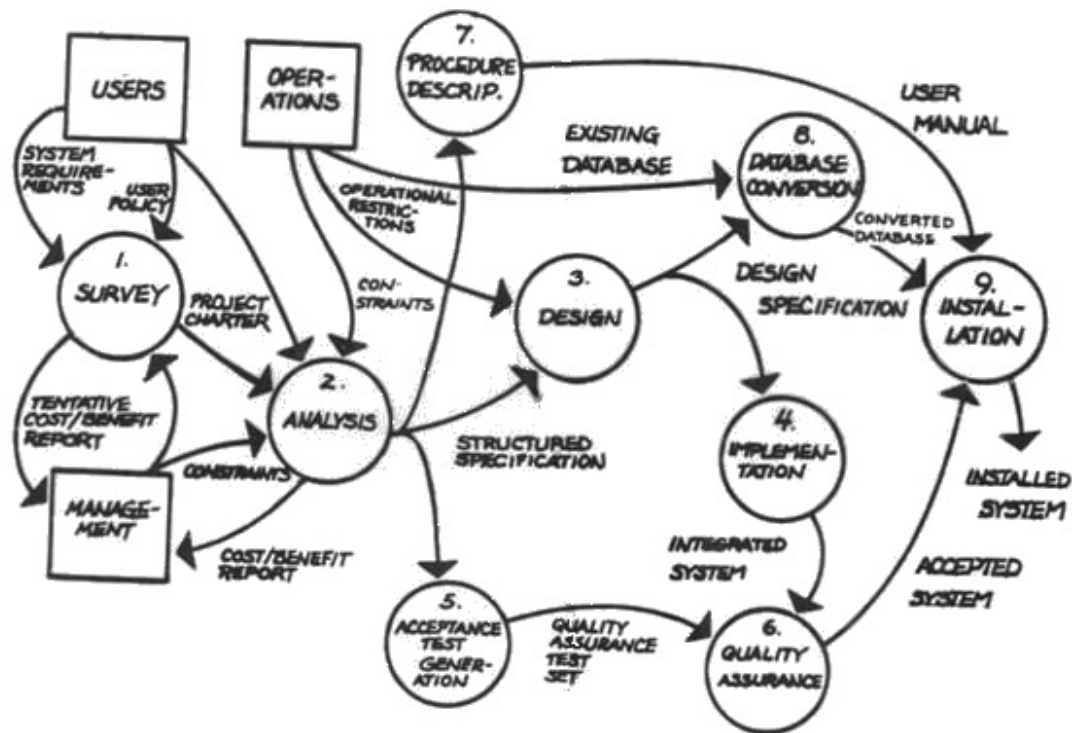


Software Development Life Cycle (SDLC)





How the customer explained it



How the Project Leader understood it



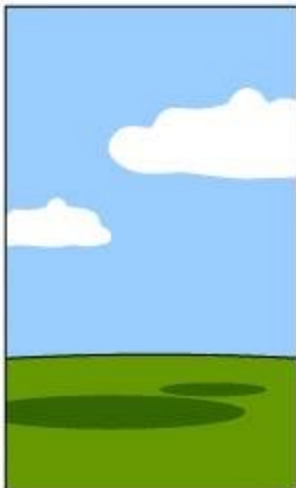
How the Analyst designed it



How the Programmer wrote it



How the Business Consultant described it



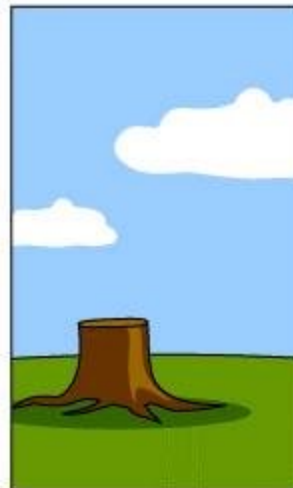
How the project was documented



What operations installed



How the customer was billed



How it was supported



What the customer really needed

SYSTEMS DEVELOPMENT LIFE CYCLE (SDLC)

- ***Systems development life cycle (SDLC)*** - a structured step-by-step approach for developing information systems
- Typical activities include:
 - Determining budgets
 - Gathering business requirements
 - Designing models
 - Writing user documentation

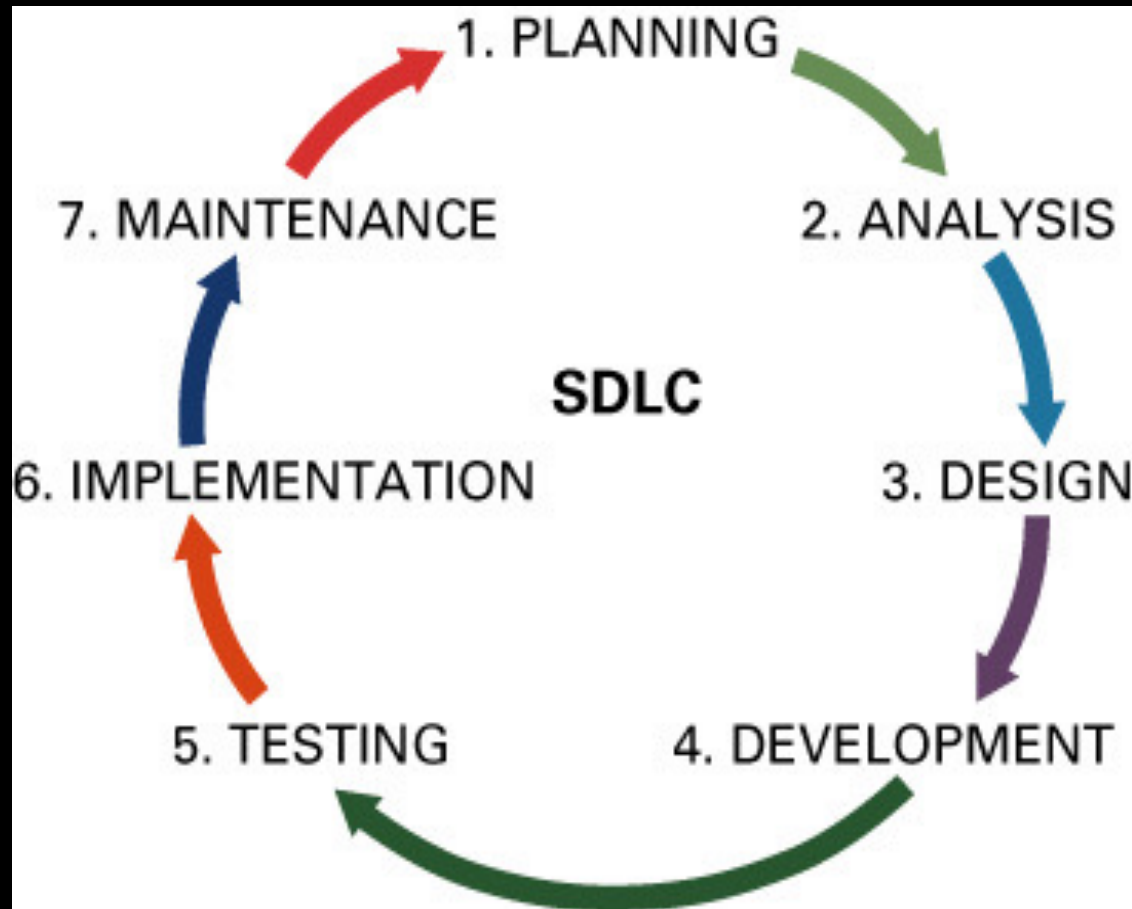
SYSTEMS DEVELOPMENT LIFE CYCLE (SDLC)

SDLC PHASE	ACTIVITIES
1. Planning	<ul style="list-style-type: none">•Define the system to be developed•Set the project scope•Develop the project plan
2. Analysis	<ul style="list-style-type: none">•Gather business requirements
3. Design	<ul style="list-style-type: none">•Design the technical architecture•Design system models
4. Development	<ul style="list-style-type: none">•Build technical architecture•Build databases and programs
5. Testing	<ul style="list-style-type: none">•Write test conditions•Perform testing
6. Implementation	<ul style="list-style-type: none">•Write user documentation•Provide training
7. Maintenance	<ul style="list-style-type: none">•Build a help desk•Support system changes

SYSTEMS DEVELOPMENT LIFE CYCLE (SDLC)

- The SDLC has 7 phases:
 1. Planning
 2. Analysis
 3. Design
 4. Development
 5. Testing
 6. Implementation
 7. Maintenance

SYSTEMS DEVELOPMENT LIFE CYCLE (SDLC)



Phase 1: Planning

- ***Planning phase*** - involves determining a solid plan for developing your information system
- Three primary planning activities:
 1. Define the system to be developed
 - ***Critical success factor (CSF)*** - a factor simply critical to your organization's success

Phase 1: Planning

2. Set the project scope

- ***Project scope*** - clearly defines the high-level system requirements
- ***Scope creep*** - occurs when the scope of the project increases
- ***Feature creep*** - occurs when developers add extra features that were not part of the initial requirements
- ***Project scope document*** - a written definition of the project scope and is usually no longer than a paragraph

Phase 1: Planning

3. Develop the project plan including tasks, resources, and timeframes

- ***Project plan*** - defines the what, when, and who questions of system development
- ***Project manager*** - an individual who is an expert in project planning and management, defines and develops the project plan and tracks the plan to ensure all key project milestones are completed on time
- ***Project milestones*** - represent key dates for which you need a certain group of activities performed

Phase 2: Analysis

- ***Analysis phase*** - involves end users and IT specialists working together to gather, understand, and document the business requirements for the proposed system

Phase 2: Analysis

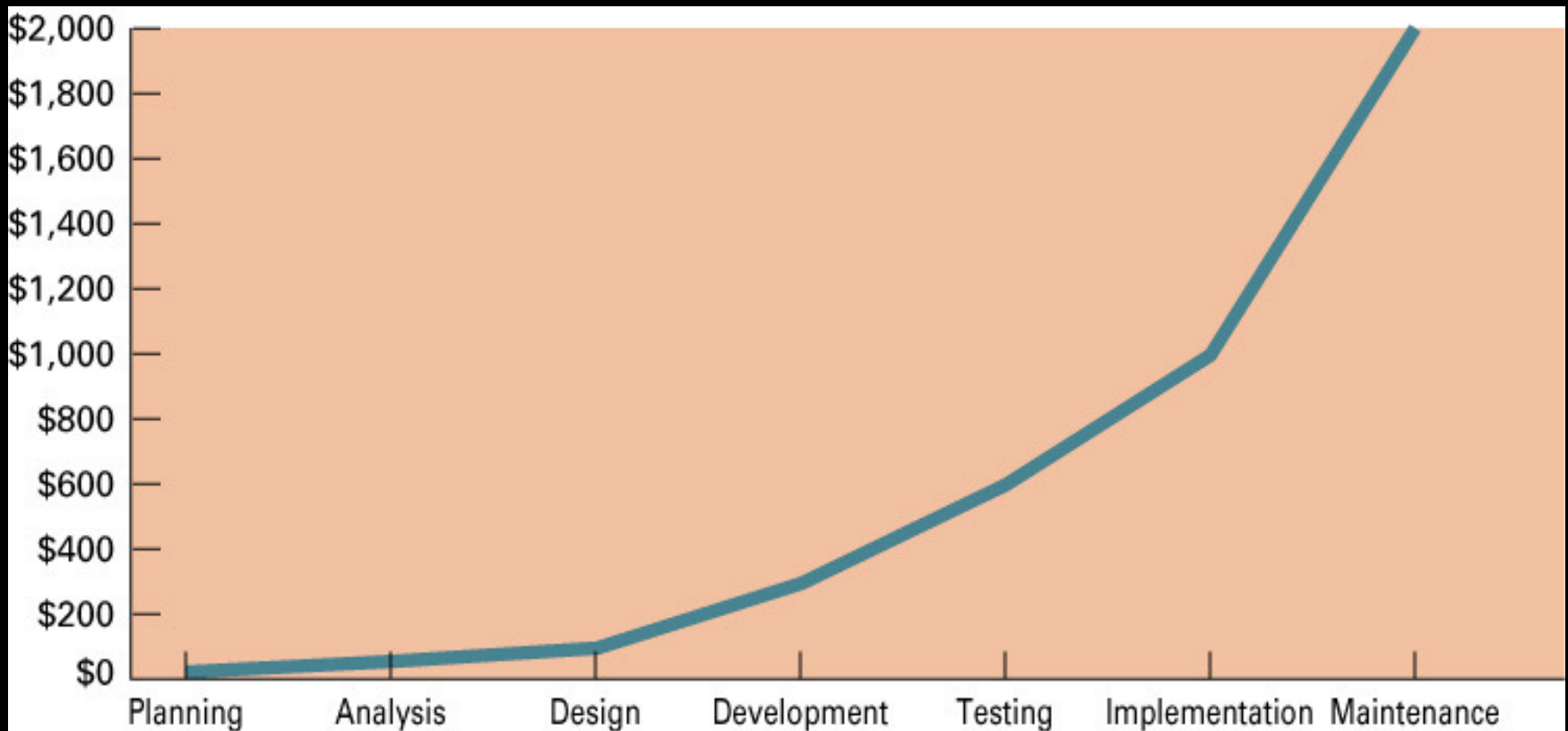
- Two primary analysis activities:
 1. Gather the business requirements
 - ***Business requirements*** - the detailed set of knowledge worker requests that the system must meet in order to be successful
 - ***Joint application development (JAD)*** - knowledge workers and IT specialists meet, sometimes for several days, to define or review the business requirements for the system

Phase 2: Analysis

2. Prioritize the requirements

- ***Requirements definition document*** – prioritizes the business requirements and places them in a formal comprehensive document

Phase 2: Analysis



Phase 3: Design

- ***Design phase*** - build a technical blueprint of how the proposed system will work
- Two primary design activities:
 1. Design the technical architecture
 - ***Technical architecture*** - defines the hardware, software, and telecommunications equipment required to run the system

Phase 3: Design

2. Design system models

- ***Modeling*** - the activity of drawing a graphical representation of a design
- ***Graphical user interface (GUI)*** - the interface to an information system
- ***GUI screen design*** - the ability to model the information system screens for an entire system

Phase 4: Development

- ***Development phase*** - take all of your detailed design documents from the design phase and transform them into an actual system
- Two primary development activities:
 1. Build the technical architecture
 2. Build the database and programs
 - Both of these activities are mostly performed by IT specialists

Phase 5: Testing

- ***Testing phase*** - verifies that the system works and meets all of the business requirements defined in the analysis phase
- Two primary testing activities:
 1. Write the test conditions
 - ***Test conditions*** - the detailed steps the system must perform along with the expected results of each step

Phase 5: Testing

2. Perform the testing of the system

- ***Unit testing*** – tests individual units of code
- ***System testing*** – verifies that the units of code function correctly when integrated
- ***Integration testing*** – verifies that separate systems work together
- ***User acceptance testing (UAT)*** – determines if the system satisfies the business requirements

Phase 6: Implementation

- ***Implementation phase*** - distribute the system to all of the knowledge workers and they begin using the system to perform their everyday jobs
- Two primary implementation activities
 1. Write detailed user documentation
 - ***User documentation*** - highlights how to use the system

Phase 6: Implementation

2. Provide training for the system users

- ***Online training*** - runs over the Internet or off a CD-ROM
- ***Workshop training*** - is held in a classroom environment and lead by an instructor

Phase 6: Implementation

- Choose the right implementation method
 - ***Parallel implementation*** – use both the old and new system simultaneously
 - ***Plunge implementation*** – discard the old system completely and use the new
 - ***Pilot implementation*** – start with small groups of people on the new system and gradually add more users
 - ***Phased implementation*** – implement the new system in phases

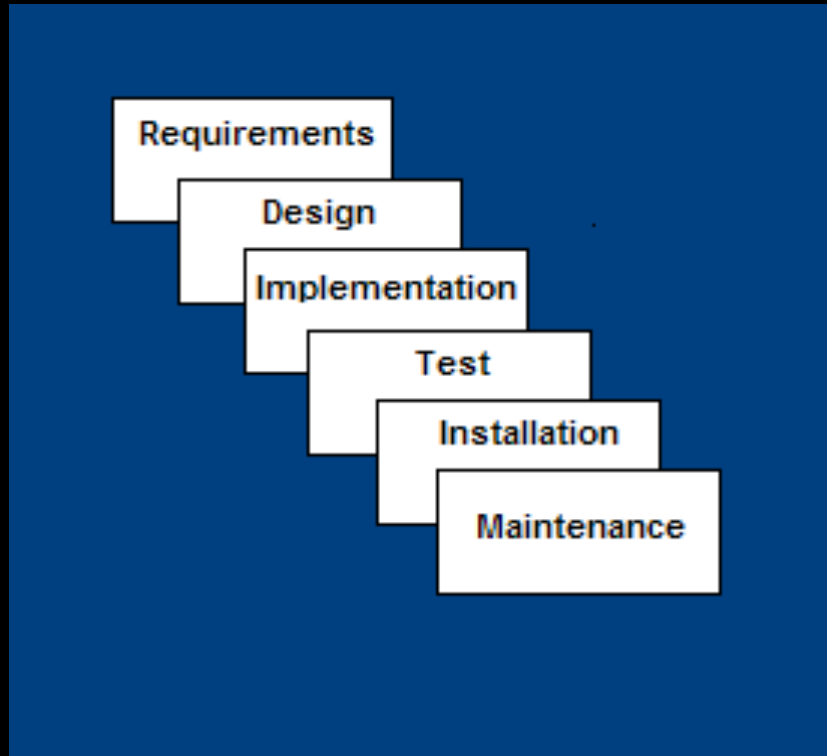
Phase 7: Maintenance

- ***Maintenance phase*** - monitor and support the new system to ensure it continues to meet the business goals
- Two primary maintenance activities:
 1. Build a help desk to support the system users
 - ***Help desk*** - a group of people who responds to knowledge workers' questions
 2. Provide an environment to support system changes

SYSTEMS DEVELOPMENT METHODOLOGIES

- Developers have different development methodologies:
 - Waterfall methodology
 - Rapid application development (RAD)
 - Extreme programming (XP)
 - Agile methodology

Waterfall Model



- **Requirements** – defines needed information, function, behavior, performance and interfaces.
- **Design** – data structures, software architecture, interface representations, algorithmic details.
- **Implementation** – source code, database, user documentation, testing.

Waterfall Strengths

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

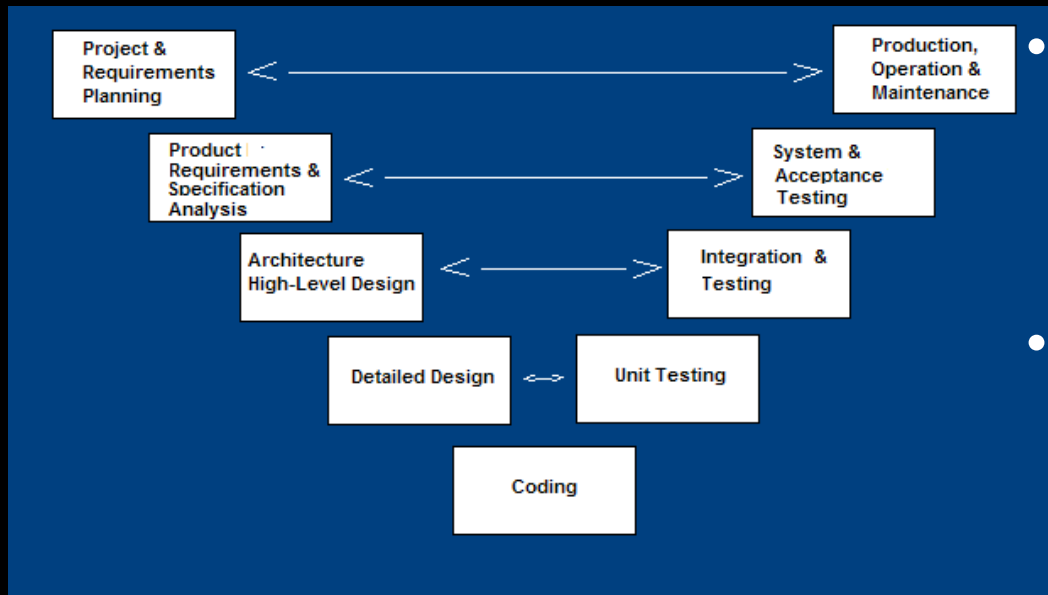
Waterfall Deficiencies

- 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)

When to use the Waterfall Model

- 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.

V-Shaped SDLC Model



- A variant of the Waterfall that emphasizes the verification and validation of the product.
- Testing of the product is planned in parallel with a corresponding phase of development

V-Shaped Steps

- **Project and Requirements Planning** – allocate resources
- **Product Requirements and Specification Analysis** – complete specification of the software system
- **Architecture or High-Level Design** – defines how software functions fulfill the design
- **Detailed Design** – develop algorithms for each architectural component
- **Production, operation and maintenance** – provide for enhancement and corrections
- **System and acceptance testing** – check the entire software system in its environment
- **Integration and Testing** – check that modules interconnect correctly
- **Unit testing** – check that each module acts as expected
- **Coding** – transform algorithms into software

V-Shaped Strengths

- Emphasize planning for **verification and validation** of the product in early stages of product development
- **Each deliverable must be testable**
- Project management can **track progress by milestones**
- **Easy to use**

V-Shaped Weaknesses

- Does not easily handle **concurrent events**
- Does not handle **iterations** or phases
- Does not easily handle **dynamic changes in requirements**
- Does not contain **risk analysis** activities

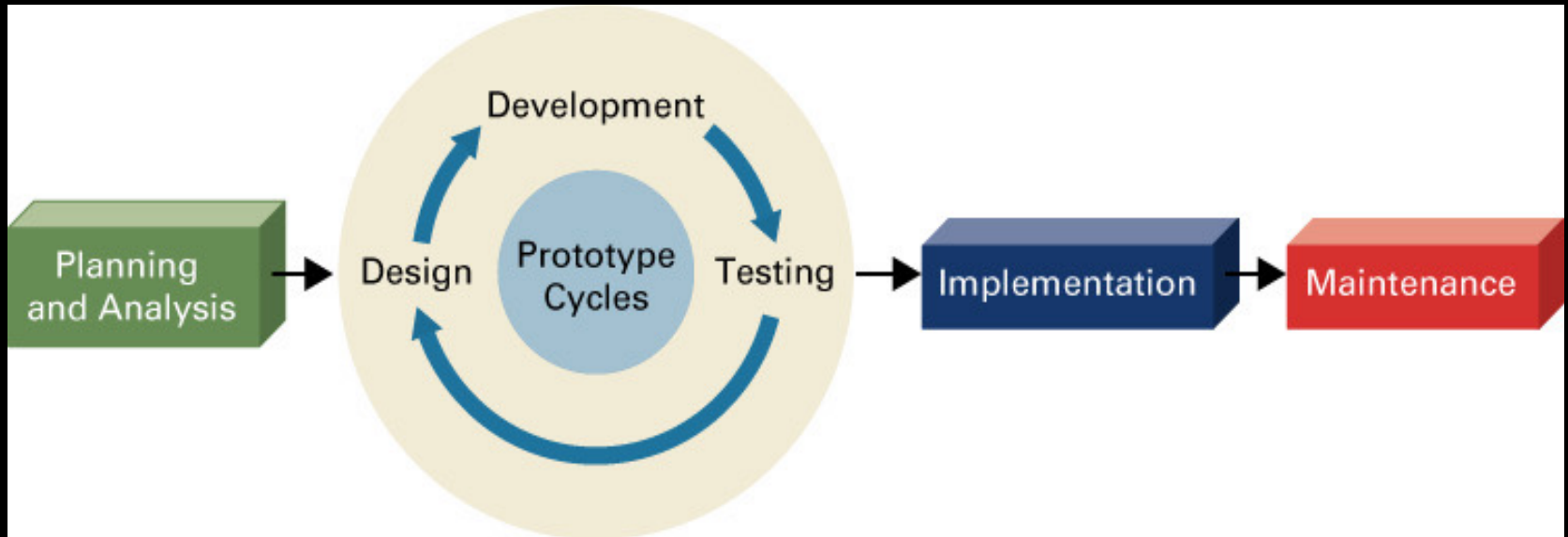
When to use the V-Shaped Model

- Excellent choice for **systems requiring high reliability**
 - hospital patient control applications
- **All requirements are known** up-front
- When it can be modified to **handle changing requirements beyond analysis phase**
- **Solution and technology are known**

Rapid Application Development (RAD)

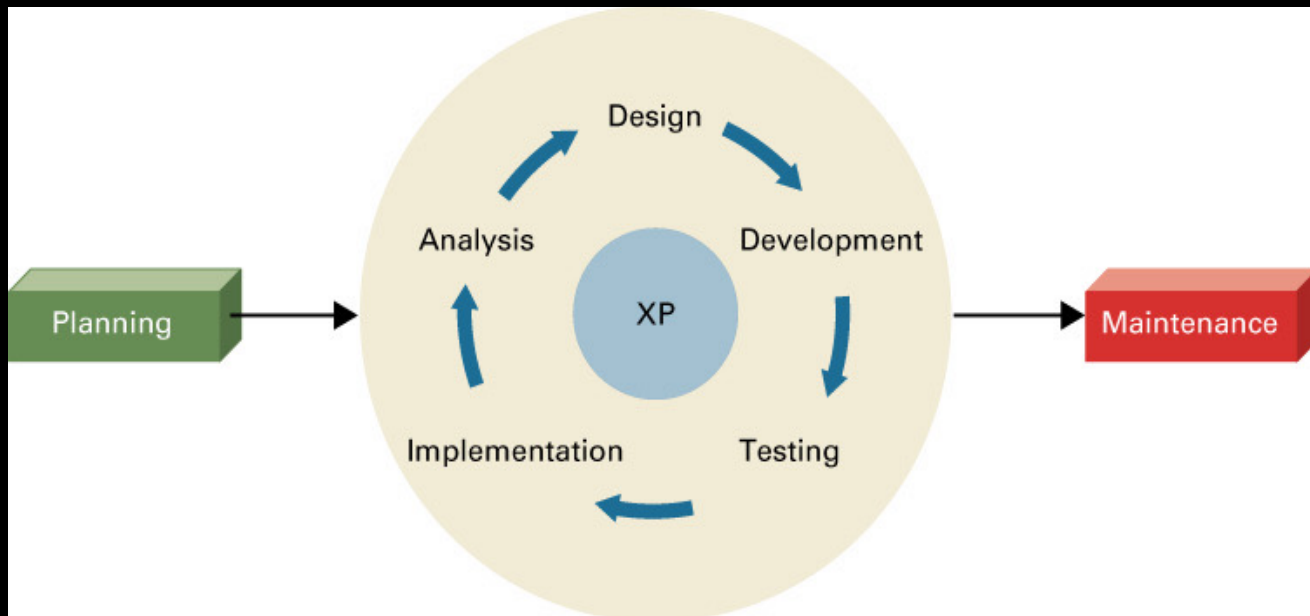
- ***Rapid application development (RAD)*** (also called ***rapid prototyping***) - emphasizes extensive user involvement in the rapid and evolutionary construction of working prototypes of a system to accelerate the systems development process
 - ***Prototype*** - a smaller-scale, representation, or working model of the user's requirements or a proposed design for an information system

Rapid Application Development (RAD)



Extreme Programming (XP)

- ***Extreme programming (XP)*** - breaks a project into tiny phases and developers cannot continue on to the next phase until the first phase is complete



Agile Methodology

- ***Agile methodology*** - a form of XP, aims for customer satisfaction through early and continuous delivery of useful software components

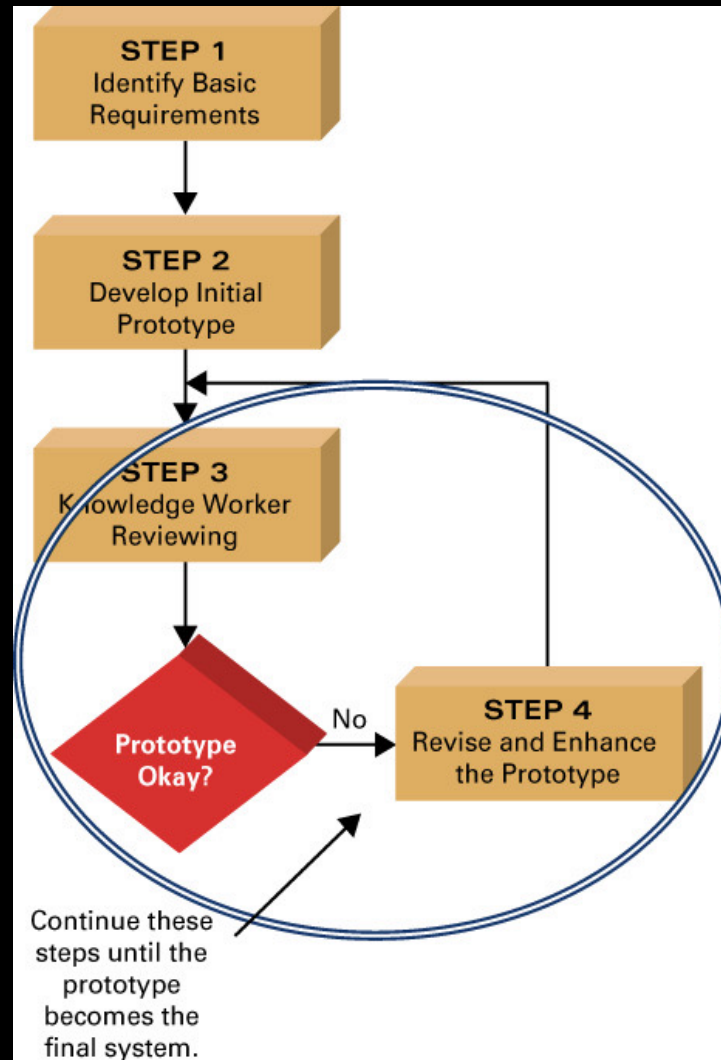
PROTOTYPING

- ***Prototyping*** - the process of building a model that demonstrates the features of a proposed product, service, or system
 - ***Proof-of-concept prototype*** - used to prove the technical feasibility of a proposed system
 - ***Selling prototype*** - used to convince people of the worth of a proposed system

The Prototyping Process

- The prototyping process involves four steps:
 1. Identify basic requirements
 2. Develop initial prototype
 3. User review
 4. Revise and enhance the prototype

The Prototyping Process



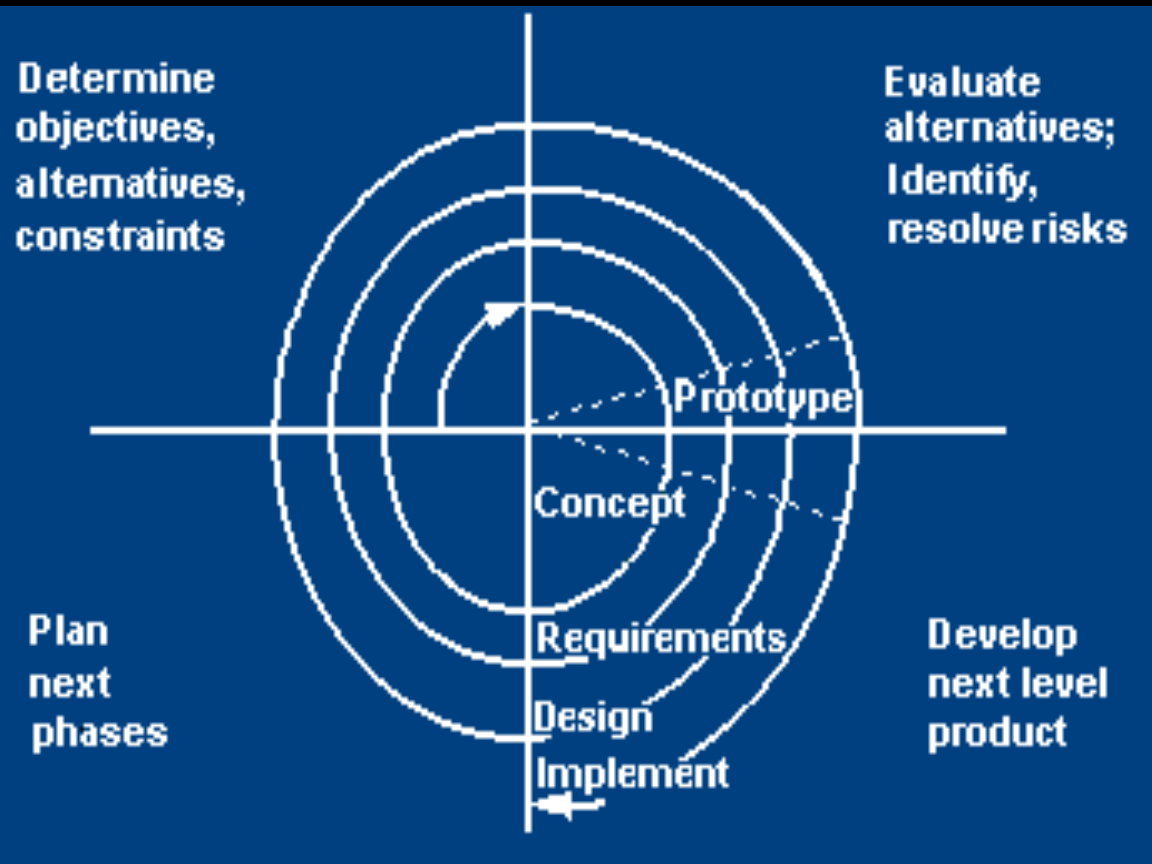
Advantages of Prototyping

- Encourages Active User Participation
- Helps Resolve Discrepancies Among Users
- Gives Users a Feel for the Final System
- Helps Determine Technical Feasibility
- Helps Sell the Idea of a Proposed System

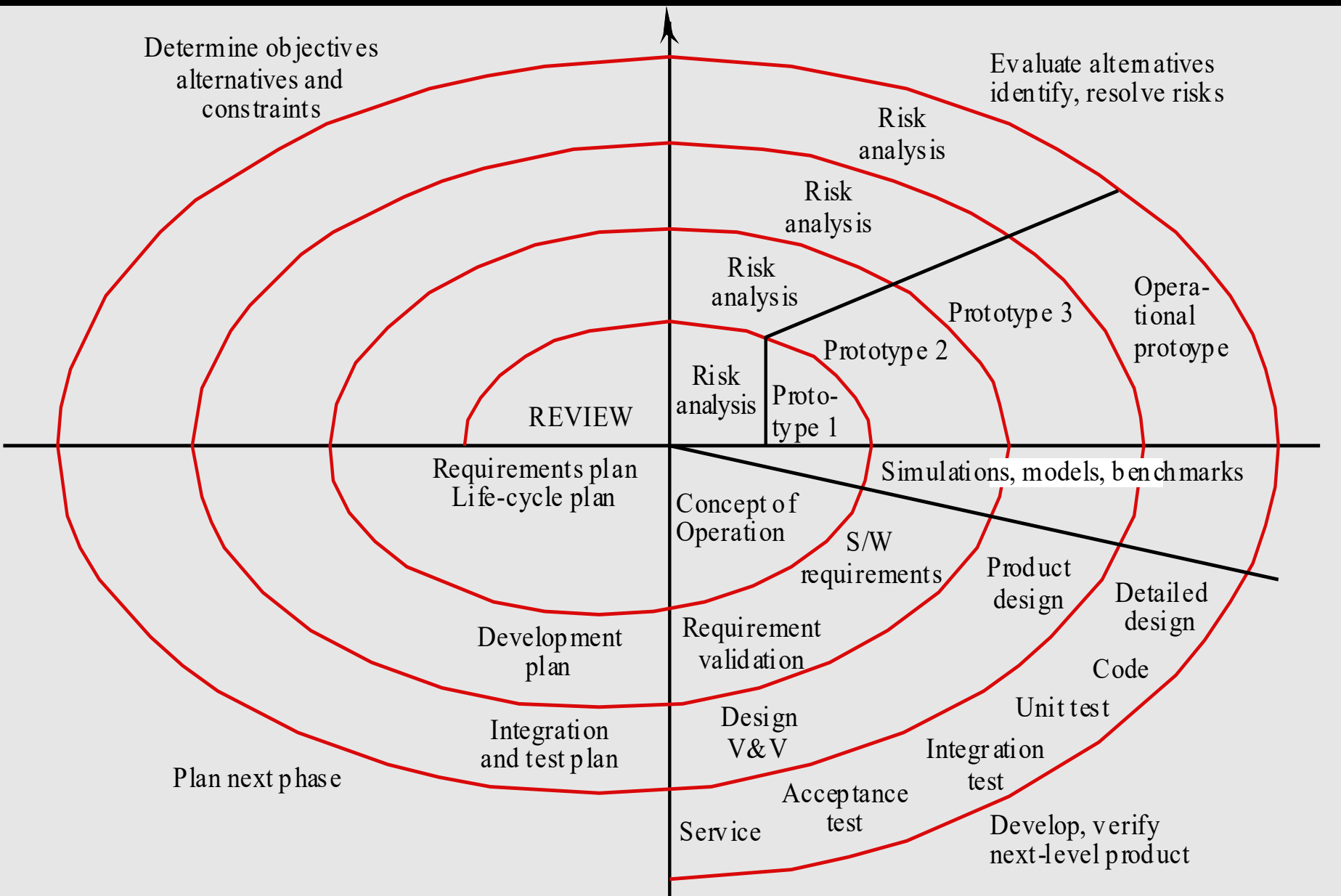
Disadvantages of Prototyping

- Leads People to Believe the Final System Will Follow
- Gives No Indication of Performance under Operational Conditions
- Leads the Project Team to Forgo Proper Testing and Documentation

Spiral SDLC Model



- Adds risk analysis, and 4gl RAD prototyping to the waterfall model
- Each cycle involves the same sequence of steps as the waterfall process model



Spiral Quadrant: Determine objectives, alternatives and constraints

- **Objectives:** functionality, performance, hardware/software interface, critical success factors, etc.
- **Alternatives:** build, reuse, buy, sub-contract, etc.
- **Constraints:** cost, schedule, interface, etc.

Spiral Quadrant: Evaluate alternatives, identify and resolve risks

- **Study alternatives** relative to objectives and constraints
- **Identify risks** (lack of experience, new technology, tight schedules, poor process, etc.
- **Resolve risks** (evaluate if money could be lost by continuing system development

Spiral Quadrant: Develop next-level product

- Typical activities:
 - Create a design
 - Review design
 - Develop code
 - Inspect code
 - Test product

Spiral Quadrant: Plan next phase

- Typical activities

- Develop project plan
- Develop configuration management plan
- Develop a test plan
- Develop an installation plan

Spiral Model Strengths

- Provides early indication of insurmountable risks, without much cost
- Users see the system early because of rapid prototyping tools
- Critical high-risk functions are developed first
- The design does not have to be perfect
- Users can be closely tied to all lifecycle steps
- Early and frequent feedback from users
- Cumulative costs assessed frequently

Spiral Model Weaknesses

- Time spent for evaluating risks too large for small or low-risk projects
- Time spent planning, resetting objectives, doing risk analysis and prototyping may be excessive
- The model is complex
- Risk assessment expertise is required
- Spiral may continue indefinitely
- Developers must be reassigned during non-development phase activities
- May be hard to define objective, verifiable milestones that indicate readiness to proceed through the next iteration

When to use Spiral Model

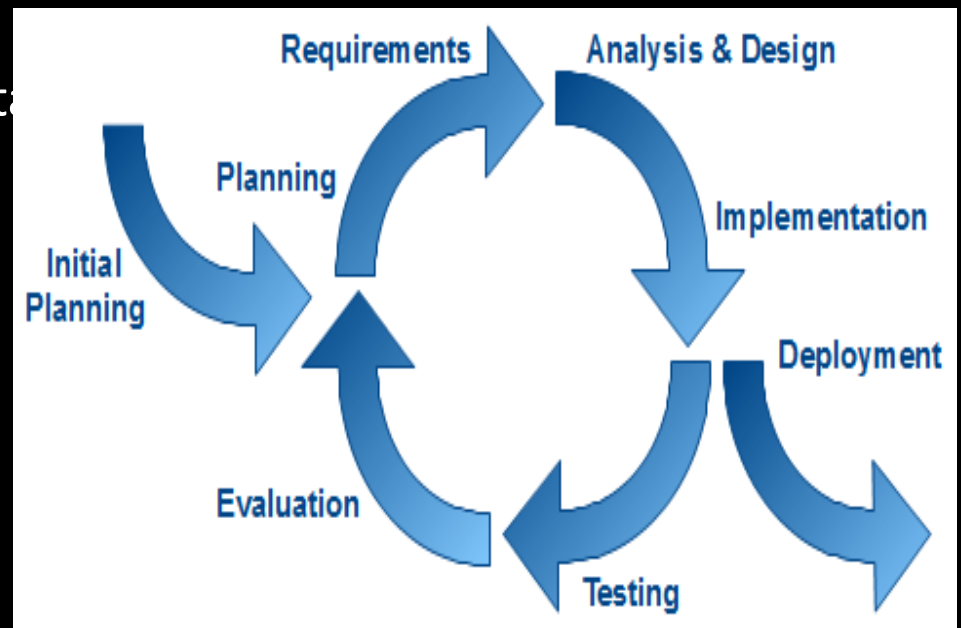
- When creation of a prototype is appropriate
- When costs and risk evaluation is important
- For medium to high-risk projects
- Long-term project commitment unwise because of potential changes to economic priorities
- Users are unsure of their needs
- Requirements are complex
- New product line
- Significant changes are expected (research and exploration)

Incremental Build Model

- The model is designed, implemented and tested incrementally (a little more is added each time).
- Finished when satisfies all the requirements.
- Combines the elements of the waterfall model with the iterative philosophy of prototyping.

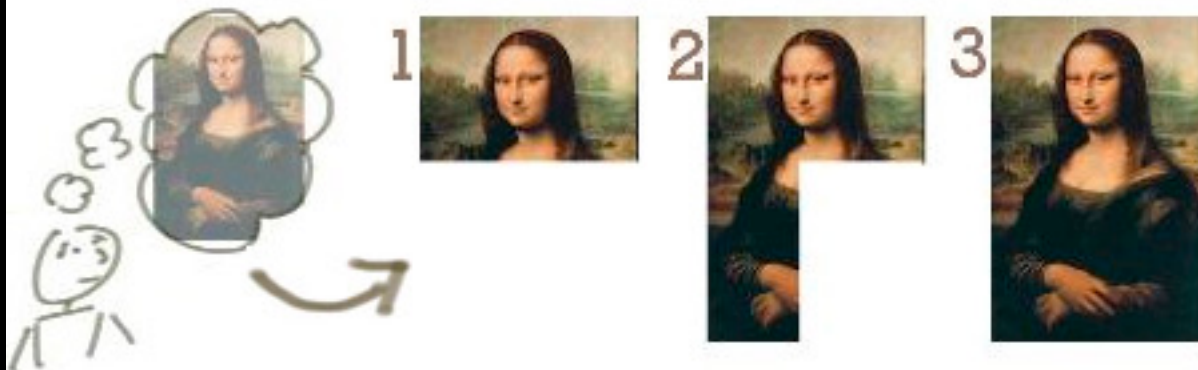
Iterative and Incremental Development

- Iterative and incremental development is any combination of both iterative design or iterative method and incremental build model for development.

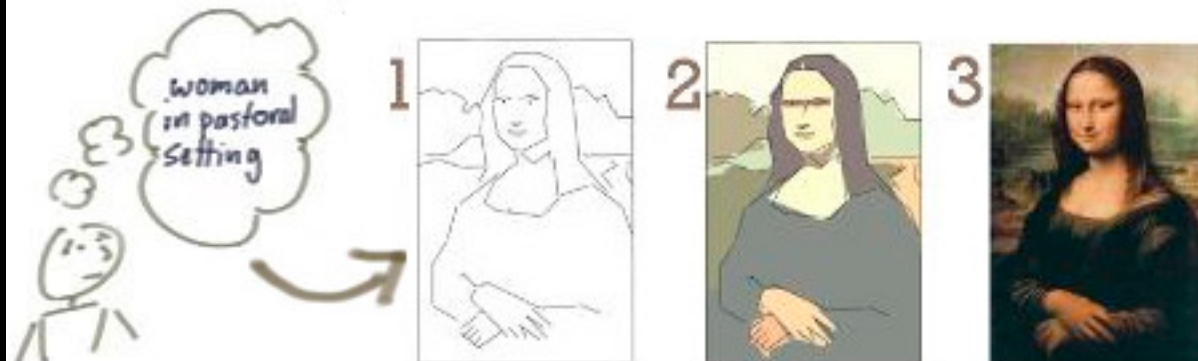


Incremental vs. Iterative

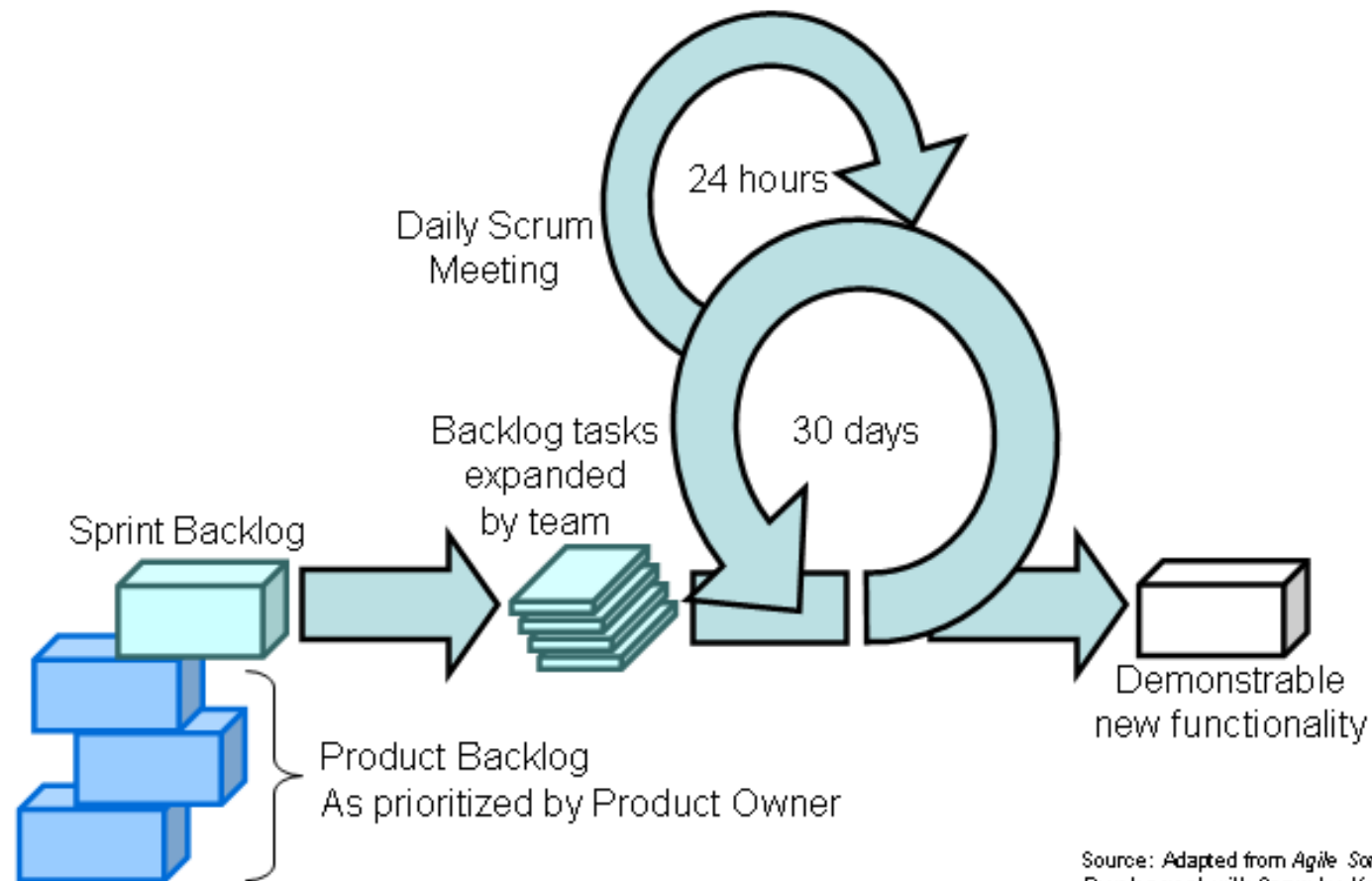
Incremental



Iterative



Agile and Scrum



Source: Adapted from *Agile Software Development with Scrum* by Ken Schwaber and Mike Beedle.

What is Agile Software Development ?

- Methodologies for designing software that have proven to be more effective in dealing with business realities such as changing requirements during development. It promotes industry best practices that emphasize teamwork, customer involvement and the frequent creation of small, working pieces of the total system.

Manifesto for Agile Software Development

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

Principles behind the Agile Manifesto

- Highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.

Principles Behind The Agile Manifesto - 2

- Build projects around motivated individuals. Give the environment, support and trust them to do it.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Continuous attention to technical excellence and good design enhances agility.

Principles Behind The Agile Manifesto - 3

- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Simplicity--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile Methods

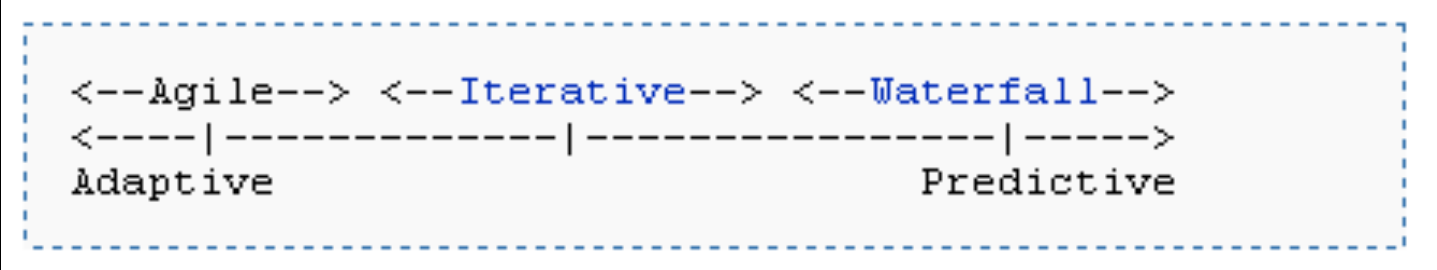
Some of well-known agile software development methods

- [Extreme Programming \(XP\)](#)
- [Scrum](#)
- [Agile Modeling](#)
- [Adaptive Software Development](#)
- [Crystal Clear and Other Crystal Methodologies](#)
- [Dynamic Systems Development Method](#)
- [Feature Driven Development](#)
- [Lean software development](#)
- [Agile Unified Process](#)

Comparison with other methods

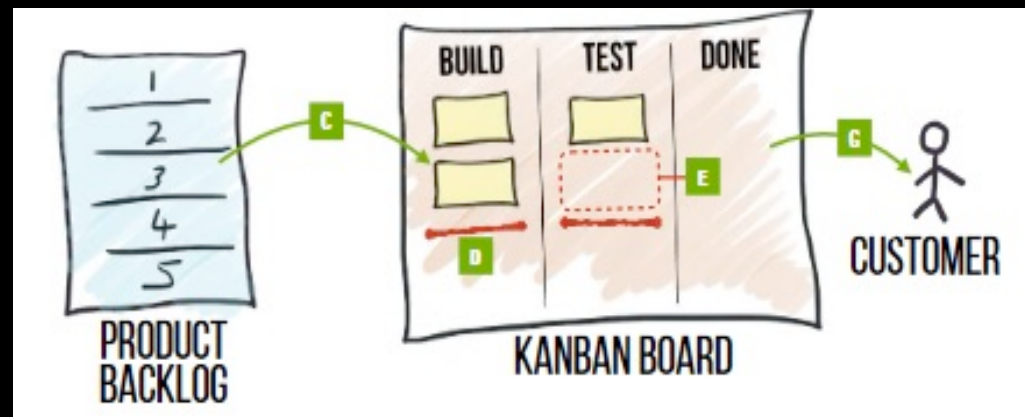
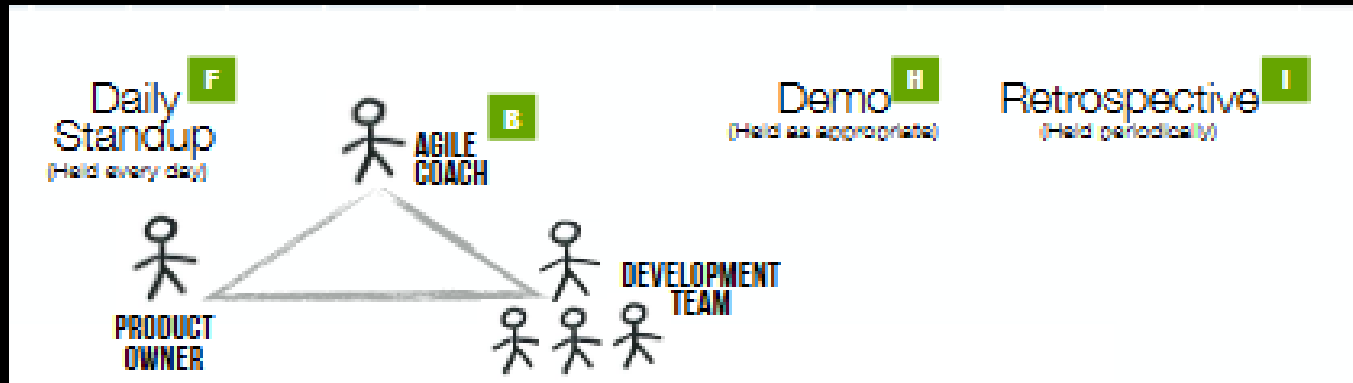
Agile methods are often characterized as being at the opposite end of the spectrum from "plan-driven" or "disciplined" methodologies. This distinction is misleading, as it implies that agile methods are "unplanned" or "undisciplined."

It is better to say that methods exist on a continuum from "adaptive" to "predictive." Agile methods exist on the "adaptive" side of this continuum.

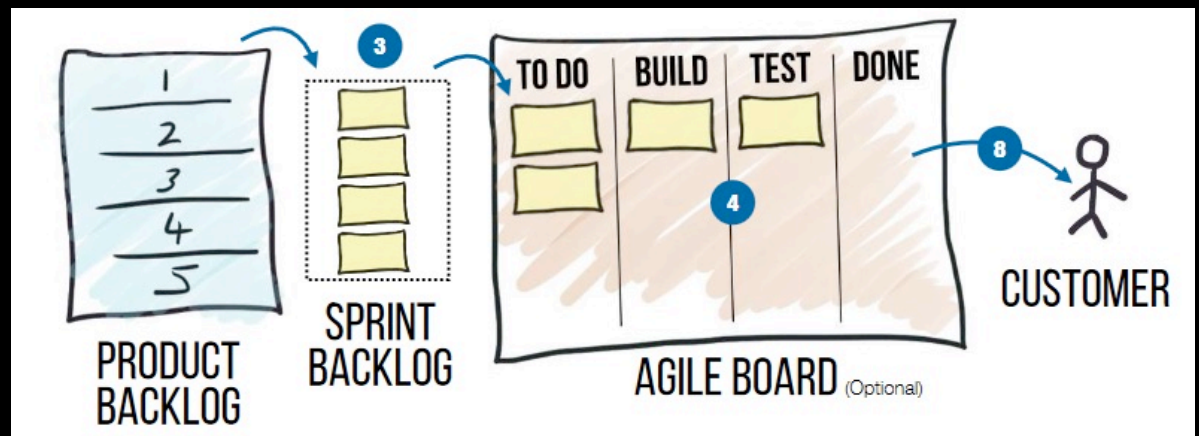
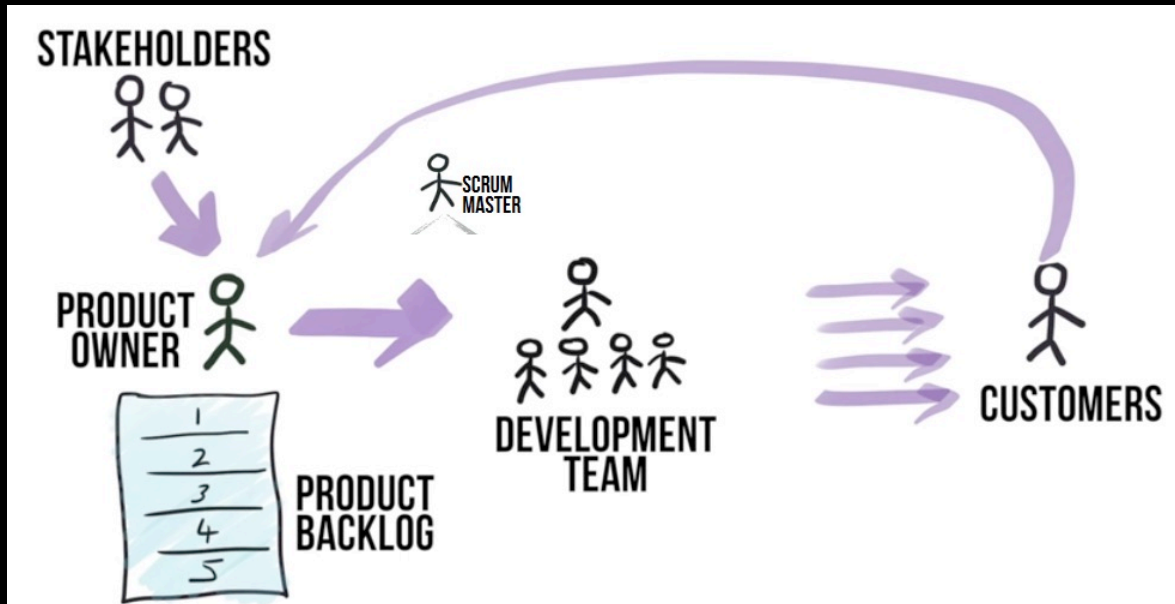


```
<--Agile--> <--Iterative--> <--Waterfall-->  
<----|-----|-----|----->  
Adaptive                                     Predictive
```

Scrum Overview



Kanban Overview



Thank You!