

Introduction to SE

Mrs. Purvi D. Sankhe



Prescriptive Process Models

Contents

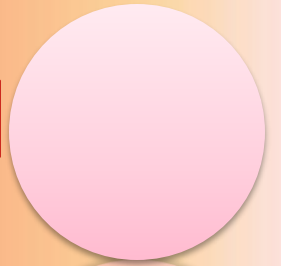
- **Generic process framework (revisited)**
- **Traditional process models**
- **Specialized process models**





Traditional Process Models

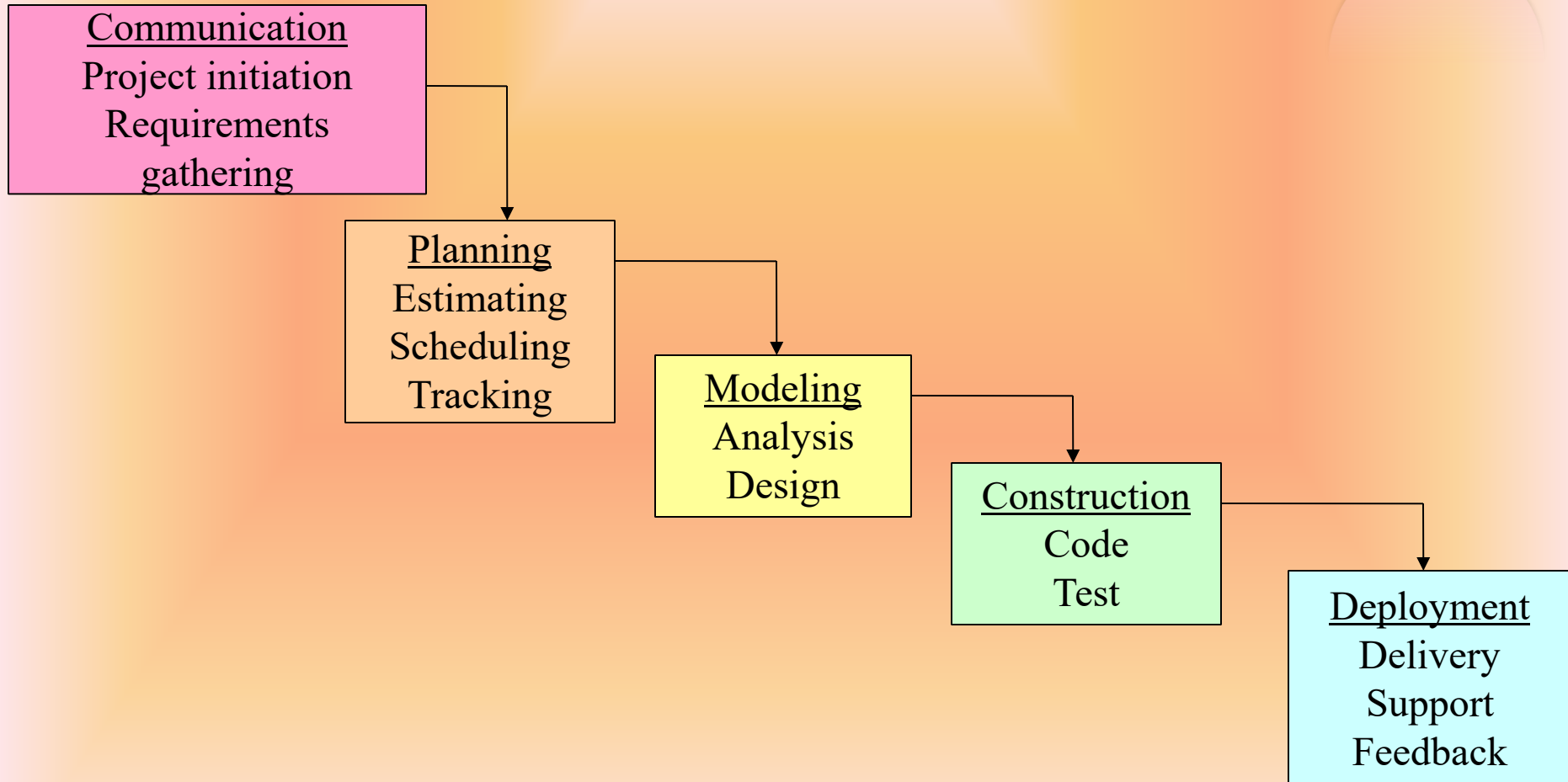
Prescriptive Process Model



- **Defines a distinct set of activities, actions, tasks, milestones, and work products that are required to engineer high-quality software**
- **The activities may be linear, incremental, or evolutionary**

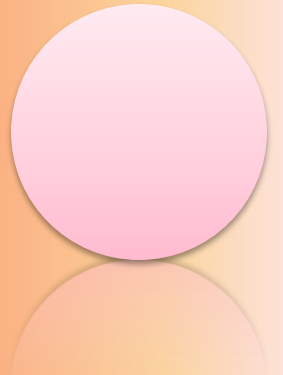
Waterfall Model

(Diagram)



Waterfall Model

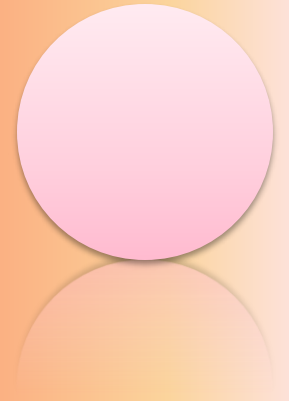
(Description)



- **Oldest software lifecycle model and best understood by upper management**
- **Used when requirements are well understood and risk is low**
- **Work flow is in a linear (i.e., sequential) fashion**
- **Used often with well-defined adaptations or enhancements to current software**

Waterfall Model

(Advantages)



Simplicity and Structure

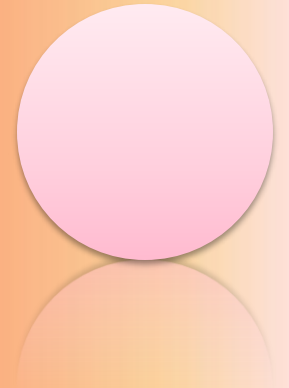
- **The linear and straightforward approach makes it easy to understand and manage.**
- **Example:** A government project (like creating a tax filing system) benefits from Waterfall because requirements are well-documented upfront, and there's minimal scope for change.

Clear Documentation and Milestones

- **Each phase has a definitive endpoint and deliverables, making it easy to track progress.**
- **Example:** Developing a school management system where all features are predefined, ensuring documentation guides development.

Waterfall Model

(Advantages) Contd..



Works Well for Fixed Requirements

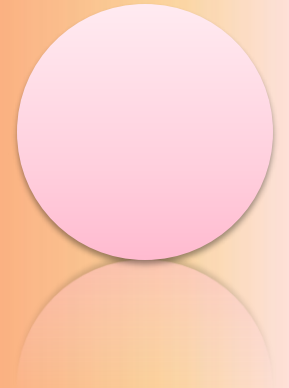
- **Suitable when the project's requirements are well-known and unlikely to change.**
- **Example:** Building a payroll management system, where calculation methods and tax rules are fixed during the development.

Easy to Test and Maintain

- **Testing occurs after the development phase is complete, ensuring all features are built before validation.**
- **Example:** Legacy system upgrades for a bank (like transitioning from COBOL to a modern language) where each phase is carefully tested before moving on.

Waterfall Model

(Advantages) Contd..



Works Well for Fixed Requirements

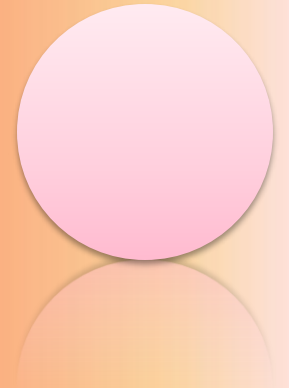
- **Suitable when the project's requirements are well-known and unlikely to change.**
- **Example:** Building a payroll management system, where calculation methods and tax rules are fixed during the development.

Easy to Test and Maintain

- **Testing occurs after the development phase is complete, ensuring all features are built before validation.**
- **Example:** Legacy system upgrades for a bank (like transitioning from COBOL to a modern language) where each phase is carefully tested before moving on.

Waterfall Model

(Disadvantages)



Lack of Flexibility

- Any changes in requirements after the development has started can be expensive and difficult to implement.

Example: In an e-commerce platform project, if the client later requests integration with social media (like Instagram), reworking may lead to delays and cost overruns.

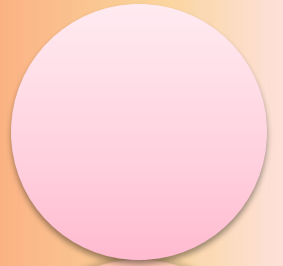
Late Discovery of Issues

- Testing happens after the development phase, leading to the late detection of errors that could require significant rework.

Example: A hospital management system where user feedback reveals that the user interface is unintuitive, requiring major revisions post-development.

Waterfall Model

(Disadvantages) contd..



High Risk with Unclear Requirements

- **If requirements are not well-understood, the project risks failure because there's little room for feedback during development.**
- **Example:** A startup mobile app project where market trends change frequently, and Waterfall doesn't allow iterative revisions.

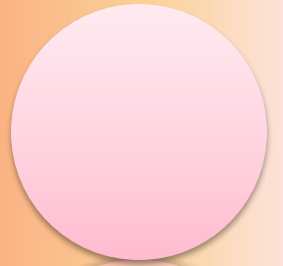
Customer Involvement is Limited

- **Customers only see the final product, and if it doesn't meet expectations, revisiting previous phases can be costly.**
- **Example:** Developing a custom CRM system where the client isn't involved during design or development phases and is dissatisfied with the final outcome.

Comparison with Real-Time Scenarios

Scenario	Why Waterfall Works	Why Waterfall Fails
Developing a library database system	Requirements are well-defined upfront.	Changes in book categorization policies mid-way.
Implementing an ERP for a small firm	Fixed business processes; easier to document.	If the firm expands or requires customizations.
Designing a government portal	High emphasis on documentation and traceability.	Delays in response to user feedback.

Waterfall Model (Example)



Civil Engineering and Construction Projects

Example: Building a bridge, dam, or highway.

- **Why Waterfall Works:** These projects require detailed planning, including blueprints, resource allocation, and timelines. Changes during construction can be expensive and difficult to implement.
- **Steps:**
 - **Requirements:** Survey and approval of the design.
 - **Design:** Finalize blueprints.
 - **Implementation:** Construction begins.
 - **Testing:** Safety and structural integrity checks.
 - **Maintenance:** Ongoing upkeep after completion.

Waterfall Model (Example) contd..

Defense and Aerospace Projects

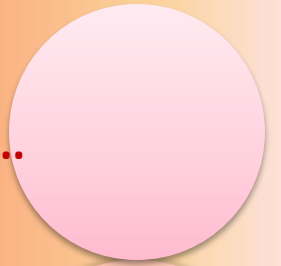
Example: Missile control system or aircraft navigation system.

Why Waterfall Works: These projects demand high precision and adherence to predefined requirements and safety standards.

Steps:

- Requirements: Define specifications for guidance systems.
- Design: Develop schematics and software.
- Implementation: Build hardware and software components.
- Testing: Perform simulations and real-world tests.
- Maintenance: Regular system upgrades.

Waterfall Model (Example) contd..



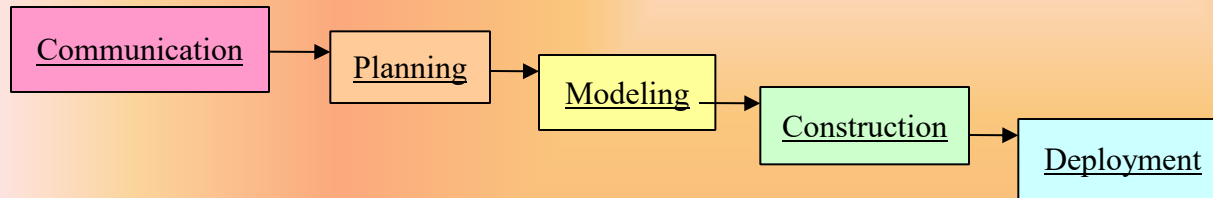
- **Government Projects.**
- **Banking and Financial Sector**
- **Embedded Systems Development**
- **Manufacturing Projects**
- **Educational Software Development**
- **Document Management Systems**

Incremental Model

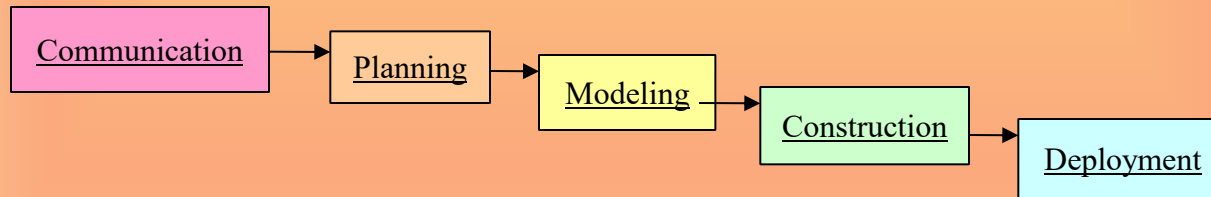
(Diagram)



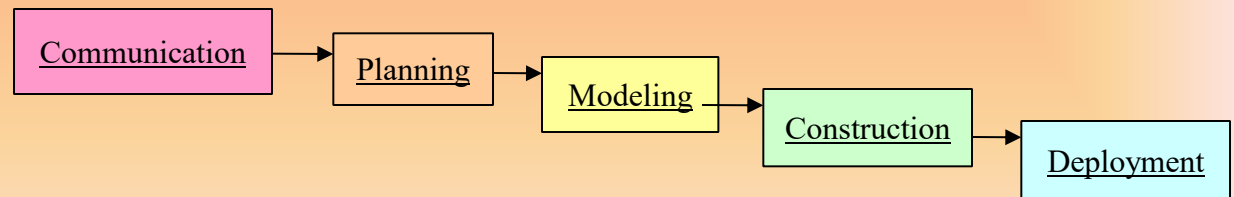
Increment #1



Increment #2

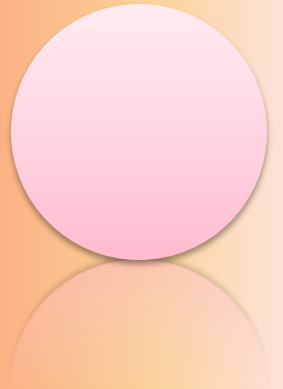


Increment #3



Incremental Model

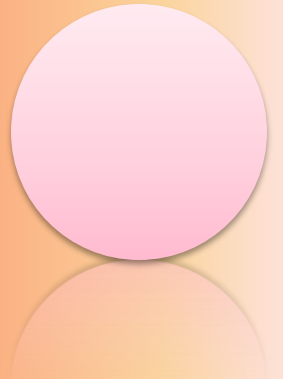
(Description)



- **Used when requirements are well understood**
- **Multiple independent deliveries are identified**
- **Work flow is in a linear (i.e., sequential) fashion within an increment and is staggered between increments**
- **Iterative in nature; focuses on an operational product with each increment**
- **Provides a needed set of functionality sooner while delivering optional components later**
- **Useful also when staffing is too short for a full-scale development**

Incremental Model

(Advantages)



Early Delivery of Functional Software

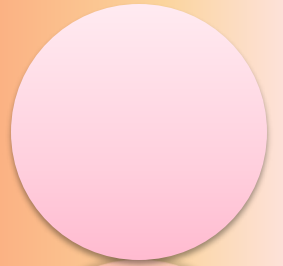
- Functional modules are delivered early, allowing clients to use and provide feedback even before the full system is completed.
- **Example:** A banking application where basic features like account creation and balance checking are delivered first, followed by loan management and fund transfers.

Flexibility to Change Requirements

- Changes in requirements can be incorporated into future increments without major rework.
- **Example:** A retail management system where customer loyalty features are added after initial feedback.

Incremental Model

(Advantages) contd..



Easier Testing and Debugging

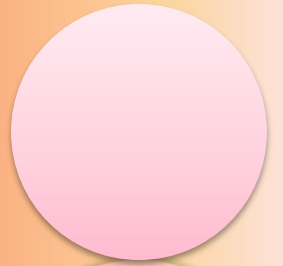
- Testing is performed incrementally for each module, making it easier to identify and fix issues.
- **Example:** An e-commerce platform where payment integration can be tested independently before adding inventory management.

Risk Management

- Risk is reduced as critical functionalities are delivered early, and feedback can guide subsequent increments.
- **Example:** A healthcare app where essential patient record modules are delivered first to ensure compliance before developing appointment scheduling.

Incremental Model

(Advantages) contd..



Resource Allocation is Easier

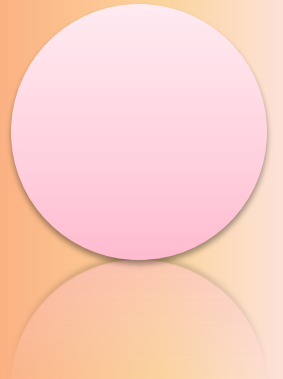
- Resources can be better managed and distributed as work is divided into smaller chunks.
- **Example:** A university portal where features like exam registration, results, and attendance tracking are developed one at a time.

Customer Satisfaction

- Continuous delivery and early access to features keep customers engaged and satisfied.
- **Example:** A news app where the basic browsing functionality is delivered first, with advanced personalization features coming later.

Incremental Model

(Disadvantage)



Requires Proper Planning

- A clear and detailed plan is essential to identify and prioritize increments, which can be time-consuming.
- **Example:** A city transport management system where improper planning could delay essential features like ticketing.

Higher Total Cost

- Frequent deliveries and testing can increase the overall development cost compared to building the system in one go.
- **Example:** A CRM system where continuous iterations incur additional costs for deployment and support.

Incremental Model

(Disadvantage) contd..



Integration Challenges

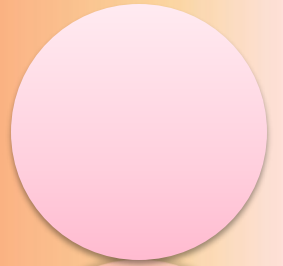
- Combining increments and ensuring compatibility between modules can be complex.
- **Example:** A warehouse management system where newly added inventory modules need to integrate seamlessly with existing order management features.

Incomplete System at Early Stages

- Customers may find early increments lacking essential features, leading to dissatisfaction.
- **Example:** A hotel booking system where early versions only allow searching for hotels but not booking.

Incremental Model

(Disadvantage) contd..



Dependency Between Increments

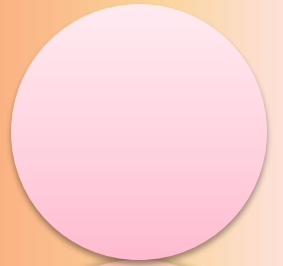
- If increments depend on earlier ones, delays in one module can cascade into others.
- **Example:** A video streaming platform where the recommendation engine relies on completed user data modules.

Requires Continuous Feedback

- Success depends on timely and consistent feedback from stakeholders, which can sometimes be challenging.
- **Example:** A fitness tracking app where user input on usability is critical for refining future features.

Incremental Model

(Example)

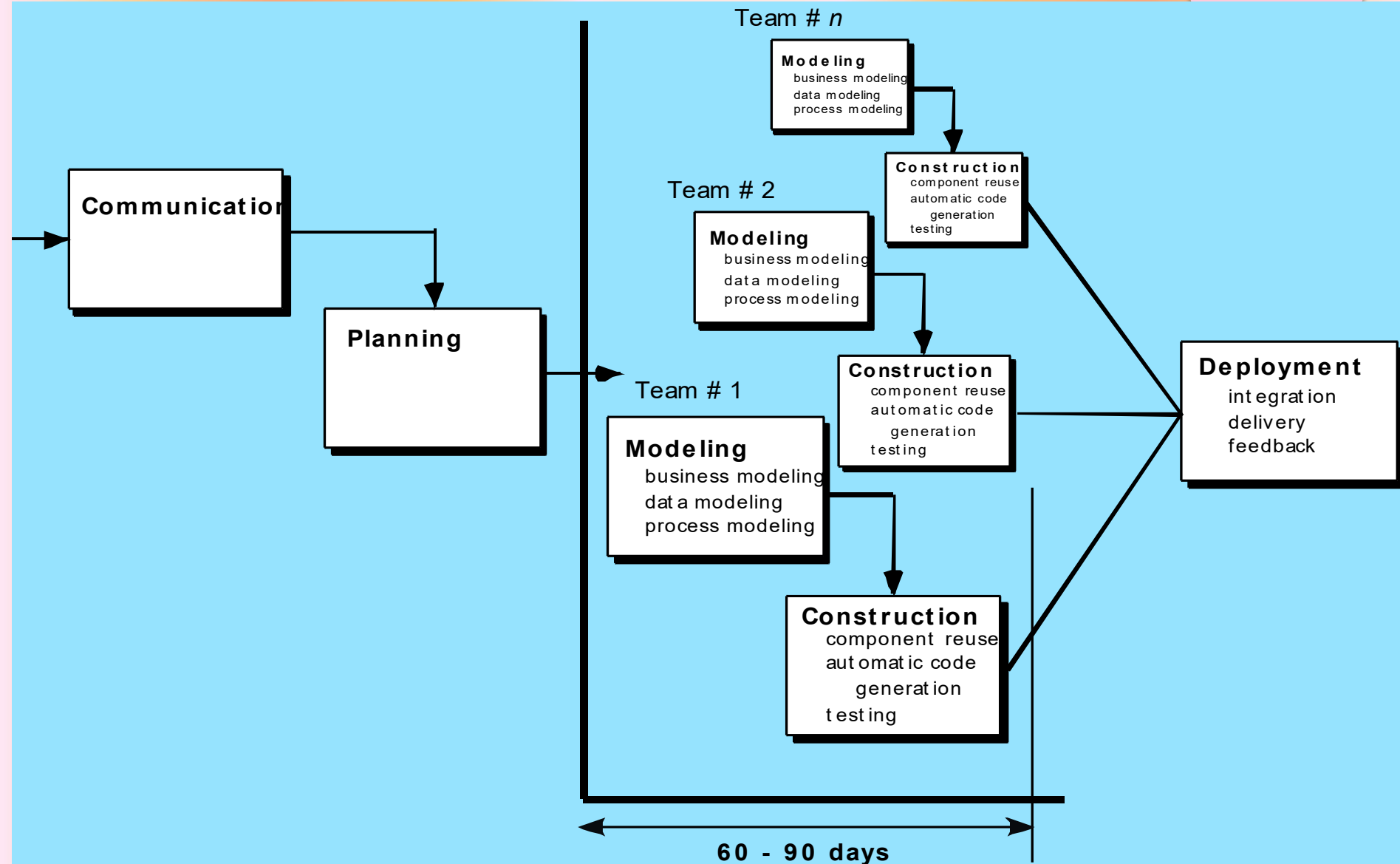


- Each linear sequence produces deliverable “increments” of the software.
- Ex: a Word Processor delivers basic file mgmt., editing, in the first increment;
- more sophisticated editing, document production capabilities in the 2nd increment;
- spelling and grammar checking in the 3rd increment.

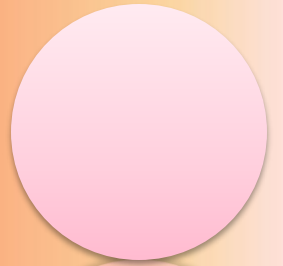
When to Use the Incremental Model

- When requirements are clear but might evolve over time.
- When early delivery of functional components is important.
- For projects requiring frequent user feedback and involvement.
- When there is a need to mitigate risks by developing high-priority modules first.

RAD model



RAD model



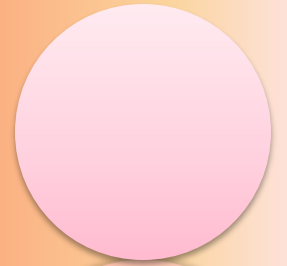
- **is an incremental software development process model**
- **“high-speed” adaptation of the linear sequential model**
- **If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a “fully functional system” within very short time periods**

RAD model



- RAD approach encompasses the following phases :
- **Business modeling.** The information flow among business functions is modeled in a way that answers the following questions:
 - What information drives the business process? What information is generated? Who generates it?
Where does the information go? Who processes it?
- **Data modeling.** The information flow defined as part of the business modeling phase is refined into a set of data objects
- **Process modeling.** The data objects defined in the data modeling phase are transformed to achieve the information flow
- **Application generation.** RAD assumes the use of fourth generation techniques
- **Testing and turnover.** Since the RAD process emphasizes reuse, many of the program components have already been tested.

RAD model Examples



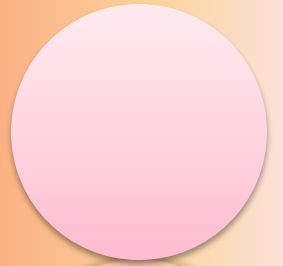
E-commerce Website Development

- A retailer needs to launch an e-commerce platform quickly for the upcoming holiday season.
- **RAD Application:** Using tools like Shopify or Magento, a prototype of the online store is developed within days. Feedback from stakeholders is incorporated rapidly to adjust features like the product catalog, payment gateway integration, and user interface.

Mobile App Development

- A startup needs to validate an idea for a food delivery app.
- **RAD Application:** Using frameworks like Flutter, React Native, or low-code platforms (e.g., Mendix or OutSystems), developers create a functional prototype with basic features (e.g., browsing restaurants, placing orders). Feedback from users helps refine the app before the final release.

RAD model Advantages



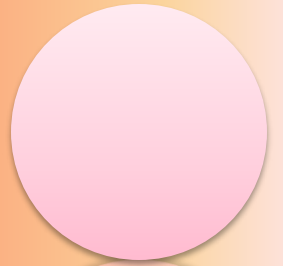
Faster Development Time

- **Example:** Developing an e-commerce platform for a seasonal sale using tools like Shopify or Magento.
- **Advantage:** RAD enables quick prototyping and delivery, allowing the system to be operational in weeks instead of months.

Encourages User Involvement

- **Example:** Building a healthcare management system with patient and staff feedback during iterative releases.
- **Advantage:** Stakeholders actively participate in defining and refining requirements, ensuring the final product aligns with user needs.

RAD model Advantages contd..



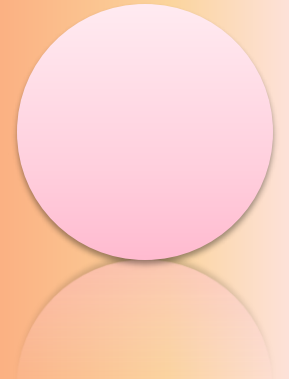
Flexibility to Changes

- **Example:** A mobile banking app where customers request features like biometric authentication after initial deployment.
- **Advantage:** RAD easily accommodates changes during the development cycle without causing delays.

Focus on Reusability

- **Example:** Creating an event management app using pre-built modules like registration forms and payment gateways.
- **Advantage:** Reusable components reduce redundancy and accelerate development.

RAD model Disadvantages



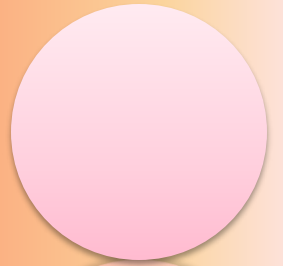
Not Suitable for Large, Complex Projects

- **Example:** Developing a large-scale ERP system for a multinational company.
- **Disadvantage:** RAD may struggle with managing extensive systems that require comprehensive planning and multiple integrations.

Requires Skilled Developers

- **Example:** Rapidly building a ride-sharing app using advanced frameworks like React Native.
- **Disadvantage:** The success of RAD heavily relies on skilled and experienced developers who can handle rapid iterations.

RAD model Disadvantages contd..



High Dependency on User Feedback

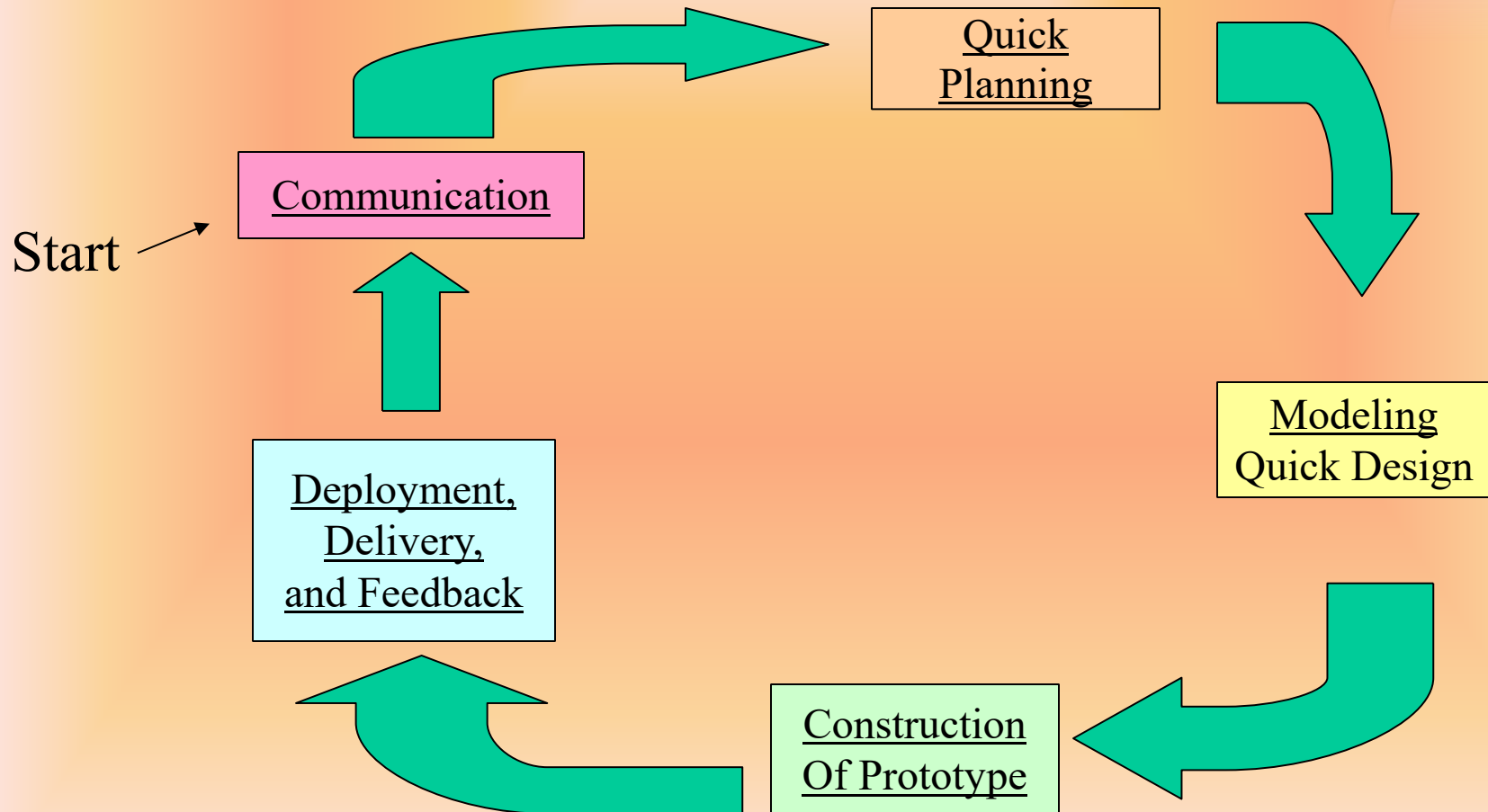
- **Example:** A prototype of an LMS for an educational institute.
- **Disadvantage:** If stakeholders fail to provide timely or meaningful feedback, the project can lose direction.

Not Ideal for Projects with Fixed Budgets

- **Example:** Developing a government-sponsored e-governance app with a strict budget.
- **Disadvantage:** Iterative changes and frequent updates can lead to cost overruns.

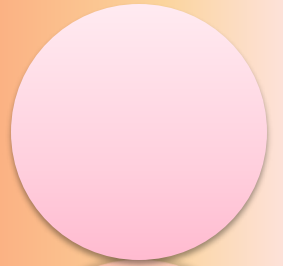
Prototyping Model

(Diagram)



Prototyping Model

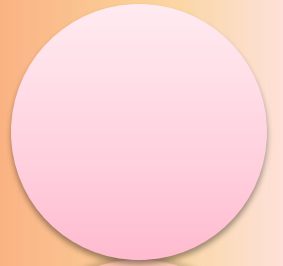
(Description)



- **Follows an evolutionary and iterative approach**
- **Used when requirements are not well understood**
- **Serves as a mechanism for identifying software requirements**
- **Focuses on those aspects of the software that are visible to the customer/user**
- **Feedback is used to refine the prototype**

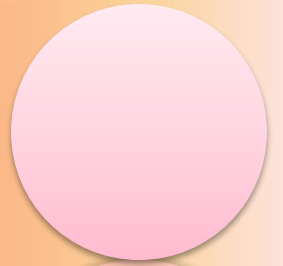
Prototyping Model

(Description)



- The prototyping paradigm begins with *communication* where requirements and goals of Software are defined.
- Prototyping iteration is *planned* quickly and modeling in the form of **quick design** occurs.
- The *quick design* focuses on a representation of those aspects of the Software that will be visible to the customer “Human interface”.
- The quick design leads to the *Construction of the Prototype*.
- The prototype is *deployed* and then *evaluated* by the customer.
- *Feedback* is used to refine requirements for the Software.
- Iteration occurs as the prototype is tuned to satisfy the needs of the customer, while enabling the developer to better understand what needs to be done.
- The prototype can serve as the “first system”. Both customers and developers like the prototyping paradigm as users get a feel for the actual system, and developers get to build Software immediately.

Prototyping Model Example



E-commerce Platforms:

- RAD accelerates the launch of websites like Shopify-based stores during festive seasons.

Mobile Applications:

- Frameworks like Flutter are used for apps requiring rapid development, such as food delivery or fitness tracking apps.

CRM Systems:

- Tools like Salesforce allow businesses to customize CRM systems quickly for immediate deployment.

Event Management Apps:

- Applications like Eventbrite are iteratively refined to manage registrations, schedules, and ticketing.

Prototyping Model Advantages



Clearer Understanding of Requirements

Advantage: Users can visualize the system and refine their requirements based on the prototype.

Example: A healthcare startup creates a prototype of an **Electronic Health Record (EHR)** system. Doctors and nurses test the prototype and request additional features like patient history charts and prescription templates. This iterative feedback clarifies the actual needs.

Early Detection of Design Flaws

Advantage: Design flaws or usability issues are identified and corrected early, saving time and cost.

Example: A fintech app prototype for loan applications reveals navigation issues during user testing. Developers address these issues before full-scale development, avoiding costly redesigns later.

Prototyping Model Advantages



Facilitates User-Centric Design

Advantage: Prototypes ensure the system is designed around user needs and behaviors.

Example: For a **fitness tracking app**, the prototype includes step tracking and calorie counting. Feedback from users helps optimize the UI for ease of access during workouts.

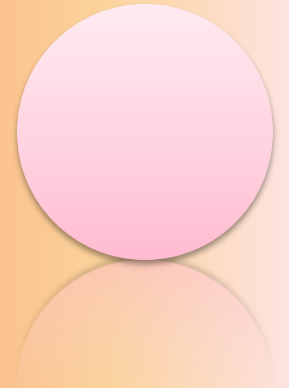
Better Cost Estimation

Advantage: Building a prototype helps estimate the final project's development cost and resource requirements.

Example: A travel agency developing an **online booking platform** prototypes the core search and booking features. This provides a clearer picture of the resources and costs required for the final system.

Prototyping Model

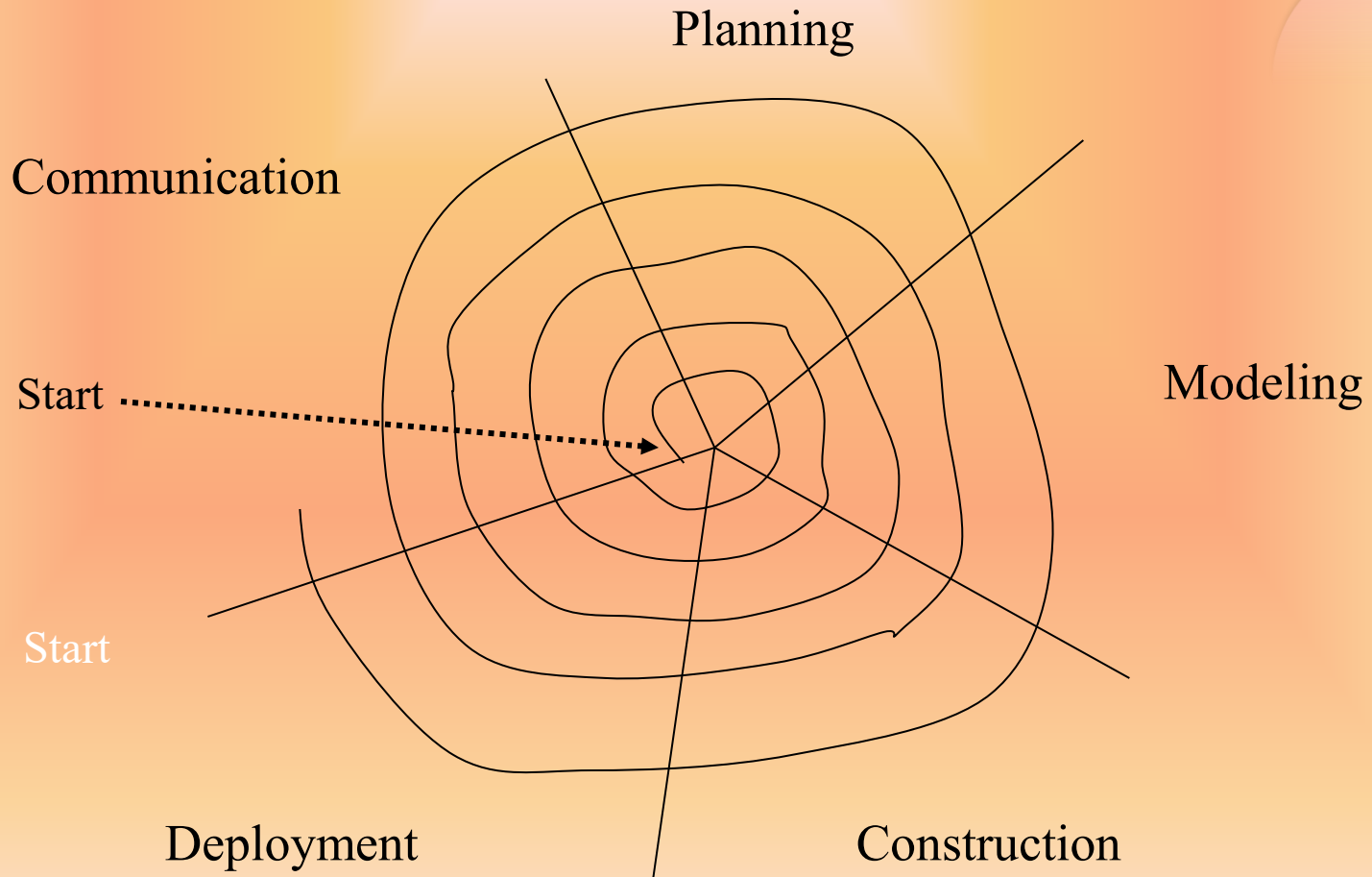
(Potential Problems)



- The customer sees a "working version" of the software, wants to stop all development and then buy the prototype after a "few fixes" are made
- Developers often make implementation compromises to get the software running quickly (e.g., language choice, user interface, operating system choice, inefficient algorithms)

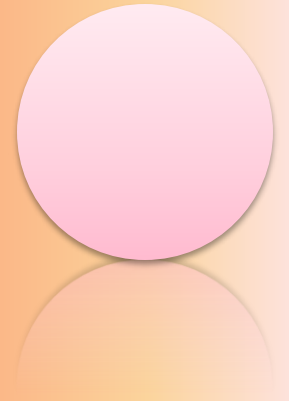
Spiral Model

(Diagram)



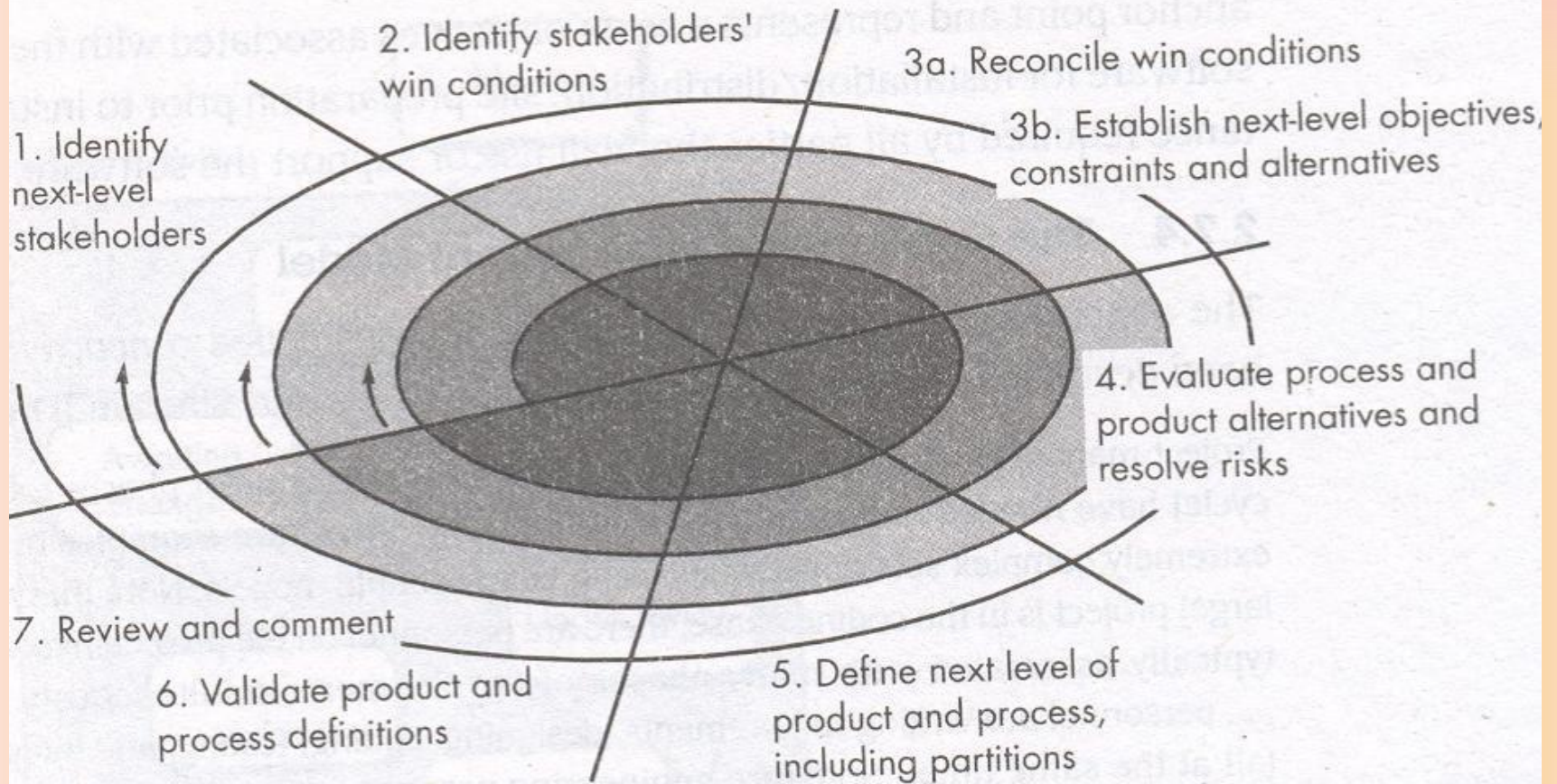
Spiral Model

(Description)



- Invented by Dr. Barry Boehm in 1988
- Follows an evolutionary approach
- Used when requirements are not well understood and risks are high
- **Inner spirals** focus on identifying software requirements and project risks; may also incorporate prototyping
- **Outer spirals** take on a classical waterfall approach after requirements have been defined, but permit iterative growth of the software
- Operates as a risk-driven model...a go/no-go decision occurs after each complete spiral in order to react to risk determinations
- Requires considerable expertise in risk assessment
- Serves as a realistic model for large-scale software development

WinWin Spiral Model



WinWin Spiral Model contd..



- The customer and the developer enter into a process of negotiation, where the customer may be asked to balance functionality, performance, and other product or system characteristics against cost and time to market.
- The best negotiations strive for a “win-win” result.
- The customer wins by getting the system or product that satisfies the majority of the customer’s needs
- The developer wins by working to realistic and achievable budgets and deadlines.

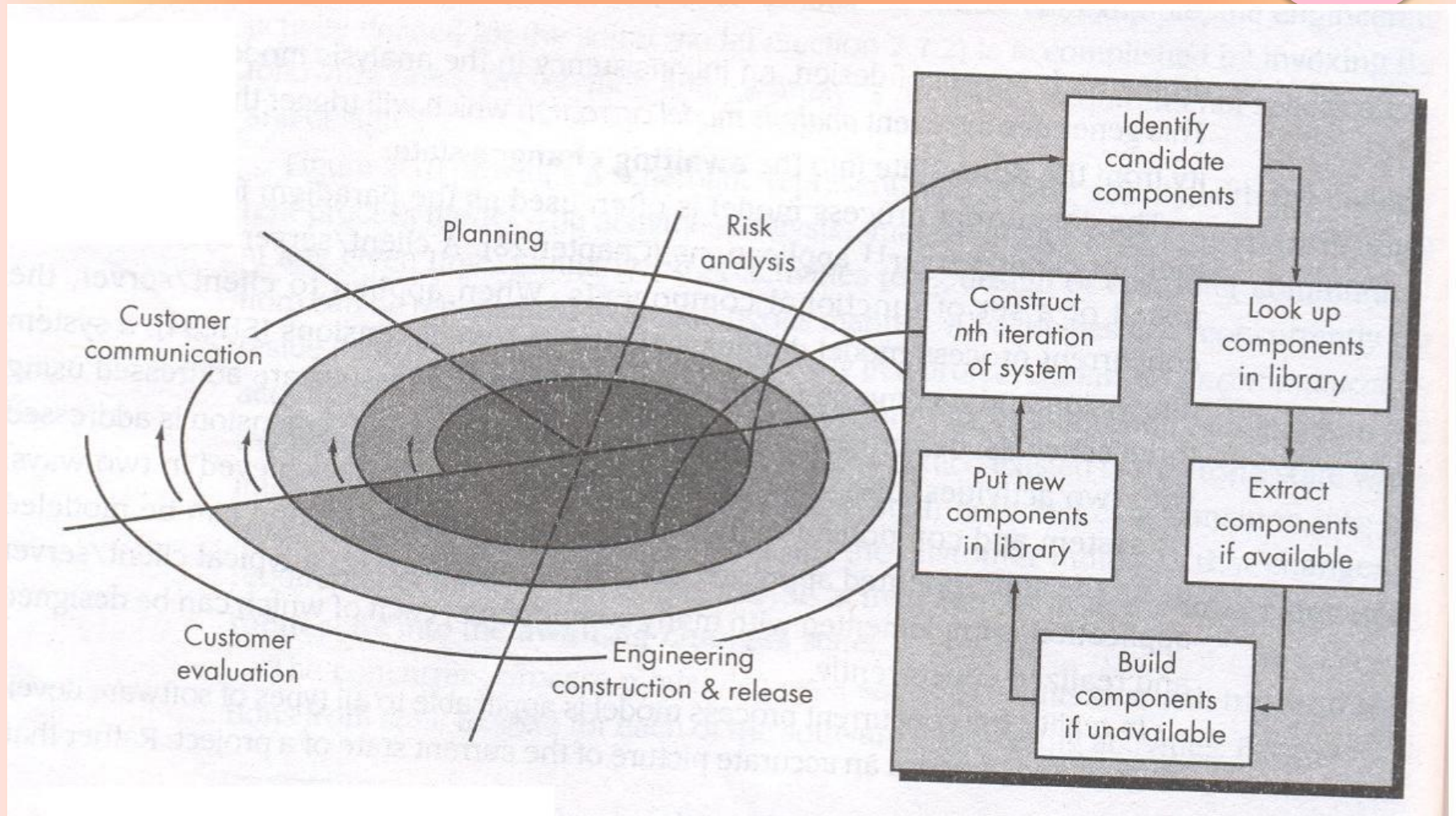
Specialized Process Models



Component-based Development Model

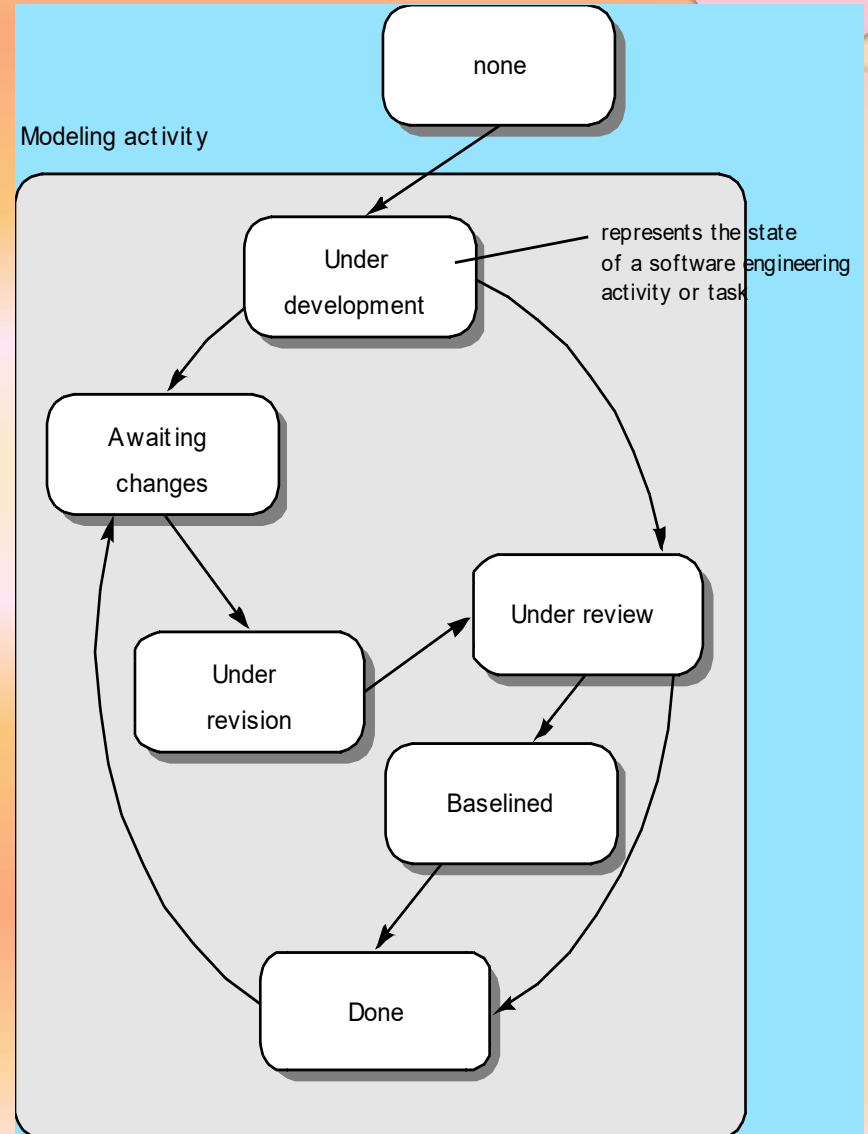
- **Consists of the following process steps**
 - **Available component-based products are researched and evaluated for the application domain in question**
 - **Component integration issues are considered**
 - **A software architecture is designed to accommodate the components**
 - **Components are integrated into the architecture**
 - **Comprehensive testing is conducted to ensure proper functionality**
- **Relies on a robust component library**
- **Capitalizes on software reuse, which leads to documented savings in project cost and time**

Component Assembly Model



Concurrent Process Model

- The concurrent development model, sometimes called concurrent engineering, can be represented schematically as a series of framework activities, Software engineering actions of tasks, and their associated states.





The concurrent model is often more appropriate for system engineering projects where different engineering teams are involved.

Figure above provides a schematic representation of one Software engineering task within the modeling activity for the concurrent process model. The activity – modeling- may be in any one of the states noted at any given time.

All activities exist concurrently but reside in different states.

For example, early in the project the communication activity has completed its first iteration and exists in the awaiting changes state. The modeling activity which existed in the none state while initial communication was completed now makes a transition into underdevelopment state.

If, however, the customer indicates the changes in requirements must be made, the modeling activity moves from the under development state into the awaiting changes state.

Comparing Models

Model	Strengths	Weaknesses
Waterfall	Disciplined approach Document driven	Final product may not meet client's real needs
Prototyping	Helps to detail user's requirements	The prototype may outlive its usefulness
Iterative	Get early results Promotes maintenance	May degenerate into CABTAB
Spiral	Incorporates features from other models	Only suited to large-scale in-house development

An Agile View of Process

Introduction



Classical methods of software development have many disadvantages:

- huge effort during the planning phase
- poor requirements conversion in a rapid changing environment
- treatment of staff as a factor of production

➤ ***New methods:***
Agile Software Development Methodology

The Manifesto for Agile Software Development

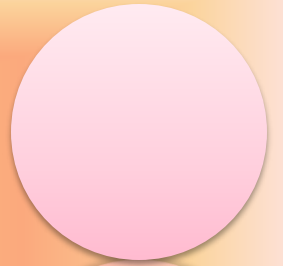
- **“We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:**
 - **Individuals and interactions over processes and tools**
 - **Working software over comprehensive documentation**
 - **Customer collaboration over contract negotiation**
 - **Responding to change over following a plan**

What is Agility?



- **Effective response to change**
- **Effective communication among all stakeholders**
- **Drawing the customer onto the team; eliminate the “us and them” attitude**
- **Organizing a team so that it is in control of the work performed**
- **Rapid, incremental delivery of software**

Agile software development



- **It promotes adaptive planning, evolutionary development, early delivery, continuous improvement, and encourages rapid and flexible response to change.**

Principles to achieve agility – by the Agile Alliance (1)

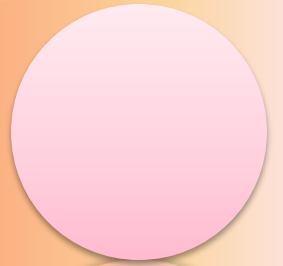
- 1. Highest priority -> satisfy the customer**
- 2. Welcome changing requirements**
- 3. Deliver working software frequently**
- 4. Business people and developers must work together**
- 5. Build projects around motivated individuals**
- 6. Emphasize face-to-face conversation**

Principles to achieve agility – by the Agile Alliance

(2)

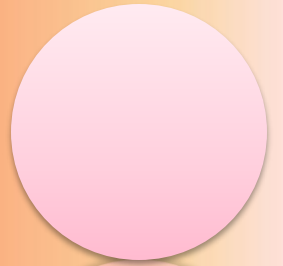
- 7. Working software is the primary measure of progress**
- 8. Agile processes promote sustainable development**
- 9. Continuous attention to technical excellence and good design enhances agility**
- 10. Simplicity – the art of maximizing the amount of work not done – is essential**
- 11. The best designs emerge from self-organizing teams**
- 12. The team tunes and adjusts its behavior to become more effective**

Agile Software Process



- **An agile process must be adaptable**
- **It must adapt incrementally**
- **Requires customer feedback**
- **An effective catalyst for customer feedback is an operational prototype**

Agile Process Models



- **Extreme Programming (XP)**
- **Adaptive Software Development (ASD)**
- **Dynamic Systems Development Method (DSDM)**
- **Scrum**
- **Crystal**
- **Feature Driven Development (FDD)**
- **Agile Modeling (AM)**

Scrum



Scrum in 100 words



Scrum is an agile process that allows us to focus on **delivering the highest business value in the shortest time.**

It allows us to rapidly and repeatedly inspect actual working software **(every two weeks to one month).**

The business sets the priorities. Our teams self-manage to determine the best way to deliver the highest priority features.

Every two weeks to a month anyone can see real working software and decide to release it as is or continue to enhance for another iteration.

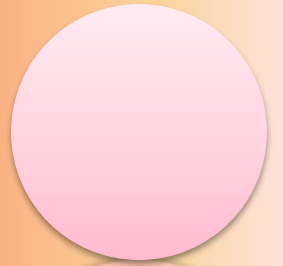
Scrum contd..

- Name is derived from activity that occurs during rugby match.
- Was conceived by Jeff Sutherland & his development team in the early 90s.
- Further development is done by Schwaber & Beedle.
- Principles are consistent with the agile manifesto & are used to guide development activities.
- Each individual iteration phase is termed as a **sprint**.
- Deliverable product is created during each sprint.
- **Product backlog:** prioritized list of requirements that provides business values for the customer. Items can be added anytime.

Scrum contd..

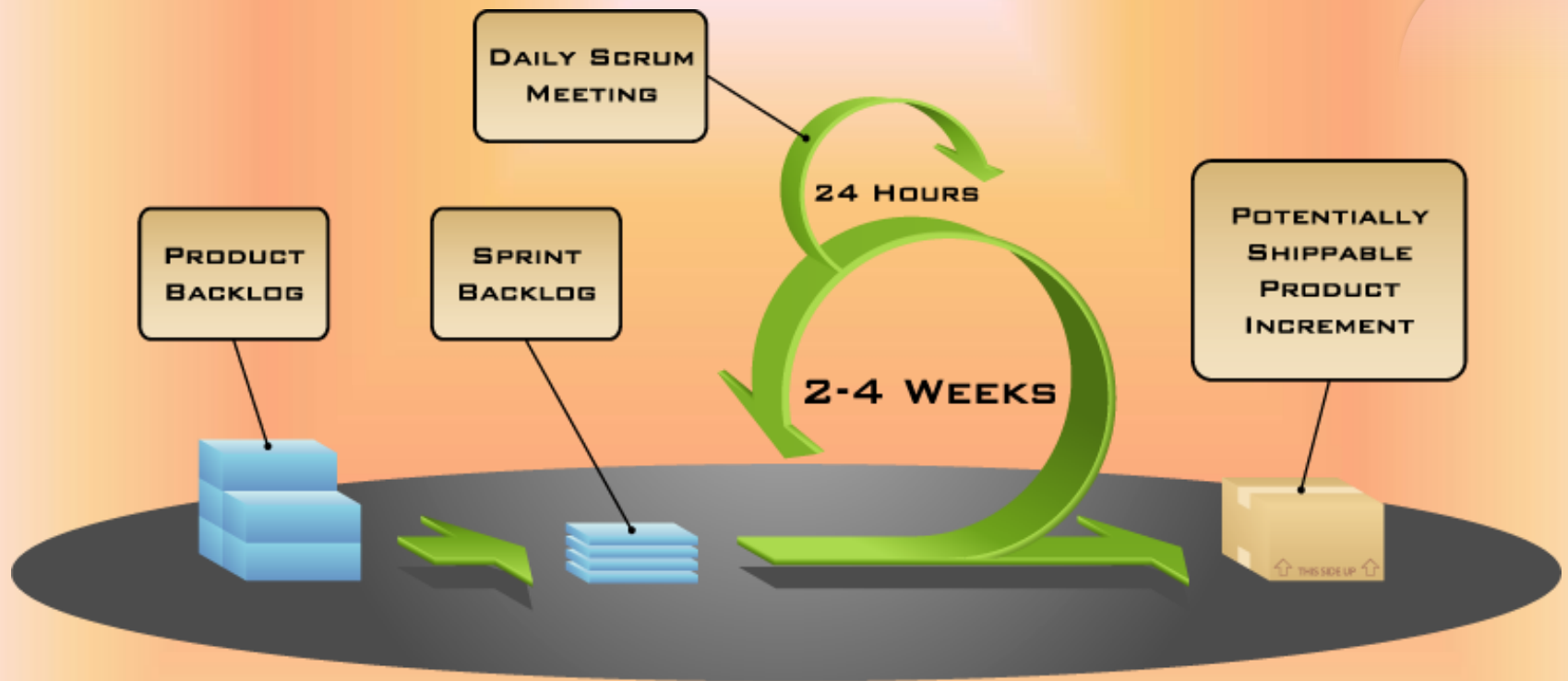
- **Scrum meetings:** are short (15 minutes) meetings held daily by the scrum team.
- 3 key questions are asked & answered by team members:
 - What did youn do since the last team meetings?
 - What obstacle are you encountering
 - What do you plan to accomplish by the next team meeting?
- **Scrum master :** a team leader, maintains the processes.
- **Demos:** delivering software increment to the customer.

Characteristics

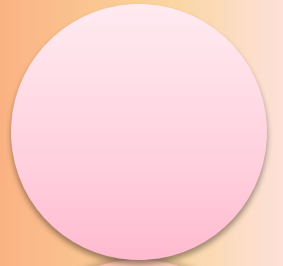


- **Self-organizing teams**
- **Product progresses in a series of month-long “sprints”**
- **Requirements are captured as items in a list of “product backlog”**
- **No specific engineering practices prescribed**
- **Uses generative rules to create an agile environment for delivering projects**
- **One of the “agile processes”**

How Scrum Works?



A REAL WORLD EXAMPLE



- Alex is assigned as the **Scrum Product Owner** of a new software development project. One of his **first tasks is to start requirement engineering**.
- **He writes down the most important use-cases** and discusses them with the architects, customer representatives and other stakeholders.
- After collecting the high-level use-cases and requirements, **he writes them into the Scrum Product Backlog** and initiates an estimation and prioritization session with the architects and some senior developers.
- As a result of this session all the items in the Scrum Product Backlog **have an initial rough estimation and a prioritization**.
- Now he starts to break-down the high-level requirements into smaller-grained user stories. With this list he then calls for the **first Sprint Planning meeting**.

- **Sprint 1 - Day 0**

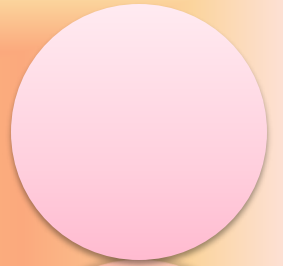
During the Sprint Planning meeting **Alex presents the Scrum Product Backlog items from the highest priority to the lowest.**

- The team clarifies open questions and for each item the team discusses if they have enough capacity, the required know-how and if everything else needed is available. After this discussion they commit to complete the stories 1,2,3,6,7 and 8 until the end of this sprint. The items 4 and 5 cannot be realized in this sprint, as some technical infrastructure is not yet in place.

After the Sprint Planning meeting Frank - **the Scrum Master of the team** - **calls the team to define the details of how the committed items are going to be implemented.**

- The resulting tasks are written down on the cards at the prepared Sprint Task board. Now everyone of the Scrum Team selects a task to work on.

Advantages of Scrum Development:



- In this methodology decision-making is entirely in the hands of the teams.
- This methodology enables project's where the business requirements documentation is not consider very significant for the successfully development.
- It is a lightly controlled method which totally empathizes on frequent updating of the progress therefore project development steps is visible in this method.
- A daily meeting easily helps developer to make it possible to measure individual productivity. This leads to the improvement in the productivity of each of the team members.

Disadvantages of Scrum Development:

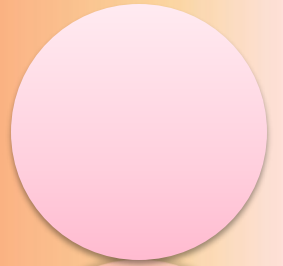
- This kind of development model is suffered, if the estimating project costs and time will not be accurate.
- It is good for small, fast moving projects but not suitable for large size projects.
- This methodology needs experienced team members only. If the team consists of people who are novices, the project cannot be completed within exact time frame.

Feature Driven Development (FDD)



- It is an agile iterative and incremental model that focuses on progressing the features of the developing software.
- The main motive of feature-driven development is to provide timely updated and working software to the client.
- In FDD, reporting and progress tracking is necessary at all levels.

Characteristics of FDD

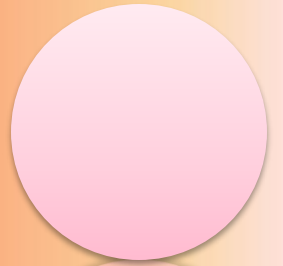


- **Short iterative:** FDD lifecycle works in simple and short iterations to efficiently finish the work on time and gives good pace for large projects.
- **Customer focused:** This agile practice is totally based on inspection of each feature by client and then pushed to main build code.
- **Structured and feature focused:** Initial activities in lifecycle builds the domain model and features list in the beginning of timeline and more than 70% of efforts are given to last 2 activities.
- **Frequent releases:** Feature-driven development provides continuous releases of features in the software and retaining continuous success of the project.

Advantages of FDD

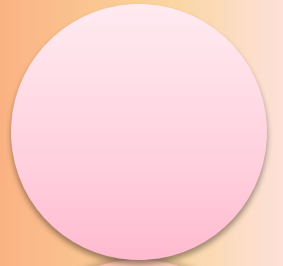
- Reporting at all levels leads to easier progress tracking.
- FDD provides continuous success for larger size of teams and projects.
- Reduction in risks is observed as whole model and design is build in smaller segments.
- FDD provides greater accuracy in cost estimation of the project due to feature segmentation.

Disadvantages of FDD



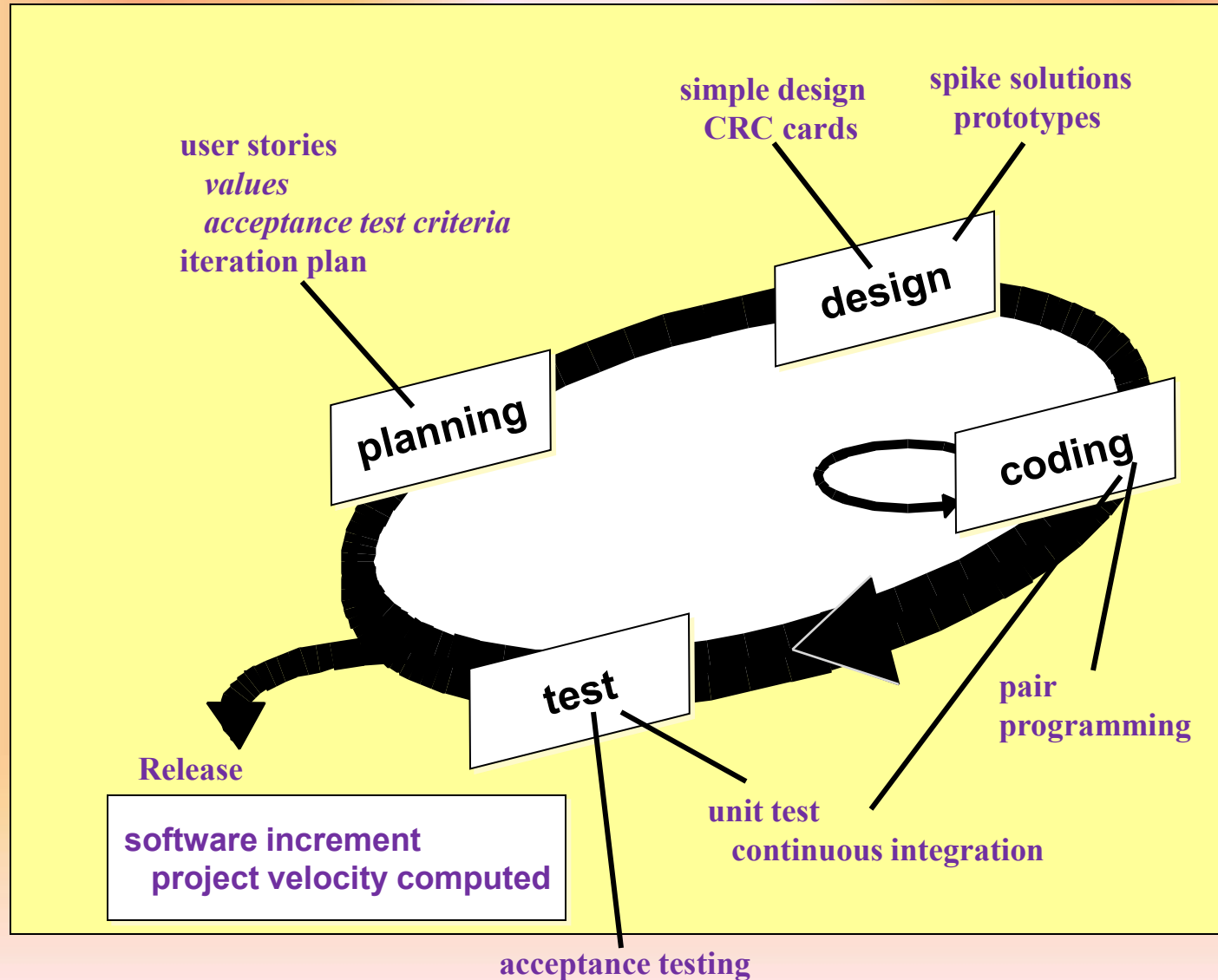
- This agile practice is not good for smaller projects.
- There is high dependency on lead programmers, designers and mentors.
- There is lack of documentation which can create an issue afterwards.

Extreme Programming (XP) - 1



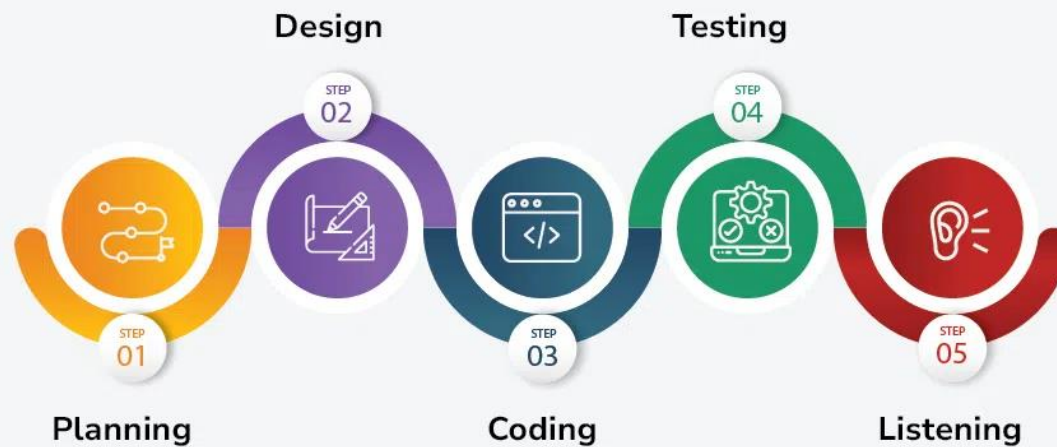
- The most widely used agile process, originally proposed by Kent Beck [BEC99]
- XP uses an object-oriented approach as its preferred development paradigm
- Defines four (4) framework activities
 - Planning
 - Design
 - Coding
 - Testing

Extreme Programming (XP) - 2





Life Cycle of Extreme Programming (XP)

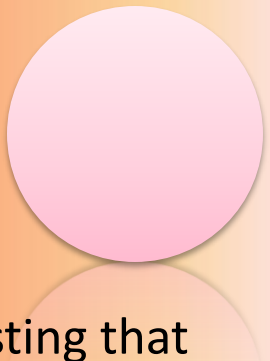


Life Cycle of Extreme Programming (XP)

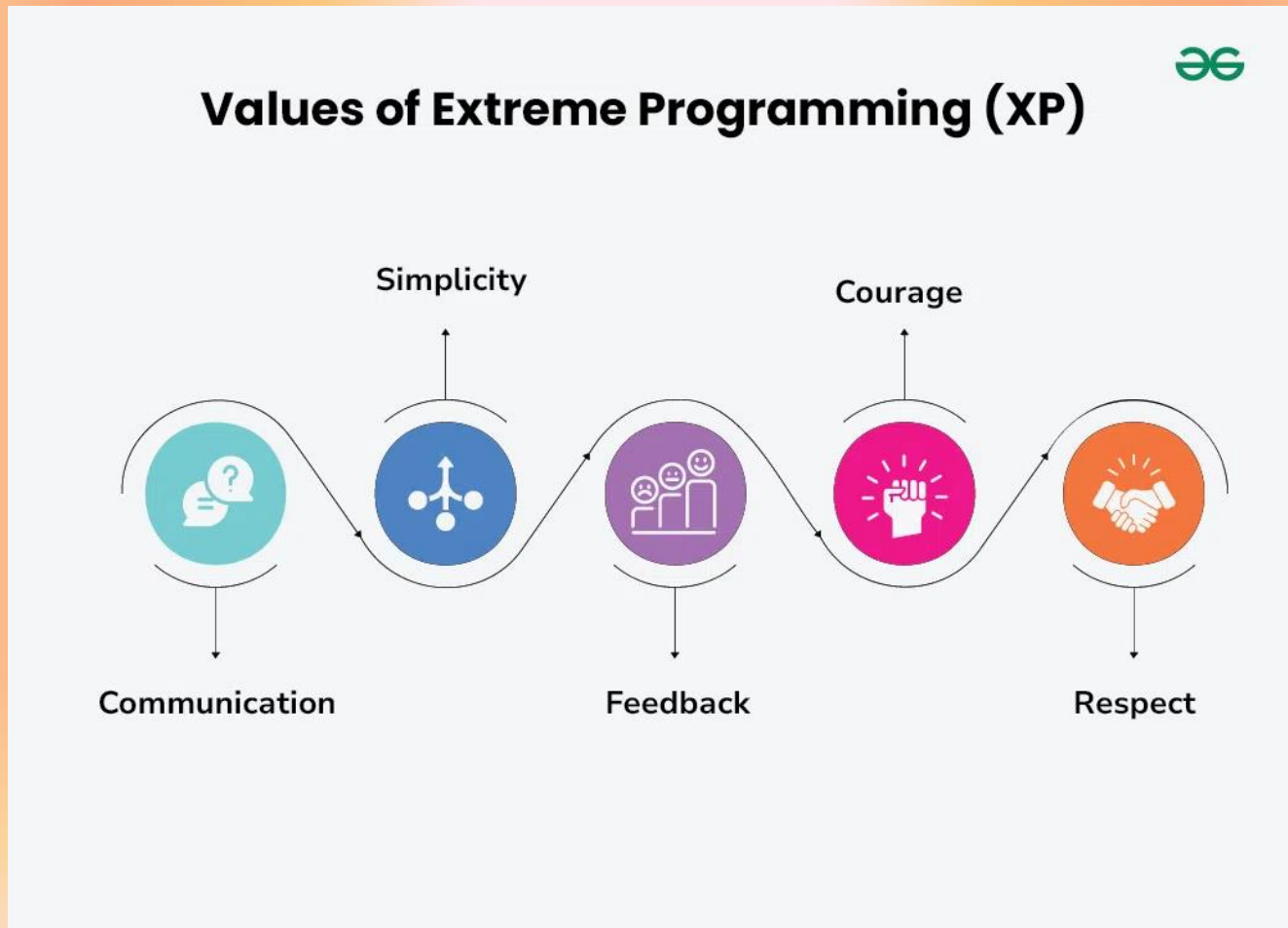


The Extreme Programming Life Cycle consist of five phases:

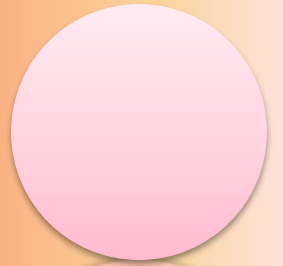
- **Planning:** During this phase, clients define their needs in concise descriptions known as user stories. The team calculates the effort required for each story and schedules releases according to priority and effort.
- **Design:** The team creates only the essential design needed for current user stories, using a common analogy or story to help everyone understand the overall system architecture and keep the design straightforward and clear.
- **Coding:** Extreme Programming (XP) promotes pair programming i.e. two developers work together at one workstation, enhancing code quality and knowledge sharing. They write tests before coding to ensure functionality from the start (TDD), and frequently integrate their code into a shared repository with automated tests to catch issues early.

- 
- **Testing:** Extreme Programming (XP) gives more importance to testing that consist of both unit tests and acceptance test. Unit tests, which are automated, check if specific features work correctly. Acceptance tests, conducted by customers, ensure that the overall system meets initial requirements. This continuous testing ensures the software's quality and alignment with customer needs.
 - **Listening:** In the listening phase regular feedback from customers to ensure the product meets their needs and to adapt to any changes.

Values of Extreme Programming (XP)

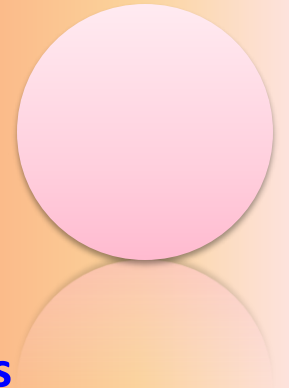


XP - Planning



- Begins with the creation of a set of stories (also called *user stories*)
 - Each story is written by the customer and is placed on an **index card**
 - The customer assigns **a value** (i.e. **a priority**) to the story
 - Agile team assesses each story and assigns a **cost**
 - Stories are grouped to for a **deliverable increment**
 - A **commitment** is made on delivery date
 - After the first increment “**project velocity**” is used to help define subsequent delivery dates for other increments
-
- “Project velocity: no of customer stories implemented during the first release.”

XP - Design

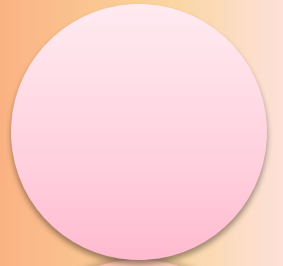


- Follows the **KIS (keep it simple)** principle
- Encourage the use of **CRC (class-responsibility-collaborator)** cards
- For difficult design problems, suggests the creation of “*spike solutions*”—a design prototype
- Encourages “*refactoring*”—an iterative refinement of the internal program design
- Design occurs both before and after coding commences

Class Name	
Responsibilities	Collaborators

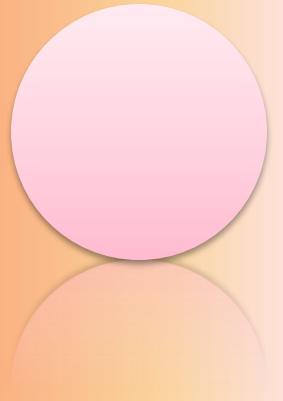
Student	
Student number Name Address Phone number Enroll in a seminar Drop a seminar Request transcripts	Seminar

XP - Coding



- Recommends the construction of a series of **unit tests** for each of the stories before coding commences
- Encourages “*pair programming*”
 - Mechanism for real-time problem solving and real-time quality assurance: **two heads are often better than one**
 - Keeps the developers focused on the problem at hand

XP - Testing



- **Unit tests** should be implemented using a framework to make testing **automated**. This encourages a **regression testing** strategy.
- **Integration and validation testing** can occur on a daily basis
- **Acceptance tests**, also called **customer tests**, are specified by the customer and executed to assess customer visible functionality
- **Acceptance tests** are derived from user stories