

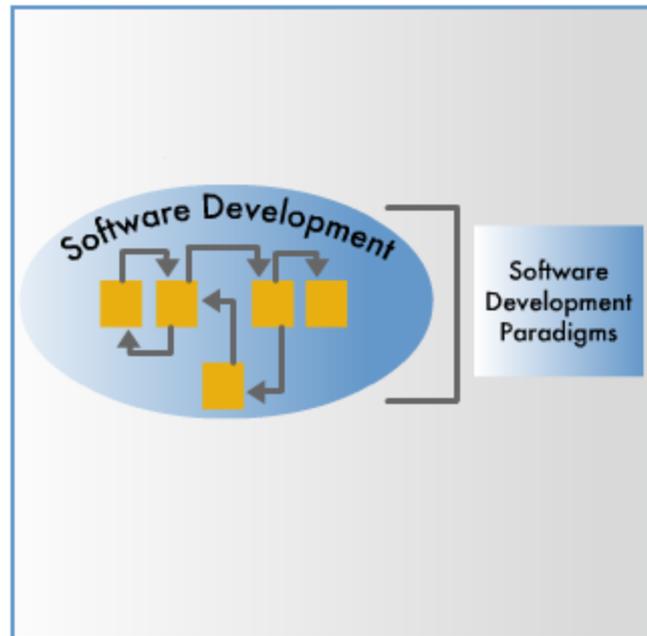
Module 6 - Acquisition Approaches and Development Paradigms

*TOC*

## Introduction: Software Development Paradigms

Software is the key part of today's defense systems. The process of how it is "put together" has a big impact on the success of a project. Because of this, knowledge of the various ways developers can structure the activities comprising software development is important.

Effective systems development is dependent on disciplined, consistent, acquisition/program strategies. This same disciplined approach is needed for software development.



TOC

## Learning Objectives

After completing this lesson, you will be able to:

- Define common software development paradigms.
- Given a list of software methodologies, match the software methodology best suited for the attributes of a given development project.



# Objectives

TOC

## Software Development Paradigms

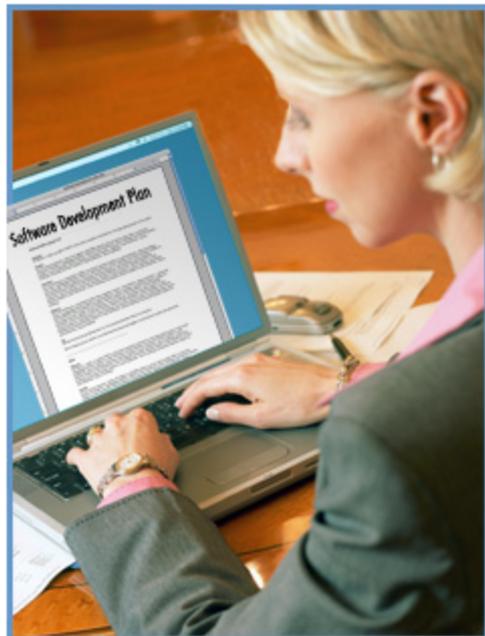
In the context of this course, models of structuring software development processes are called "paradigms." Paradigms, chosen by the supplier, apply to the structuring of software development activities. Paradigm selection is an essential part of a supplier's software planning process.

A process model answers two main questions:

1. What should be done next?
2. For how long should it continue?

The supplier frequently documents key information, including descriptions of software development approaches, in a plan. This plan is usually called the Software Development Plan (SDP).

Even though they are chosen by the supplier, since they are so critical to life-cycle management, it is important for acquirers to understand the strengths and weaknesses of commonly used software development paradigms.



## Characteristics of an Effective Software Development Approach

There is not one software methodology that can be used for all software. Several characteristics that are important to selecting an effective software development method include:

- Stability of technology base
- Stability of program office leadership and staff
- Availability of support contractors to provide continuity with the program office
- Anticipated duration of initial development project
- Anticipated life span of product
- Size and number of contractors to be managed
- Level of change being experienced in the operational environment
- Degree and frequency of expected requirement changes during the life of the program
- Cultural norms of the developer and program office
- Contracting constraints
- Alignment of software acquisition goals with the larger program, where software is a component of a larger hardware-based system
- Alignment of acquisition goals (for delivery, functionality, and funding) with operational goals

**Software Development Paradigms: Development Approaches**

There are four common development approaches used by most Department of Defense (DOD) software suppliers. The approaches are Waterfall, Incremental, Spiral and Agile.

*Select each approach to learn more.*

## Waterfall

## Incremental

## Spiral

## Agile

## Traditional Versus Agile Considerations

### TOC

Program Managers should weigh the advantages when choosing between Agile and Traditional software practices.

In the government context, Agile represents a good development approach when customizing an existing system or building a small-scale or self-constrained application.

A larger program could initially use a traditional approach to build the initial increment that meets the baseline architecture requirements. Once the program has established the baseline and framed the overall conceptual design, program managers could consider shifting to an Agile approach for subsequent increments that build additional functionality into the operational baseline.

Several large acquisition programs, such as the Global Combat Support System-Joint (GCSS-J), have adopted Agile methods to build a future capability increments.

**Click image to view larger.**

Consider Agile Practices	Assessment Areas	Consider Traditional Practices
Requirements cannot be well defined upfront due to a dynamic operational environment.	Requirements Stability	Requirements have been relatively well defined by the operational sponsor.
Requirements can be decomposed into small tasks to support iterative development.	Requirements Divisibility	Requirements are tightly integrated and are difficult to decompose.
Users welcome iterative development and require frequent capability upgrades	User Timelines	Operational environment does not allow iterative development or lacks
User representatives and end users are able to frequently engage throughout development.	User Involvement	Users cannot support frequent interaction with the development team or the target end user cannot
Program scope is mostly limited to the application layer while using existing infrastructure.	Program Scope	Program spans core capabilities and underlying platform or infrastructure.
The government is responsible for primary systems integration.	Systems Integration	The government does not want to own systems integration responsibilities.
Capabilities are operational at a basic level, with some defects that can be addressed in future releases.	System Criticality	Program supports a critical mission in which defects may result in loss of life or high security risks.
Industry has relevant domain experience and Agile development expertise.	Developer Expertise	Agile development expertise is unavailable or lacks domain experience.
Program office has Agile training, experience, and/or coaches.	Government Expertise	Program office has no Agile experience or funding for Agile training or coaches.
Program contract strategy supports short Agile development timelines.	Contracting Timelines	Contract strategy cannot support short Agile development timelines.
Program Executive Office (PEO) or subordinate has authority for most program decisions.	Level of Oversight	Office of the Secretary of Defense (OSD) or Service Acquisition Executive (SAE) is the Milestone Decision Authority (MDA) and requires most decisions to be made at that level.
Development can be effectively managed by a small cross-functional government team.	Team Size	Many government stakeholders will be involved in the software development and decision-making process.
Government and developers can collaborate frequently and effectively.	Collaboration	Stakeholders physically located across multiple locations and have limited bandwidth to support frequent collaboration.
One or a few contractor(s) or teams can perform development.	Complexity	Many contractors are required to develop program elements.
Program can leverage test infrastructure and automated tests, and testers are active throughout development.	Test Environment	Extensive development and operational testing is conducted serially following development. Limited resources and tools available to conduct parallel development testing.
Leadership actively supports Agile development practices and provides "top cover" to use non-traditional processes and methods.	Leadership Support	Leadership prefers a traditional development approach or is unfamiliar with Agile practices.

## Observations About Software Development Methodologies

Before making a choice of which software development methodology to choose, here are some observations to consider:

- Requirements change so any method that makes it exceedingly difficult to change requirements will just increase the cost of the program and reduce the ability of the contractor to deliver functionality that has value to the operational user.
- Incentives matter, and they are difficult to get right. Each life cycle method has an explicit set of [incentives](#). If the inherent incentives of the development method and the explicit incentives of the contract are mismatched, success of any kind is difficult to achieve and likely to take longer than planned.
- The Office of the Secretary of Defense encourages "thoughtful tailoring," but acquisition professionals are encouraged to take the "safe path." Thoughtful tailoring is consistent with the need to match project characteristics with different methods' attributes. However, in practice it's easier for acquisition staff to use "what's worked before."

## Knowledge Review

What is an incremental paradigm?

- Characterized by the development of software in small pieces.
- A single-pass model that can be fast and inexpensive under the right circumstances.
- Is most appropriate for Unprecedented Systems because it uses risk analysis (to identify candidates for prototyping) and controlled prototyping (to mitigate risks).
- Is best used on Precendented Systems with stable, up-front requirements.
- Produces software requirements specifications and design documentation early.

**Check Answer**

TOC

## Knowledge Review

The IPT team is requesting that you pair the software methodology to each attribute.

**Software  
Methodology**

SAFE

Incremental

Waterfall

**Check Answer****Attribute**

Allows the user to gain value from a portion of the system prior to the entire system being released.

Software is developed in one long release cycle.

Combination of LD and Agile principles. Currently the most used scaling framework in DoD.

## Summary

TOC

This lesson covered software development paradigms and software methodologies. The four common development approaches that are chosen (with variations) and software methodologies that are chosen by most DoD Software suppliers include Waterfall, Incremental, Spiral and Agile.

**Select the tabs to review the software development paradigms and methodologies.**

**Waterfall**

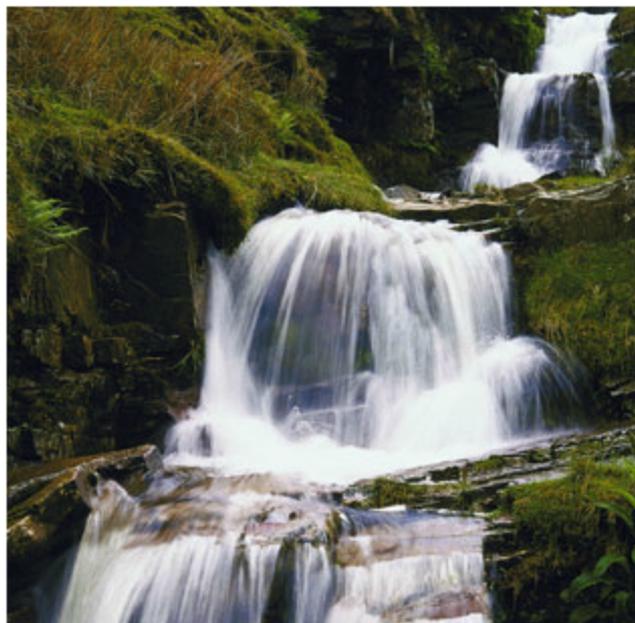
Incremental

Spiral

Agile

The Waterfall paradigm:

- Formalizes a framework for software development phases.
- Places emphasis on up-front, stable software requirements and design activities.
- Produces software requirements and design documentation during early phases.
- Is a single-pass model and thus can be fast and inexpensive when used under the right conditions.



## Lesson Completion

[TOC](#)

You have completed the content for this lesson.

To continue, select another lesson from the Table of Contents on the left.

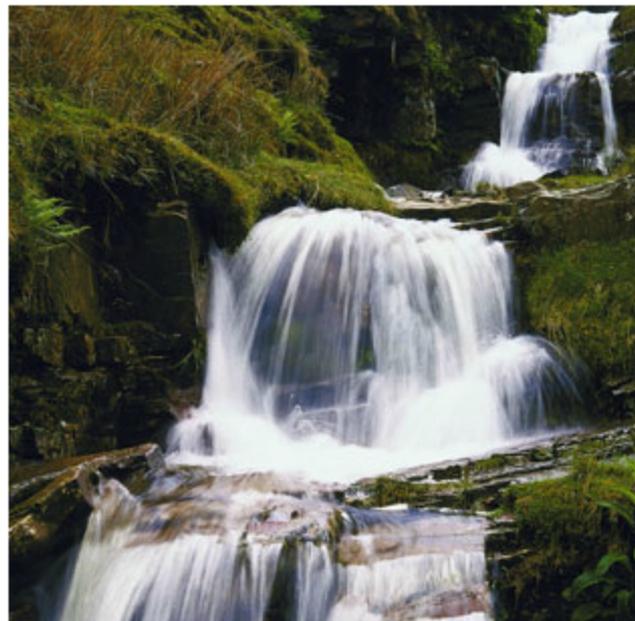
If you have closed or hidden the Table of Contents, click the Show TOC button at the top in the Atlas navigation bar.

TOC

**Definition**

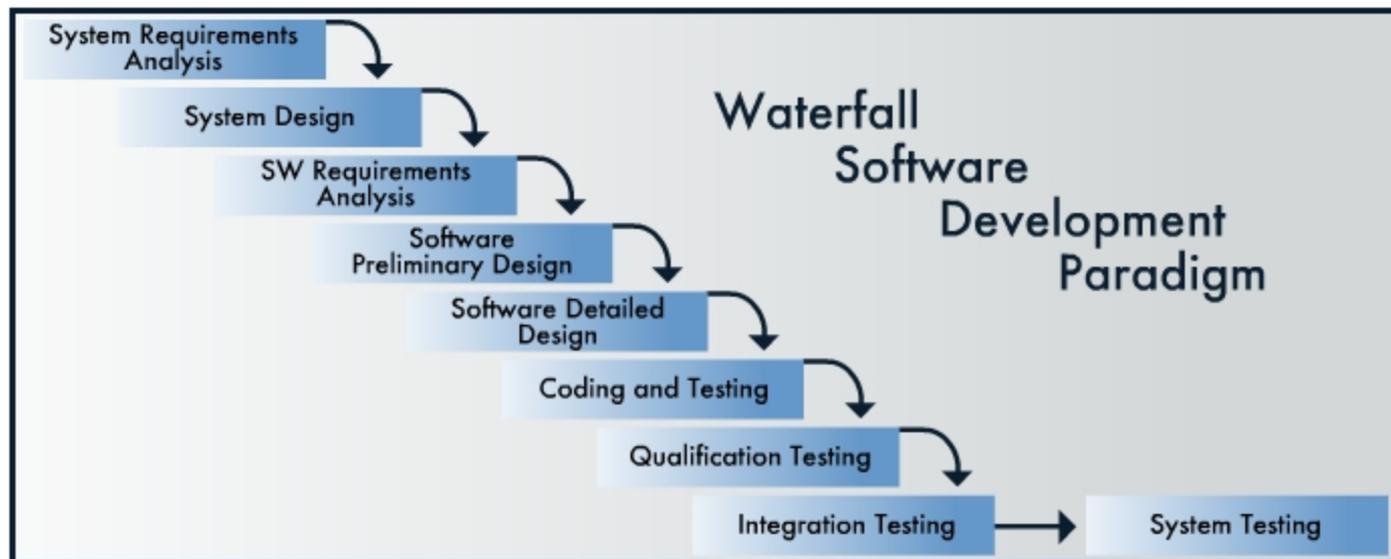
The Waterfall Paradigm:

- Formalizes a framework for software development phases.
- Places emphasis on up-front, stable software requirements and design activities.
- Produces software requirements and design documentation during early phases.
- Is a single-pass model and thus can be fast and inexpensive when used under the right conditions.



## Successful Staging

In the traditional Waterfall paradigm, each stage is a prerequisite for beginning subsequent activities. Successful completion of a stage is required before starting the next one.



TOC

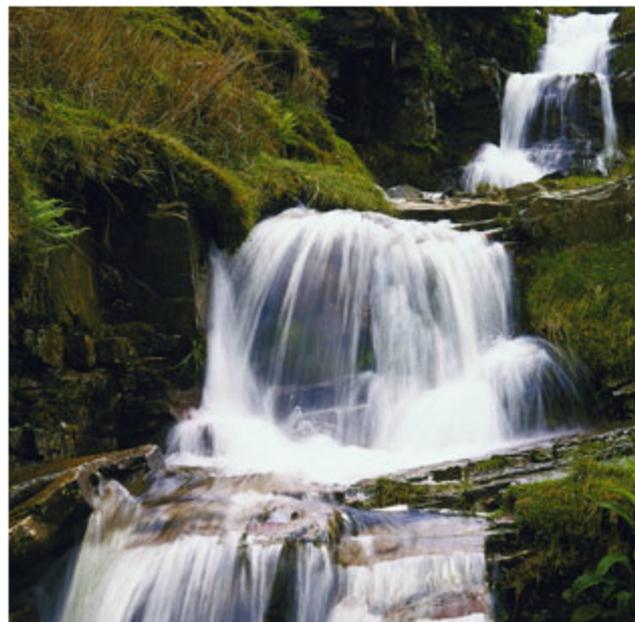
## Attributes

Initially the Waterfall Software Development Methodology was used where software was developed in one long release cycle. This methodology elaborates fundamental steps required to develop software.

However, it has one major flaw that it assumes that once the requirements process is complete, the requirements will remain unchanged throughout the development cycle. This rarely holds true in practice as change is inevitable in all large software projects. Therefore, long waterfall-like development cycles do not allow for inevitable requirement changes.

Waterfall approaches can be applied to small or large projects. When applied to larger projects, considering how the project can be broken up into smaller increments is typical.

When multiple waterfall cycles (that result in fieldable software) are sequenced together, the approach is usually termed "incremental".

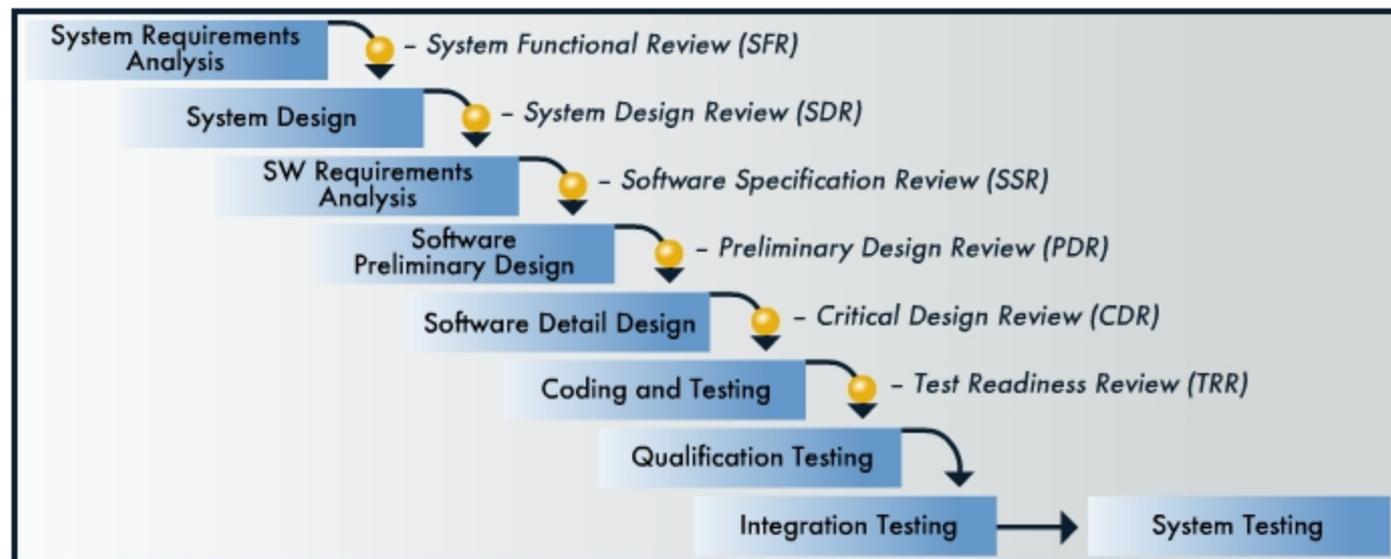


## Life-Cycle Reviews

Life-cycle reviews should be used to assess progress and determine whether or not to proceed to the next phase of software development.

Depending on the needs of the Government and the commercial life cycle standard being used, a variety of such reviews can be used.

The graphic illustrates where reviews are normally conducted as part of a Waterfall paradigm.



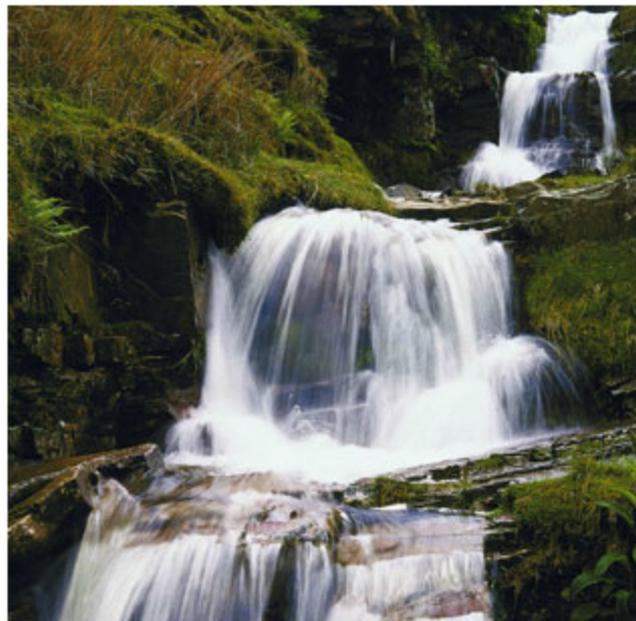
TOC

## Precedented Systems

For some categories of software systems, the Waterfall paradigm can be a good choice. These types of systems are called "Precedented Systems."

A Precedented System is characterized by:

- Stable system and software requirements, similar to previously developed systems.
- System architecture and a software design that are mature and capable of meeting requirements.
- Systems engineering and software development teams that communicate well and have previous experience with similar systems.



**Example**

An example of when the Waterfall software development paradigm could be appropriate is the [LRATS Training and Maintenance System \(TAMS\)](#).

One of the TAMS components is the Tactical Maintenance and Supply Subsystem (TMSS). The TMSS is a fielded, highly precedented system in the support phase.

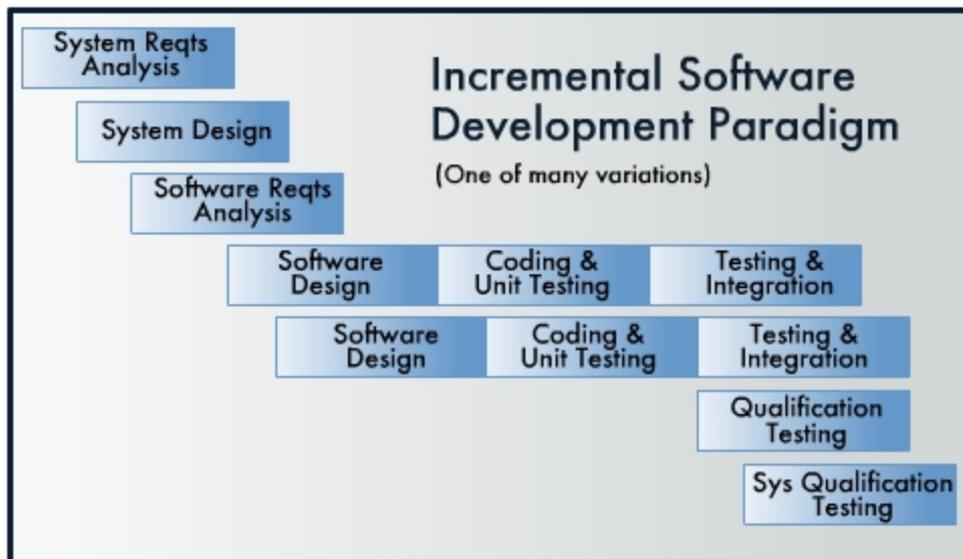
Waterfall software development paradigms are typically used for these types of systems which are highly -precedented or in the software support stage.



**Definition**

The Incremental software development paradigm involves developing software in 'pieces' or increments. A contractor-government Integrated Product Team (IPT) may be used to define the number, size, and phasing of the increments required to satisfy the user's software requirements.

One risk of incremental software development is the tendency for "hard" software requirements to gradually migrate to later and later increments. This can lead to cost and schedule overruns if problems develop in implementing deferred requirements. Software measures exist that allow the Program Management Office (PMO) visibility into such migration.



## Attributes

Breaking software cycles into a series of increments allows one to better adapt to changing requirements. Incremental delivery allows the user to gain value from a portion of the system prior to the entire system being released.

**Select the tabs to learn about the advantages and disadvantages of the Incremental Paradigm.**

**Advantages**   **Disadvantages**

Advantages of the Incremental Paradigm include:

- Requirements are relatively stable and may be better understood with each increment if the increment is appropriately scoped.
- Allows some requirements modification and may allow the addition of new requirements as system attributes are learned about with the implementation of each increment.
- More responsive to user needs than the Waterfall model.
- A usable product is available with the first release, and each cycle results in greater functionality.
- The project can be stopped any time after the first cycle and leave a working product.
- Risk is spread out over multiple cycles.
- This method can usually be performed with fewer people than the waterfall model.
- Return on investment is visible earlier in the project.
- Project management may be easier for smaller, incremental projects.
- Testing may be easier on smaller portions of the system.
- Incremental life cycle can be a frame for Waterfall, Spiral or Agile methods.

TOC

**Example**

An incremental software development paradigm would be appropriate for developing the software operating the LRATS E-SENTINEL weapons system.

The E-SENTINEL Guidance and Control Subsystem ([EGCS](#)) consists of several Software Items (SIs) that:

- Manage launch sequence activities
- Perform in-flight control
- Do dynamic retargeting
- Initiate warhead activities

These EGCS SIs would be candidates for an Incremental approach.



## The Incremental Model

Essentially a series of waterfall cycles



## Definition and Attributes

The Spiral Model emphasizes risk-driven prototyping. Risk analysis is used to identify high-risk candidates for prototyping. Feedback from the results of controlled prototyping is evaluated and knowledge obtained is used to mitigate risks.

This risk-driven process continues until major software development risks are understood.

It is used to guide multi-stakeholder concurrent engineering of software intensive systems. It has two main distinguishing features which include:

- Cyclic approach for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk
- Set of anchor point milestones for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions



TOC

## Unprecedented Systems

An Unprecedented System is a system for which design examples do not exist and requirements and risks are not completely understood.

The system and software architecture alternatives are unconstrained by previous system descriptions or implementations and development is likely to be risky.

Spiral Models use risk-driven prototyping to fully define requirements. Spiral Models for software development are especially appropriate for unprecedented systems, as they frequently have undefined requirements.

The term "Spiral Development" is sometimes used within the DoD to refer to the process of implementing a complete system (hardware, software, facilities, etc.) via Evolutionary Acquisition. This is a much broader approach than the software-specific development paradigm discussed here.

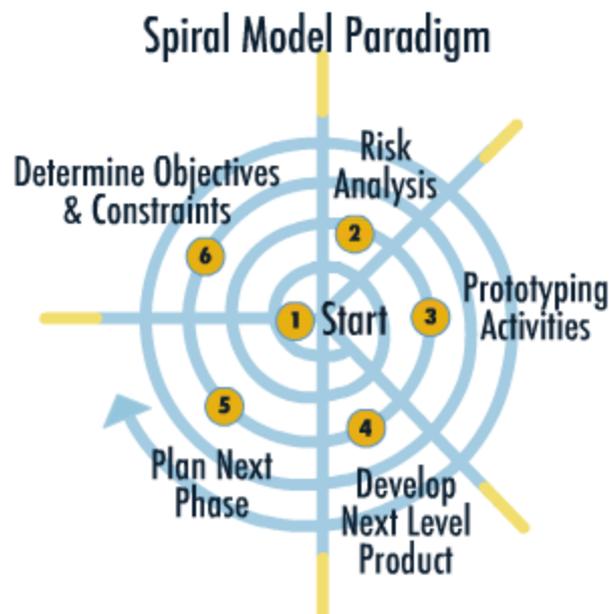


TOC

## Characteristics

The Spiral Model is a risk-driven paradigm. Key characteristics include:

- Use of prototyping to identify risks.
- Feedback from prototypes to mitigate risks.
- Elaboration of draft documentation during the life-cycle.
- Deferment of formal baselines.

[D](#)

TOC

## Example

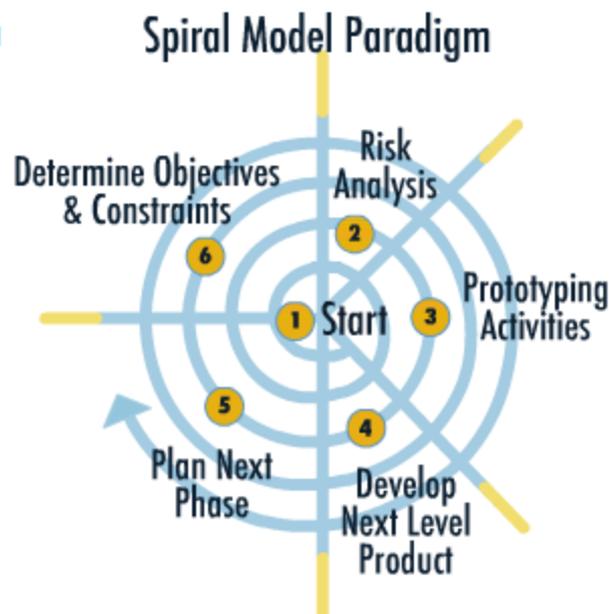
The Spiral Model software development may be appropriate for the LRATS Mission Operational Planning and Targeting Intelligence System ([MOPTIS](#)).

MOPTIS is:

- An unprecedented C4I system using a variety of state-of-the-art technologies
- Critically dependent on the human-computer interface

Software for the MOPTIS would be an ideal candidate for a Spiral Model software development approach. A formal risk analysis would identify high-risk modules.

These would be prototyped and refined, and risk-driven prototyping would continue until risks have been mitigated to an acceptable level.

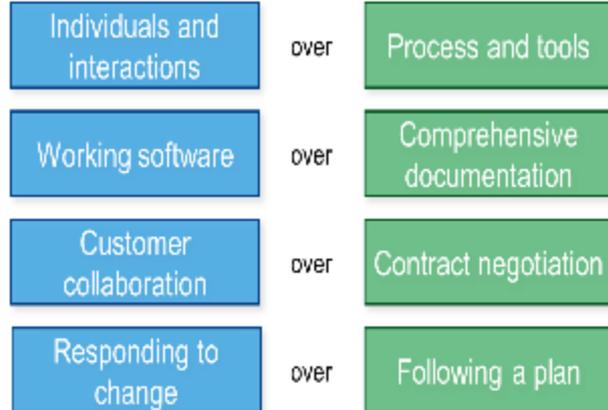


## Description and Principles

Agile Methods are a reaction to traditional ways of developing software and acknowledge the need for an alternative to documentation driven, heavyweight software development processes. What emerged was the [Agile Software Development Manifesto](#).

Developers declared they were "uncovering better ways of developing software by doing it and helping others do it." The developers believed value should be placed on:

- Individuals and interaction over process and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan



***[Click here to read the underlying principles of the Agile Manifesto.](#)***

**Attributes and Methods**

Because Agile is based on principles rather than practices in a defined sequence, there is no accepted criteria or "checklist" to determine if an organization is using agile development methods. The Agile Manifesto only lays the ground work for a collection of methods to include Scrum, Extreme Programming (XP), Crystal, Lean Development (LD), Scaled Agile Framework for Enterprises (SAFE), Kanban and Dynamic Systems Development Methodology (DSDM).

Select each method to learn more.

**Scrum**

**Extreme Programming**

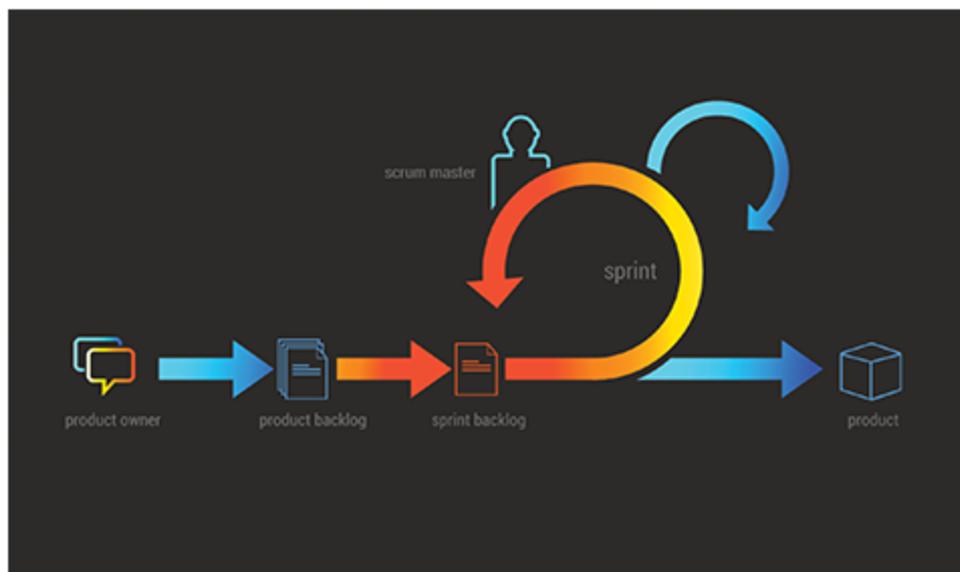
**Crystal**

**Lean Development**

**Scaled Agile Framework for Enterprises**

**Kanban**

**Dynamic Systems Development Methodology**



TOC

## Lessons Learned

Agile development has been widely seen as being more suitable for certain types of environment, including small teams of experts. Some things that may negatively impact the success of an agile project are:

- Large-scale development efforts (>20 developers).
- Distributed development efforts (non-colocated teams).
- Forcing an agile process on a development team.
- Mission-critical systems where failure is not an option at any cost (e.g. software for avionics).

Teams implementing Agile often face difficulties transitioning from more traditional software development paradigms. Common problems faced by teams implementing agile processes include:

- Lack of sponsor support
- Insufficient Training

