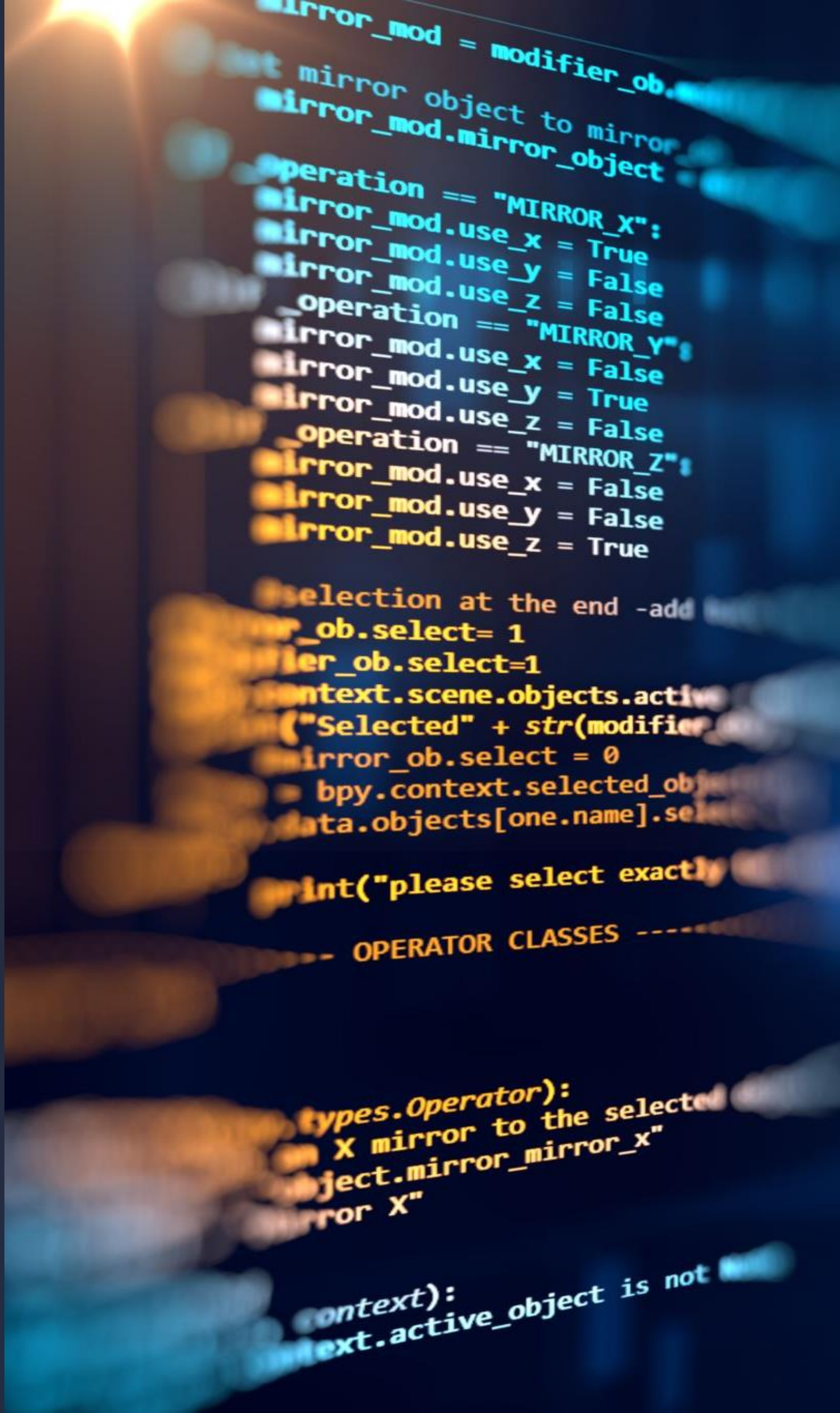
CSE216: SOFTWARE ENTERPRISE: PERSONAL PROCESSES AND QUALITY



```
irror_mod = modifier_ob.
t mirror object to mirror
irror_mod.mirror_object
peration == "MIRROR_X":
irror_mod.use_x = True
irror_mod.use_y = False
irror_mod.use_z = False
_operation == "MIRROR_Y":
irror_mod.use_x = False
irror_mod.use_y = True
irror_mod.use_z = False
_operation == "MIRROR_Z":
irror_mod.use_x = False
irror_mod.use_y = False
irror_mod.use_z = True

election at the end -add
_ob.select= 1
er_ob.select=1
ntext.scene.objects.active
("Selected" + str(modifier
irror_ob.select = 0
bpy.context.selected_obj
ata.objects[one.name].sel

int("please select exactly

OPERATOR CLASSES ----

types.Operator):
X mirror to the selected
ject.mirror_mirror_x"
rror X"

context):
ext.active_object is not
```

Assist. Prof.

Dr. Noha El-Sayad

WEEK 2

- Personal Software Process

Why Small or Large Projects Fail ?

- Project commitments are often unrealistic
- The larger the project, the less influence we have
- If we don’t have anything to say, nobody will listen
- Larger projects are harder to control
- Quality problems get worse with project size

1. In software systems, if any part has quality problems, the system will have quality problems)
2. If the developers do not manage quality, their teams cannot manage quality
3. When unmanaged, quality will always be poor

To be effective, need Team Software Process (TSP)

The PSP provides the knowledge and skill that developers need to work on TSP teams.

- Teams need leadership and coaching
- Leaders build team motivation and commitment
- Coaching develops team cohesion
- Cohesive, motivated, and committed teams do the best work

PSP Principles

1. The quality of a software system is determined by the *quality of its worst components*
2. The quality of a software component is governed by the *individual* who developed it
3. The quality of a software component is governed by the *quality of the process* used to develop it
4. The key to quality is the individual developer’s skill, commitment, and personal process discipline
5. As a software professional, you are responsible for your personal process
6. You should *measure*, *track*, and *analyze* your work
7. You should *learn* from your performance variations
8. You should *incorporate lessons learned* into your personal practices

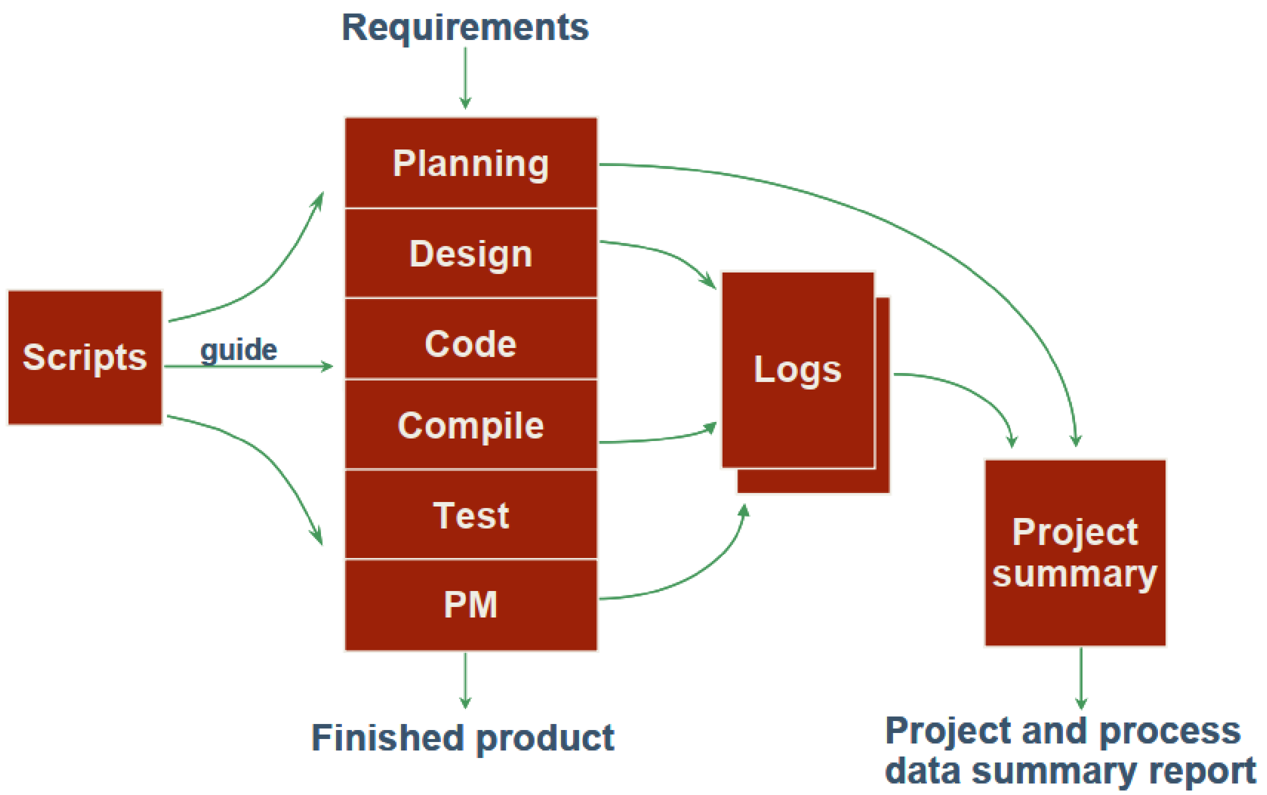
What is the PSP?

The PSP is a personal process for developing software or for doing any other defined activity

The PSP includes:

1. defined step
2. Forms
3. standards

The PSP Process Flow (defined step)



Learning the PSP

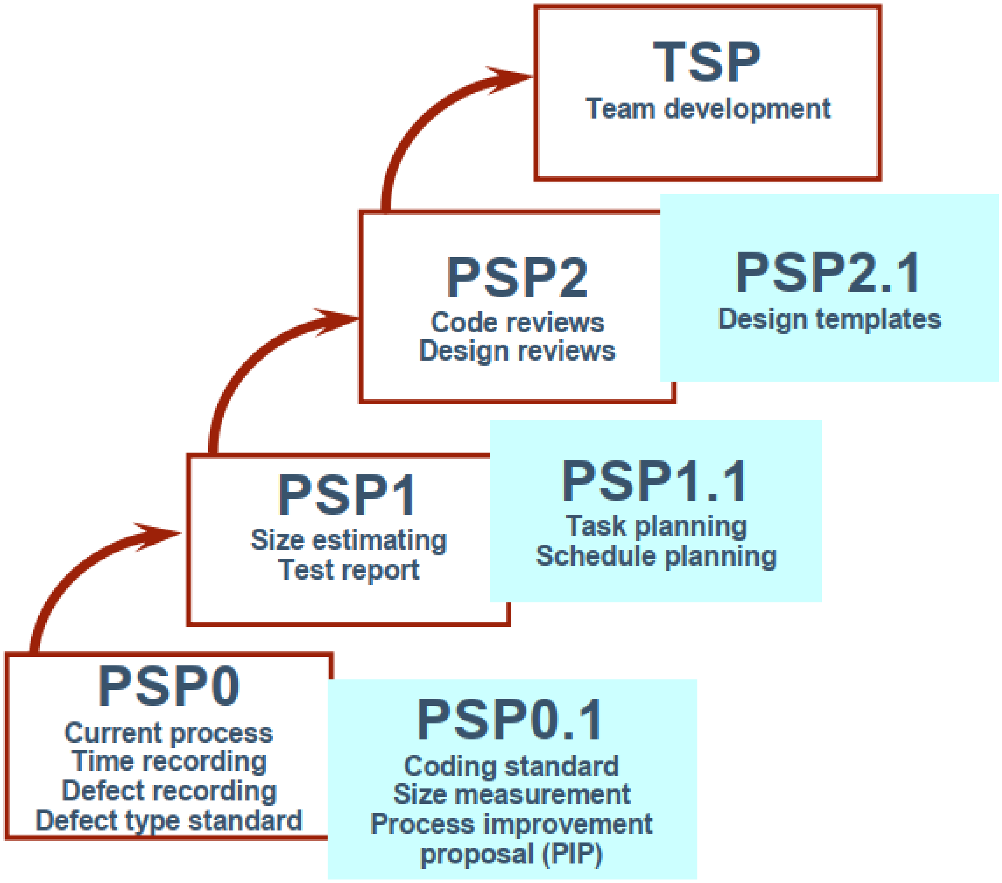
The PSP is introduced in six upward-compatible steps

- write one or more module-sized programs at each step
- gather and analyze data on your work
- use the results to improve your personal performance

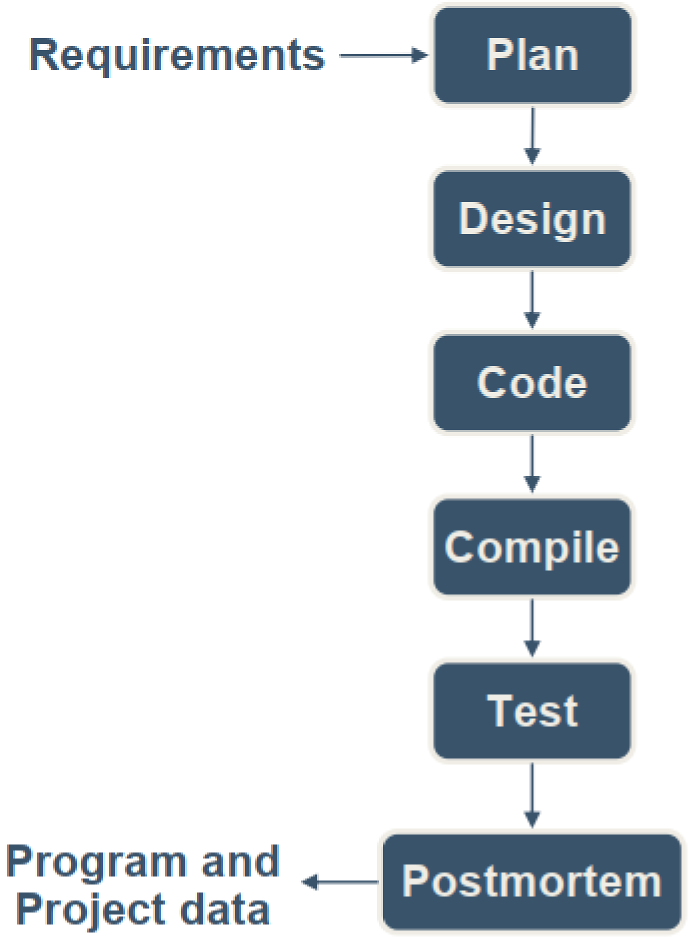
PSP0: You establish a measured performance baseline.

PSP1: You make size, resource, and schedule plans.

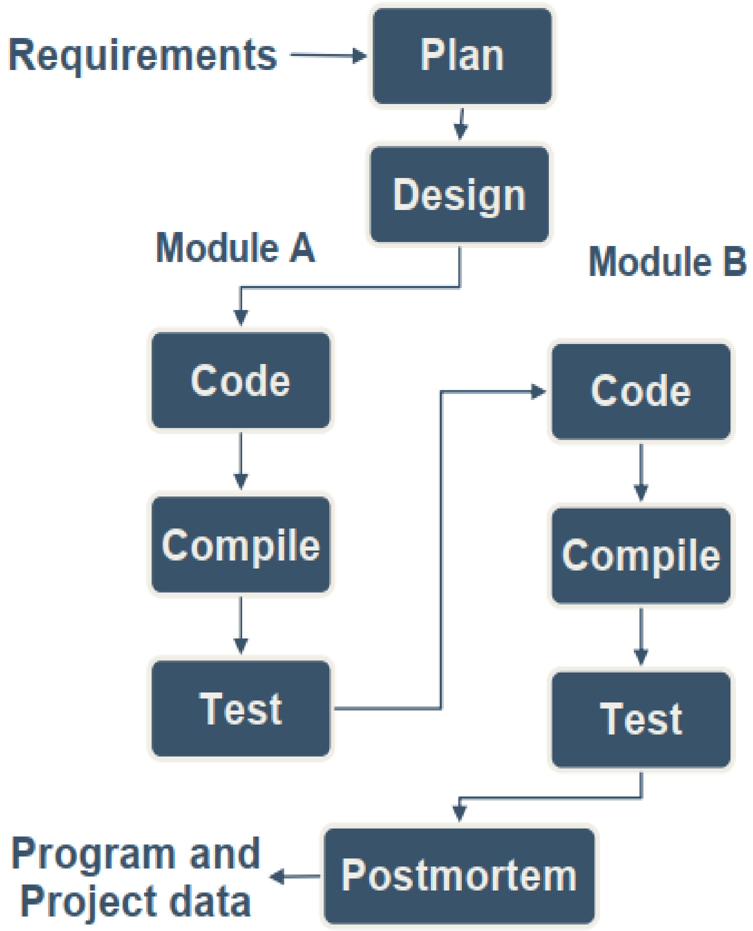
PSP2: You practice defect and yield management



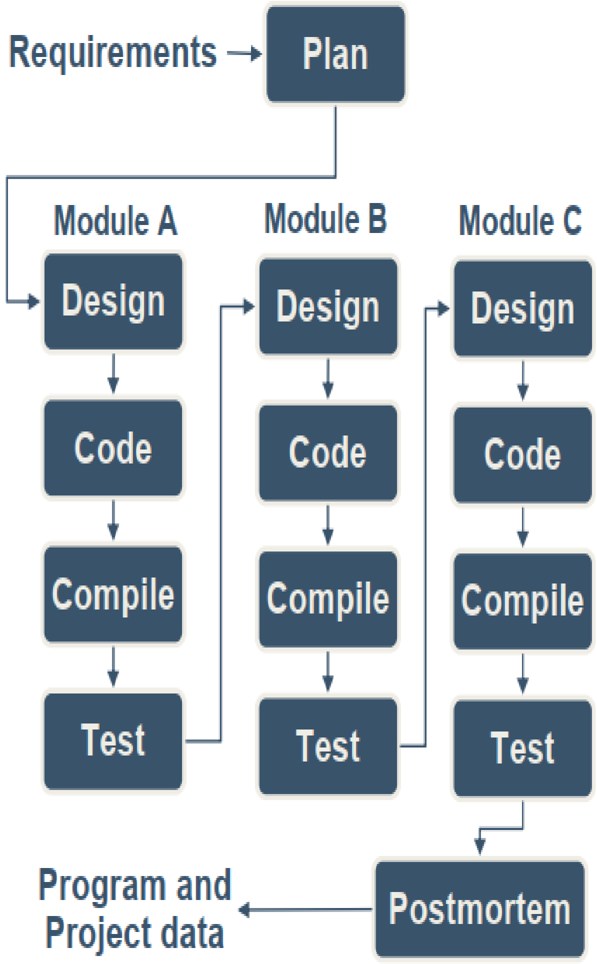
Process Flow



Cyclic Process Flow -1

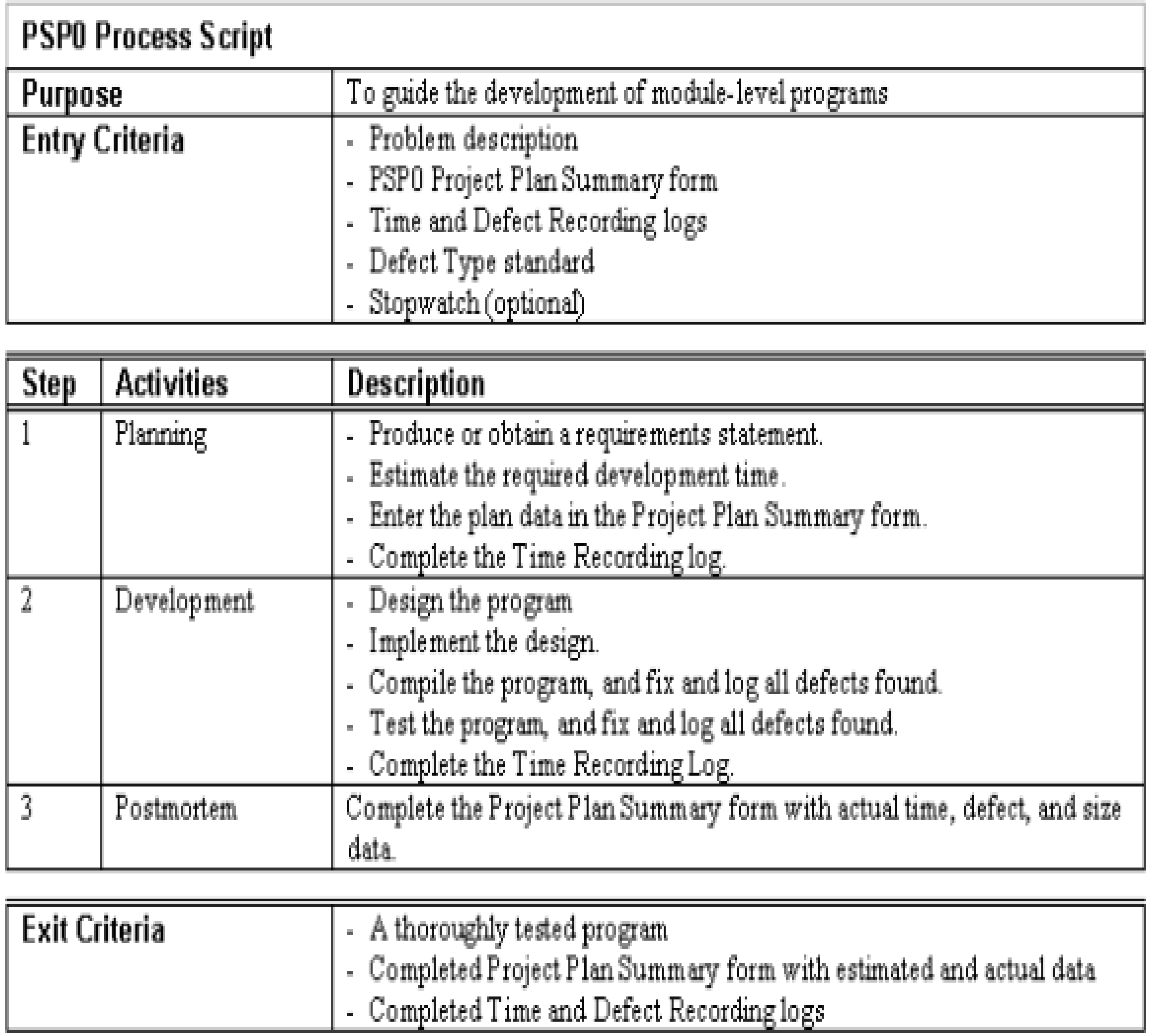


Cyclic Process Flow -2



The PSP Scripts

- Planning: Estimate the development time
- Development: Develop the product using your current methods
- Postmortem: Complete the project plan summary with the time spent and defects found and injected in each phase.
- Design: Design the program using your current design methods
- Coding: Implement the program
- Compile: Compile until defect-free
- Test: Test the program and fix all defects
- Record defects in the defect log and time per phase in the time log



PSP0 Process Script	
Purpose	To guide the development of module-level programs
Entry Criteria	- Problem description - PSP0 Project Plan Summary form - Time and Defect Recording logs - Defect Type standard - Stopwatch (optional)

Step	Activities	Description
1	Planning	- Produce or obtain a requirements statement. - Estimate the required development time. - Enter the plan data in the Project Plan Summary form. - Complete the Time Recording log.
2	Development	- Design the program - Implement the design. - Compile the program, and fix and log all defects found. - Test the program, and fix and log all defects found. - Complete the Time Recording Log.
3	Postmortem	Complete the Project Plan Summary form with actual time, defect, and size data.

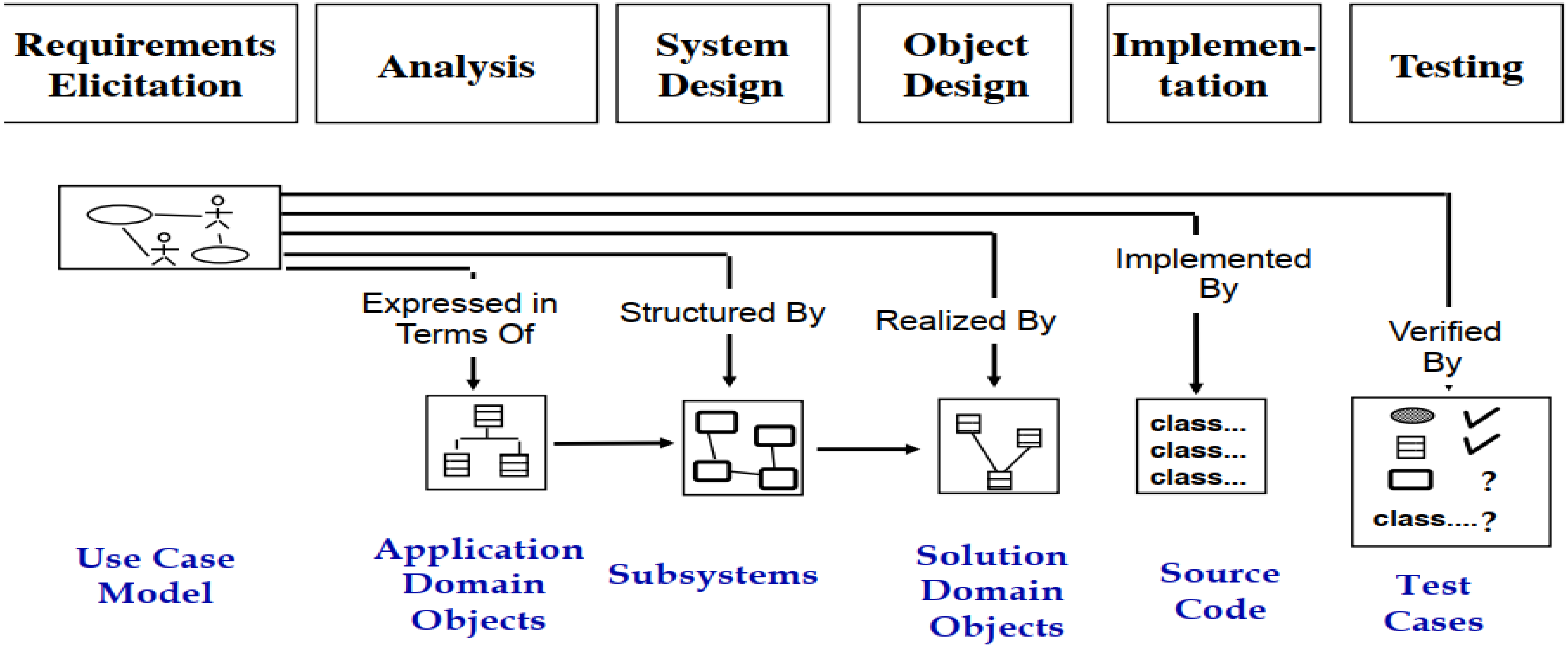
Exit Criteria	- A thoroughly tested program - Completed Project Plan Summary form with estimated and actual data - Completed Time and Defect Recording logs

Software Development Lifecycle (SDLC)

Software lifecycle:

- Set of activities and their relationships to each other to support the development of a software system

Software Lifecycle Activities



Requirements Elicitation

Analysis

System Design

Object Design

Implemen-tation

Testing

Expressed in Terms Of

Structured By

Realized By

Implemented By

Verified By

Use Case Model

Application Domain Objects

Subsystems

Solution Domain Objects

Source Code

Test Cases

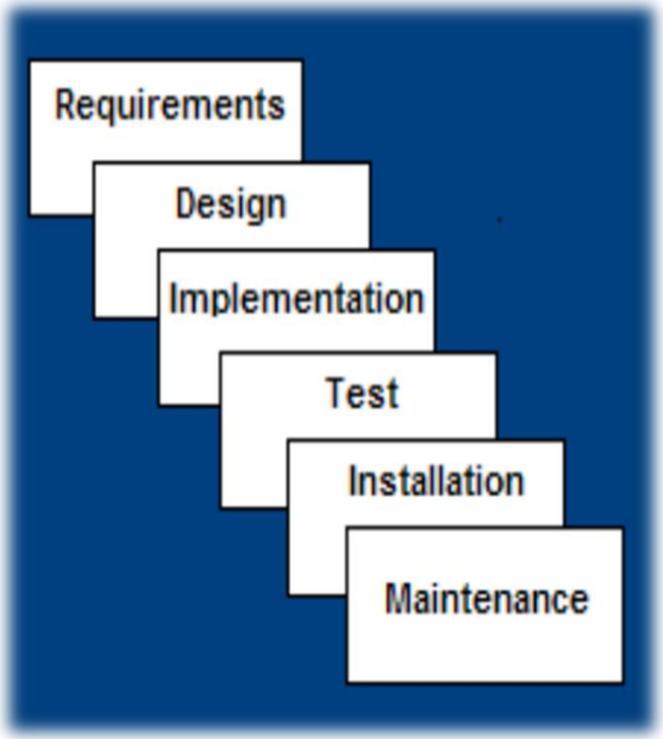
SDLC Model

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

1. Waterfall Model
2. V-Shaped SDLC Model
3. Structured Evolutionary Prototyping Model
4. Incremental SDLC Model
5. Spiral SDLC Model
6. Agile SDLC’s
7. Extreme Programming – XP
8. Rapid Application Model (RAD)
9. Phases in the Rational Unified Process (RUB)

1. Waterfall Model

	When to Use
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	1. Requirements are very well known 2. Product definition is stable 3. Technology is understood 4. New version of an existing product 5. Porting an existing product to a new platform



1. Waterfall Model

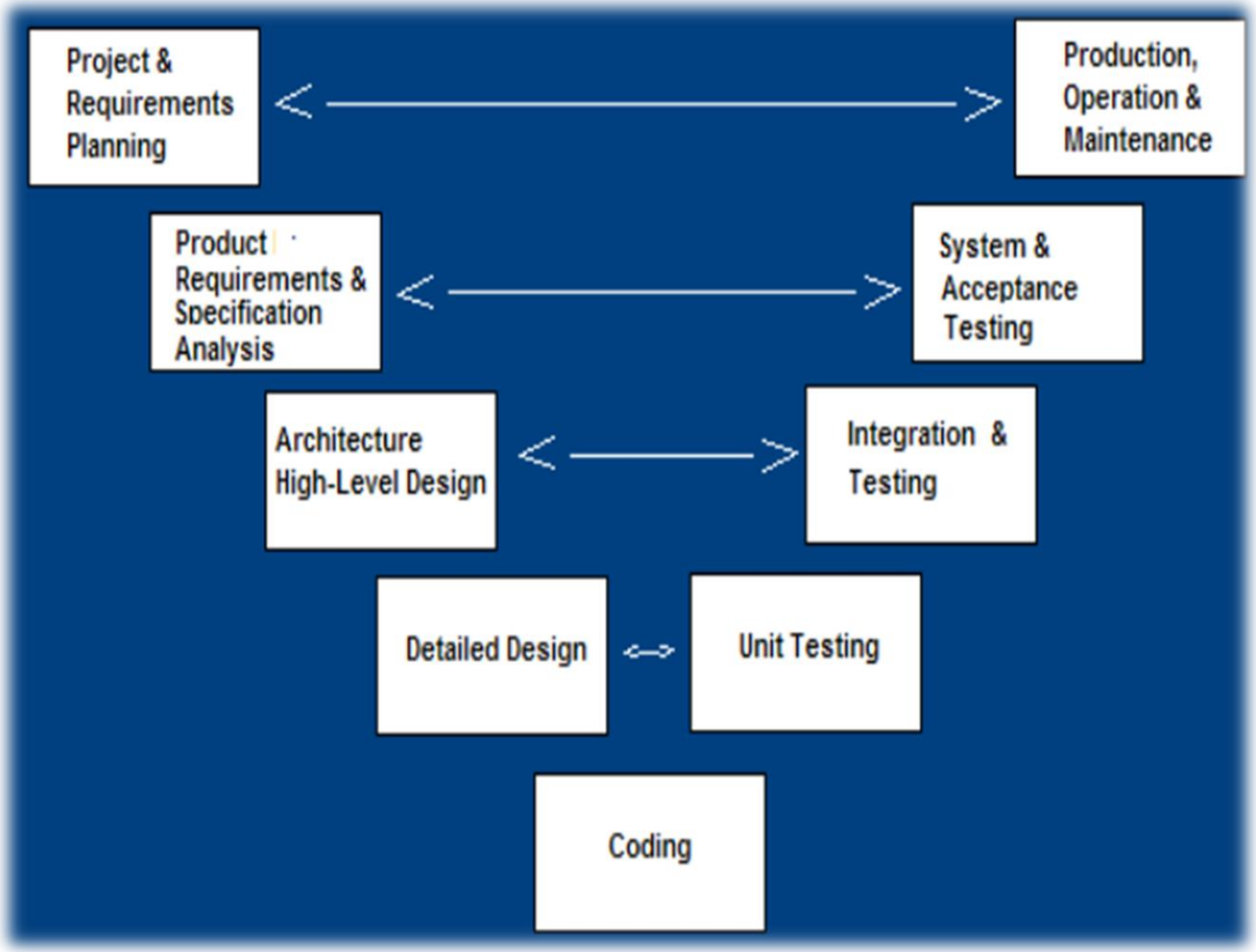
Strengths	Deficiencies
1. Easy to understand and easy to use. 2. Provides structure to inexperienced staff 3. Milestones are well understood. 4. Sets requirements stability 5. Good for management control (plan, staff, track). 6. Works well when quality is more important than cost or schedule	1. All requirements must be known upfront 2. Deliverables created for each phase are considered frozen – inhibits flexibility 3. Can give a false impression of progress 4. Does not reflect problem-solving nature of software development – iterations of phases 5. Integration is one big bang at the end 6. Little opportunity for customer to preview the system (until it may be too late)

2. V-Shaped SDLC Model

	When to Use
emphasizes the verification and validation of the product. • Testing of the product is planned in parallel with a corresponding phase of developmentg	1. Excellent choice for systems requiring high reliability (hospital patient control applications) 2. All requirements are known up-front 3. When it can be modified to handle changing 4. requirements beyond analysis phase 5. • Solution and technology are known

2. V-Shaped SDLC Model

- 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


Project & Requirements Planning
Production, Operation & Maintenance
Product Requirements & Specification Analysis
System & Acceptance Testing
Architecture High-Level Design
Integration & Testing
Detailed Design
Unit Testing
Coding

2. V-Shaped SDLC Model

Strengths	Deficiencies
1. Emphasize planning for verification and validation of the product in early stages of product development 2. Each deliverable must be testable 3. Project management can track progress by milestones 4. Easy to use	1. Does not easily handle concurrent events 2. Does not handle iterations or phases 3. Does not easily handle dynamic changes in requirements 4. Does not contain risk analysis activities

3. Structured Evolutionary Prototyping Model

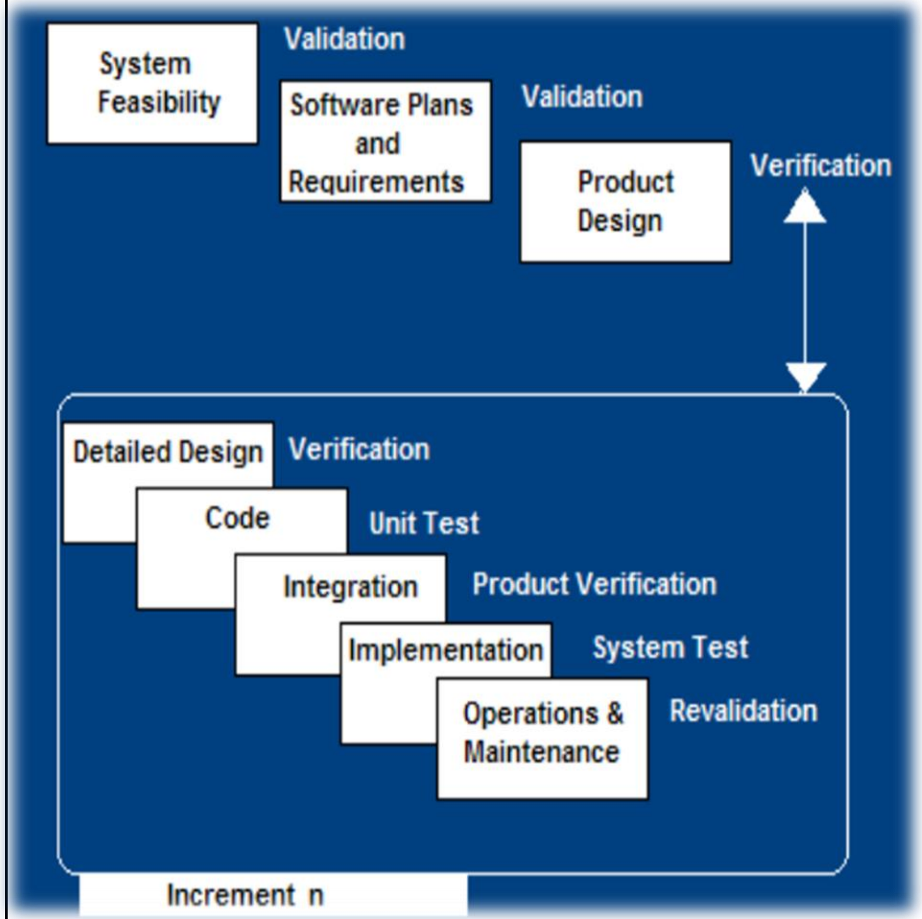
	When to Use
Developers build a prototype during the requirements phase then users evaluate it and give corrective feedback. Steps 1. A preliminary project plan is developed. 2. A partial high-level paper model is created. 3. The model is source for a partial requirements specification. 4. A prototype is built with basic and critical attributes. 5. The designer builds the database user interface algorithmic functions. 6. The designer demonstrates the prototype, the user evaluates for problems and suggests improvements. 7. This loop continues until the user is satisfied.	1. Requirements are unstable or have to be clarified 2. As the requirements clarification stage of a waterfall model 3. Develop user interfaces 4. Short-lived demonstrations 5. New, original development 6. With the analysis and design portions of object-oriented development

3. Structured Evolutionary Prototyping Model

Strengths	Weakness
1. Customers can “see” the system requirements as they are being gathered 2. Developers learn from customers 3. A more accurate end product 4. Unexpected requirements accommodated 5. Allows for flexible design and development 6. Steady, visible signs of progress produced 7. Interaction with the prototype stimulates awareness of additional needed functionality	1. Tendency to abandon structured program development for “code-and-fix” development 2. Bad reputation for “quick-and-dirty” methods 3. Overall maintainability may be overlooked 4. The customer may want the prototype delivered. 5. Process may continue forever

4. Incremental SDLC Model

	When to Use
• 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.	1. Risk, funding, schedule, program complexity, or need for early realization of benefits. 2. Most of the requirements are known up-front but are expected to evolve over time 3. On projects which have lengthy development schedules 4. On a project with new technology



4. Incremental SDLC Model

Strengths	Weakness
1. Develop high-risk or major functions first 2. Each release delivers an operational product 3. Customer can respond to each build 4. Uses “divide and conquer” breakdown of tasks 5. Lowers initial delivery cost 6. Initial product delivery is faster 7. Customers get important functionality early 8. Risk of changing requirements is reduced	1. Requires good planning and design 2. Requires early definition of a complete and fully functional system to allow for the definition of increments 3. Well-defined module interfaces are required (some will be developed long before others) 4. Total cost of the complete system is not lower

5. 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

Spiral Quadrant

1. **Determine objectives, alternatives and constraints**
2. **Evaluate alternatives, identify and resolve risks**

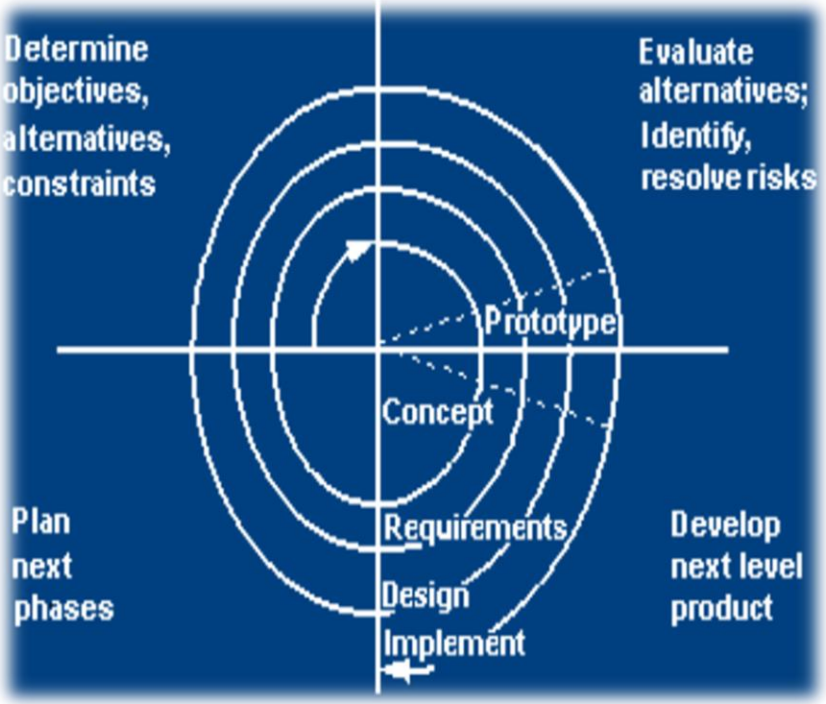
Study alternatives relative to objectives and constraints then Identify risks and Resolve risks.

3. Develop next-level product

Typical activates: (Create a design, Review design, Develop code, Inspect code, Test product)

4. Plan next phase

Typical activities (Develop project plan, Develop configuration management plan, Develop a test plan, Develop an installation plan)


Determine objectives, alternatives, constraints
Evaluate alternatives; Identify, resolve risks
Prototype
Concept
Requirements
Design
Implement
Plan next phases
Develop next level product

5. Spiral SDLC Model

When to use

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

5. Spiral SDLC Model

Strengths	Weakness
1. Provides early indication of insurmountable risks, without much cost 2. Users see the system early because of rapid prototyping tools 3. Critical high-risk functions are developed first 4. The design does not have to be perfect 5. Users can be closely tied to all lifecycle steps 6. Early and frequent feedback from users 7. Cumulative costs assessed frequently	1. Time spent for evaluating risks too large for small or low risk projects 2. Time spent planning, resetting objectives, doing risk analysis and prototyping may be excessive 3. The model is complex 4. Risk assessment expertise is required 5. Spiral may continue indefinitely 6. May be hard to define objective, verifiable milestones that indicate readiness to proceed through the next iteration

6. 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

Methods

1. Adaptive Software Development (ASD)
2. Feature Driven Development (FDD)
3. Crystal Clear
4. Dynamic Software Development Method (DSDM)
5. Rapid Application Development (RAD)
6. Scrum
7. Extreme Programming (XP)
8. Rational Unified Process (RUP)

6. Agile SDLC's

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

Extreme Programming - XP

XP Practices (12 Principles)

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
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

6. Agile SDLC's

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

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

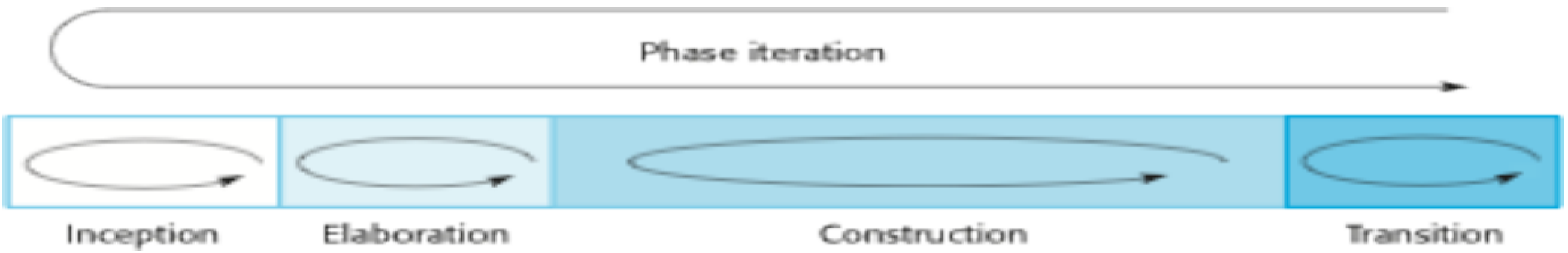
6. Agile SDLC's

Rapid Application Model (RAD)

Strengths	Weakness
1. Reduced cycle time and improved productivity with fewer people means lower costs 2. Time-box approach mitigates cost and schedule risk 3. Customer involved throughout the complete cycle and minimizes risk of not achieving customer satisfaction and business needs 4. • Focus moves from documentation to code (WYSIWYG). 5. Uses modeling concepts to capture information about business, data, and processes	1. Accelerated development process must give quick responses to the user 2. Risk of never achieving closure 3. Hard to use with legacy systems 4. Requires a system that can be modularized 5. Developers and customers must be committed to 6. rapid-fire activities in an abbreviated time frame

6. Agile SDLC's

Phases in the Rational Unified Process



RUP phases

- Inception
  - Establish the business case for the system.
- Elaboration
  - Develop an understanding of the problem domain and the system architecture.
- Construction
  - System design, programming and testing.
- Transition
  - Deploy the system in its operating environment.

RUP iteration

- In-phase iteration
  - Each phase is iterative with results developed incrementally.
- Cross-phase iteration
  - As shown by the loop in the RUP model, the whole set of phases may be enacted incrementally.