



LT 2: Mobile App Development Approaches

CS 504

Drake Patrick Mirembe

School of Professional and Vocational Education
Uganda Technology and Management University

2015



Outline of the talk

- Introduction to ***software development life cycle***
- The classic waterfall model
- Structure evolutionary model
- Feature driven design
- Spiral SDLC model
- Rapid Application Development (RAD)



What is a Mobile Application?

A wireless mobile application is a software application, a wireless service or a mobile service that can be either pushed to users' handheld wireless devices or downloaded and installed, over the air, on these devices.

OR

An application which resides in the mobile phone or which is accessed/used by a mobile phone over any channel such as SMS, MMS, GPRS, Voice, DTMF

What is a Mobile Application?

- Two types of mobile applications can be accessed by wireless devices, the first type:
- Browser-Based
 - A Browser-Based application is an application that is accessed through the use of the mobile device's web browser
 - Browser-Based applications are coded with the use of a markup language
- Native Applications
 - Native applications are those applications that are found entirely on the mobile device
 - These applications have their own runtime environment for execution
 - Highly interactive applications are really only feasible when they are native applications



Why Develop Mobile Applications?

- It is estimated that by December 2014, there were 6.7 billion mobile phone subscribers of which 1.82 billion were smart phones
- Provide mobile phone users with applications that can keep them productive, informed, entertained, or connected whenever they feel the need
- Large potential for financial gain in the field of mobile applications
- Solve problems which have many challenges and obstacles



Mobile Applications

- Mobile Applications can be found in any industry, they have been developed for:
 - Mobile Gaming (see [gameloft](#))
 - Mobile Banking (see [RBC](#))
 - Mobile Text, Presentation, and Spreadsheet (see [Microsoft Office Mobile](#))
 - Social Networking (see [Facebook](#))
 - Mobile News (see [Yahoo! Mobile News](#))
 - Location Aware Services (see [Loopt](#))
 - Healthcare – Matibabu and WinSenga



Mobile Application Development Challenges

- Development of mobile applications provides for many challenges and obstacles that are not commonly found in the development of applications for desktop computers. Hence need to we thought approach
- The challenges faced by developers are found in:
 - Heterogeneity of mobile devices
 - Security
 - Network
 - Unknown user requirements



Challenge: Mobile Devices

- Java is a portable implementation language, any application created with Java can be run on any machine which contains a Java Runtime Environment (JRE)
- J2ME, is similarly a portable language, which can be run on any mobile device which contains a JRE, however this portability is severely affected by the heterogeneity of the mobile devices currently on the market
- Mobile devices display a wide range of characteristics that will greatly effect a mobile application's performance, usability, functionality, etc.



Challenge: Mobile Devices Resources

- Small screen size
 - Mobile devices come in many different screen sizes
 - Consider the differentiating screen sizes between smartphones and cell phones
 - Smartphones offer the user a generally larger and higher resolution display screen, contrasted to cell phones which generally provide lower resolution and smaller display size
- Memory
 - Just as screen size differs from device to device, the amount of available memory and differs from device to device
 - Developers must create applications which have a minimal memory footprint on the device while being of service to the user
 - Memory must also be carefully managed during the execution of any mobile application as it can potentially render the phone unusable until termination of the application



Challenge: Mobile Devices Resources

- Processing Power
 - Another sign of the heterogeneity of mobile devices is the processing power
 - The CPUs differ from phone to phone and this must be taken into consideration by developers
 - Developers cannot create applications that require the user to wait an unreasonable amount of time for the service to load
- Input Devices
 - The input devices on mobile devices range from full QWERTY keyboards to three letter button inputs
 - This means developers must take into account how much text is required by the user to input into their application and what kind of difficulties they may experience based on their device



Challenge: Network Issues

- Transmission Errors
 - When creating mobile applications that utilize network connections there is a variety of issues that can effect the application
 - Wireless networks are exposed to interference which can alter the message received by the client or the server then what was originally sent
 - Applications must take into account these potential problems especially in financially sensitive services
- Message Latency
 - Messages that are to be sent to clients or servers can be delayed due to a variety of reasons such as overloaded network nodes or servers, dead or turned off cell phones, distance to travel
 - Applications must take this into account so as to avoid sending servers or clients stale information



More challenges

- Bandwidth Usage
 - Wireless customers are forced to pay fees to access the wireless network and internet
 - While phones with WIFI capabilities allow for some users to have free connectivity at times it is important to keep messages to a minimum and compact
 - Applications that cost a lot to use will not be popular with many of the financially conscious users
- Wireless networks by default are not as secure as wired networks, it is important to note that message can be intercepted when travelling through the air
- Mobile applications must secure the sensitive data that is being transmitted over the air
- There are different methods to implement security but it must be relative to the information we want to secure and the resources that we wish to use for securing it



Solution: Mobile Devices

- Screen Size
 - There is no one single method to overcome to problem of different screen sizes however there are some ways to help
 - 1: When dealing with graphics that should be placed on edges use methods which retrieve the edge of the display
 - 2: When creating an for a particular set of mobile devices (ie. Blackberry's, cell phones) create the layout to the smallest display size
- Memory
 - Compact data representation will help reduce the amount of memory it requires to load and use your application
 - Use optimization techniques to reduce the amount of code required to write your application (see [J2ME tech tips](#))
 - Compress any graphic images that you use in your application and save graphics in a format which takes the least space



Solution: Mobile Devices

- Processing Power
 - A result of reducing the memory consumption and footprint of the application should help time required to load applications
 - If the mobile application has a client-server architecture consider the partitioning of the application
 - Allow the server to do the brunt of the calculations and processing work and pass the information to the mobile device for less CPU intensive calculations
- Transmission Errors
 - Transmissions errors may be inevitable when dealing with wireless networks but there are some wireless network protocols than can correct or at the least detect these errors
 - One solution does not exist for every single type of transmission error that may occur, it is important to plan for these types of errors and be able to deal with them accordingly



Solution: Network

- Message Latency
 - In a client-server architecture the server can store messages that do not arrive at the mobile device and attempt to resend them at specific intervals
 - Servers can also store the message and send it when the mobile device reconnects to the system
 - Let the user know if they receive a message that can possibly be out of date or no longer valid, this could be done using timestamps
- Bandwidth Usage
 - Pass as little messages as required between the client and the server
 - Keep the messages as short as possible, you can use symbols to represent commands for the server
 - If your application must use a lot of bandwidth at least notify the user of this fact



Solution: Security

- Important to implement security to a level which is appropriate for the data being exchanged
- Mobile devices, having limited processing power, cannot generate large cryptographic keys in a reasonable amount of time
- There has been research into creating keys for algorithms such as RSA and others and sending this to the mobile device to use but this is an area that is still developing



Mobile Application Development

- Knowing the challenges faced by developers with mobile applications we can look at the tools and steps developers take when creating applications
- Mobile application development differs from development of applications on desktops because mobile applications are developed on one platform and then deployed on a totally different platform
- This leads to many issues that developers face after moving their application to another platform and stresses an importance on testing

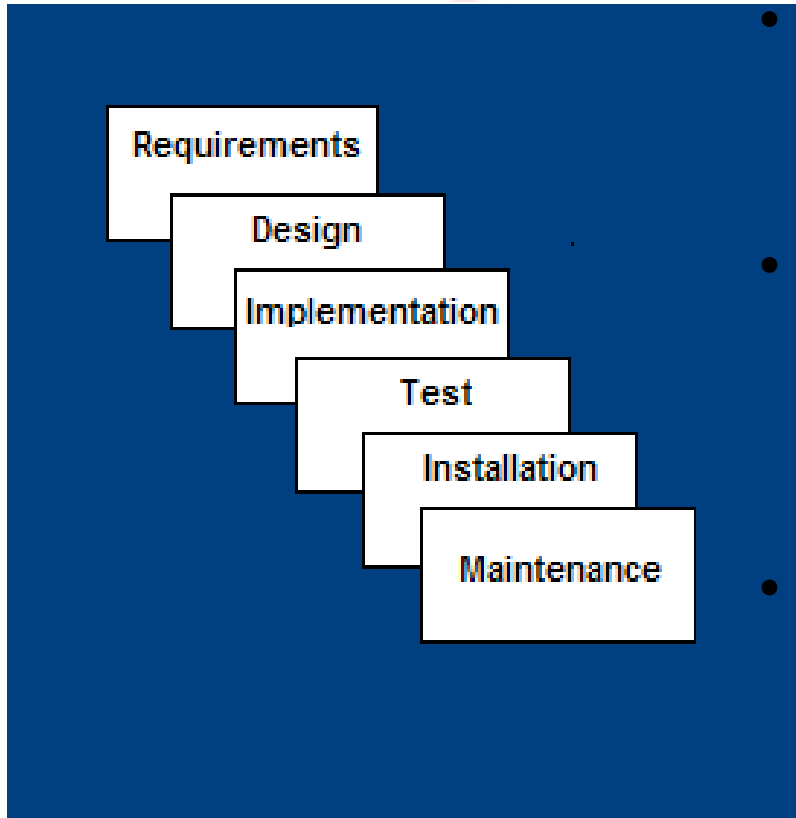


SDLC Model

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



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

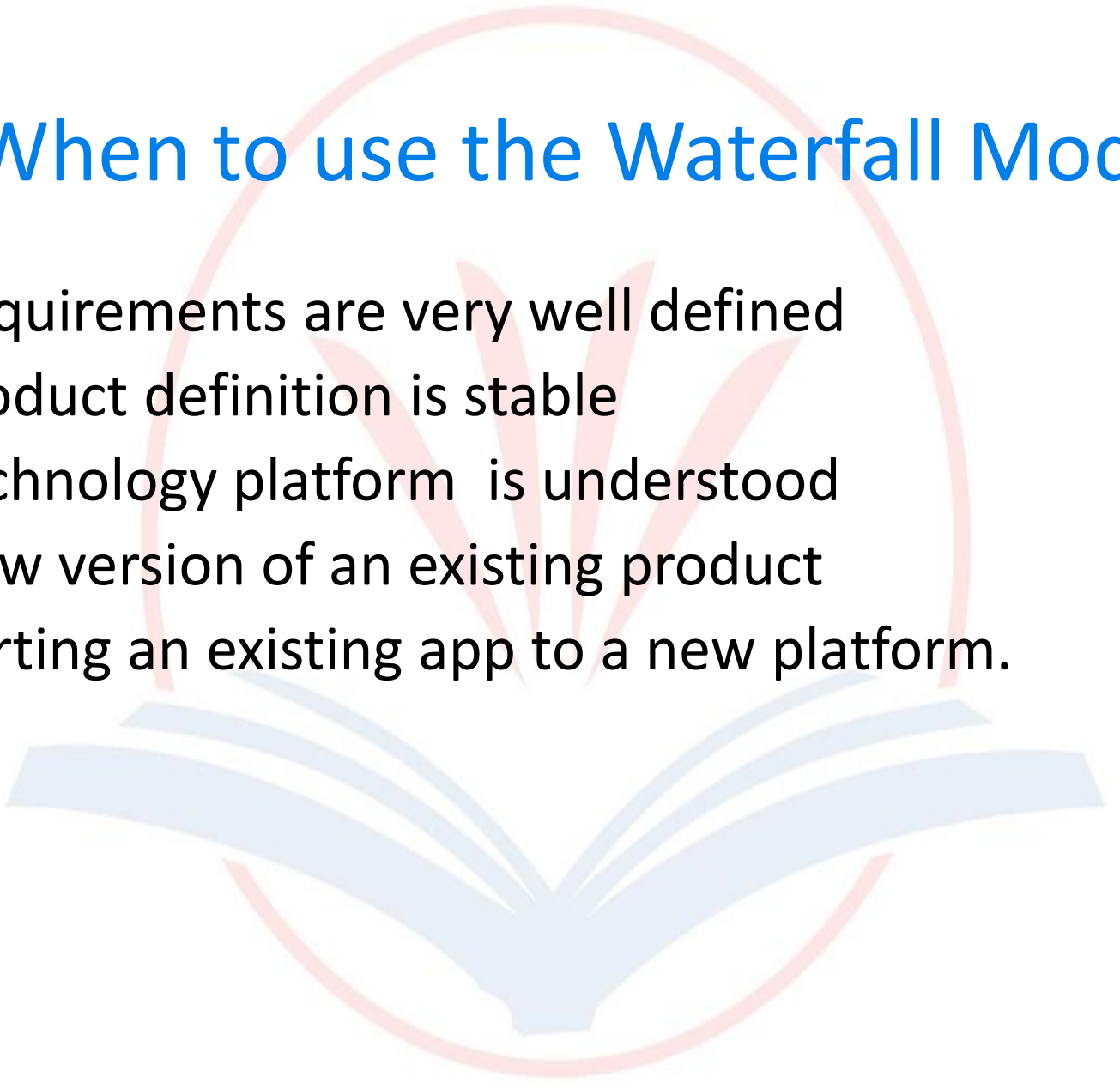


Waterfall Deficiencies

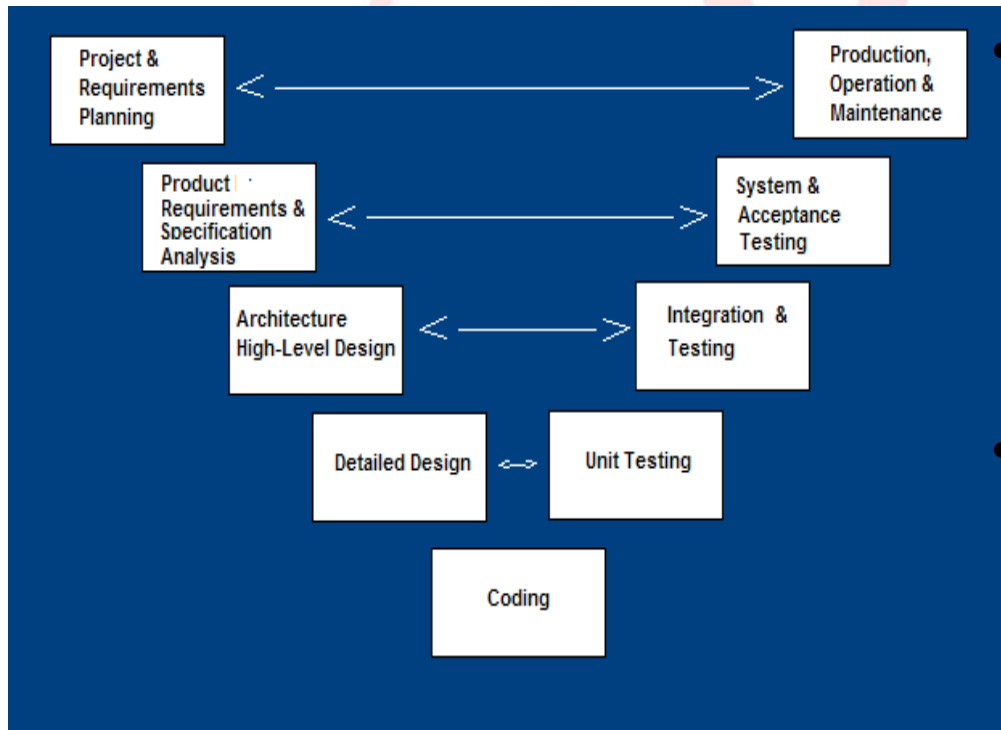
- All requirements must be known upfront
- Deliverables created for each phase are considered frozen – inhibits flexibility
- Can give a false impression of progress
- Does not reflect problem-solving nature of software development – iterations of phases
- Integration is one big bang at the end
- Little opportunity for customer to preview the system (until it may be too late)



When to use the Waterfall Model

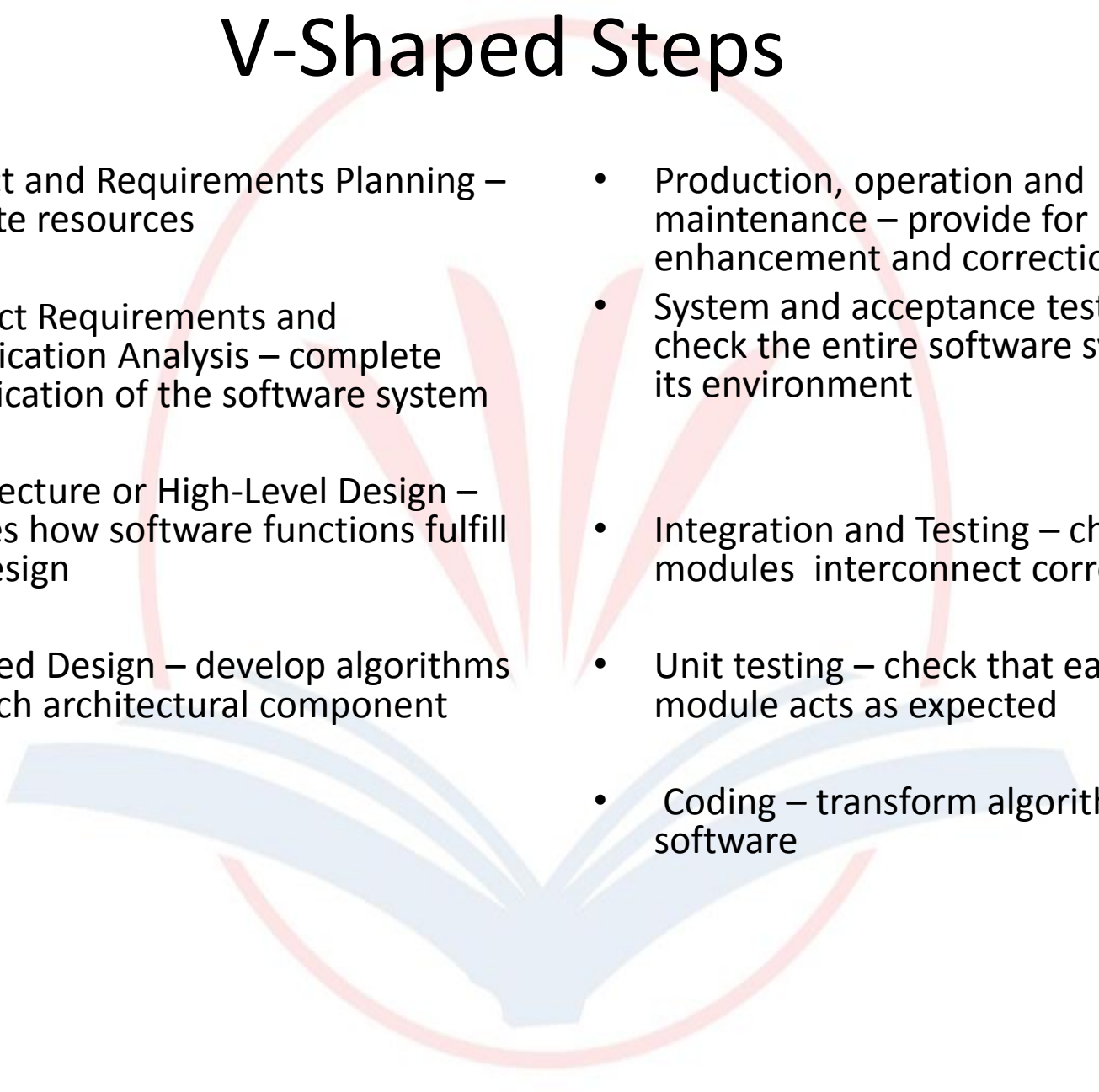
- Requirements are very well defined
 - Product definition is stable
 - Technology platform is understood
 - New version of an existing product
 - Porting an existing app to a new platform.
- 

V-Shaped SDLC Model



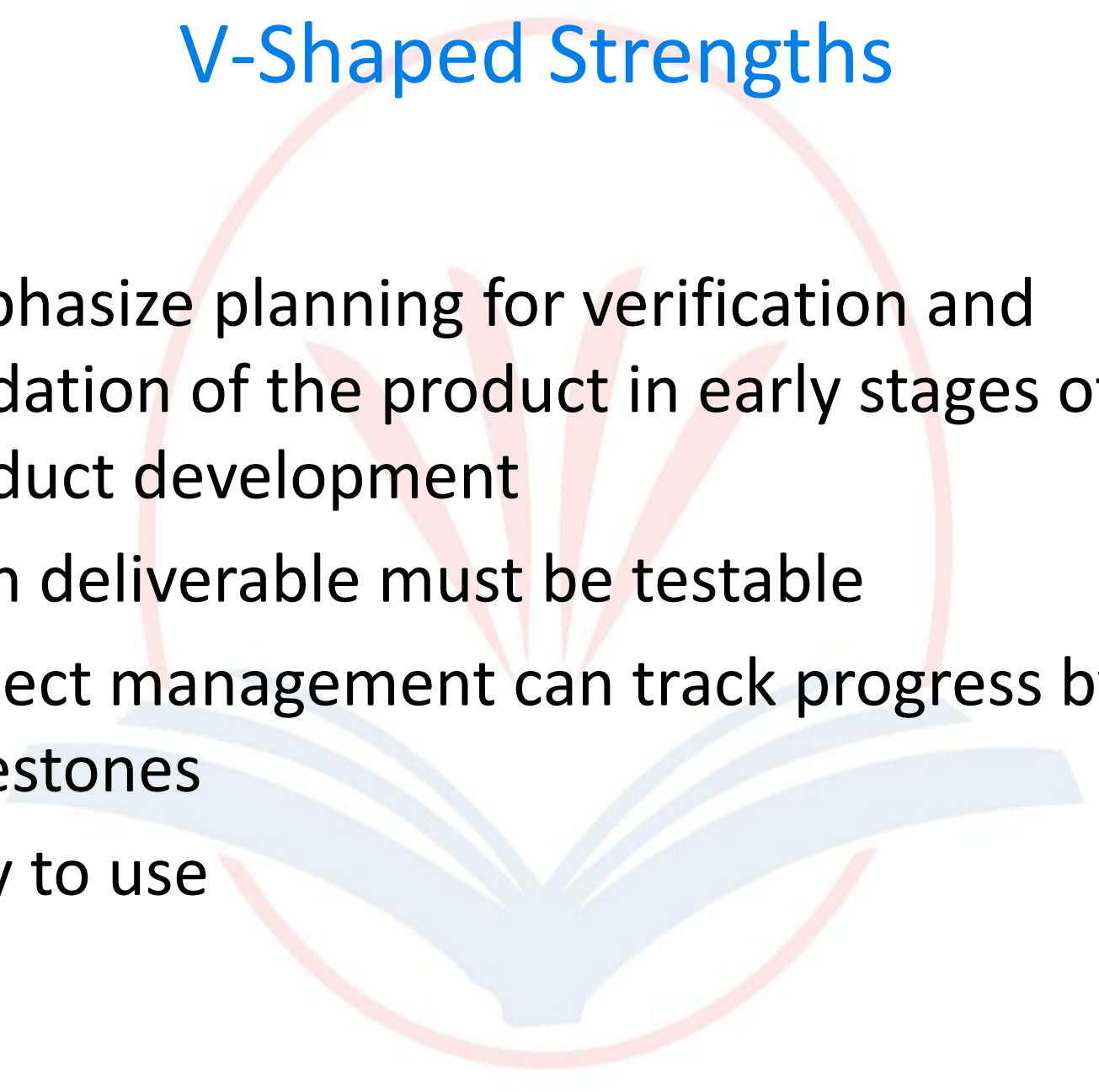
- A variant of the Waterfall that emphasizes the verification and validation of the product.
- Testing of the product is planned in parallel with a corresponding phase of development

V-Shaped Steps

- 
- The diagram illustrates the V-model of software development. It features a large, faint V-shape in the background. The left side of the V is composed of four red and blue diamond shapes. The right side of the V is composed of four red and blue diamond shapes. The bottom of the V is a light blue shape. The steps are listed in two columns, corresponding to the left and right sides of the V.
- 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

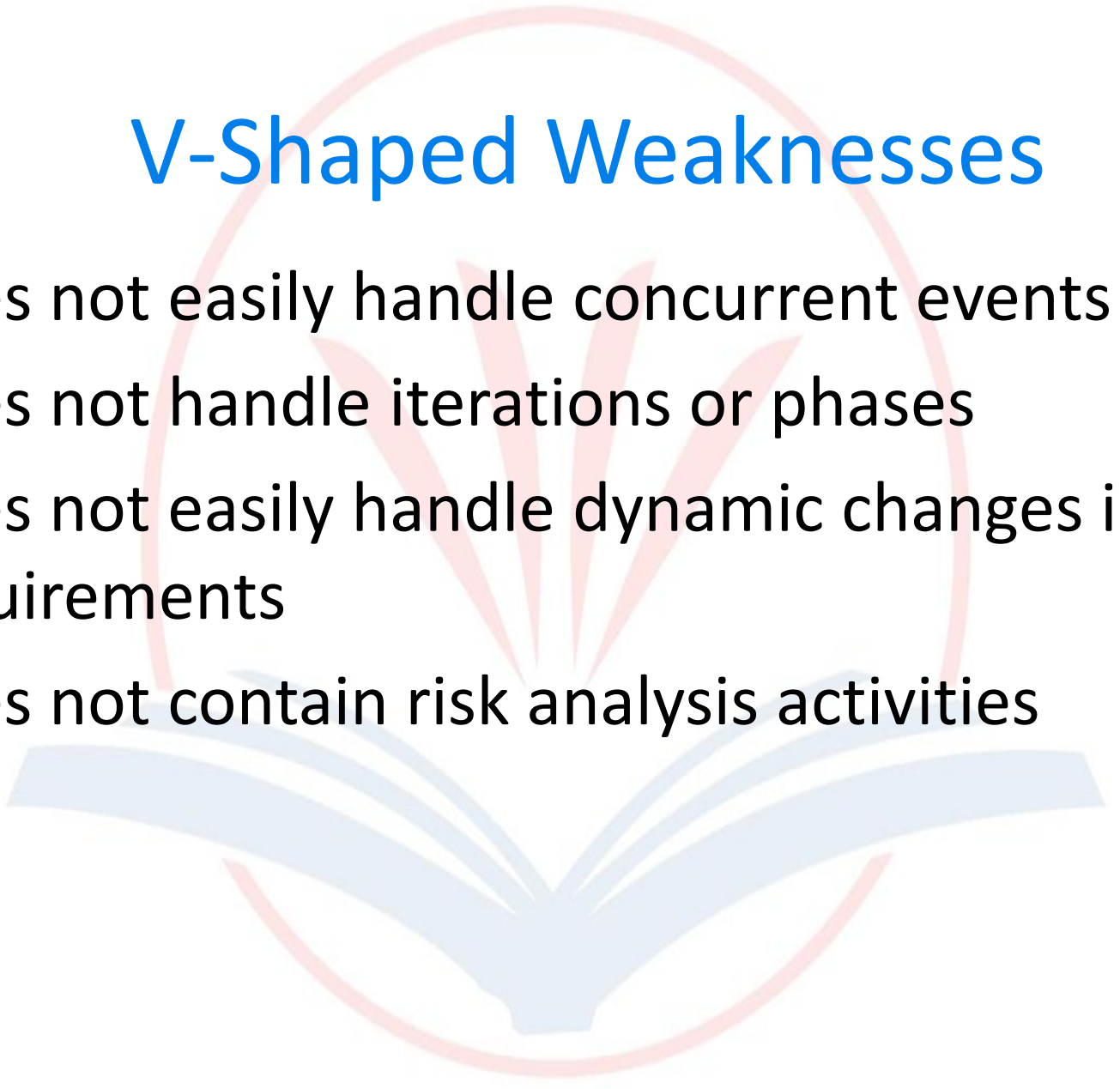


V-Shaped Strengths

- 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
- 

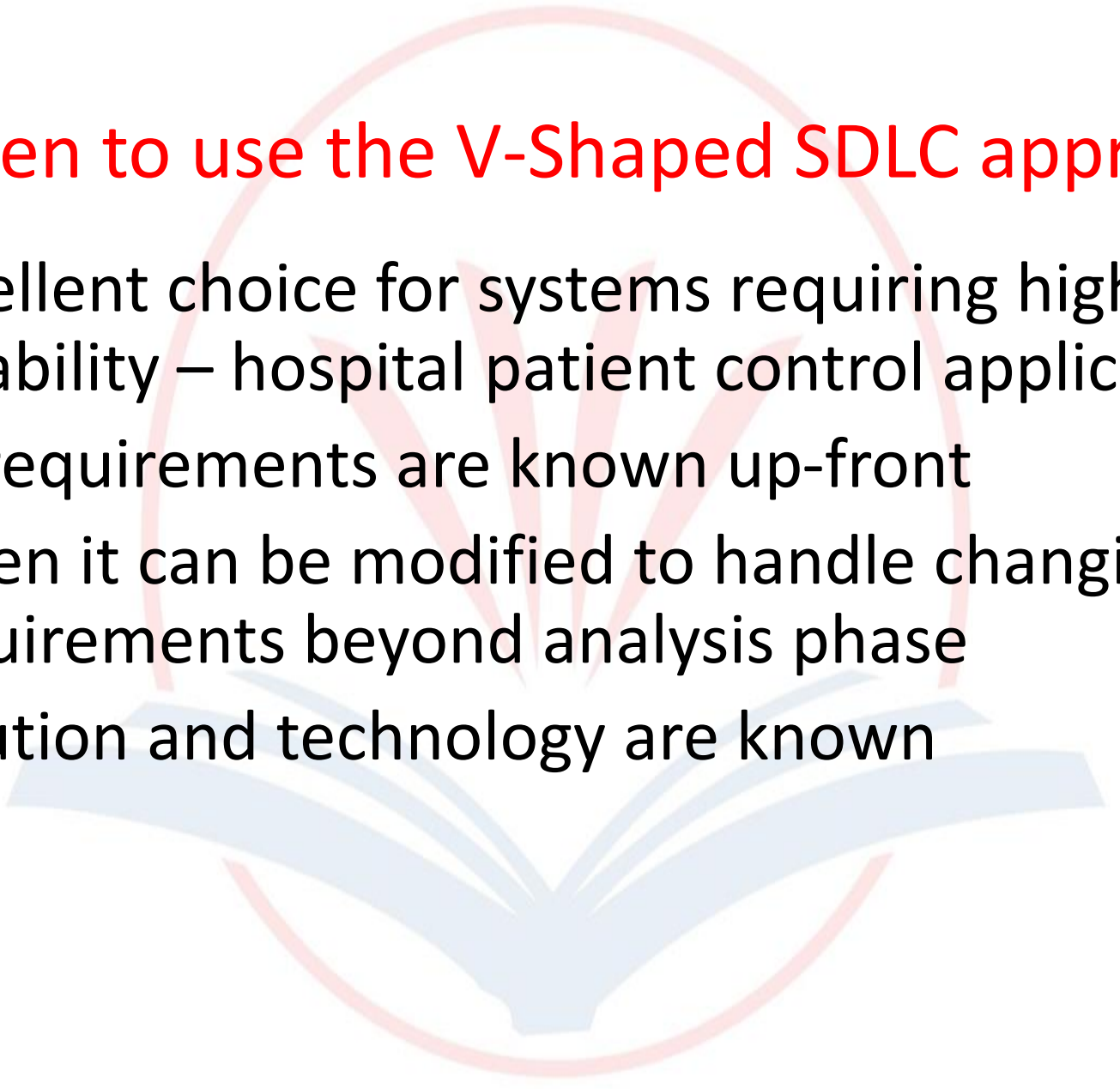


V-Shaped Weaknesses

- 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 SDLC approach

- 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 (SEP)

- 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.



SEP Phases

- A preliminary project plan is developed
- An 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

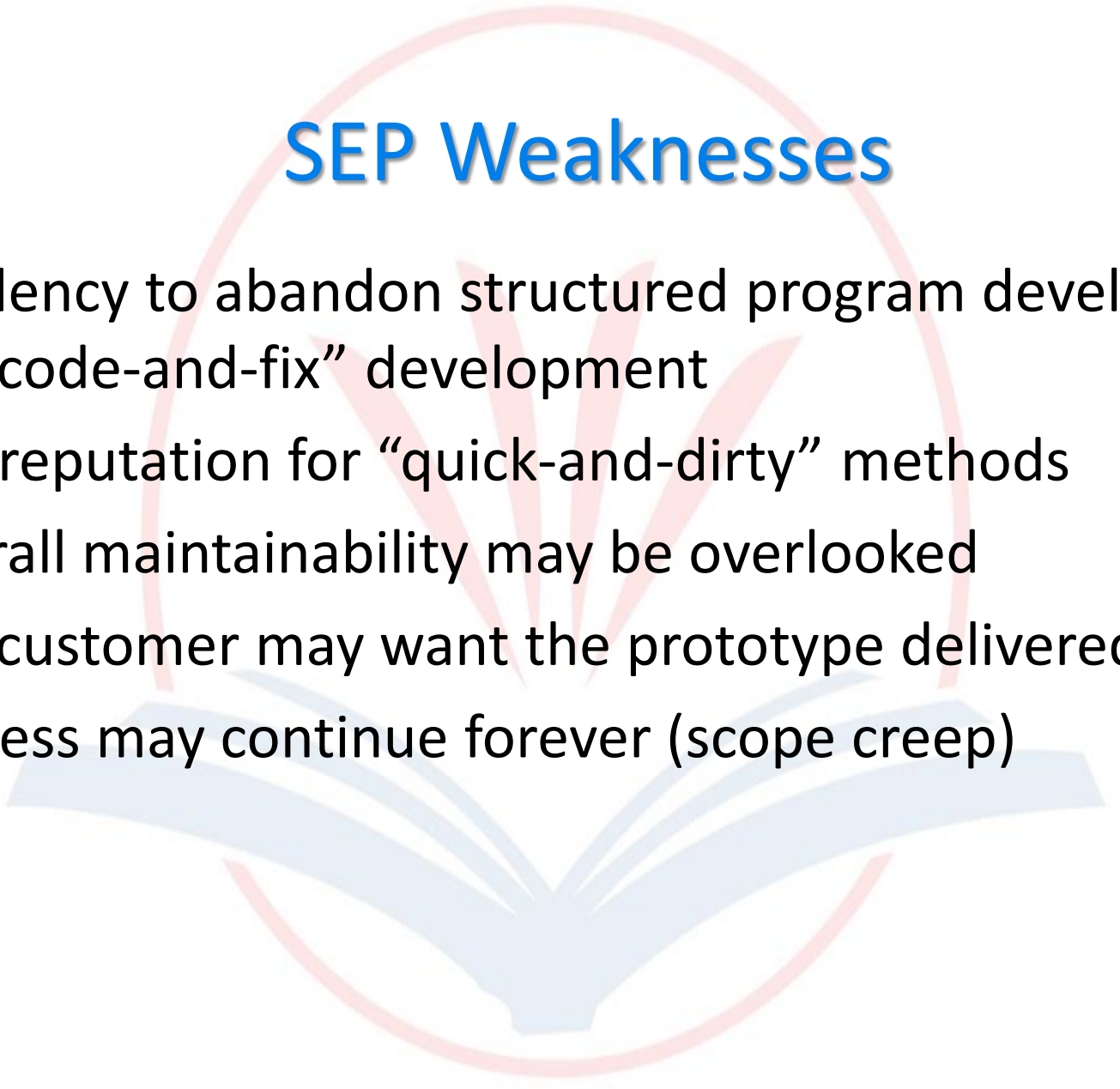


SEP Strengths

- 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



SEP 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 SEP

- 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



RAD Strengths

- 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 (WYSIWYG).
- Uses modeling concepts to capture information about business, data, and processes.

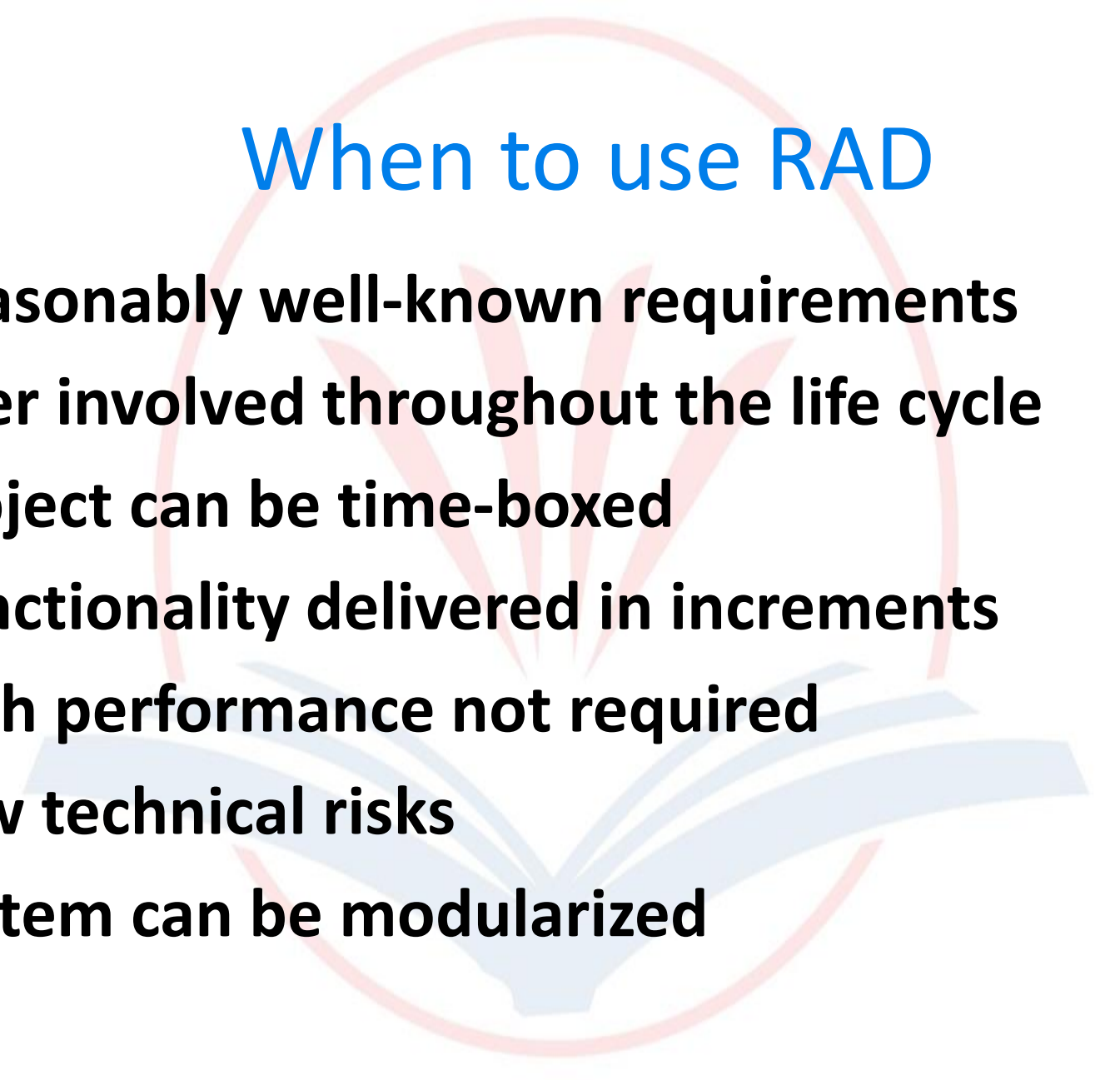


RAD Weaknesses

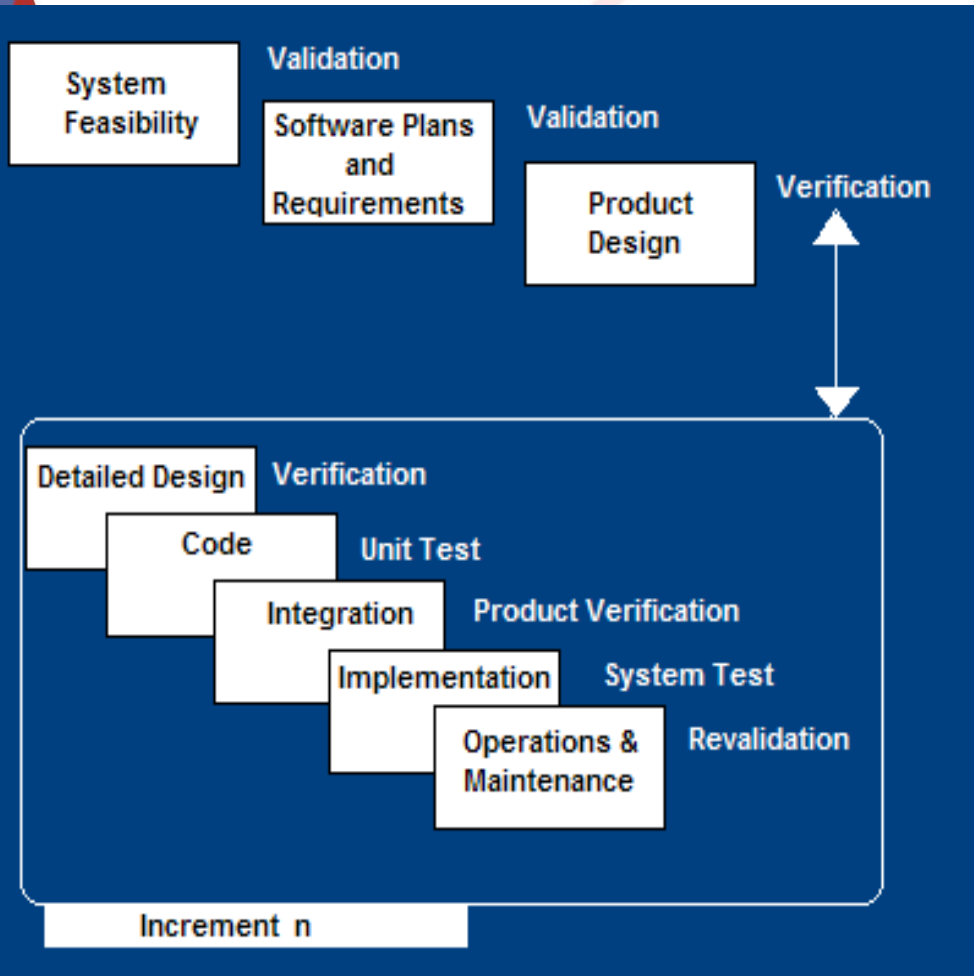
- Accelerated development process must give quick responses to the user
- Risk of never achieving closure
- Hard to use with legacy systems
- Requires a system that can be modularized
- Developers and customers must be committed to rapid-fire activities in an abbreviated time frame.



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.

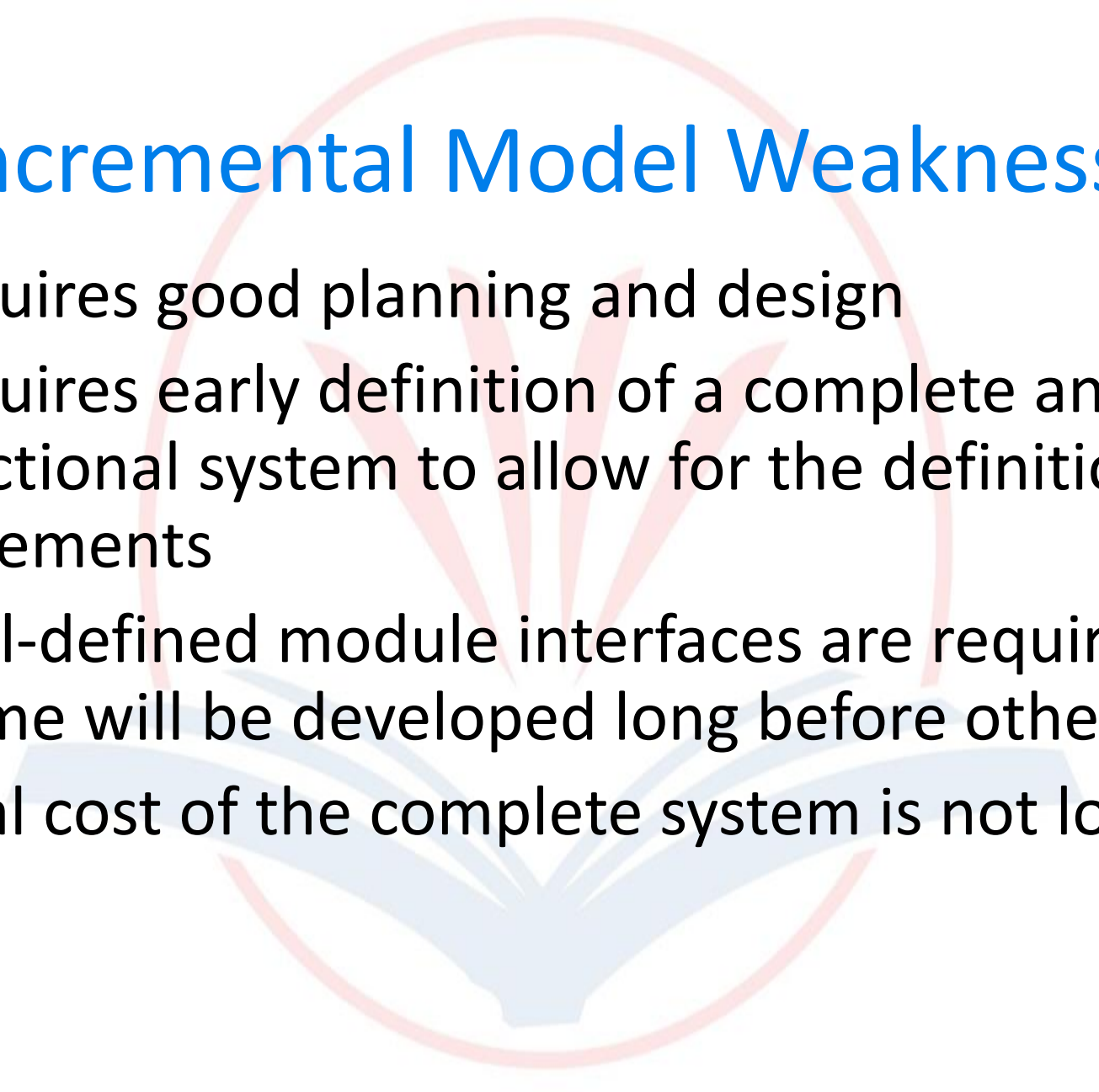


Incremental Model Strengths

- 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

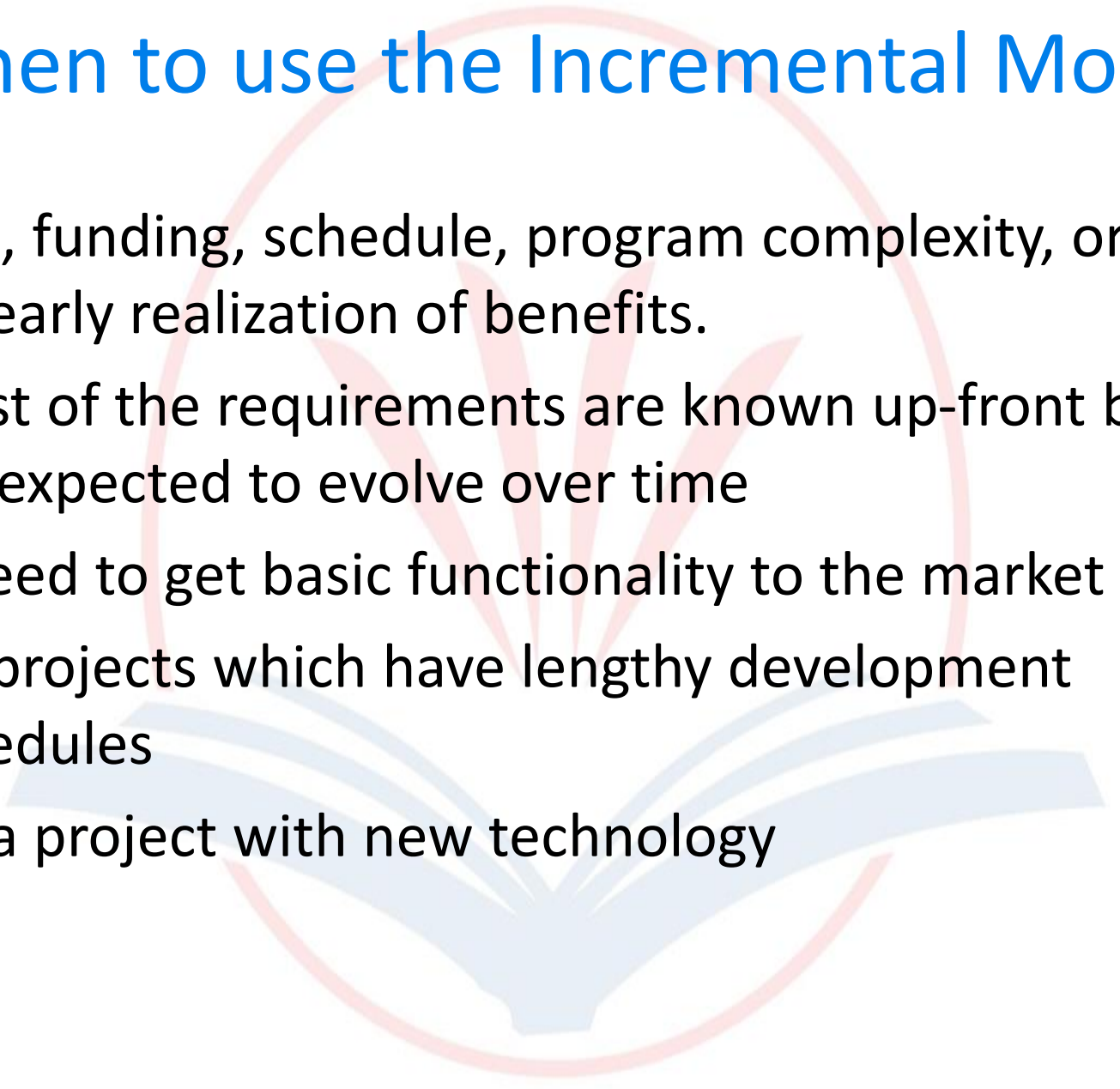


Incremental Model Weaknesses

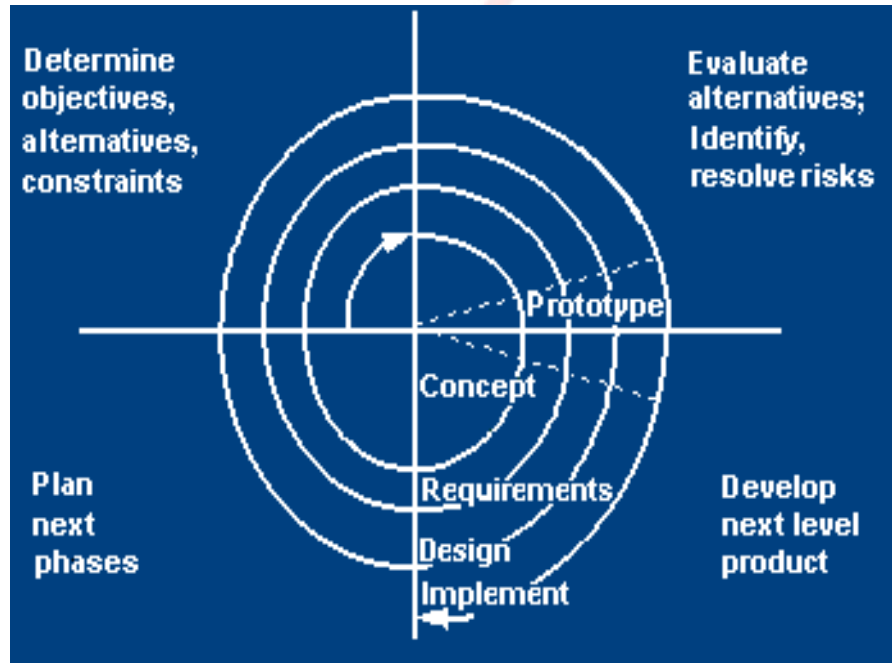
- Requires good planning and design
 - Requires early definition of a complete and fully functional system to allow for the definition of increments
 - Well-defined module interfaces are required (some will be developed long before others)
 - Total cost of the complete system is not lower
- 



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
- 

Spiral SDLC Model

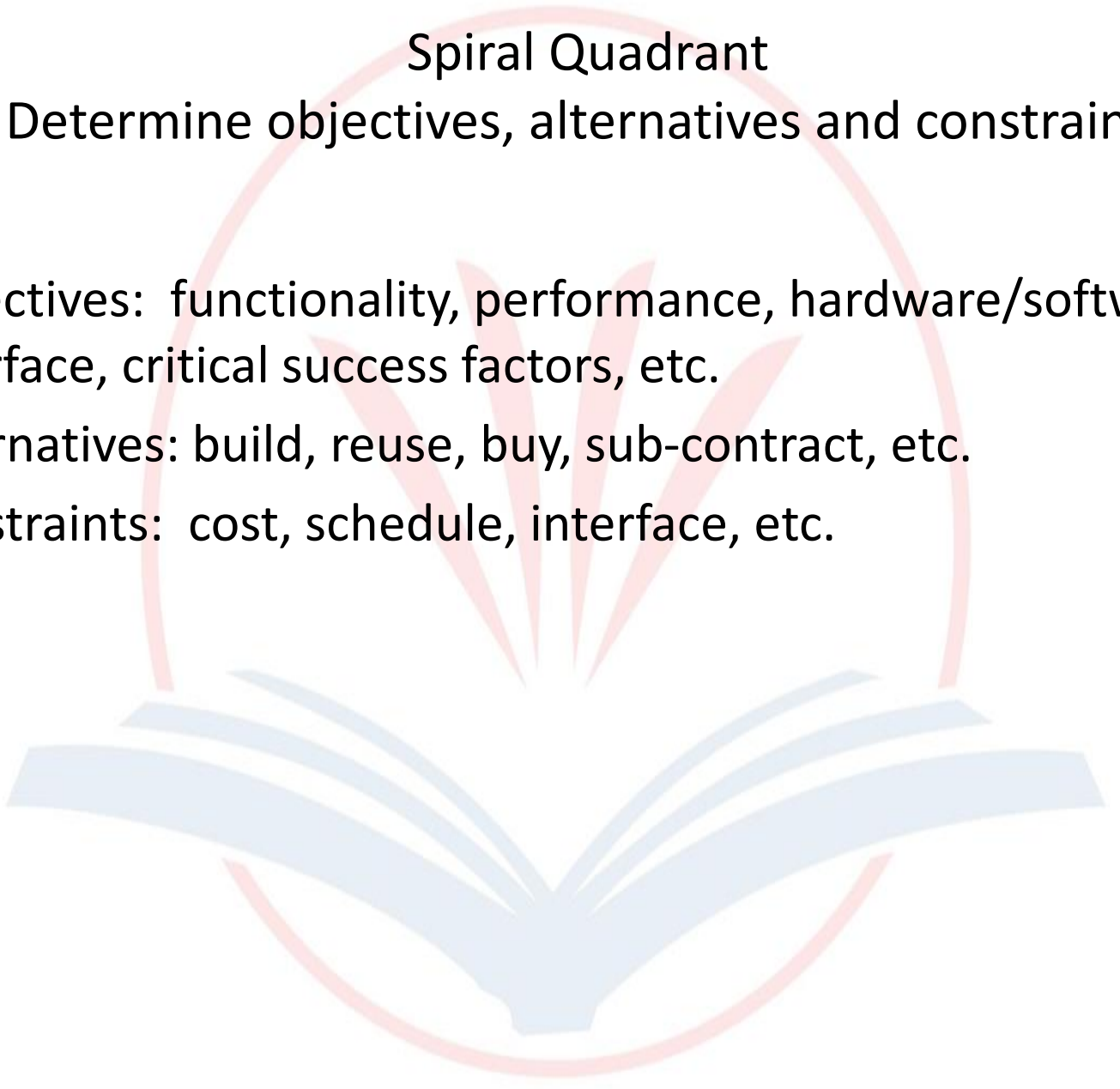


- Adds risk analysis, and 4gl RAD prototyping to the waterfall model
- Each cycle involves the same sequence of steps as the waterfall process model



Spiral Quadrant

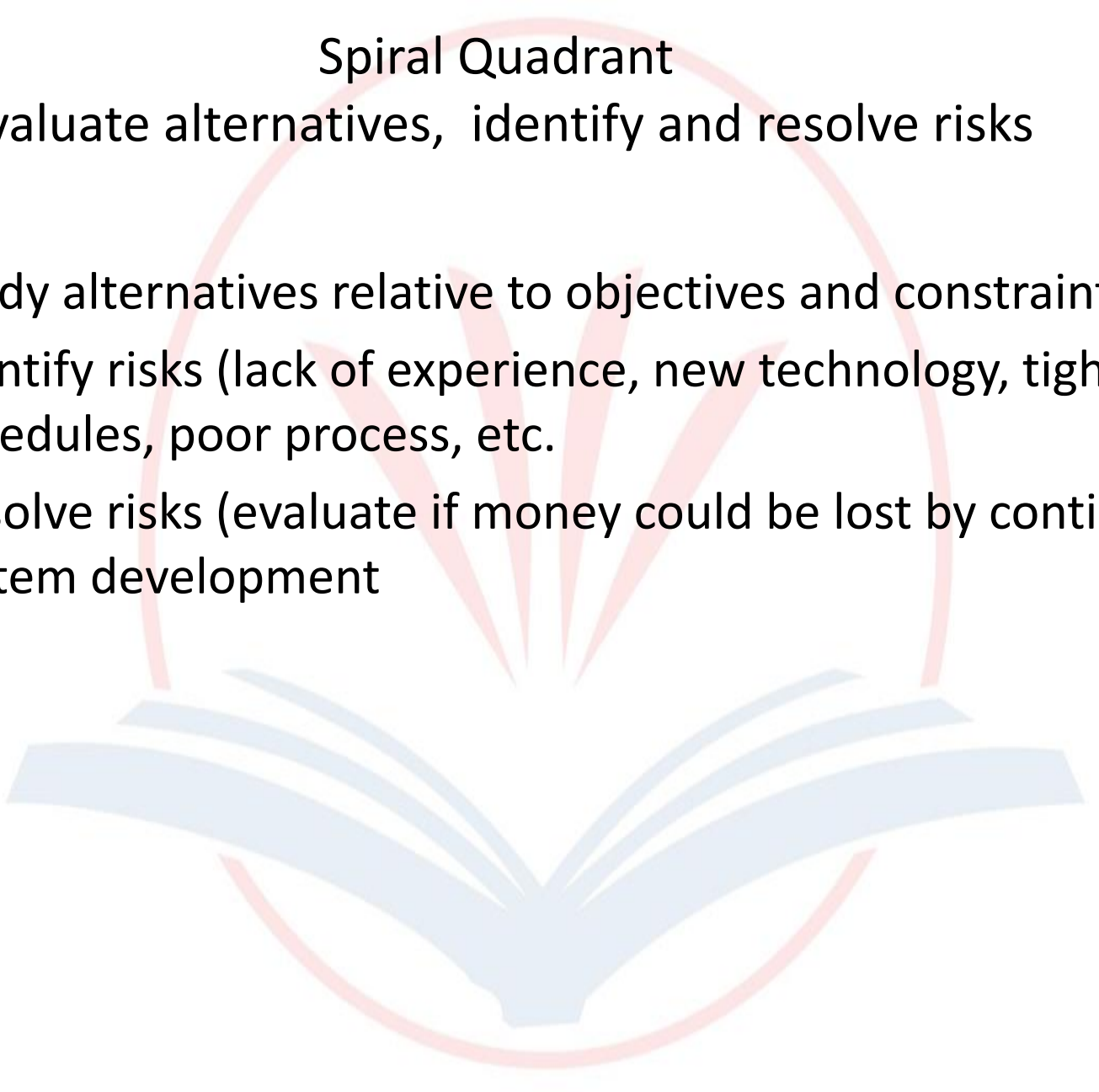
Determine objectives, alternatives and constraints


- 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

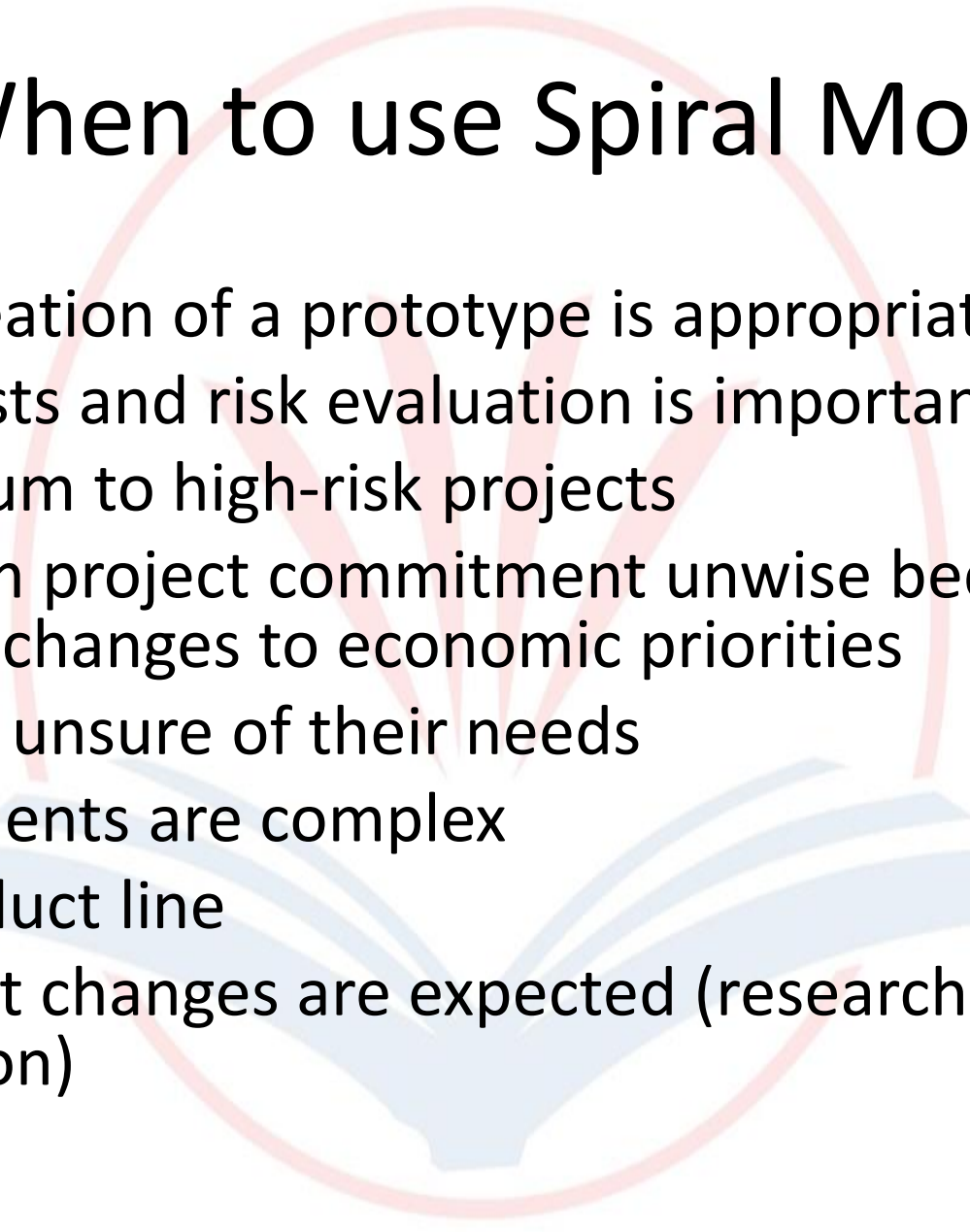


Spiral Model Weaknesses

- 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

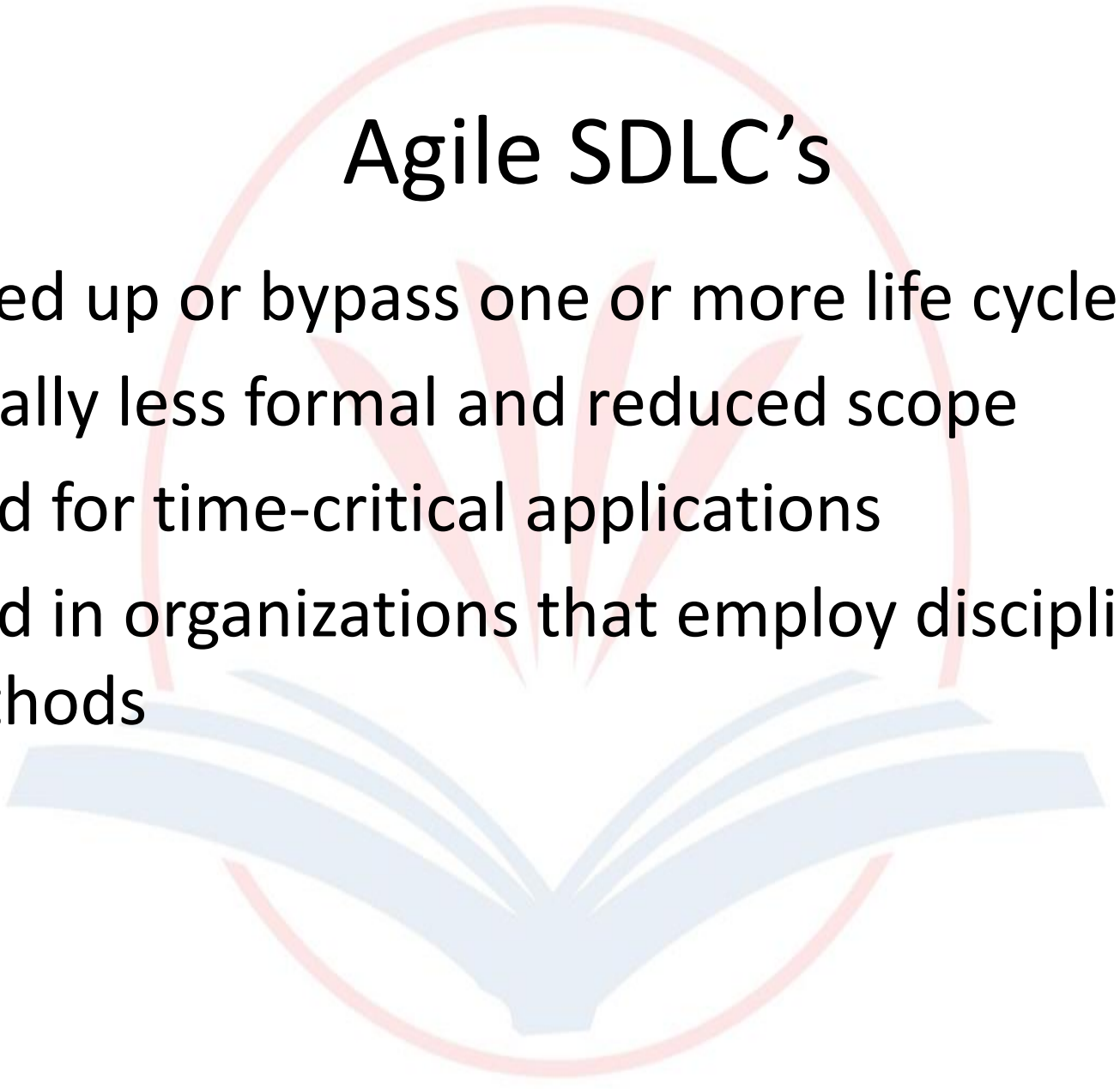


When to use Spiral Model

- 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)
- 



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

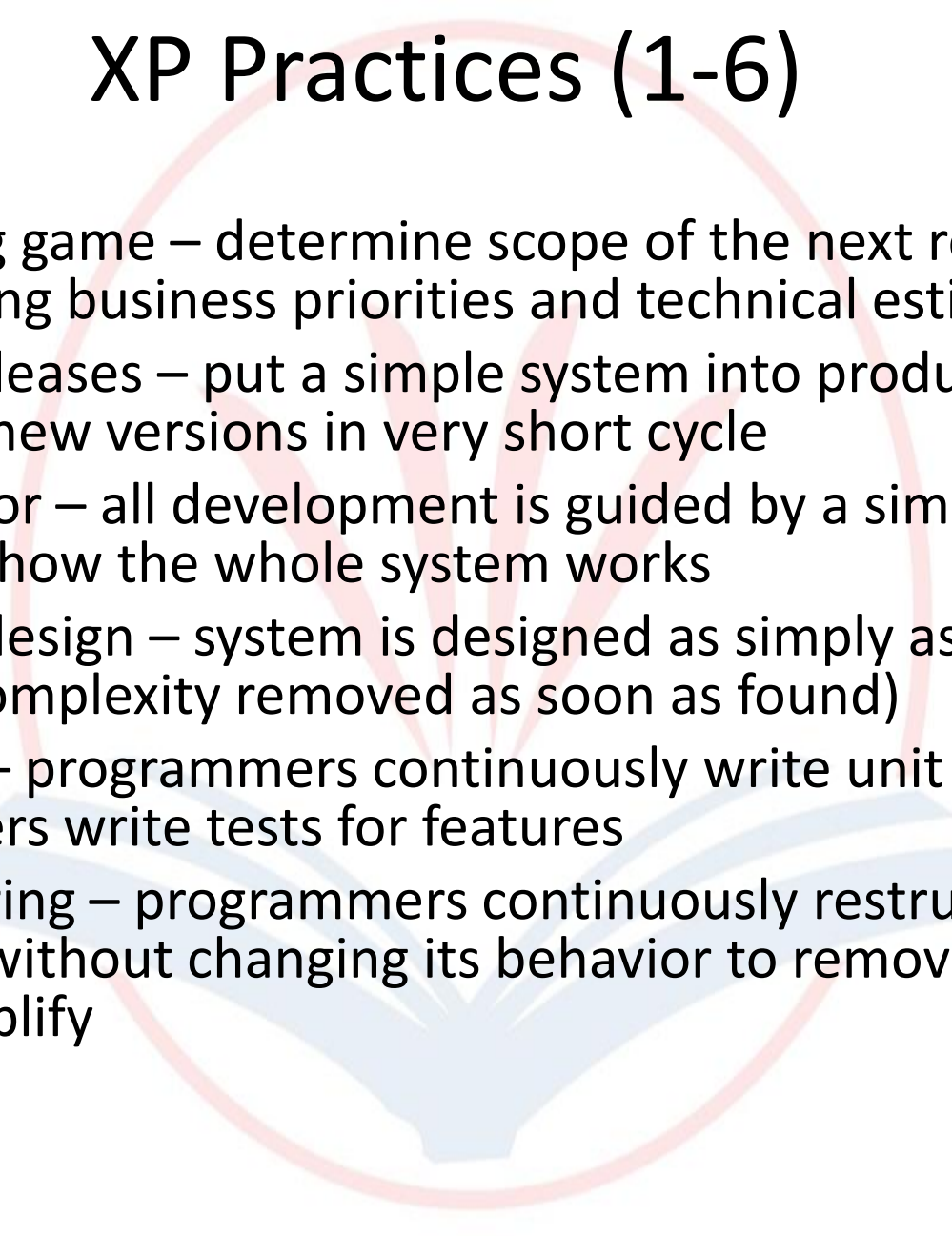
For small-to-medium-sized teams developing software with vague or rapidly changing requirements

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

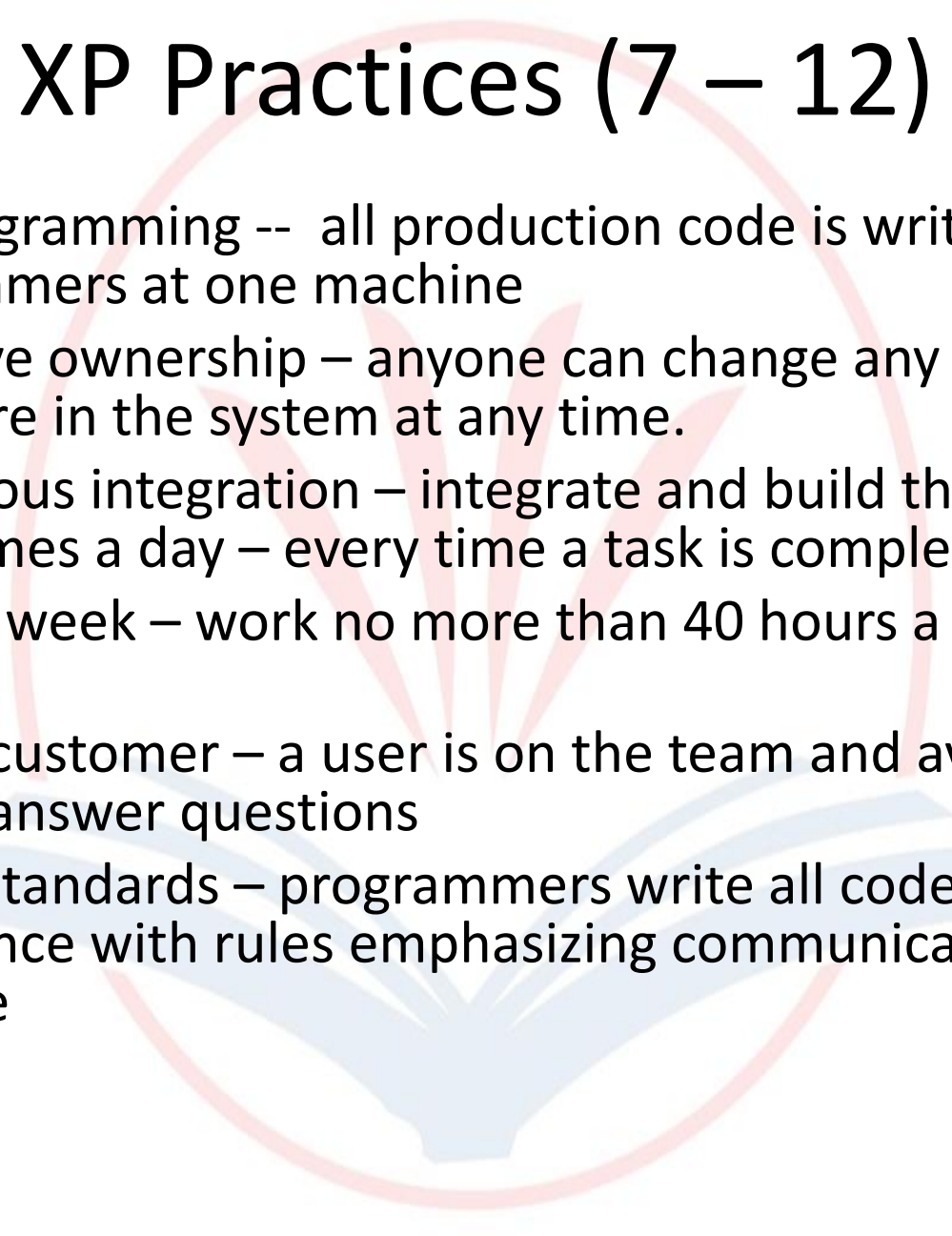


XP Practices (1-6)

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
- 



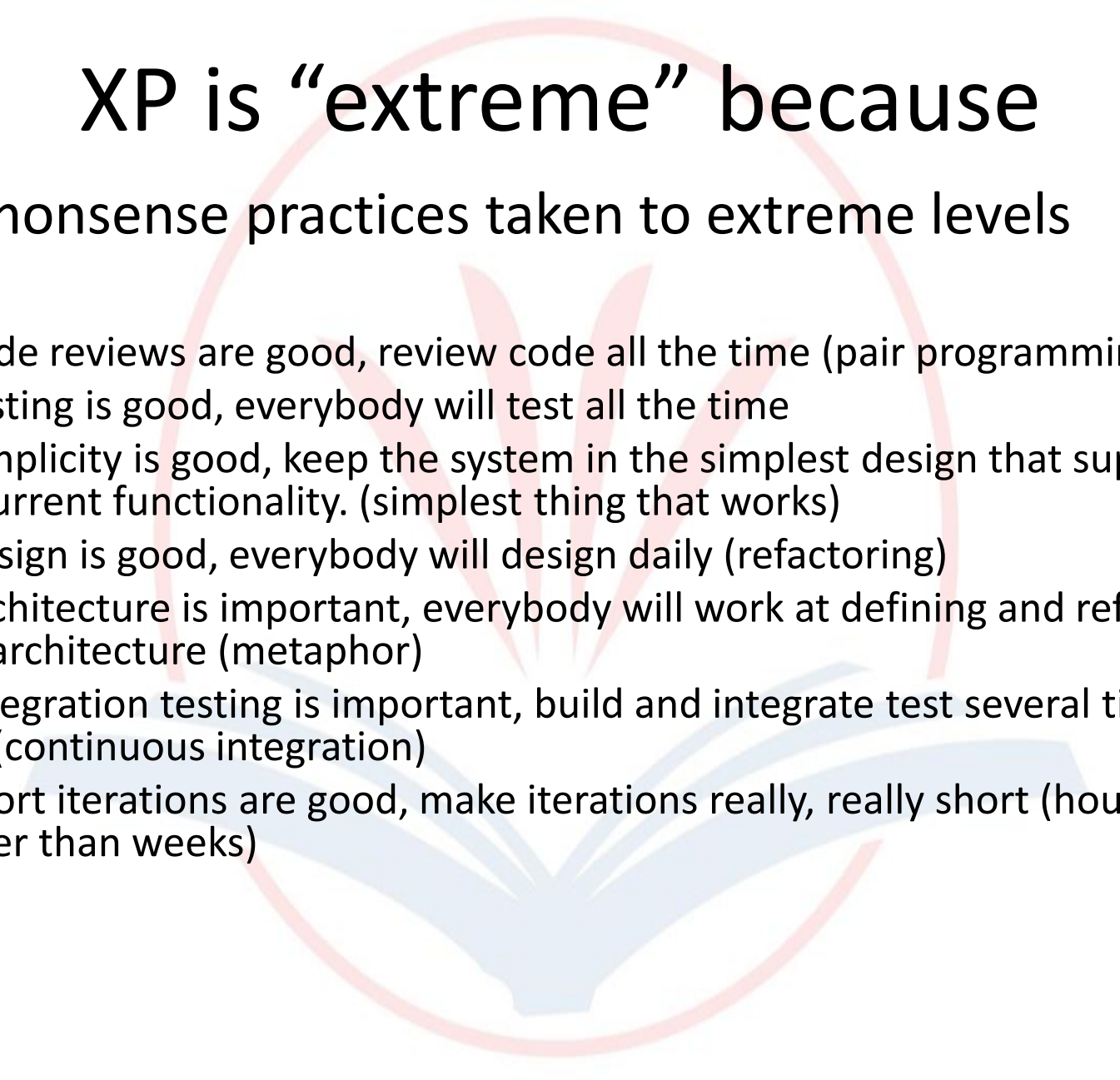
XP Practices (7 – 12)

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

Commonsense 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)
- 



XP References

Online references to XP at

- <http://www.extremeprogramming.org/>
 - <http://c2.com/cgi/wiki?ExtremeProgrammingRoadmap>
 - <http://www.xprogramming.com/>
- 



Feature Driven Design (FDD)

Five FDD process activities

1. Develop an overall model – Produce class and sequence diagrams from chief architect meeting with domain experts and developers.
2. Build a features list – Identify all the features that support requirements. The features are functionally decomposed into Business Activities steps within Subject Areas.

Features are functions that can be developed in two weeks and expressed in client terms with the template: <action> <result> <object>

i.e. Calculate the total of a sale
3. Plan by feature -- the development staff plans the development sequence of features
4. Design by feature -- the team produces sequence diagrams for the selected features
5. Build by feature – the team writes and tests the code

<http://www.nebulon.com/articles/index.html>



Dynamic Systems Development Method (DSDM)

Applies a framework for RAD and short time frames

Paradigm is the 80/20 rule

- majority of the requirements can be delivered in a relatively short amount of time.
- 

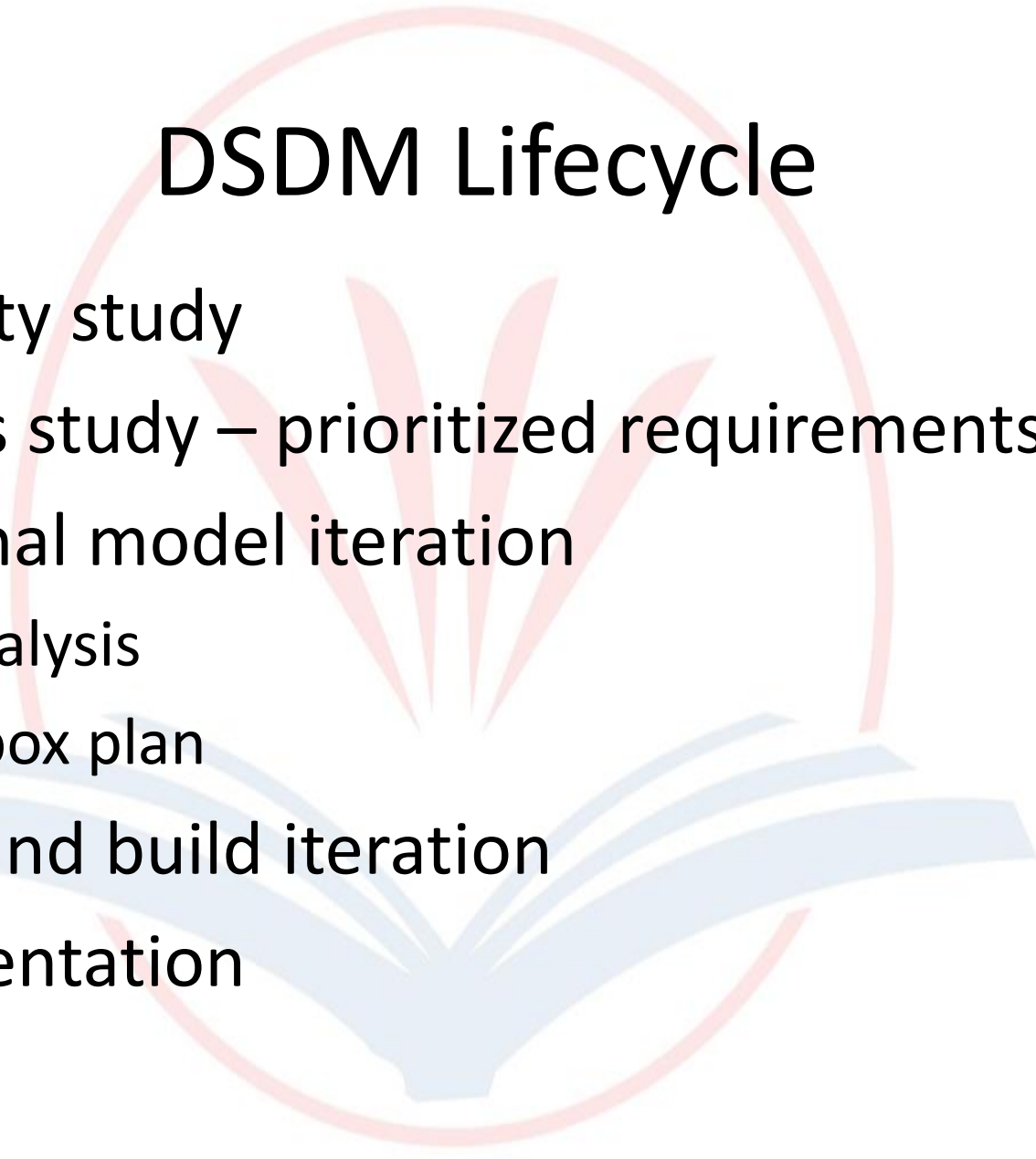


DSDM Principles

1. Active user involvement imperative (Ambassador users)
2. DSDM teams empowered to make decisions
3. Focus on frequent product delivery
4. Product acceptance is fitness for business purpose
5. Iterative and incremental development - to converge on a solution
6. Requirements initially agreed at a high level
7. All changes made during development are reversible
8. Testing is integrated throughout the life cycle
9. Collaborative and co-operative approach among all stakeholders essential



DSDM Lifecycle

- Feasibility study
 - Business study – prioritized requirements
 - Functional model iteration
 - risk analysis
 - Time-box plan
 - Design and build iteration
 - Implementation
- 



Adaptive SDLC

Combines RAD with software engineering best practices

- Project initiation
- Adaptive cycle planning
- Concurrent component engineering
- Quality review
- Final QA and release

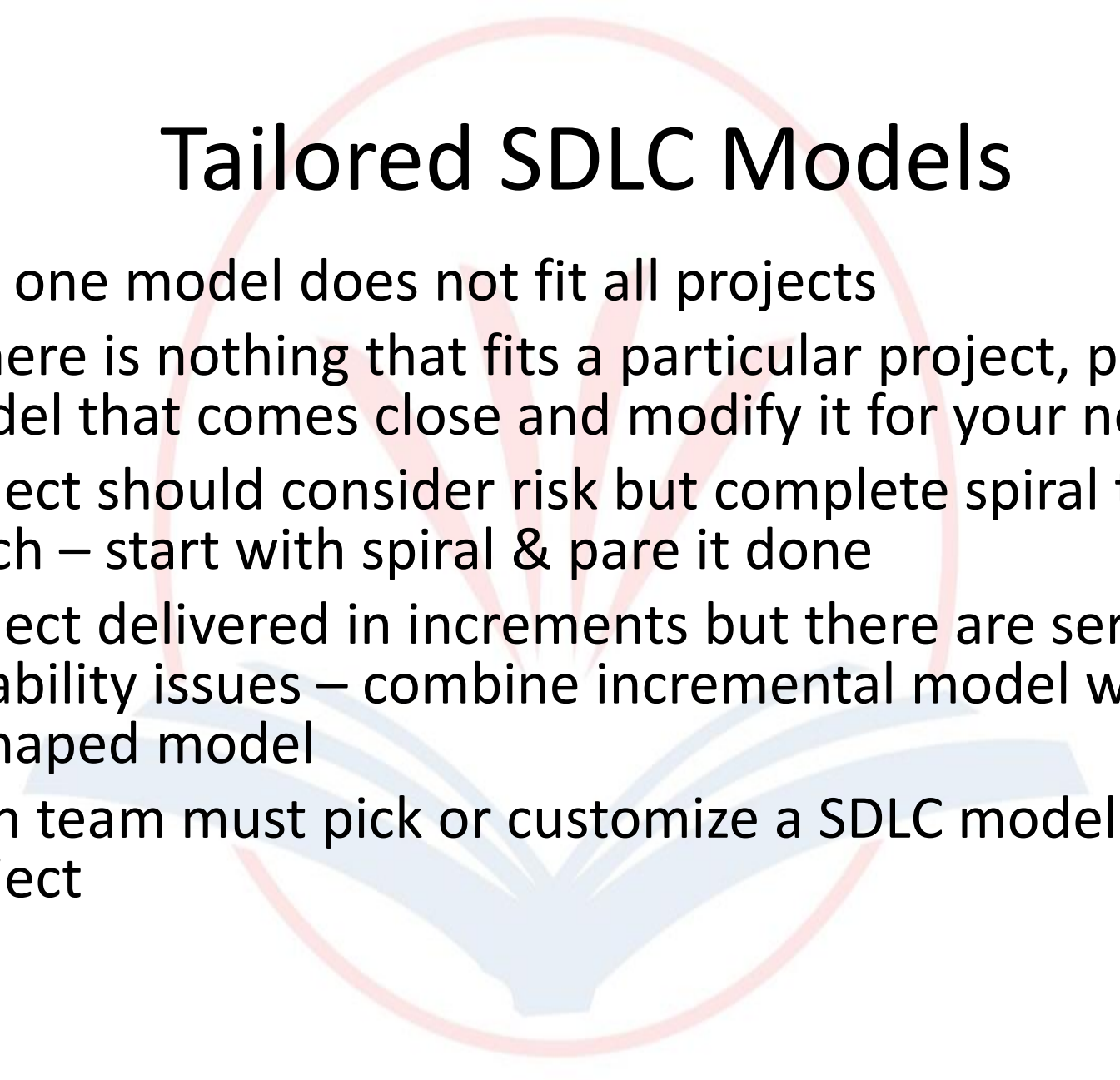


Adaptive Steps

1. Project initialization – determine intent of project
2. Determine the project time-box (estimation duration of the project)
3. Determine the optimal number of cycles and the time-box for each
4. Write an objective statement for each cycle
5. Assign primary components to each cycle
6. Develop a project task list
7. Review the success of a cycle
8. Plan the next cycle



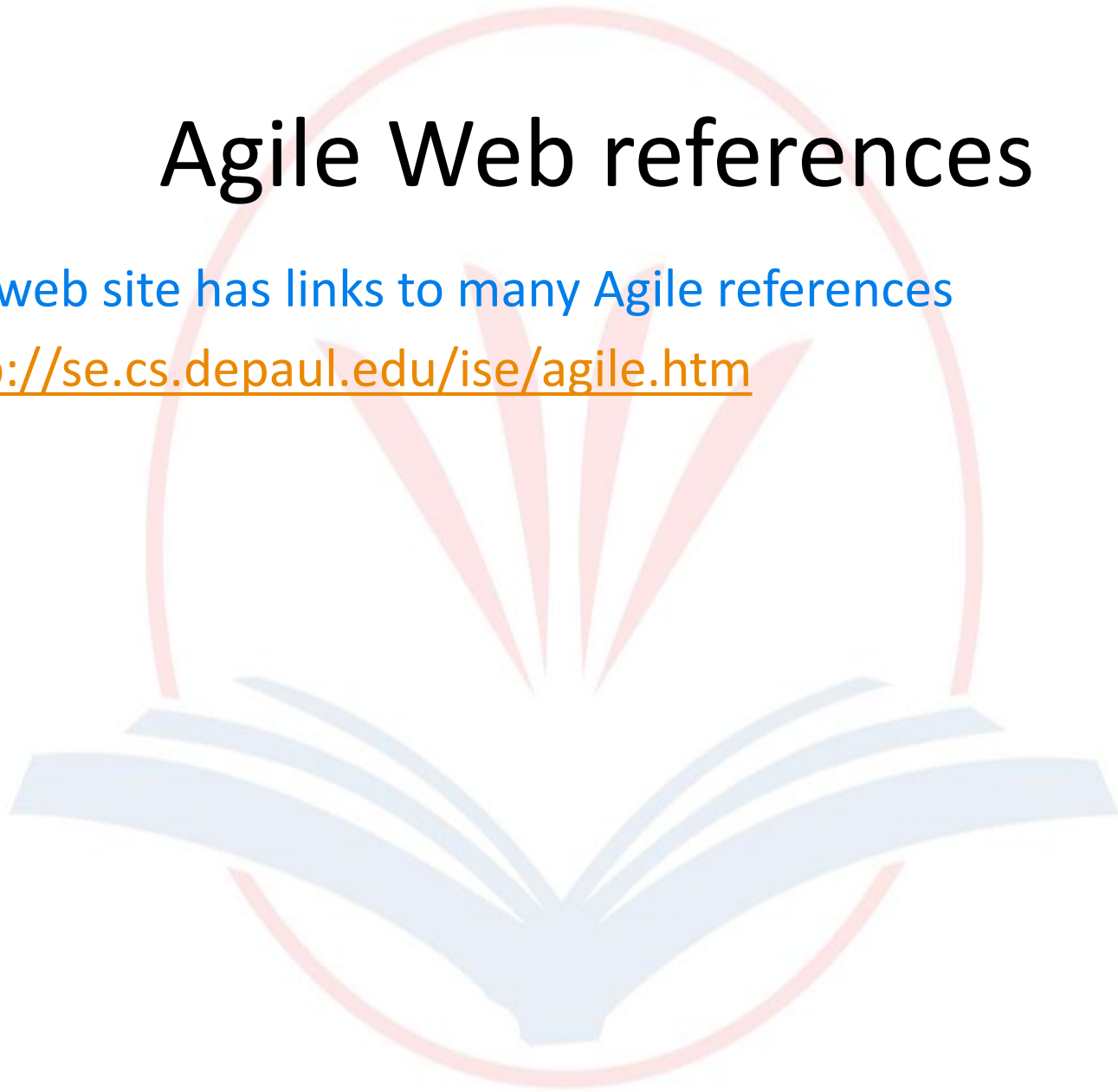
Tailored SDLC Models


- Any one model does not fit all projects
 - If there is nothing that fits a particular project, pick a model that comes close and modify it for your needs.
 - Project should consider risk but complete spiral too much – start with spiral & pare it done
 - Project delivered in increments but there are serious reliability issues – combine incremental model with the V-shaped model
 - Each team must pick or customize a SDLC model to fit its project
- 

Agile Web references

DePaul web site has links to many Agile references

<http://se.cs.depaul.edu/ise/agile.htm>





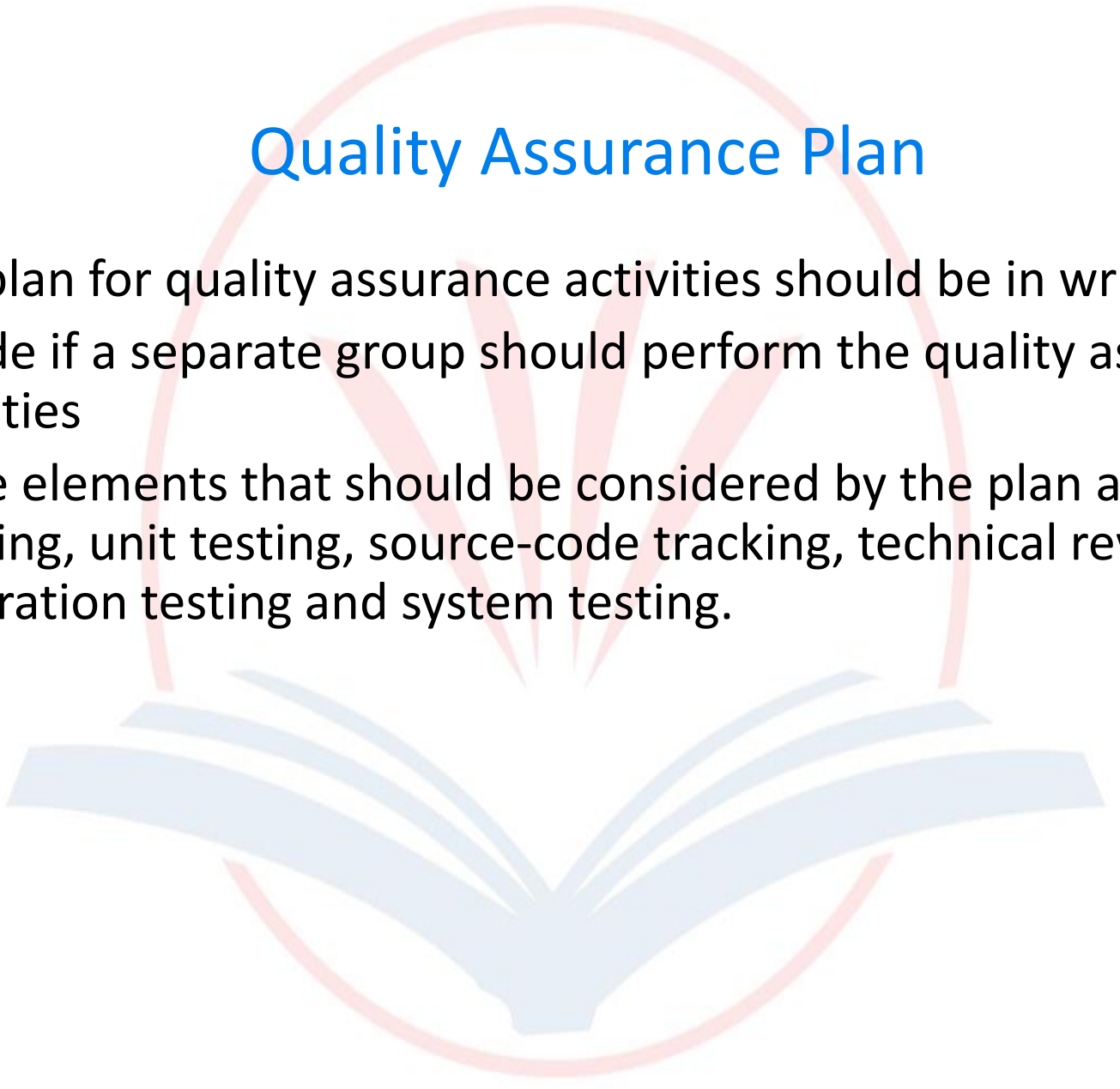
Quality – the degree to which the software satisfies stated and implied requirements

- Absence of system crashes
- Correspondence between the software and the users' expectations
- Performance to specified requirements

Quality must be controlled because it lowers production speed, increases maintenance costs and can adversely affect business



Quality Assurance Plan

- The plan for quality assurance activities should be in writing
 - Decide if a separate group should perform the quality assurance activities
 - Some elements that should be considered by the plan are: defect tracking, unit testing, source-code tracking, technical reviews, integration testing and system testing.
- 



Quality Assurance Plan

- Defect tracing – keeps track of each defect found, its source, when it was detected, when it was resolved, how it was resolved, etc
- Unit testing – each individual module is tested
- Source code tracing – step through source code line by line
- Technical reviews – completed work is reviewed by peers
- Integration testing -- exercise new code in combination with code that already has been integrated
- System testing – execution of the software for the purpose of finding defects.