole, making provision for an incremental approach
- A usable set of functions are designed in detail and released in a base version
- A next increment is then designed and implemented

Though all requirements are gathered at the start, the model allows for additional requirements to be added later.
APPLYING INCREMENTAL

Incremental is works when time to market is critical
- Where essential to bring innovation to market and grab commercially viable market share

If a subset of functionality provides benefit
- Business value of a subset of all required functions can be realised quickly

Limited Resources and Budget
- The application has to pay for itself before more functionality can be added

There is Technology risk
- Technology risk can be reduced when applying it over multiple increments and incorporating lessons learnt
INCREMENTAL ADVANTAGES

- Generate working software quickly and earlier
- More flexible, less costly to change scope and requirements
- Easier to test and debug during a smaller increments
- Opportunity for customer feedback in each increment
- Lower delivery cost of initial functionality
- Advantage to handling risk, as riskier items can be focused on early

OVER WATERFALL
INCREMENTAL DISADVANTAGES

- More complicated planning and design
- Total cost is higher than waterfall
- Requires complete definition of the system before increments can be planned effectively
- Implementation of Requirements tend to get postponed
ITERATIVE AND INCREMENTAL (IID) DEVELOPMENT

By the late 1980s, the DoD was experiencing significant failure in acquiring software based on the strict, document-driven, single-pass waterfall model that DoD-Std-2167 required.

A 1999 review of failure rates in a sample of earlier DoD projects drew grave conclusions: “Of a total $37 billion for the sample set, 75% of the projects failed or were never used, and only 2% were used without extensive modification.”

IID MODEL

Initial Planning

Requirements

Planning

Analysis & Design

Implementation

Evaluation

Testing

Deployment
IID CHARACTERISTICS

Start with limited set of requirements, then refine and expand in iterations

All system requirements are not captured prior to design start as in Plan Driven
• Known requirements are captured and delivered in an **Initialisation Step**
• Learning from **development** leads to design modification
• Learning from **use** leads to capture of new requirements
• Delivered in next **Iteration**, broken down into **Increments**
• Work done in **Increments** are integrated as they are completed
IID ADVANTAGES AND DISADVANTAGES

Advantages
- Successive iterations allows user feedback throughout development
- Benefits in defect handling, defects can be picked up earlier and resolved cheaper
- Learning from both development and use can be applied in iterations
- Lowers initial delivery cost, and initial delivery can occur faster
- The risk of costly system architecture changes are minimised in the approach, as cognition of full lifecycle requirements are progressively and incrementally improved

Disadvantages
- Integration may be more problematic than plan-driven development
PROTOTYPING MODEL

**Throwaway Prototyping**
- Outline spec
- Develop Prototype
- Evaluate Prototype
- Specify System
- Throw away prototype

**Evolutionary Prototyping**
- Develop abstract specification
- Build prototype
- Adequate?
  - No
  - Yes
  - Evaluate Prototype
- Delivery System
PROTOTYPING

Iterative development with regular user evaluation
- Prototype is developed with multiple iterations.
- The target users evaluate and experiment.
- User suggestions are incorporated into the next iteration.

Rapid turnaround of prototype
- Each iteration of prototype development is time boxed (1 to 3 weeks).

Validated requirements are documented
- Knowledge gained of features required are formally captured in a software requirements specification.

System developed from requirements
- At the completion of the last iteration system is developed from the resulting requirements.

Prototype is discarded.
MANY METHODOLOGIES CAME TO BE AGILE

Created prior to 2001:

• Rapid Application Development (1986)
• Scrum (1986),
• Adaptive Software Development (1995)
• Extreme Programming (1996)
AGILE MODEL

User Stories:

Capture Requirements
Design, Build & Test (BDT)

Complete?

Release 1

Capture Requirements
Design, Build & Test (BDT)

2-4 Weeks

Release 2

Capture Requirements
Design, Build & Test (BDT)

2-4 Weeks

Complete?
THE AGILE MANIFESTO (2001)

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

**Individuals and interactions** over **processes and tools**

**Working software** over **comprehensive documentation**

**Customer collaboration** over **contract negotiation**

**Responding to change** over **following a plan**

That is, while there is value in the items on the right, we value the items on the left more.
12 AGILE PRINCIPLES

- **Quality Means:**
  - System adheres to documentation
  - VS
  - Customer is happy

- Sustainable development, a
- Continuous attention to technical debt,
- Simplicity—the art of maximizing the amount of work not done,
- Best architectures, requirements, and designs emerge from self-organizing teams
- Regularly, the team reflects on how to become more effective, and adjusts accordingly

- **Discipline means:**
  - Compliance with established process
  - VS
  - Self-discipline

- **Agility Means:**
  - Inconsistent purpose, not stick to the plan
  - VS
  - Dexterity and skill
AGILE CHARACTERISTICS

• Embrace Change

  • Requirement Change is expected

  • Documentation through the SDLC

---

**Diagram:**

- Total Effort vs. Time
- Detailed Design Specification
- Detailed Requirements Specification
- Detailed Project Plan
- Deliverable Docs
- Support, Operations, and User Docs

---

Copyright 2006-2014 Scott W. Ambler
WHEN TO USE AGILE

- Time to market is critical
- Cost control is fixed, and features are variable
- Project scope lacks specificity, and is unlikely to remain stable
- Customer is willing and able to offer flexibility on project scope
WHEN NOT TO USE AGILE

- Fixed scope projects
- Geographically distributed teams and customers *
- Incremental delivery not possible
- Stakeholders are not committed to active involvement
- Safety/Mission critical applications
SPIRAL MODEL

1. Set goals
   - Determine, objectives, goals, alternatives, constraints
   - Define/refine the plan

2. Analyse risks
   - Evaluate alternatives
   - Select alternative
   - Identify and resolve risks
   - Plan work product development

3. Develop product
   - Develop work product
   - Verify the work product

4. Evaluate project
   - Evaluate work product
   - Evaluate project
   - Plan next cycle
     OR
   - Stop
WHY SPIRAL?

The Spiral Model:

• Provides a formal risk driven approach to iterative development
• Focuses developers on taking small, low cost steps with regular analysis of project viability
• Provides an overall framework within which you can apply other life cycle models as components.
APPLYING THE SPIRAL MODEL

High risk projects
• The target product is large & complex with ambitious use of new technologies

Mission critical projects
• The project is mission critical. Application failure will trigger company failure and bankruptcy

Fuzzy concepts
• System concepts must be formalized and evaluated before substantial funds are applied to development

Basic research involved
• The project will attempt to develop new technologies to implement the customer's requirements.
SPIRAL PHASE 1: SET GOALS

• Determine the project's objectives, goals, alternatives and constraints

• What is the project supposed to achieve?

• What problems will it solve?

• In the first pass around the spiral goal setting will focus on which user needs should be satisfied

• In later passes the objectives of designs and implementation tools and technologies will be the focus.
SPIRAL PHASE 2: ANALYZE RISKS

- Identify alternative courses of action
- Select the best alternative
- Analyze the risks and risk avoidance strategies
- Use various methods to evaluate alternatives.
SPIRAL PHASE 3: DEVELOP PRODUCT

Develop the work product using best alternative approach

- Initial cycles: Software Requirements Specification
- Later cycles: System Architecture Specification
- Still later cycles: Software product

Verify that it complies with its specification

- Initial cycles: verification is requirements and design reviews and traceability analysis
- Later cycles: verification is product testing.
SPIRAL PHASE 4: EVALUATE PROJECT

Evaluate the output of the current cycle
Review the progress of the project
Are we doing the right things?
YES => plan the next spiral
NO => Either
  • Change the project approach
  OR
  • Shut the project down.
OUTER SPIRALS
SPIRAL MODEL ADVANTAGES

Accurate requirements
- Provides a framework for developers and users to progressively learn about the product at low cost

Reduced risk
- Reduces project risk by dealing with risk in a formal manner

Early error detection
- Eliminates requirements errors and unworkable alternatives early

Failure avoidance
- Terminates non-viable projects early.
SPIRAL MODEL DISADVANTAGES

Difficult to perform at a fixed price
  • Progressive discovery is not applicable to fixed price fixed time contracted software development

Mature risk recognition skills required
  • Not recommended for use by immature developers who cannot recognize risk

Management overhead
  • Not recommended for projects where the application is well understood.
MODEL CONCLUSION

Consider the lifecycle drivers:

• Requirement comprehension and stability
• Scope, Budget and Time flexibility
• Minimal Viable product
• Technology Risk
• Criticality

One model does not fit all.