SOFTWARE DEVELOPMENT LIFE CYCLE

Software Development Life Cycle Model

A framework that describes the activities performed at each stage of a software development project.

Software Development Life Cycle Model

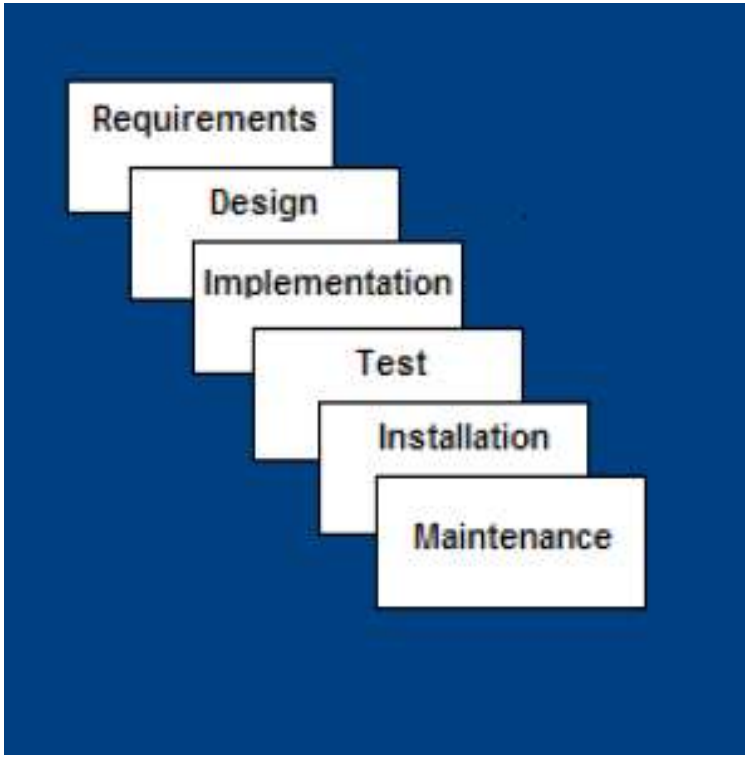
- WATER FALL MODEL
- SPIRAL MODEL
- INCREMENTAL MODEL
- V-SHAPED MODEL
- RAD MODEL
- RUP MODEL
- AGILE WITH SCRUB/scrum/server
- AGILE WITH XP

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, and track)
- Works well when quality is more important than cost or schedule

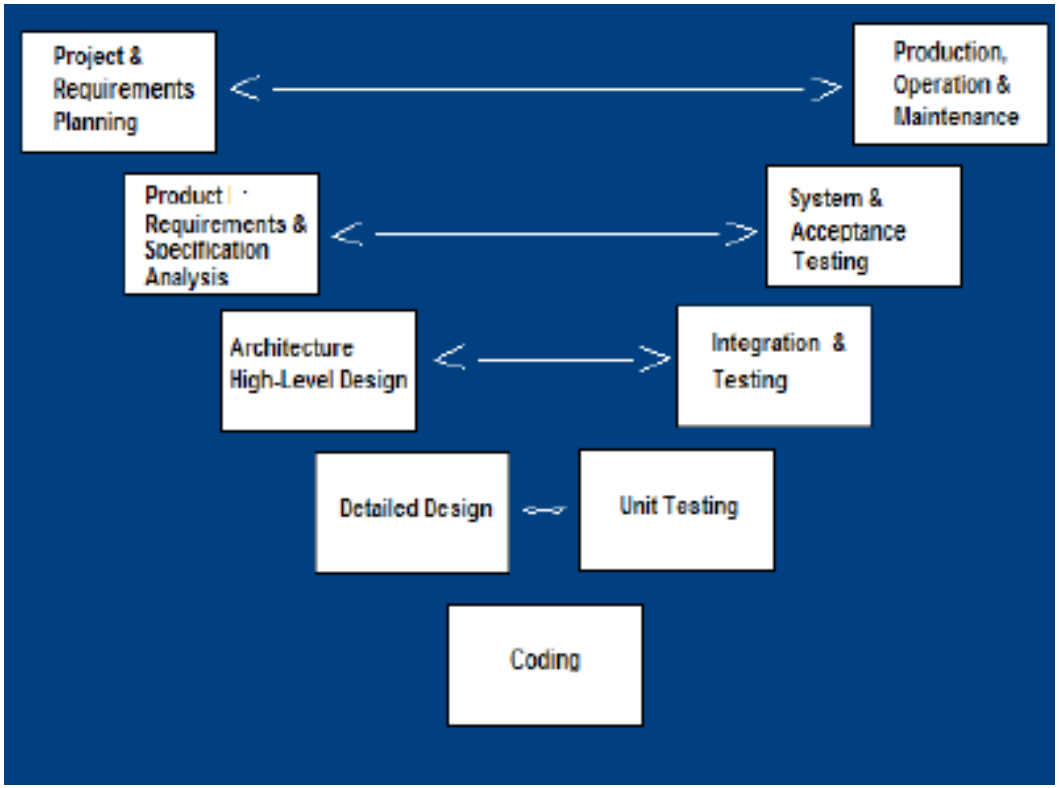
Waterfall Deficiencies

- 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

V-Shaped Steps

- 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

Structured Evolutionary Prototyping Model

•Developers build a prototype during the requirements phase

•Prototype is evaluated by end users

•Users give corrective feedback

•Developers further refine the prototype

•When the user is satisfied, the prototype code is brought up to the standards needed for a final product.

Structured Evolutionary Prototyping Steps

- A preliminary project plan is developed
- A partial high-level paper model is created
- The model is source for a partial requirements specification
- A prototype is built with basic and critical attributes
- The designer builds
  - –the database
  - –user interface
  - –algorithmic functions
- The designer demonstrates the prototype, the user evaluates for problems and suggests improvements.
- This loop continues until the user is satisfied

Structured Evolutionary Prototyping Strengths

•Customers can “see“the system requirements as they are being gathered

•Developers learn from customers

•A more accurate end product

•Unexpected requirements accommodated

•Allows for flexible design and development

•Steady, visible signs of progress produced

•Interaction with the prototype stimulates awareness of additional needed functionality

Structured Evolutionary Prototyping Weaknesses

•Tendency to abandon structured program development for “code-and-fix “development

•Bad reputation for “quick-and-dirty “methods

•Overall maintainability may be overlooked

•The customer may want the prototype delivered.

•Process may continue forever (scope creep)

When to use Structured Evolutionary Prototyping

•Requirements are unstable or have to be clarified

•As the requirements clarification stage of a waterfall model

•Develop user interfaces

•Short-lived demonstrations

•New, original development

•With the analysis and design portions of object-oriented development.

Rapid Application Model (RAD)

•Requirements planning phase (a workshop utilizing structured discussion of business problems)

•User description phase –automated tools capture information from users

•Construction phase –productivity tools, such as code generators, screen generators, etc. inside a time-box. ("Do until done")

•Cutover phase --installation of the system, user acceptance testing and user training

Definition:

•Rapid application development

•RAD is incremental software development process model that allows usable systems to be built in as little as 60-90 days.

•The RAD model used for information systems development.

RAD Strength

•Reduced cycle time and improved productivity with fewer people means lower costs

•Time-box approach mitigates cost and schedule risk

•Customer involved throughout the complete cycle minimizes risk of not achieving customer satisfaction and business needs

•Focus moves from documentation to code.

•Uses modeling concepts to capture information about business, data, and processes.

RAD Weakness

•Accelerated development process must give quick responses to the user

•Risk of never achieving closure

•Hard to use with legacy systems

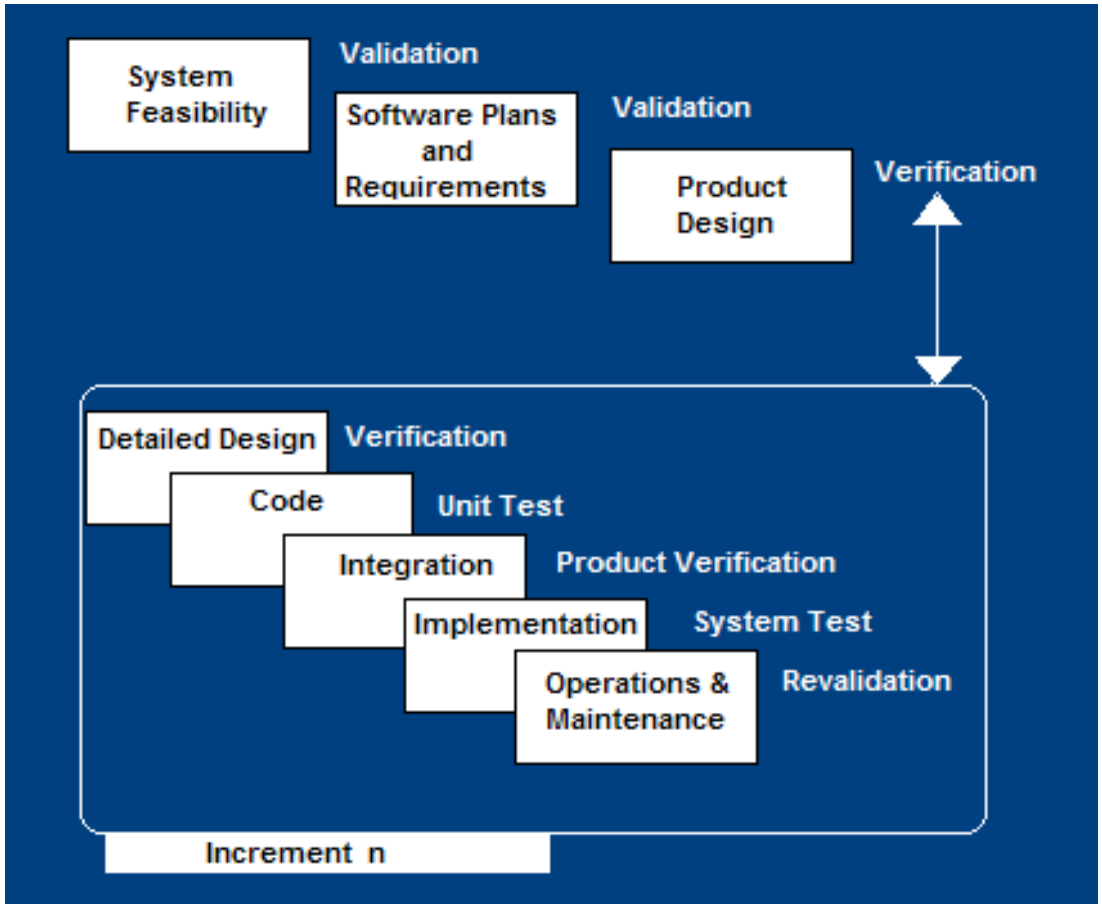
•Requires a system that can be modularized

•Developers and customers must be committed to rapid-fire activities in an abbreviated time frame.

When to use RAD

- Reasonably well-known requirements
- User involved throughout the life cycle
- Project can be time-boxed
- Functionality delivered in increments
- High performance not required
- Low technical risks
- System can be modularized

Incremental SDLC Model



- Construct a partial implementation of a total system
- Then slowly add increased functionality
- The incremental model prioritizes requirements of the system and then implements them in groups.
- Each subsequent release of the system adds function to the previous release, until all designed functionality has been implemented.

Incremental Model Strengths

- Develop high-risk or major functions first
- Each release delivers an operational product
- Customer can respond to each build
- Uses "divide and conquer" breakdown of tasks
- Lowers initial delivery cost
- Initial product delivery is faster
- Customers get important functionality early
- Risk of changing requirements is reduced

Incremental Model Weaknesses

•Requires good planning and design

•Requires early definition of a complete and fully functional system to allow for the definition of increments

•Well-defined module interfaces are required (some will be developed long before others)

•Total cost of the complete system is not lower

When to use the Incremental Model

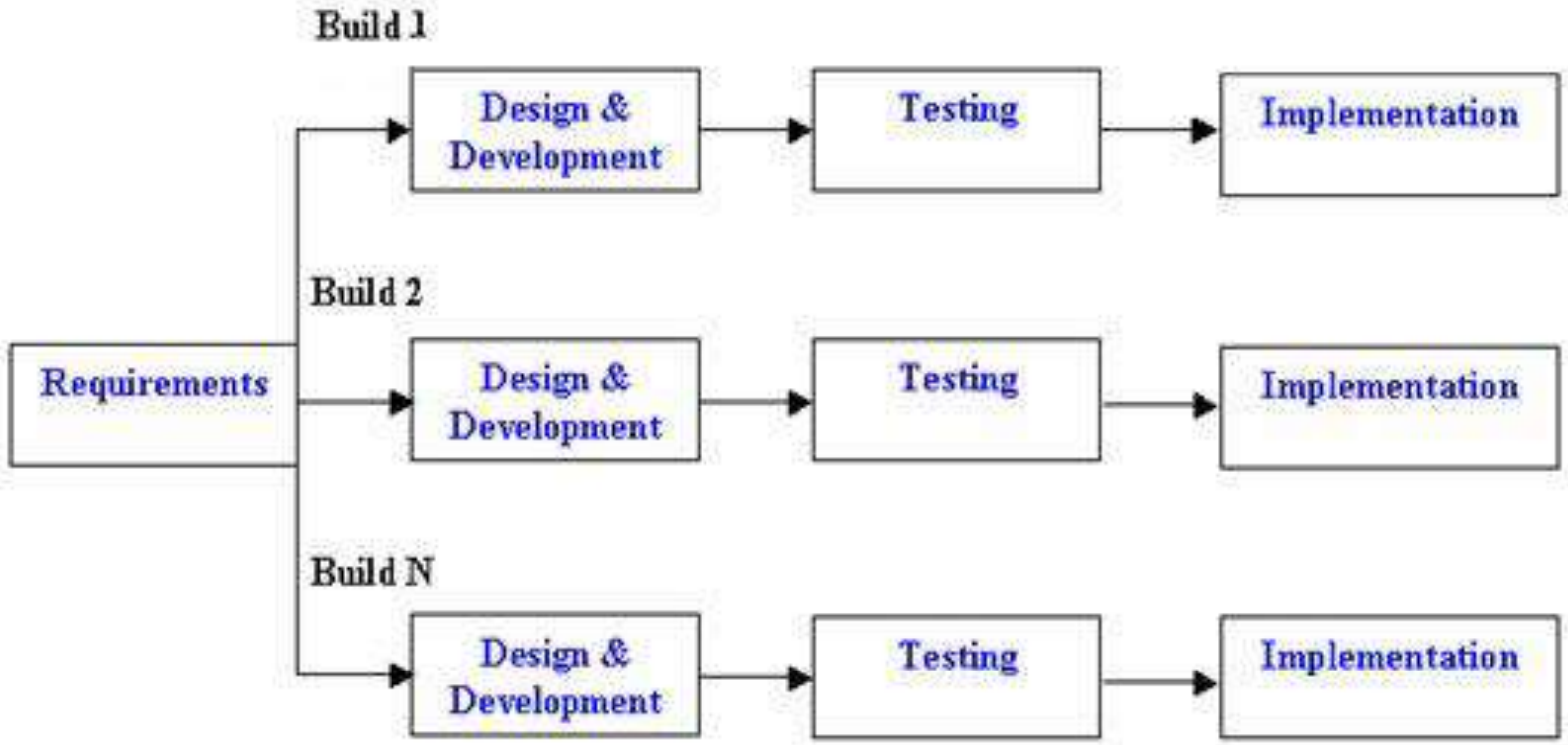
•Risk, funding, schedule, program complexity, or need for early realization of benefits.

•Most of the requirements are known up-front but are expected to evolve over time

•A need to get basic functionality to the market early

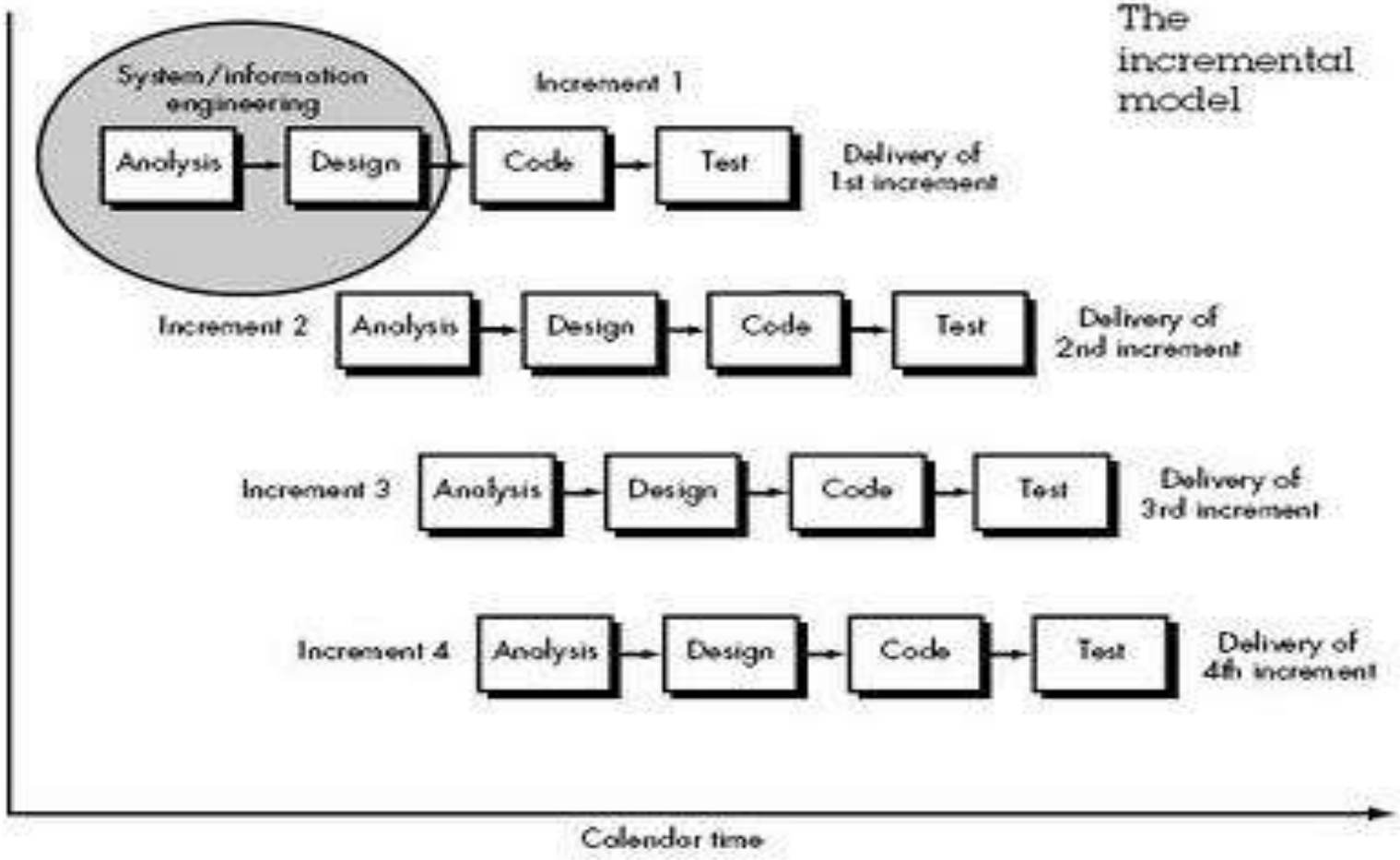
•On projects which have lengthy development schedules

•On a project with new technology

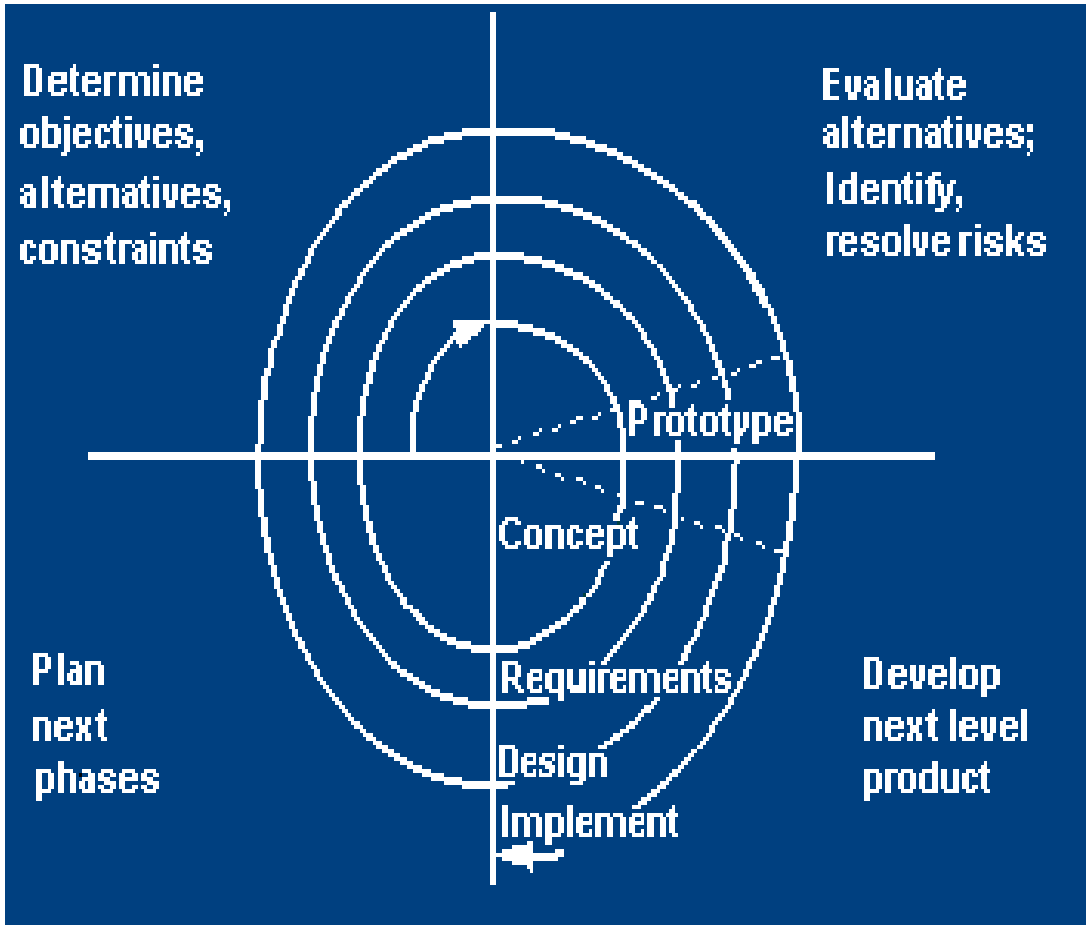


Incremental Life Cycle Model

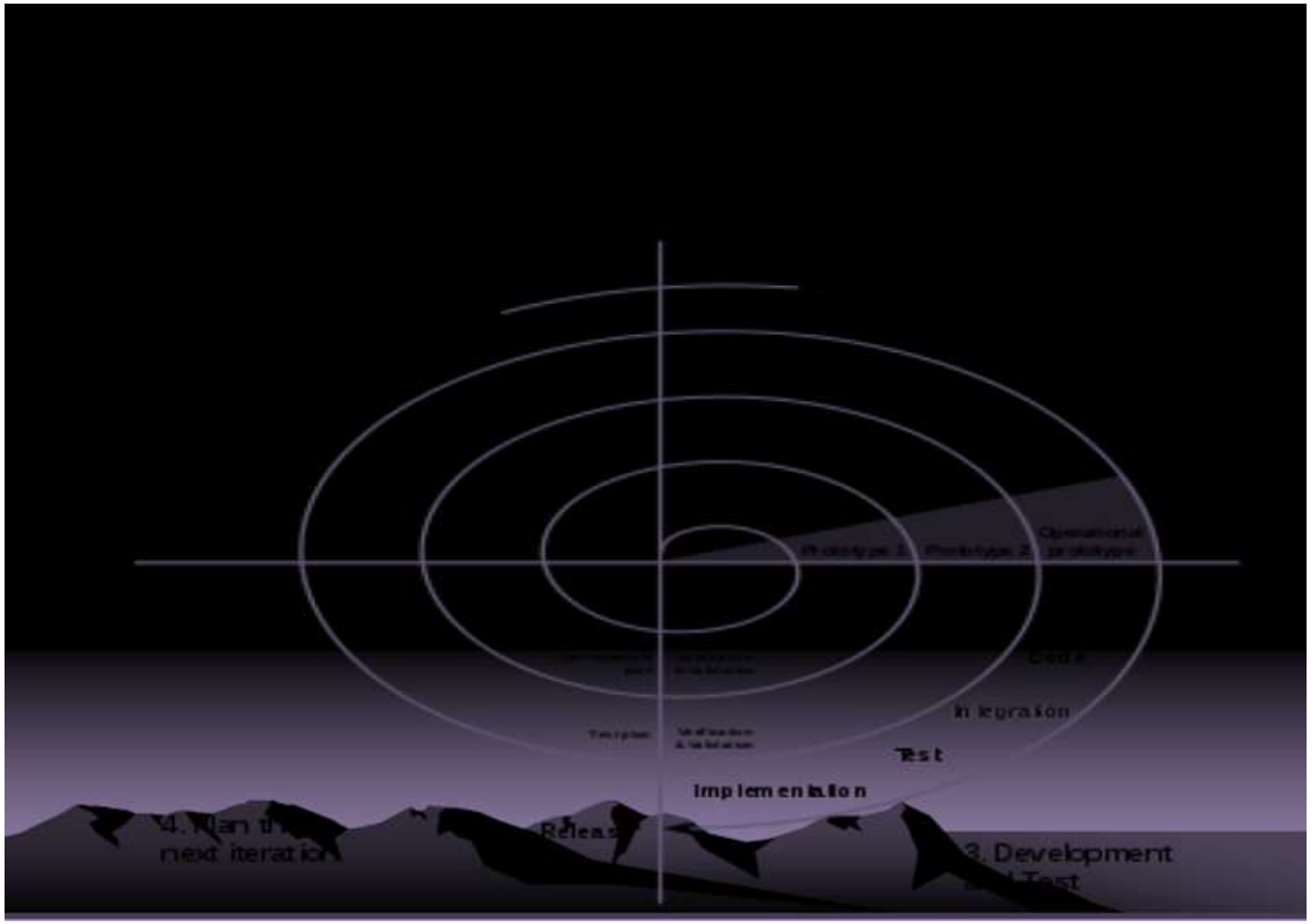
The incremental model



Spiral SDLC Model



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


Integration
Test
Implementation
3. Development

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)

Agile SDLC's

- Speed up or bypass one or more life cycle phases
- Usually less formal and reduced scope
- Used for time-critical applications
- Used in organizations that employ disciplined methods

Some Agile Methods

- Adaptive Software Development (ASD)
- Feature Driven Development (FDD)
- Crystal Clear
- Dynamic Software Development Method (DSDM)
- Rapid Application Development (RAD)
- Scrum
- Extreme Programming (XP)
- Rational Unify Process (RUP)

Extreme Programming –XP

For small-to-medium-sized teams developing software with vague or rapidly changing requirements

Coding is the key activity throughout a software project

•Communication among teammates is done with code

•Life cycle and behavior of complex objects defined in test cases –again in code

XP Practices (1-6)

1. Planning game –determine scope of the next release by combining business priorities and technical estimates

2. Small releases –put a simple system into production, then release new versions in very short cycle

3. Metaphor –all development is guided by a simple shared story of how the whole system works

4. Simple design –system is designed as simply as possible (extra complexity removed as soon as found)

5. Testing –programmers continuously write unit tests; customers write tests for features

6. Refactoring –programmers continuously restructure the system without changing its behavior to remove duplication and simplify

XP Practices (7 –12)

7. Pair-programming --all production code is written with two programmers at one machine

8. Collective ownership –anyone can change any code anywhere in the system at any time.

9. Continuous integration –integrate and build the system many times a day –every time a task is completed.

10.40-hour week –work no more than 40 hours a week as a rule

11. On-site customer –a user is on the team and available full-time to answer questions

12. Coding standards–programmers write all code in accordance with rules emphasizing communication through the code

XP is "extreme" because common sense practices taken to extreme levels

•If code reviews are good, review code all the time (pair programming)

•If testing is good, everybody will test all the time

•If simplicity is good, keep the system in the simplest design that supports its current functionality. (Simplest thing that works)

•If design is good, everybody will design daily (refactoring)

•If architecture is important, everybody will work at defining and refining the architecture (metaphor)

•If integration testing is important, build and integrate test several times a day (continuous integration)

•If short iterations are good, make iterations really, really short (hours rather than weeks)

AGILE WITH SCRUM

•As a brief introduction, Scrum is an agile process for software development. With Scrum, projects progress via a series of iterations called sprints. Each sprint is typically 2-4 weeks long. While an agile approach can be used for managing any project, Scrum is ideally suited for projects with rapidly changing or highly emergent requirements such as we find in software development.

SCRUM
What is Scrum?

•Scrum is an agile approach to software development. Rather than a full process or methodology, it is a framework. So instead of providing complete, detailed descriptions of how everything is to be done on the project, much is left up to the software development team. This is done because the team will know best how to solve the problem they are presented. This is why, for example, a sprint planning meeting is described in terms of the desired outcome (a commitment to a set of features to be developed in the next sprint) instead of a set of Entry criteria, Task definitions, Validation criteria, and Exit criteria (ETVX) as would be provided in most methodologies.

•Scrum relies on a self-organizing, cross-functional team. The scrum team is self-organizing in that there is no overall team leader who decides which person will do which task or how a problem will be solved. Those are issues that are decided by the

team as a whole. The scrum team is cross-functional so that everyone necessary to take a feature from idea to implementation is involved.

•These agile development teams are supported by two specific individuals: a Scrum Master and a product owner. The Scrum Master can be thought of as a coach for the team, helping team members use the Scrum framework to perform at their highest level. The product owner represents the business, customers or users and guides the team toward building the right product.

•Scrum projects make progress in a series of sprints, which are time boxed iterations no more than a month long. At the start of a sprint, team members commit to delivering some number of features that were listed on the project's scrum product backlog. At the end of the sprint, these features are done--they are coded, tested, and integrated into

the evolving product or system. At the end of the sprint a sprint review is conducted during which the team demonstrates the new functionality to the product owner and other interested stakeholders who provide feedback that could influence the next sprint.

Sprint Planning Meeting

•The Sprint Planning Meeting is attended by the product owner, Scrum Master, and the entire Scrum Team. Outside stakeholders may attend by invitation of the team, although this is rare in most companies.

•During the sprint planning meeting the product owner describes the highest priority features to the team. The team asks enough questions that they can turn a high-level user story of the product backlog into the more detailed tasks of the sprint backlog.

•The product owner doesn't have to describe every item being tracked on the product backlog. A good guideline is for the product owner to come to the sprint planning meeting prepared to talk about two sprint's worth of product backlog items. To make an example really simple, suppose a team always finishes five product backlog items. Their product owner should enter the sprint planning meeting prepared to talk about the top ten priorities.

•There are two defined artifacts that result from a sprint planning meeting:

- •1. A sprint goal
- •2. A sprint backlog

•A sprint goal is a short, one-or two-sentence, description of what the team plans to achieve during the sprint. It is written collaboratively by the team and the product owner. The following are typical sprint goals on an eCommerce application:

••Implement basic shopping cart functionality including add, remove, and update quantities.

••The checkout process—pay for an order, pick shipping, order gift wrapping, etc.

•The sprint goal can be used for quick reporting to those outside the sprint. There are always stakeholders who want to know what the team is working on, but who do not need to hear about each product backlog item (user story) in detail. The success of the sprint will later be assessed during the Sprint Review Meeting against the sprint goal, rather than against each specific item selected from the product backlog.

•The sprint backlog is the other output of sprint planning. A sprint backlog is a list of the product backlog items the team commits to delivering plus the list of tasks necessary to delivering those product backlog items. Each task on the sprint backlog is also usually estimated.

•An important point to reiterate here is that it is the team that selects how much work they can do in the coming sprint. The product owner does not get to say, "We have four

sprints left so you need to do one-fourth of everything I need." We can hope the team does that much (or more) but it is up to the team to determine how much they can do in the sprint.

•In product development, a scrum sprint is a set period of time during which specific work has to be completed and made ready for review.

•Each sprint begins with a planning meeting. During the meeting, the product owner (the person requesting the work) and the development team agree upon exactly what work will be accomplished during the sprint. The development team has the final say when it comes to determining how much work can realistically be accomplished during the sprint, and the product owner has the final say on what criteria needs to be met for the work to be approved and accepted.

•The duration of a sprint is determined by the scrum master, the team's facilitator. Once the team reaches a consensus for how many days a sprint should last, all future sprints should be the same. Traditionally, a sprint lasts 30 days.

•After a sprint begins, the product owner must step back and let the team do their work. During the sprint, the team holds daily stand up meeting to discuss progress and brainstorm solutions to challenges. The project owner may attend these meetings as an observer but is not allowed to participate unless it is to answer questions. (See pigs and chickens). The project owner may not make requests for changes during a sprint and only the scrum master has the power to interrupt or stop the sprint.

•At the end of the sprint, the team presents its completed work to the project owner and the project owner uses the criteria established at the sprint planning meeting to either accept or reject the work.

ITERATIVE

•Iteration means the act of repeating a process usually with the aim of approaching a desired goal or target or result. Each repetition of the process is also called an "iteration," and the results of one iteration are used as the starting point for the next iteration

RUP

•The unified process groups increments/iterations into phases: inception, elaboration, construction, and transition.

•Inception identifies project scope, requirements (functional and non-functional) and risks at a high level but in enough detail that work can be estimated.

•Elaboration delivers a working architecture that mitigates the top risks and fulfills the non-functional requirements.

•Construction incrementally fills-in the architecture with production-ready code produced from analysis, design, implementation, and testing of the functional requirements.

RUP

•Transition delivers the system into the production operating environment.

•Each of the phases may be divided into 1 or more iterations, which are usually time-boxed rather than feature-boxed. Architects and analysts work one iteration ahead of developers and testers to keep their work-product backlog full.

•Within each iteration, the tasks are categorized into nine disciplines:

•Six "engineering disciplines"

•Business Modeling

•Requirements

•Analysis and Design

•Implementation

•Test

•Deployment

•Three supporting disciplines

•Configuration and Change Management

•Project Management

•Environment

•The RUP has determined a project life cycle consisting of four phases. These phases allow the process to be presented at a high level in a similar way to how a 'waterfall'-styled project might be presented, although in essence the key to the process lies in the iterations of development that lie within all of the phases. Also, each phase has one key objective and milestone at the end that denotes the objective being accomplished. The visualization of RUP phases and disciplines over time is referred to as the RUP hump chart

Inception Phase

•The primary objective is to scope the system adequately as a basis for validating initial costing and budgets. In this phase the business case which includes business context, success factors (expected revenue, market recognition, etc.), and financial forecast is established. To complement the business case, a basic use case model, project plan, initial risk assessment and project description (the core project requirements, constraints and key features) are generated. After these are completed, the project is checked against the following criteria:

•Stakeholder concurrence on scope definition and cost/schedule estimates.

•Requirements understanding as evidenced by the fidelity of the primary use cases.

•Credibility of the cost/schedule estimates, priorities, risks, and development process.

•Depth and breadth of any architectural prototype that was developed.

Inception Phase

•Establishing a baseline by which to compare actual expenditures versus planned expenditures.

•If the project does not pass this milestone, called the Lifecycle Objective Milestone, it either can be cancelled or repeated after being redesigned to better meet the criteria.

Elaboration Phase

•The primary objective is to mitigate the key risk items identified by analysis up to the end of this phase. The elaboration phase is where the project starts to take shape. In this phase the problem domain analysis is made and the architecture of the project gets its basic form.

•The outcome of the elaboration phase is:

•A use-case model in which the use-cases and the actors have been identified and most of the use-case descriptions are developed. The use-case model should be 80% complete.

•A description of the software architecture in a software system development process.

•An executable architecture that realizes architecturally significant use cases.

Elaboration Phase

•Business case and risk list which are revised.

•A development plan for the overall project.

•Prototypes that demonstrably mitigate each identified technical risk.

•A preliminary user manual (optional)

•This phase must pass the Lifecycle Architecture Milestone criteria answering the following questions:

•Is the vision of the product stable?

•Is the architecture stable?

•Does the executable demonstration indicate that major risk elements are addressed and resolved?

•Is the construction phase plan sufficiently detailed and accurate?

Elaboration Phase

•Do all stakeholders agree that the current vision can be achieved using current plan in the context of the current architecture?

•Is the actual vs. planned resource expenditure acceptable?

•If the project cannot pass this milestone, there is still time for it to be cancelled or redesigned. However, after leaving this phase, the project transitions into a high-risk operation where changes are much more difficult and detrimental when made.

•The key domain analysis for the elaboration is the system architecture

Construction Phase

•The primary objective is to build the software system. In this phase, the main focus is on the development of components and other features of the system. This is the phase when the bulk of the coding takes place. In larger projects, several construction iterations may be developed in an effort to divide the use cases into manageable segments that produce demonstrable prototypes.

•This phase produces the first external release of the software. Its conclusion is marked by the Initial Operational Capability Milestone

Transition Phase

•The primary objective is to 'transit' the system from development into production, making it available to and understood by the end user. The activities of this phase include training the end users and maintainers and beta testing the system to validate it against the end users' expectations. The product is also checked against the quality level set in the Inception phase.

•If all objectives are met, the Product Release Milestone is reached and the development cycle is finished.