UNIT 1

19CSE314 SOFTWARE ENGINEERING

What is Software?

Software is: (1) instructions (computer programs) that when executed provide desired features, function, and performance; (2) data structures that enable the programs to adequately manipulate information and (3) documentation that describes the operation and use of the programs.

What is Software?

- *Software is developed or engineered, it is not manufactured in the classical sense.*
- *Software doesn't "wear out."*
- *Although the industry is moving toward component-based construction, most software continues to be custom-built.*

Key aspects in software engineering

1. Requirement Analysis
2. Software Design
3. Software Development
4. Test and Integration
5. Deployment
6. Operationalization and Maintenance

1. Requirement Analysis

- Stakeholder interaction
- Requirement Elicitation
- Requirement Documentation
- Requirement Validation

- Scope definition
- Requirement traceability
- Prototyping
- Change management
- Risk analysis

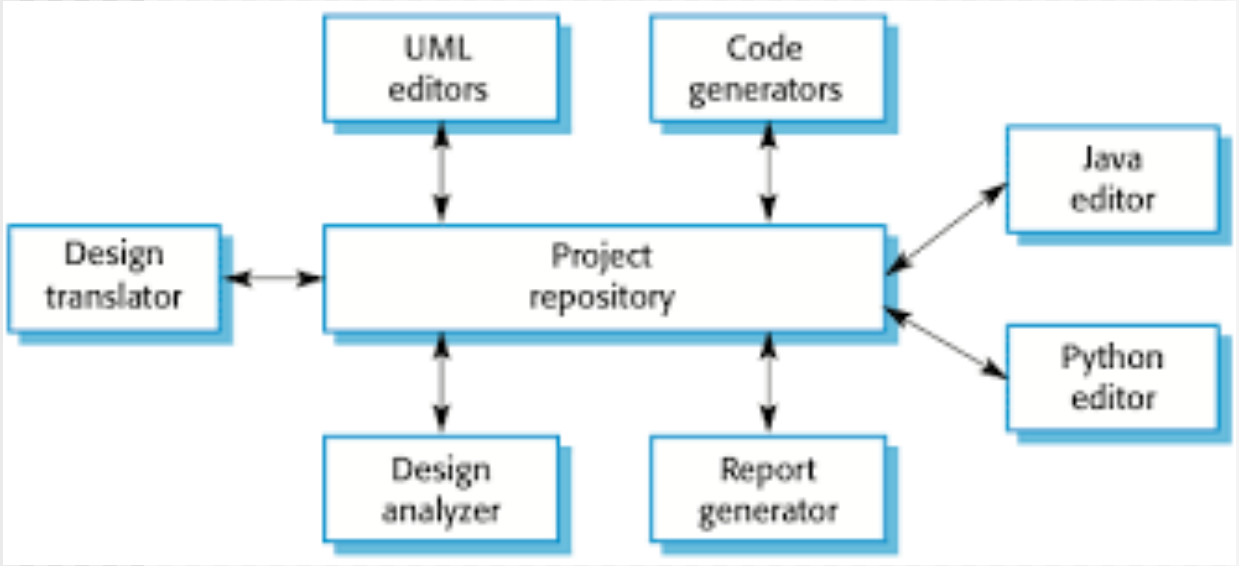
2. Software Design

Defining the architecture, components, modules and interfaces of the software system.

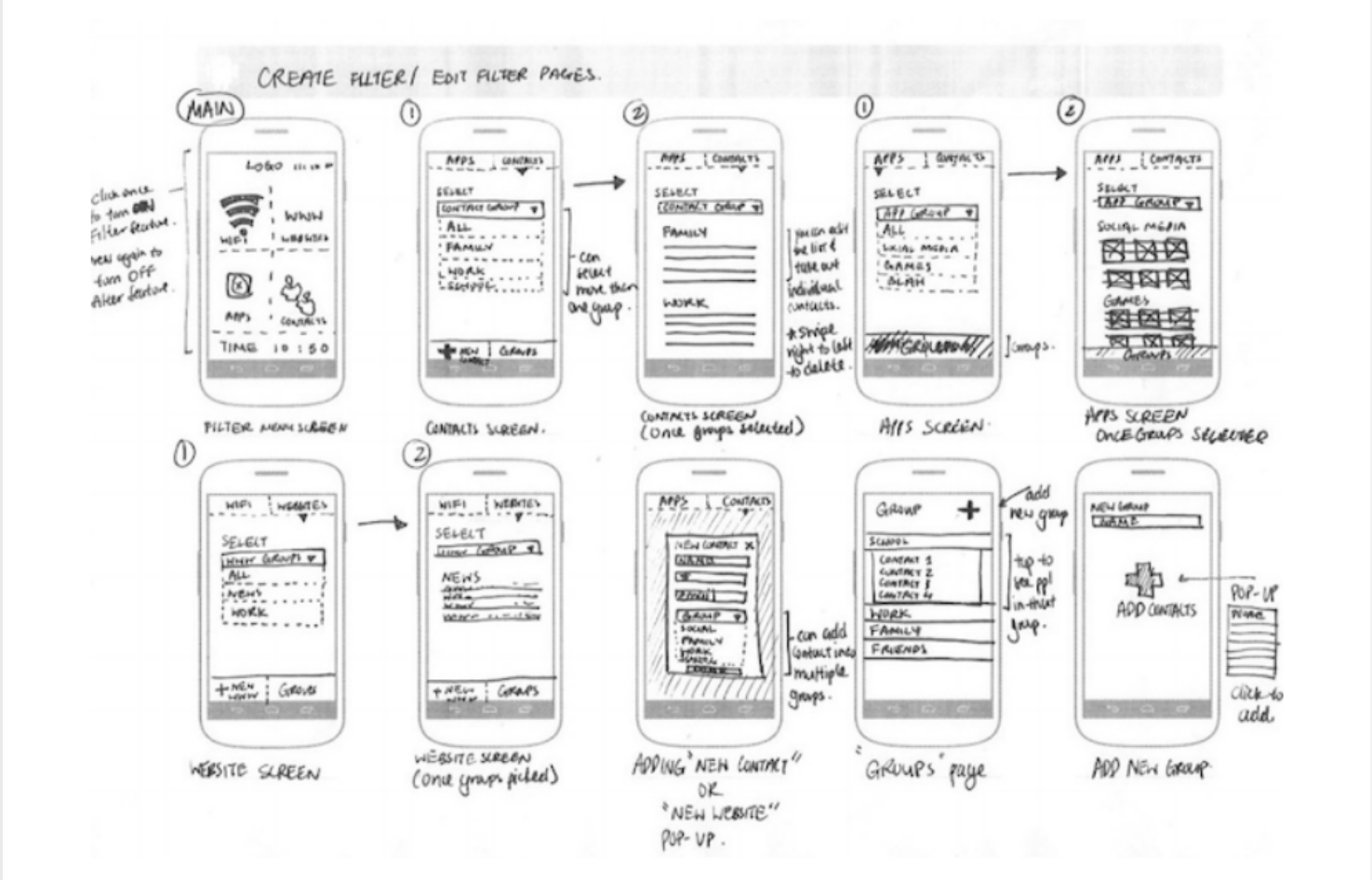
SRS is a reference for software developers to design best architecture for the software.

1. Architectural design
2. Interface design
3. Detailed design

Architectural design



User interface design



DDS

Design document specification

- Description of a software project in terms of architecture of software with various components with specified functionality.
- Software engineer/designer or project manager can create DDS.

Contents in DDS

1. Reference documents
2. Modules of the product
3. Scope
4. Design description
5. Test Provisions
6. Packaging
7. File structures and global data
8. Requirement cross reference

Reference documents

Examples,

- Existing software documentation
- System documentation
- Vendor documents
- Technical reference

Scope

- Hardware, Software and human interfaces
- Major software functions.
- Externally defined databases.

Importance of design documentation

1. Requirements are well understand
2. Architecture/ design of the product
3. New person can also work for the project
4. Everything is well stated

3. Developing product/ Coding

Implementing product by using programming languages by following the protocols set by the team.

4. Testing and Integration

- To ensure the smooth execution of the product.
- Minimal testing in each phase.
- Tracking all the probable flaws and fix them and retested.
- Ensure the product meets the quality standards.

5. Deployment and maintenance

- After detailed testing final product is released.
- Testing in real industrial environment.
- If performs well, organization deliver the product
- After retrieving beneficial feedback,company releases it **as it is** or with **auxiliary improvement** to make it further helpful for the customers.

Software Applications

- System software
- Application software
- Engineering/scientific software
- Embedded software
- Product-line software
- WebApps (Web applications)
- AI software

Software—New Categories

- Open world computing—pervasive, distributed computing
- Ubiquitous computing—wireless networks
- Netsourcing—the Web as a computing engine
- Open source—”free” source code open to the computing community (a blessing, but also a potential curse!)
- Also … (see Chapter 31)
  - Data mining
  - Grid computing
  - Cognitive machines
  - Software for nanotechnologies

Legacy Software

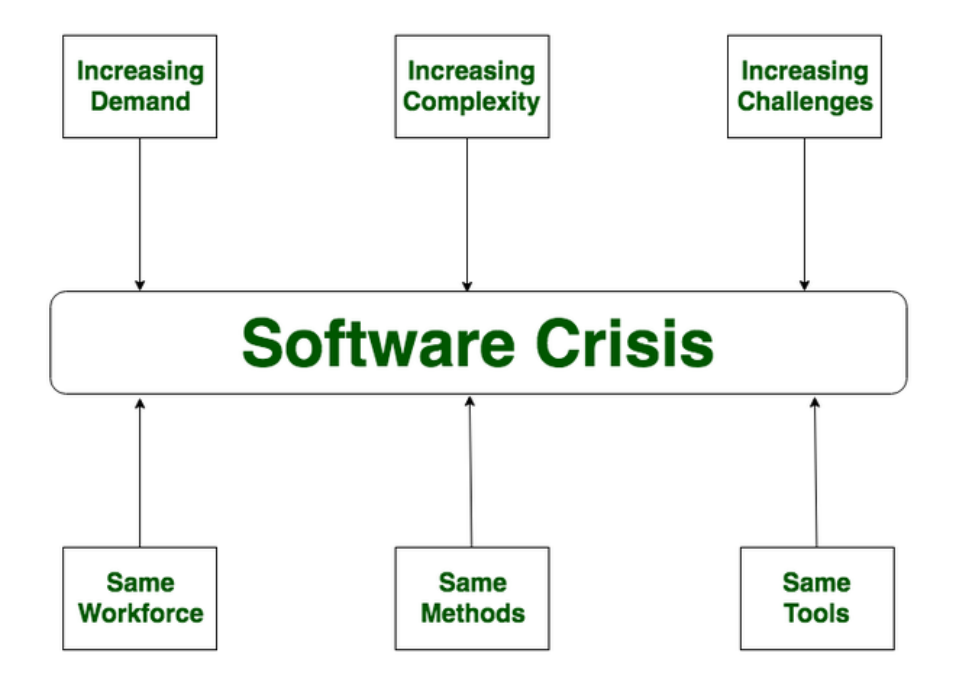
Why must it change?

- software must be adapted to meet the needs of new computing environments or technology.
- software must be enhanced to implement new business requirements.
- software must be extended to make it interoperable with other more modern systems or databases.
- software must be re-architected to make it viable within a network environment.

Software crisis

- Challenges and issues associated with developing, maintaining and managing software systems.
- In early days of software engineering, projects exceeds their budgets and schedules.
- Quality was major concern.
- Same workforce, same methods, and same tools.
- Rapidly increasing software demand, the complexity of software, and software challenges.

- Structured programming, software engineering methodologies are the best practices to address these challenges.
- Still faces issues like complexity, security vulnerabilities and changing requirements.


Increasing Demand
Increasing Complexity
Increasing Challenges
Software Crisis
Same Workforce
Same Methods
Same Tools

Causes of Software Crisis

- The cost of owning and maintaining software was as expensive as developing the software.
- At that time Projects were running overtime.
- Software was very inefficient.
- The quality of the software was low quality.
- Software often did not meet user requirements.

- The average software project overshoots its schedule by half.
- Software was never delivered.
- Non-optimal resource utilization.
- Challenging to alter, debug, and enhance.

- Poor project management.
- Lack of adequate training in software engineering.
- Less skilled project members.
- Low productivity improvements.

Solution of Software Crisis

- There is no single solution to the crisis.
- One possible solution to software crisis is *Software engineering*.

1. Reduction in software over budget.
2. The quality of the software must be high.
3. Less time is needed for a software project.
4. Experienced and skilled people working on the project.
5. Software must be delivered.
6. Software must meet user requirements.

A Layered Technology

Software Engineering

- Quality focus

  - Bedrock that supports software engineering.

  - Continuous process improvement principles of software

- Process

  1. Basis for management control of software projects.
  2. Establishes context in which technical methods are applied.
  3. Work products are produced
  4. Milestones are established
  5. Change is properly managed

- Methods

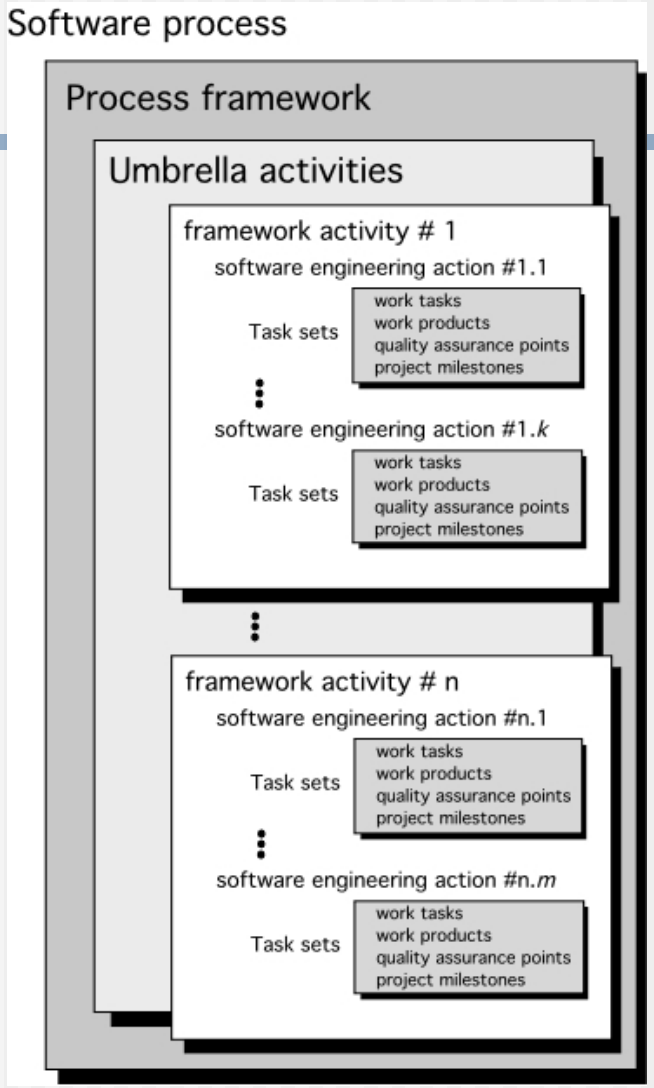
  Technical how to do's for building software

  1. Communication
  2. Requirement analysis
  3. Design modelling
  4. Program construction
  5. Testing and support

- Tools

  Automated or semi automated support for the process and methods.

A Generic Process Model



- Process framework establishes foundation for a complete SE process.
  - Framework activities
  - Umbrella activities

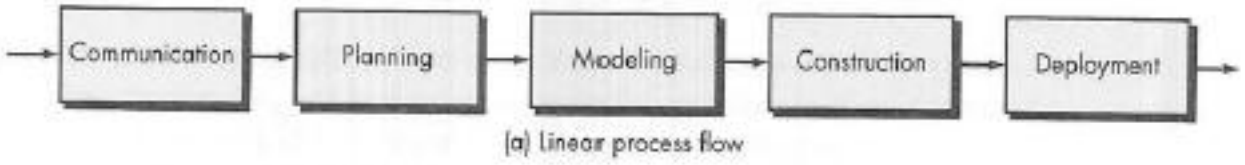
Framework activities

- Communication
- Planning
- Modeling
- Construction
- Deployment

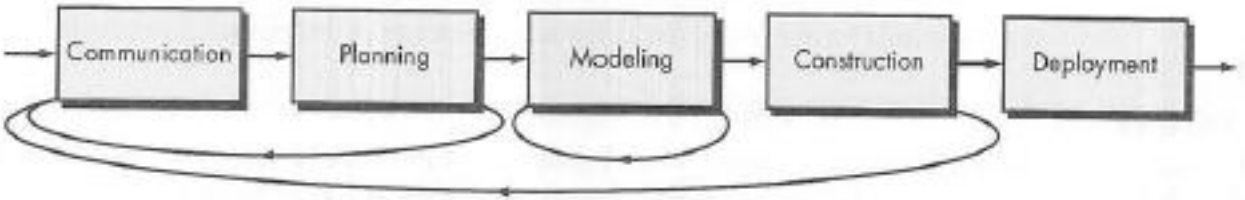
Umbrella Activities

- Software project tracking and control
- Risk management
- Software quality assurance
- Technical reviews
- Measurements
- Software configuration management
- Reusability management
- Work project preparation and production

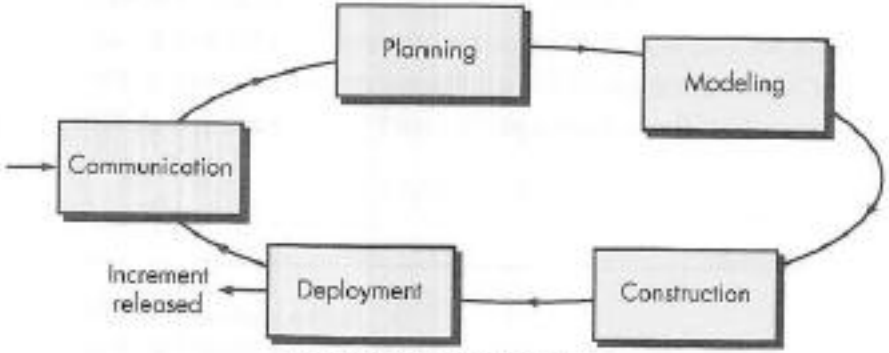
Process Flow



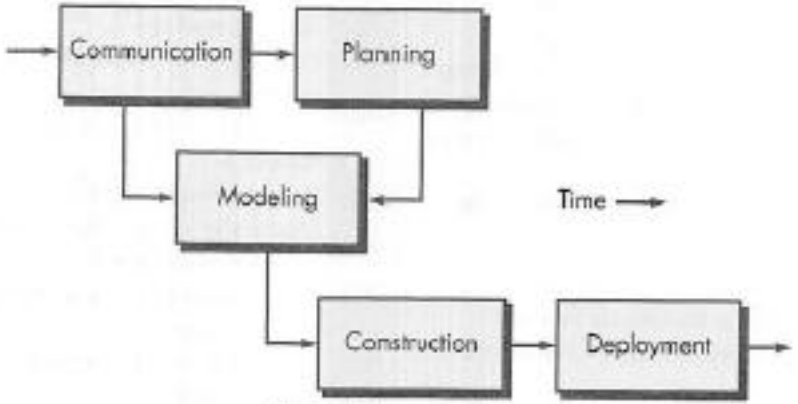
(a) Linear process flow



(b) Iterative process flow



(c) Evolutionary process flow



(d) Parallel process flow

Process Assessment and Improvement

- **Standard CMMI Assessment Method for Process Improvement (SCAMPI)** — provides a five step process assessment model that incorporates five phases: initiating, diagnosing, establishing, acting and learning.
- **CMM-Based Appraisal for Internal Process Improvement (CBA IPI)**— provides a diagnostic technique for assessing the relative maturity of a software organization; uses the SEI CMM as the basis for the assessment [Dun01]
- **SPICE—The SPICE (ISO/IEC15504)** standard defines a set of requirements for software process assessment. The intent of the standard is to assist organizations in developing an objective evaluation of the efficacy of any defined software process. [ISO08]
- **ISO 9001:2000 for Software—**a generic standard that applies to any organization that wants to improve the overall quality of the products, systems, or services that it provides. Therefore, the standard is directly applicable to software organizations and companies. [Ant06]

Prescriptive Models

- Prescriptive process models advocate an orderly approach to software engineering

That leads to a few questions ...

- If prescriptive process models strive for structure and order, are they inappropriate for a software world that thrives on change?
- Yet, if we reject traditional process models (and the order they imply) and replace them with something less structured, do we make it impossible to achieve coordination and coherence in software work?

- Prescriptive software models are those which prescribe the components which make up a software model.
- Including the activities, the inputs and outputs of the activities, how quality assurance is performed, how change is managed.
- Representation of the **order of activities** of the process and the **sequence** in which they are performed.

A model will define the following:

- The tasks to be performed
- The input and output of each task
- The pre and post-conditions for each task
- The flow and sequence of each task

The goal of a software process model is to provide guidance for controlling and coordinating the tasks to achieve the end product and objectives as effectively as possible.

SDLC models

(Software Development Life Cycle models)

- Waterfall model
- V model
- Incremental model
- Iterative model
- Prototype model
- Spiral model

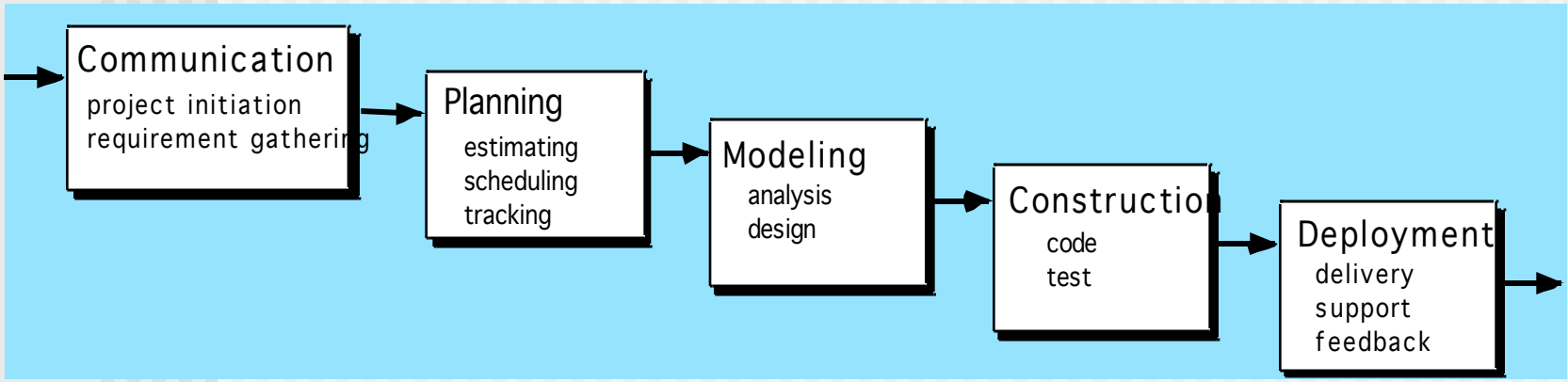
Factors in choosing a software process

1. Project requirements
2. Project size
3. Project complexity
4. Cost of delay
5. Customer involvement
6. Project resources

1. Waterfall model

- The waterfall model is a **sequential, plan driven-process**.
- Plan and schedule all the activities before starting the project.
- Each activity in the waterfall model is represented as a separate phase arranged in linear order.

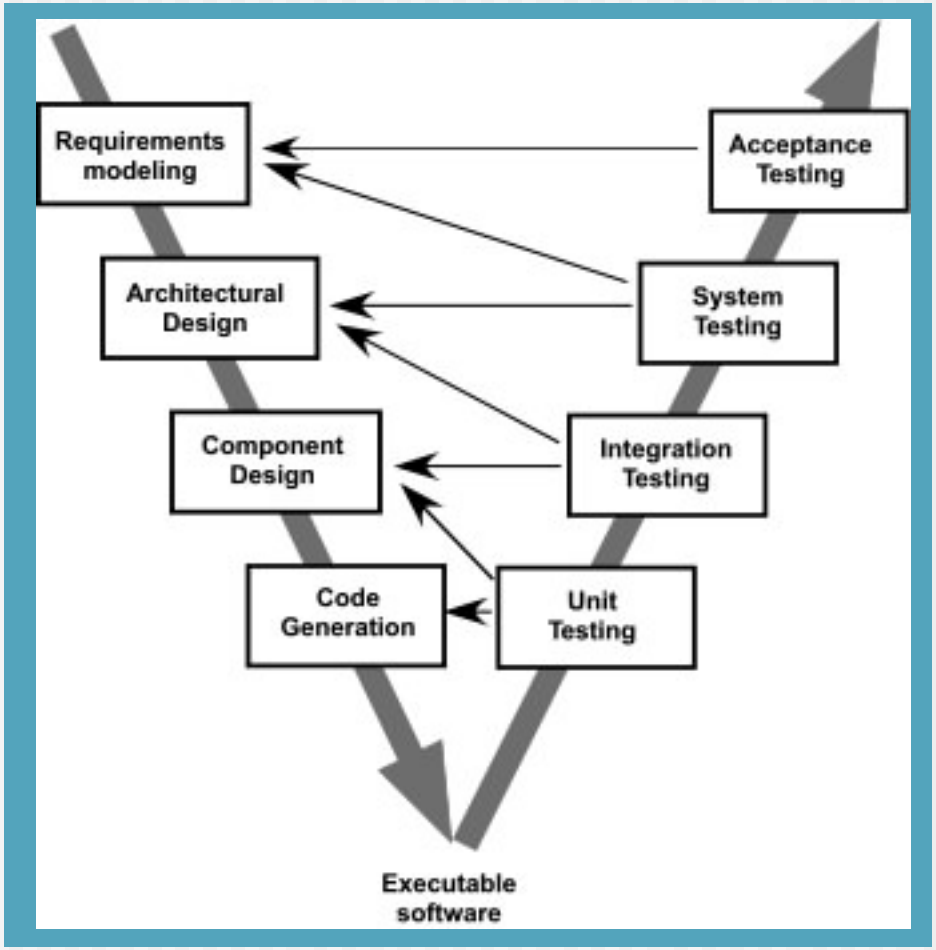
The Waterfall Model



V Model

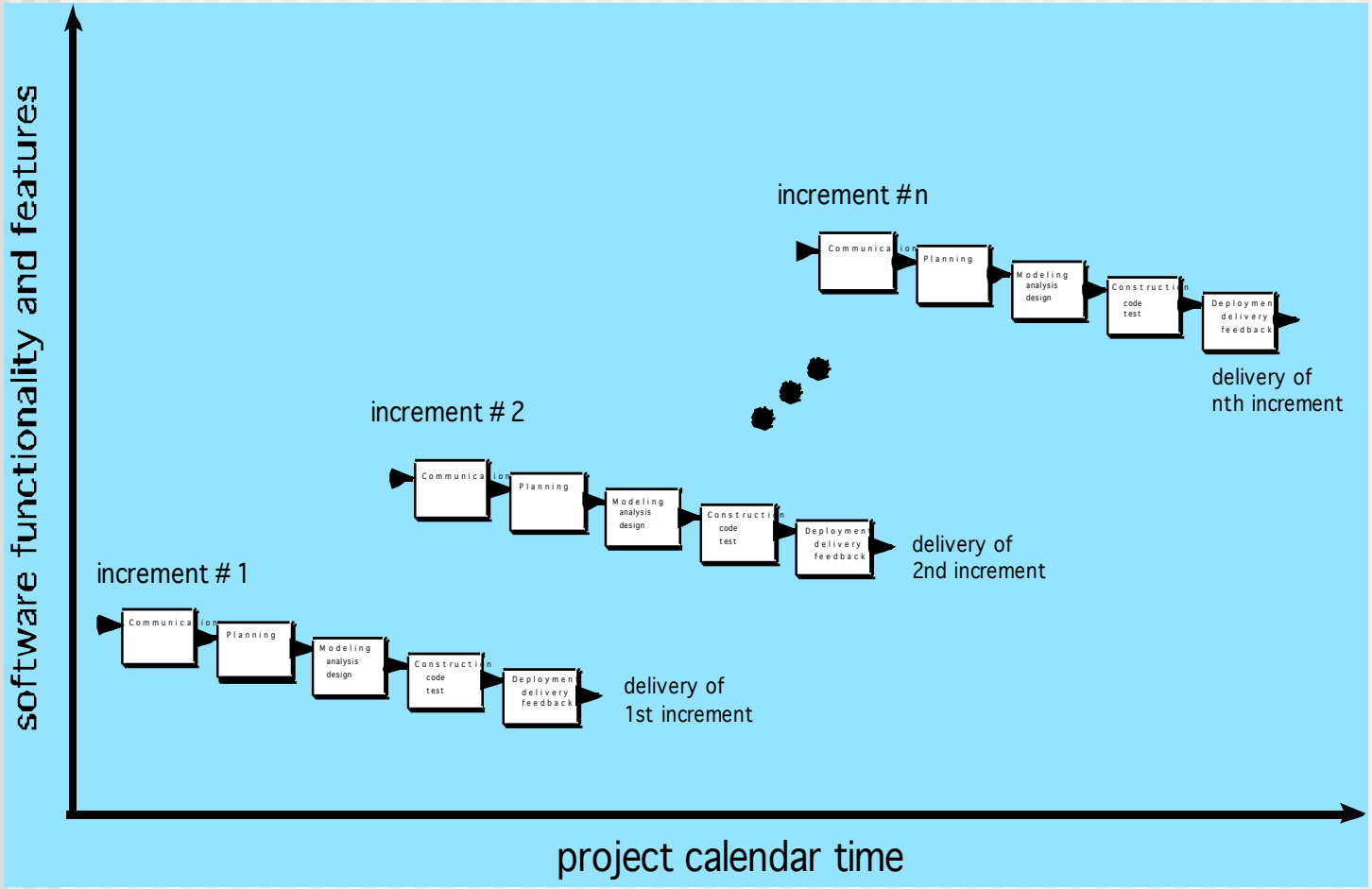
- The V model (Verification and Validation model) is an extension of the waterfall model.
- All the requirements are gathered at the start and cannot be changed.

The V-Model

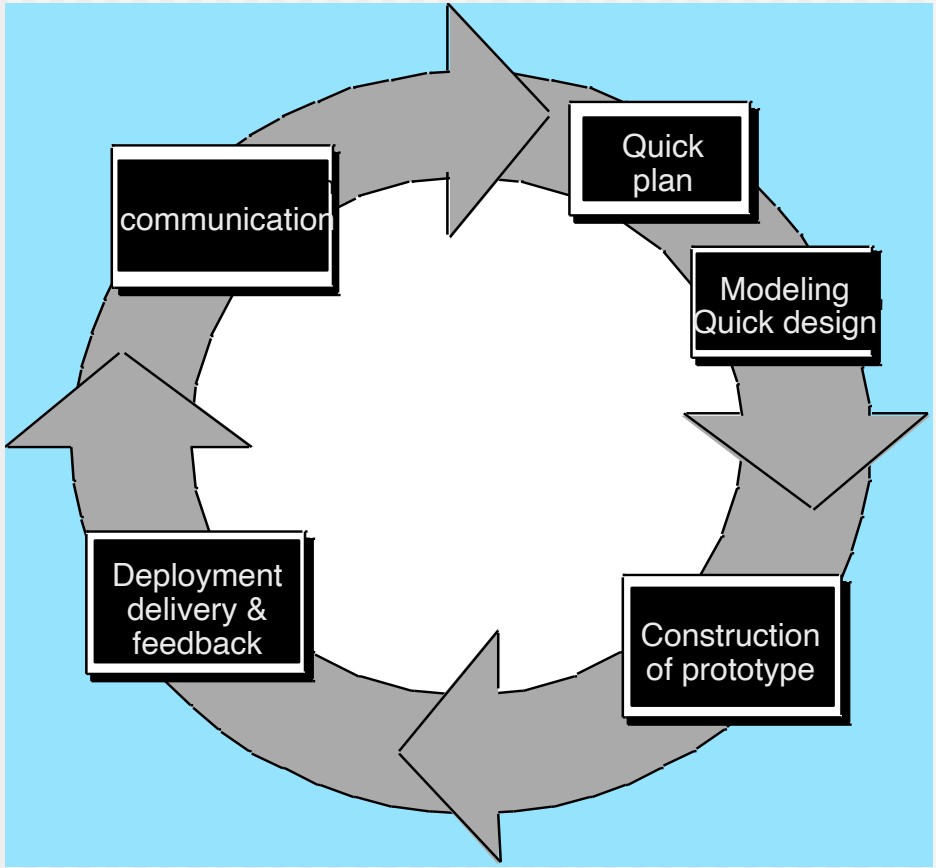

Requirements modeling
Acceptance Testing
Architectural Design
System Testing
Component Design
Integration Testing
Code Generation
Unit Testing
Executable software

- The incremental model divides the system's functionality into **small increments** that are delivered one after the other in quick succession.
- The incremental model lets stakeholders and developers see results with the first increment.
- It is efficient as the developers only focus on what is important and bugs are fixed as they arise.
- Need a **clear and complete definition** of the whole system before start developing the model.

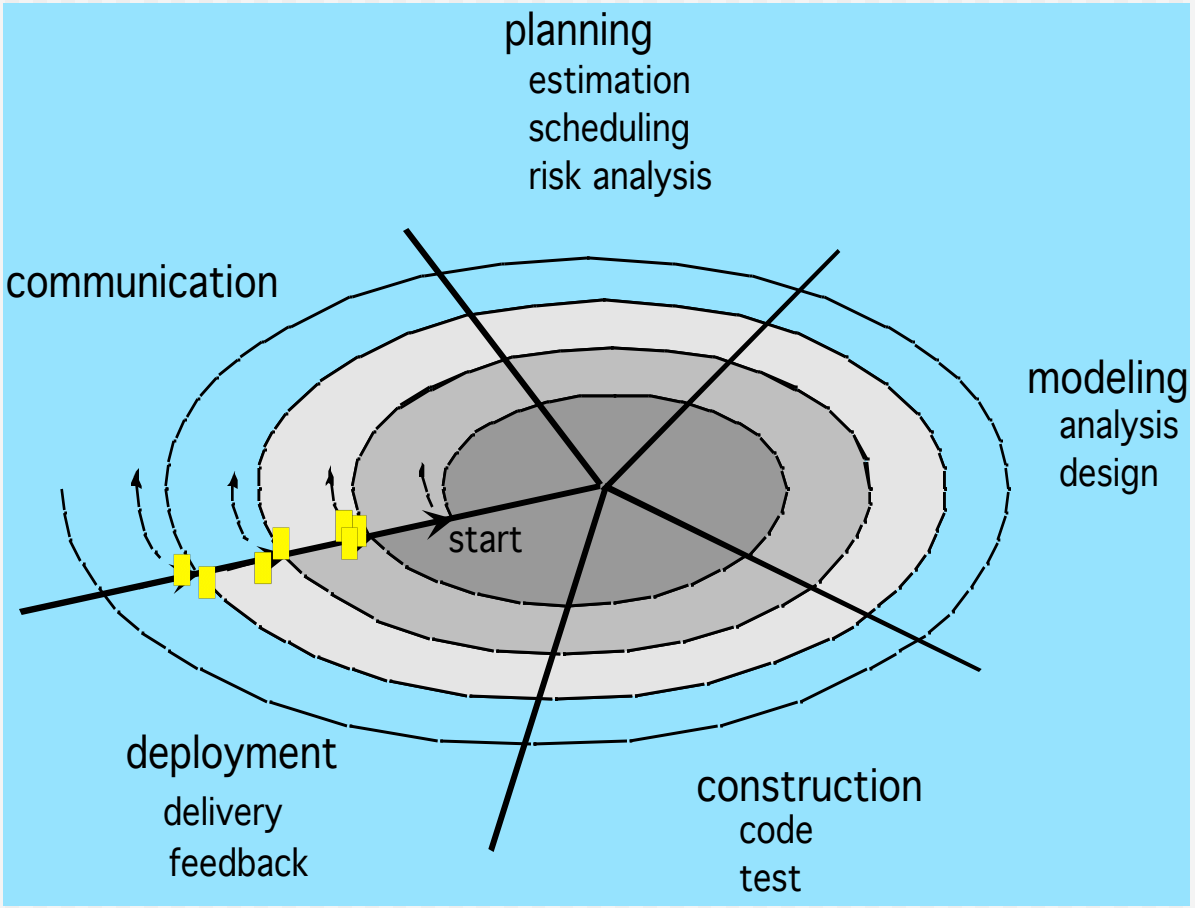
The Incremental Model



Evolutionary Models: Prototyping



Evolutionary Models: The Spiral



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**

That is, while there is value in the items on the right, we value the items on the left more."

Kent Beck et al

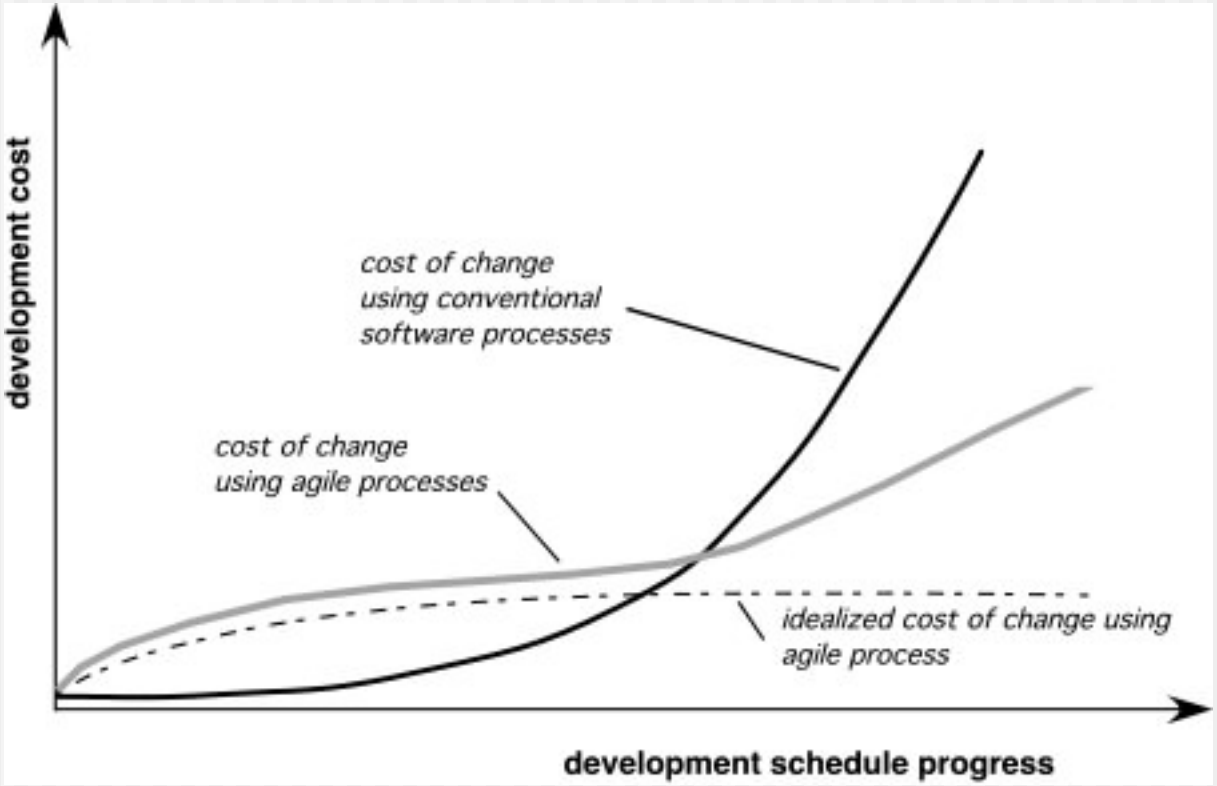
What is “Agility”?

- Effective (rapid and adaptive) response to change.
- Effective communication among all stakeholders
- Drawing the customer onto the team
- Organizing a team so that it is in control of the work performed

Yielding …

- Rapid, incremental delivery of software

Agility and the Cost of Change



An Agile Process

- Is driven by customer descriptions of what is required (scenarios)
- Recognizes that plans are short-lived
- Develops software iteratively with a heavy emphasis on construction activities
- Delivers multiple ‘software increments’
- Adapts as changes occur

Agility Principles - I

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face–to–face conversation.

Agility Principles - II

7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
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 architectures, requirements, and designs emerge from self–organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Human Factors

- *the process molds to the needs of the people and team,* not the other way around
- key traits must exist among the people on an agile team and the team itself:
  - **Competence.**
  - **Common focus.**
  - **Collaboration.**
  - **Decision-making ability.**
  - **Fuzzy problem-solving ability.**
  - **Mutual trust and respect.**
  - **Self-organization.**

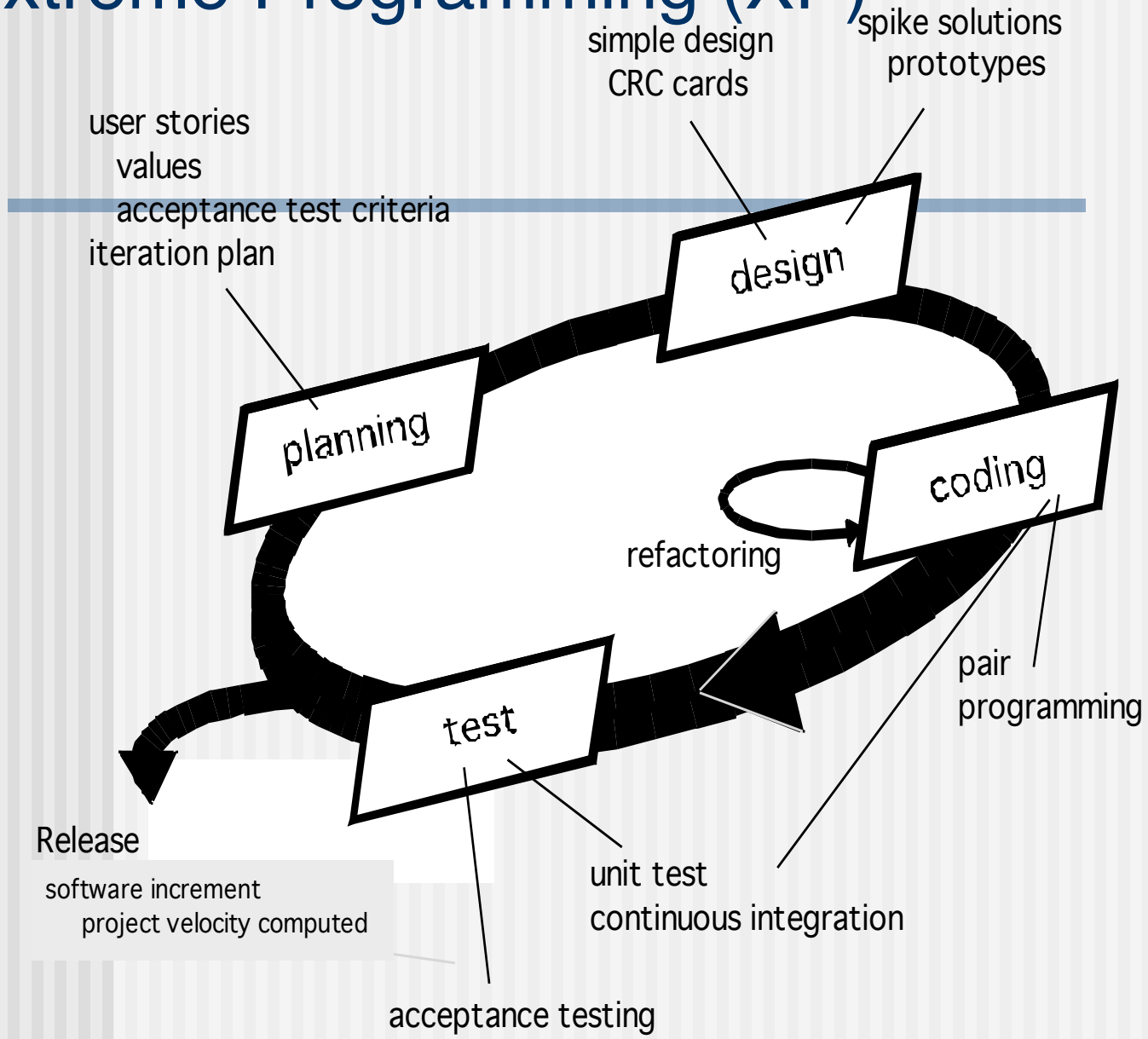
Extreme Programming (XP)

- The most widely used agile process, originally proposed by Kent Beck
- XP Planning
  - Begins with the creation of "user stories"
  - Agile team assesses each story and assigns a cost.
  - Stories are grouped 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.

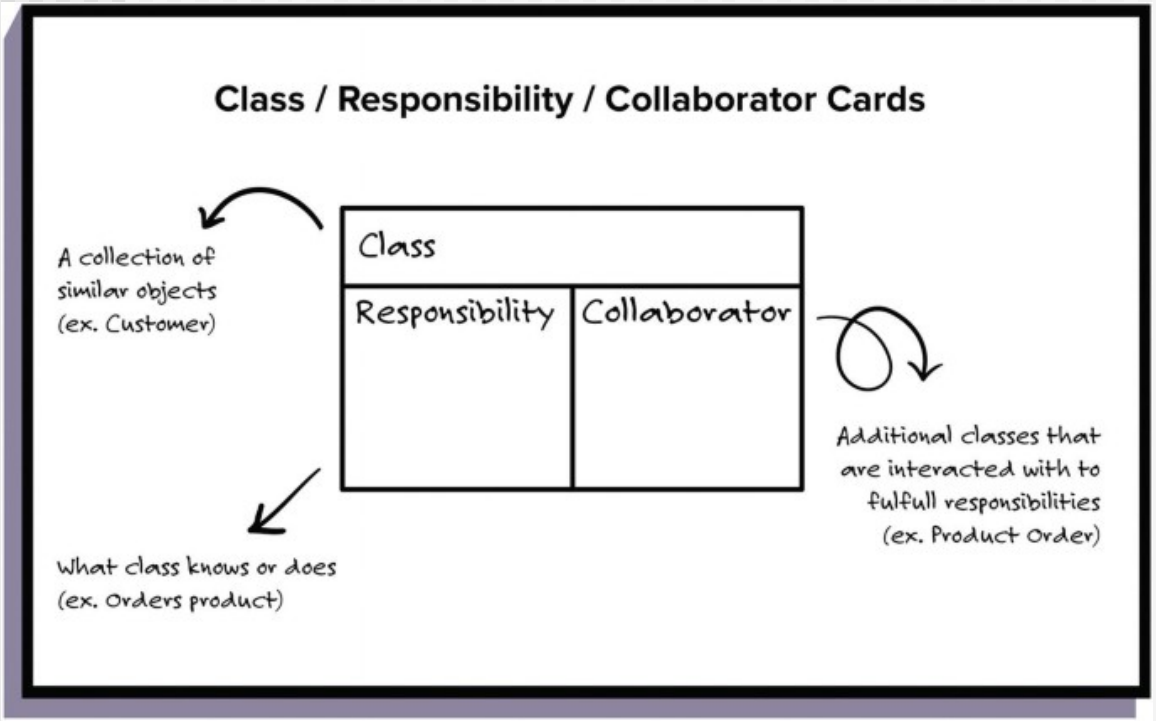
Extreme Programming (XP)

- XP Design
  - Follows the KIS principle
  - Encourage the use of CRC cards.
  - For difficult design problems, suggests the creation of "spike solutions"—a design prototype
  - Encourages "refactoring"—an iterative refinement of the internal program design
- XP Coding
  - Recommends the construction of a unit test for a story *before* coding commences
  - Encourages "pair programming"
- XP Testing
  - All unit tests are executed daily
  - "Acceptance tests" are defined by the customer and executed to assess customer visible functionality

Extreme Programming (XP)



CRC Cards

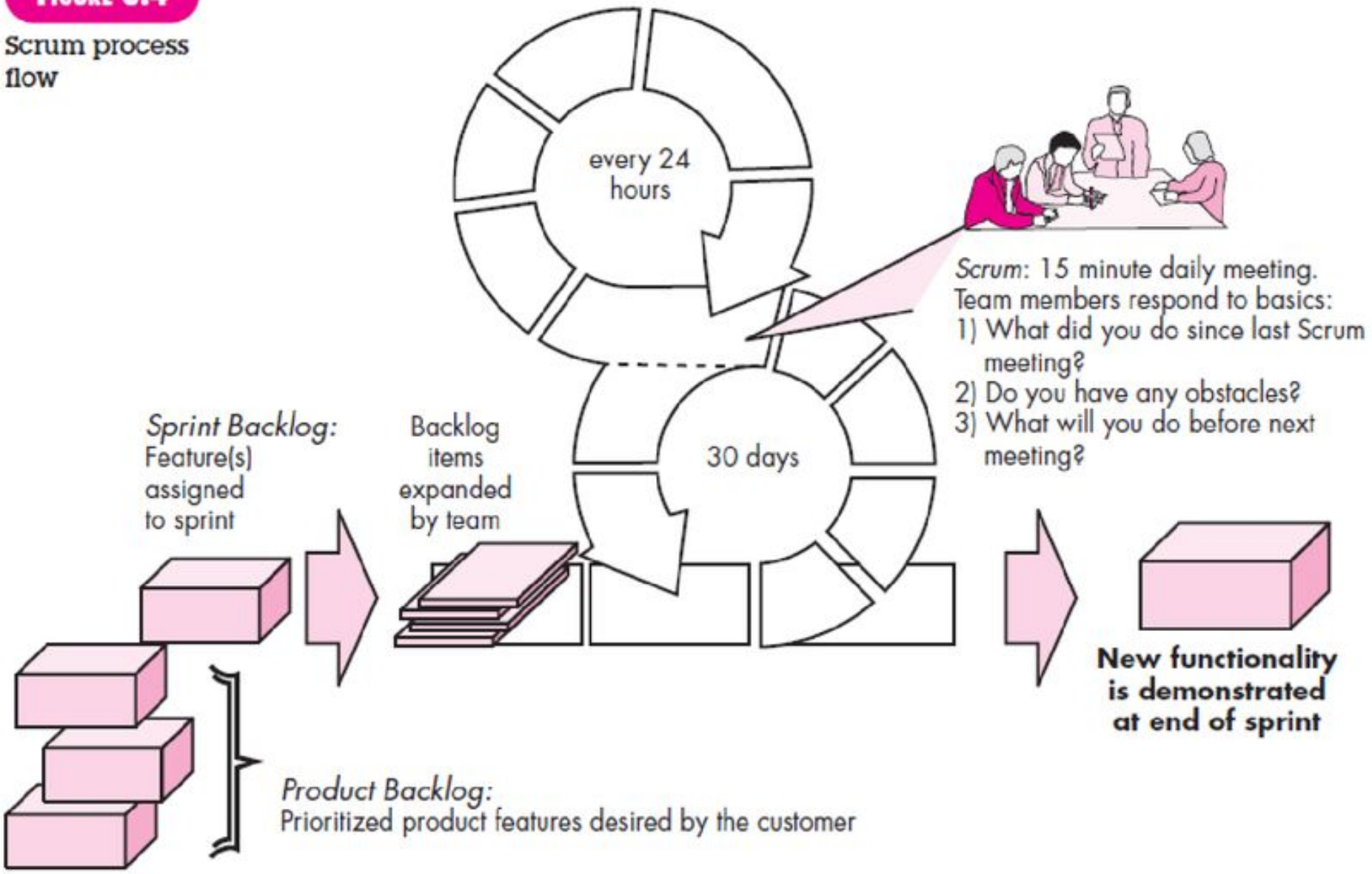

Class / Responsibility / Collaborator Cards
A collection of similar objects (ex. Customer)
Class
Responsibility
Collaborator
Additional classes that are interacted with to fulfull responsibilities (ex. Product Order)
What class knows or does (ex. Orders product)

Scrum

- Originally proposed by Schwaber and Beedle
- Scrum—distinguishing features
  - Development work is partitioned into "packets"
  - Testing and documentation are on-going as the product is constructed
  - Work occurs in "sprints" and is derived from a "backlog" of existing requirements
  - Meetings are very short and sometimes conducted without chairs
  - "demos" are delivered to the customer with the time-box allocated

FIGURE 3.4

Scrum process flow



- **Backlog:** Prioritized list of project requirements or features that provide business value for the customer. Items can be added to backlog at any time.

- **Sprint:** Work units that are required to achieve a requirement defined in a backlog that must be fit into a predefined time box.

  A Sprint is a time box of one month or less. A new Sprint starts immediately after the completion of the previous Sprint.

- **Scrum meeting:** short meeting held daily by the scrum team.(typically 15 minutes).

  1. What did you do since the last team meeting?
  2. What obstacles are you encountering?
  3. What do you plan to accomplish by the next team meeting?

  Scrum master is responsible for scheduling team meetings and assesses the responses from each persons.
- **Release:** When the product is completed, it goes to the Release stage.