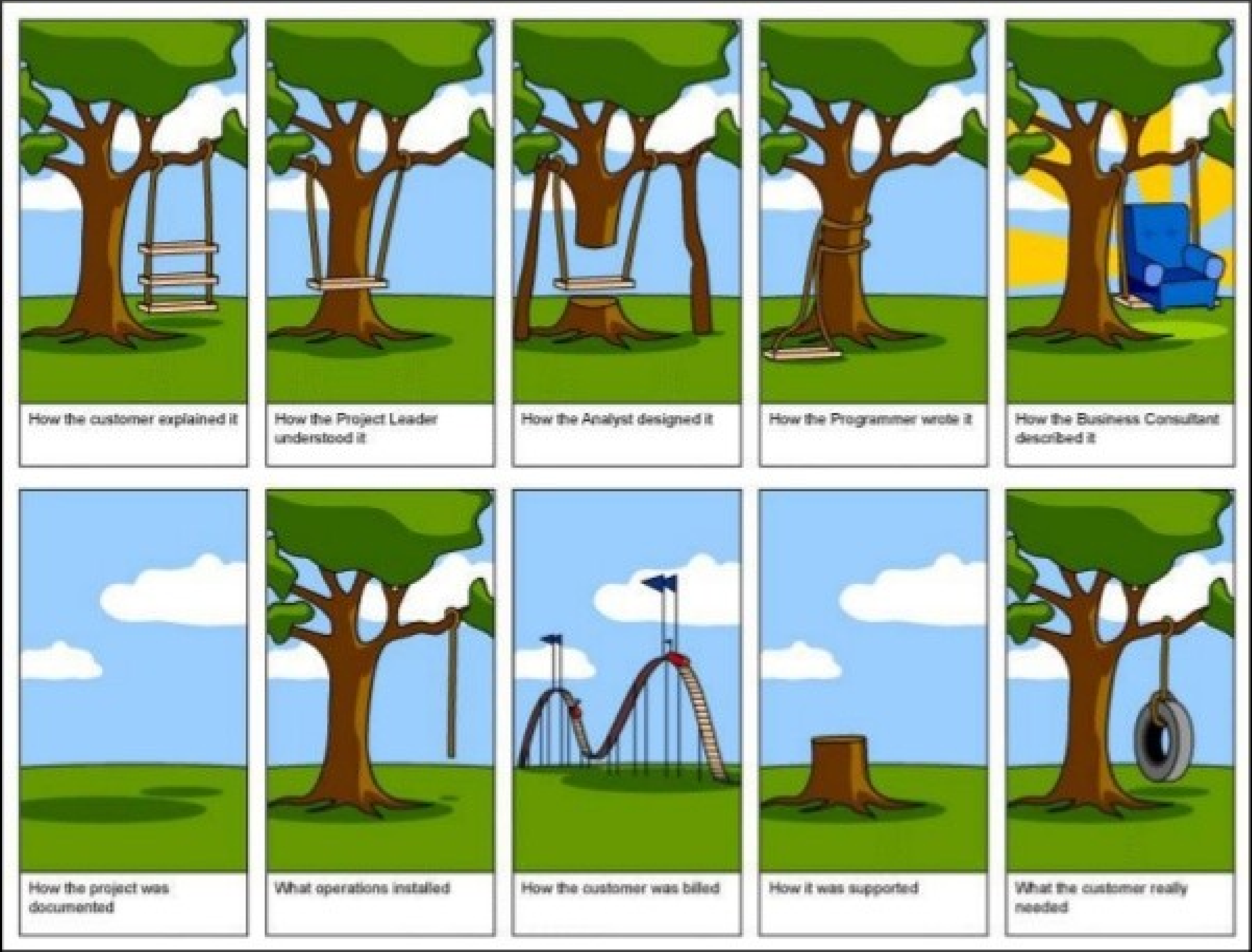
Session – 7

Understanding Requirements

Need to Understand Requirements (Quick Look)

- What is it?
- Who does it?
- Why is it important?
- What are the Steps?
- What is the work product?
- How do I ensure that I’ve done it right?



How the customer explained it

How the Project Leader understood it

How the Analyst designed it

How the Programmer wrote it

How the Business Consultant described it

How the project was documented

What operations installed

How the customer was billed

How it was supported

What the customer really needed

Types of Requirements

Broadly requirements can be categorized into

- Functional
- Non Functional

Requirement Engineering

Requirement Engineering is the process of defining, documenting and maintaining the requirements. It is a process of gathering and defining service provided by the system.

Stages in Requirements Engineering

- Elicitation
- Elaboration
- Negotiation
- Specification
- Validation
- Requirements management

Contd.,

- Inception—ask a set of questions that establish …
  - basic understanding of the problem
  - the people who want a solution
  - the nature of the solution that is desired, and
  - the effectiveness of preliminary communication and collaboration between the customer and the developer
- Elicitation—elicit requirements from all stakeholders
- Elaboration—create an analysis model that identifies data, function and behavioral requirements
- Negotiation—agree on a deliverable system that is realistic for developers and customers

Contd.,

- Specification—can be any one (or more) of the following:
  - A written document
  - A set of models
  - A formal mathematical
  - A collection of user scenarios (use-cases)
  - A prototype
- Validation—a review mechanism that looks for
  - errors in content or interpretation
  - areas where clarification may be required
  - missing information
  - inconsistencies (a major problem when large products or systems are engineered)
  - conflicting or unrealistic (unachievable) requirements.
- Requirements management

Establish the Ground Work

- **Identify Stakeholders**

A stakeholder as " anyone who benefits in a direct or indirect way from the system which is being developed.

- **Recognizing multiple viewpoints**

  - The Marketing group
  - Business Mangers
  - Software Engineers
  - Support Engineers

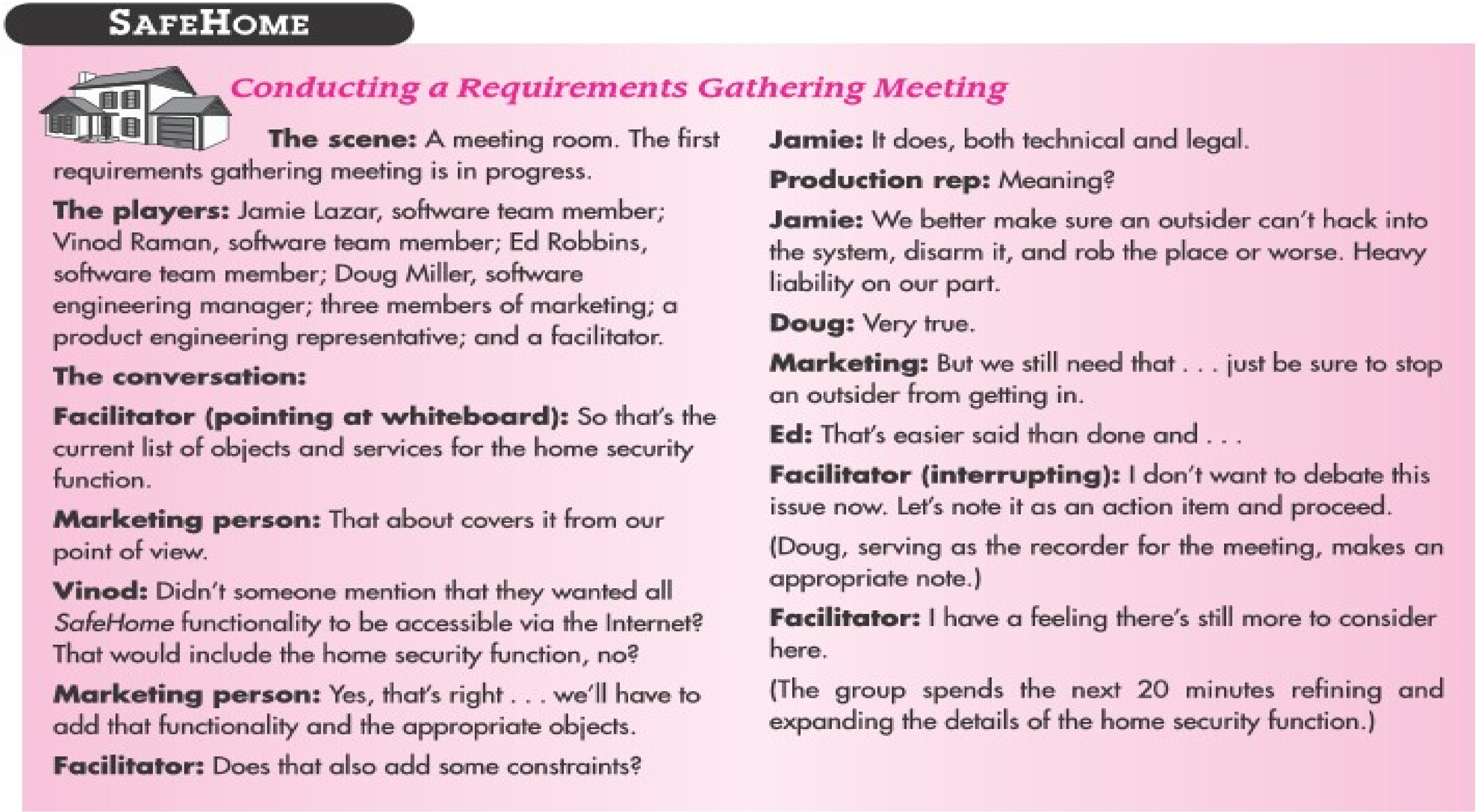
- **Working toward collaboration**
- **Asking the First Questions**

Eliciting Requirements

Requirements elicitation (also called requirements gathering) combines elements of problem solving, elaboration, negotiation, and specification. In order to encourage a collaborative , team-oriented approach to requirements gathering, stakeholders work together to identify the problem, propose elements of the solution, negotiate different approaches and specify a preliminary set of solution requirement.

- **Collaborative Requirements Gathering.**
- **Quality Function Deployment**
- **Usage Scenarios**
- **Elicitation work products**

Collaborative Requirements Gathering.



SafeHome

Conducting a Requirements Gathering Meeting

The scene: A meeting room. The first requirements gathering meeting is in progress.

The players: Jamie Lazar, software team member; Vinod Raman, software team member; Ed Robbins, software team member; Doug Miller, software engineering manager; three members of marketing; a product engineering representative; and a facilitator.

The conversation:

Facilitator (pointing at whiteboard): So that's the current list of objects and services for the home security function.

Marketing person: That about covers it from our point of view.

Vinod: Didn't someone mention that they wanted all *SafeHome* functionality to be accessible via the Internet? That would include the home security function, no?

Marketing person: Yes, that's right . . . we'll have to add that functionality and the appropriate objects.

Facilitator: Does that also add some constraints?

Jamie: It does, both technical and legal.

Production rep: Meaning?

Jamie: We better make sure an outsider can't hack into the system, disarm it, and rob the place or worse. Heavy liability on our part.

Doug: Very true.

Marketing: But we still need that . . . just be sure to stop an outsider from getting in.

Ed: That's easier said than done and . . .

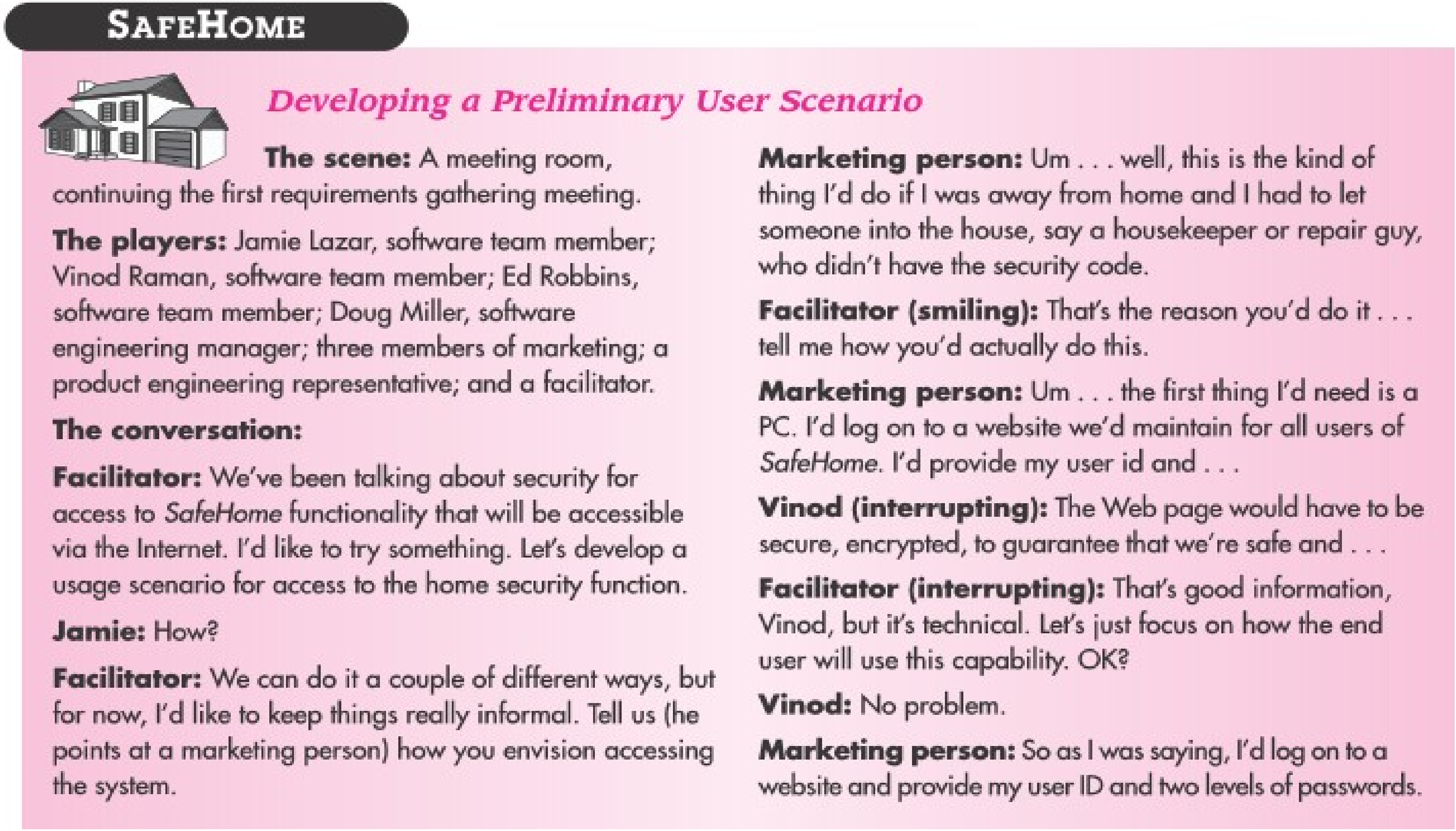
Facilitator (interrupting): I don't want to debate this issue now. Let's note it as an action item and proceed.

(Doug, serving as the recorder for the meeting, makes an appropriate note.)

Facilitator: I have a feeling there's still more to consider here.

(The group spends the next 20 minutes refining and expanding the details of the home security function.)

Usage Scenarios



SafeHome

Developing a Preliminary User Scenario

The scene: A meeting room, continuing the first requirements gathering meeting.

The players: Jamie Lazar, software team member; Vinod Raman, software team member; Ed Robbins, software team member; Doug Miller, software engineering manager; three members of marketing; a product engineering representative; and a facilitator.

The conversation:

Facilitator: We've been talking about security for access to *SafeHome* functionality that will be accessible via the Internet. I'd like to try something. Let's develop a usage scenario for access to the home security function.

Jamie: How?

Facilitator: We can do it a couple of different ways, but for now, I'd like to keep things really informal. Tell us (he points at a marketing person) how you envision accessing the system.

Marketing person: Um . . . well, this is the kind of thing I'd do if I was away from home and I had to let someone into the house, say a housekeeper or repair guy, who didn't have the security code.

Facilitator (smiling): That's the reason you'd do it . . . tell me how you'd actually do this.

Marketing person: Um . . . the first thing I'd need is a PC. I'd log on to a website we'd maintain for all users of *SafeHome*. I'd provide my user id and . . .

Vinod (interrupting): The Web page would have to be secure, encrypted, to guarantee that we're safe and . . .

Facilitator (interrupting): That's good information, Vinod, but it's technical. Let's just focus on how the end user will use this capability. OK?

Vinod: No problem.

Marketing person: So as I was saying, I'd log on to a website and provide my user ID and two levels of passwords.

Questions:

1. Explain the requirements engineering process with help of a diagram? And also explain the spiral model of requirements?
2. Explain in detail on requirements Elicitation and Analysis process?
3. Elaborate on requirements Discovery and view points?
4. What are the goals of requirements engineering process?

Session – 8

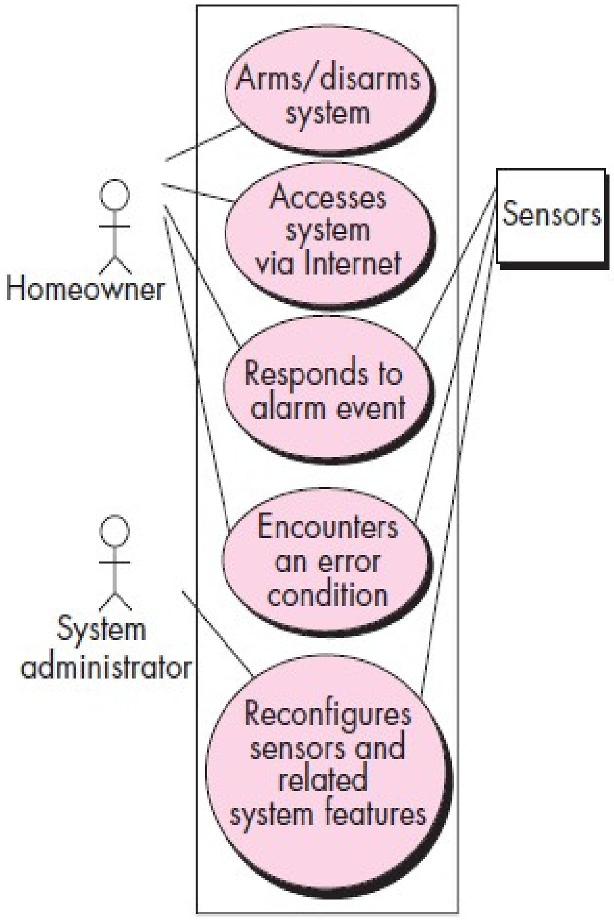
Building Requirements Model

Elements of the Requirements Model

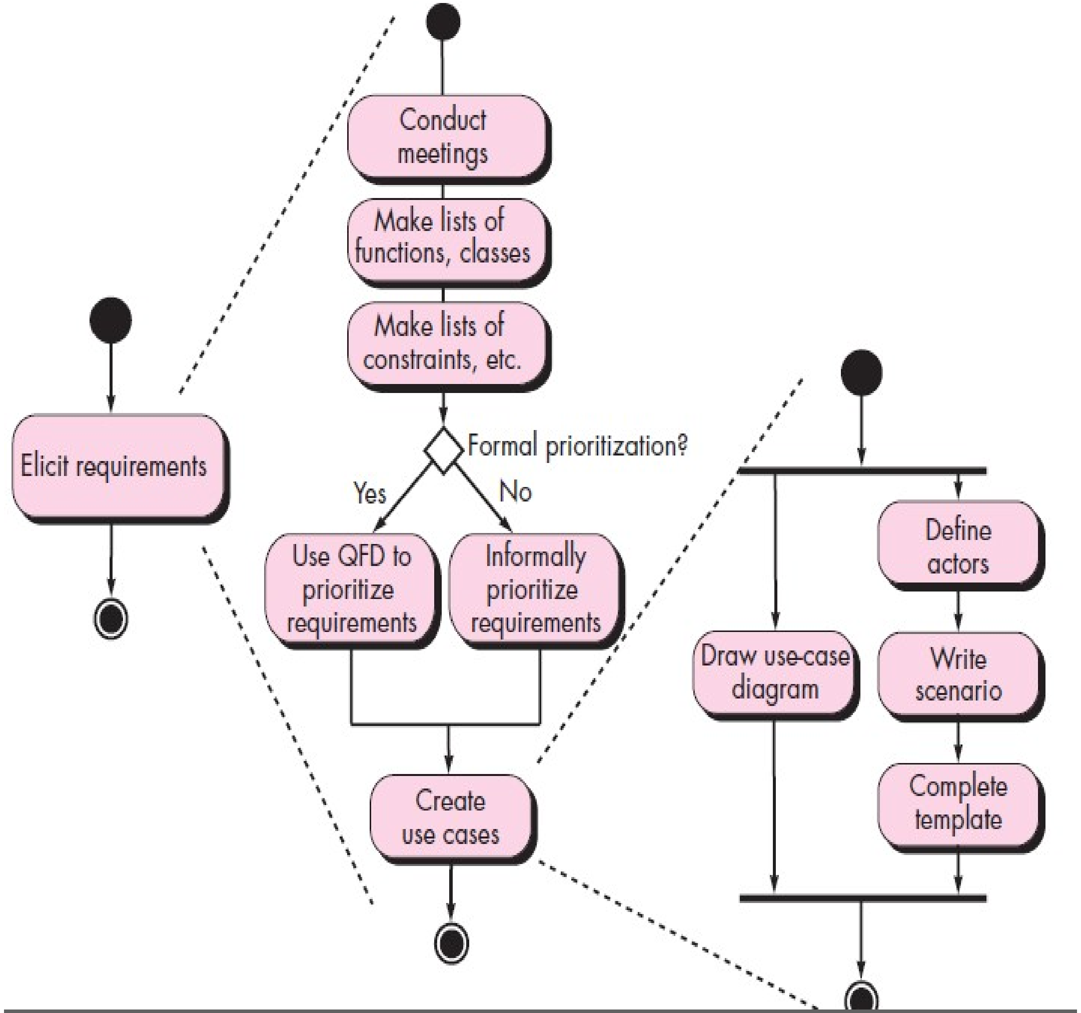
The specific elements of the requirements model are dictated by the analysis modeling method that is to be used. However, a set of generic elements is common to most requirements models.

- **Scenario-based elements**
- **Class-based elements**
- **Behavioral elements**
- **Flow-oriented elements**
- **Analysis Patterns**

Scenario-based elements

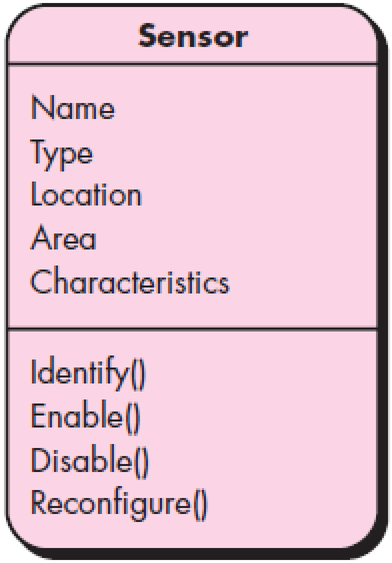

Arms/disarms system
Accesses system via Internet
Responds to alarm event
Encounters an error condition
Reconfigures sensors and related system features
Sensors
Homeowner
System administrator

Use Case Diagram for Safe home



Activity Diagram for safe home

Class-based elements



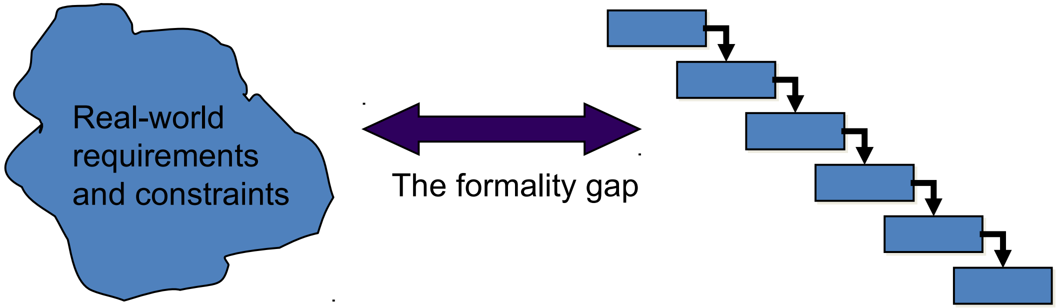
Class Diagram for Sensor

Negotiating Requirements

Boehm [Boe98] defines a set of negotiation activities at the beginning of each software process iteration. Rather than a single customer communication activity, the following activities are defined:

1. **Identification of the system or subsystem’s key stakeholders.**
2. **Determination of the stakeholders’ “win conditions.”**
3. **Negotiation of the stakeholders’ win conditions to reconcile them into a set of win-win conditions for all concerned (including the software team).**

Validating Requirements



Verification

designing the product right

Validation

designing the right product

The formality gap

validation will always rely to some extent on subjective means of proof

Management and contractual issues

design in commercial and legal contexts

Validating Requirements

In the Validation process few questions should be asked and answered to ensure that the requirements model is an accurate reflection of stakeholder needs and that it provides a solid foundation for design.

Validating Requirements

1. Is each requirement consistent with the overall objectives for the system/product?
2. Have all requirements been specified at the proper level of abstraction? That is, do some requirements provide a level of technical detail that is inappropriate at this stage?
3. Is the requirement really necessary or does it represent an add-on feature that may not be essential to the objective of the system?
4. Is each requirement bounded and unambiguous?
5. Does each requirement have attribution? That is, is a source (generally, a specific individual) noted for each requirement?

Cont..

6. Do any requirements conflict with other requirements?
7. Is each requirement achievable in the technical environment that will house the system or product?
8. Is each requirement testable, once implemented?
9. Does the requirements model properly reflect the information, function, and behavior of the system to be built?
10. Has the requirements model been "partitioned" in a way that exposes progressively more detailed information about the system?
11. Have requirements patterns been used to simplify the requirements model? Have all patterns been properly validated? Are all patterns consistent with customer requirements?

SRS Vs User Stories

User Stories

- Provide a small scale and easy to use presentation of information. Are generally formulated in the everyday language of the user and contain little detail, thus remaining open to interpretation. They should help the reader understand what the software should accomplish.
- Must be accompanied by acceptance testing procedures for clarification of behavior where stories appear ambiguous.

SRS

Software requirements specification establishes the basis for an agreement between customers and contractors or suppliers on how the software product should function.

Questions

1.Develop a complete use case for one of the following activities:

   a. Making a withdrawal at an ATM

   b. Using your charge card for a meal at a restaurant

   c. Buying a stock using an on-line brokerage account

   d. Searching for books (on a specific topic) using an on-line bookstore.

2. Develop a checklist for conducting a requirements gathering meeting.
3. Why do we say that the requirements model represents a snapshot of a system in time?
4. Why is it that many software developers don’t pay enough attention to requirements engineering? Are there ever circumstances where you can skip it?
5. Discuss some of the problems that occur when requirements must be elicited from three or four different customers.
6. Define the metrics for specifying the NON-functional Requirements.
7. Discuss in detail the user Requirements?
8. Elaborate on software requirements document in detail?

Session – 9

Agile

Agile Development

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

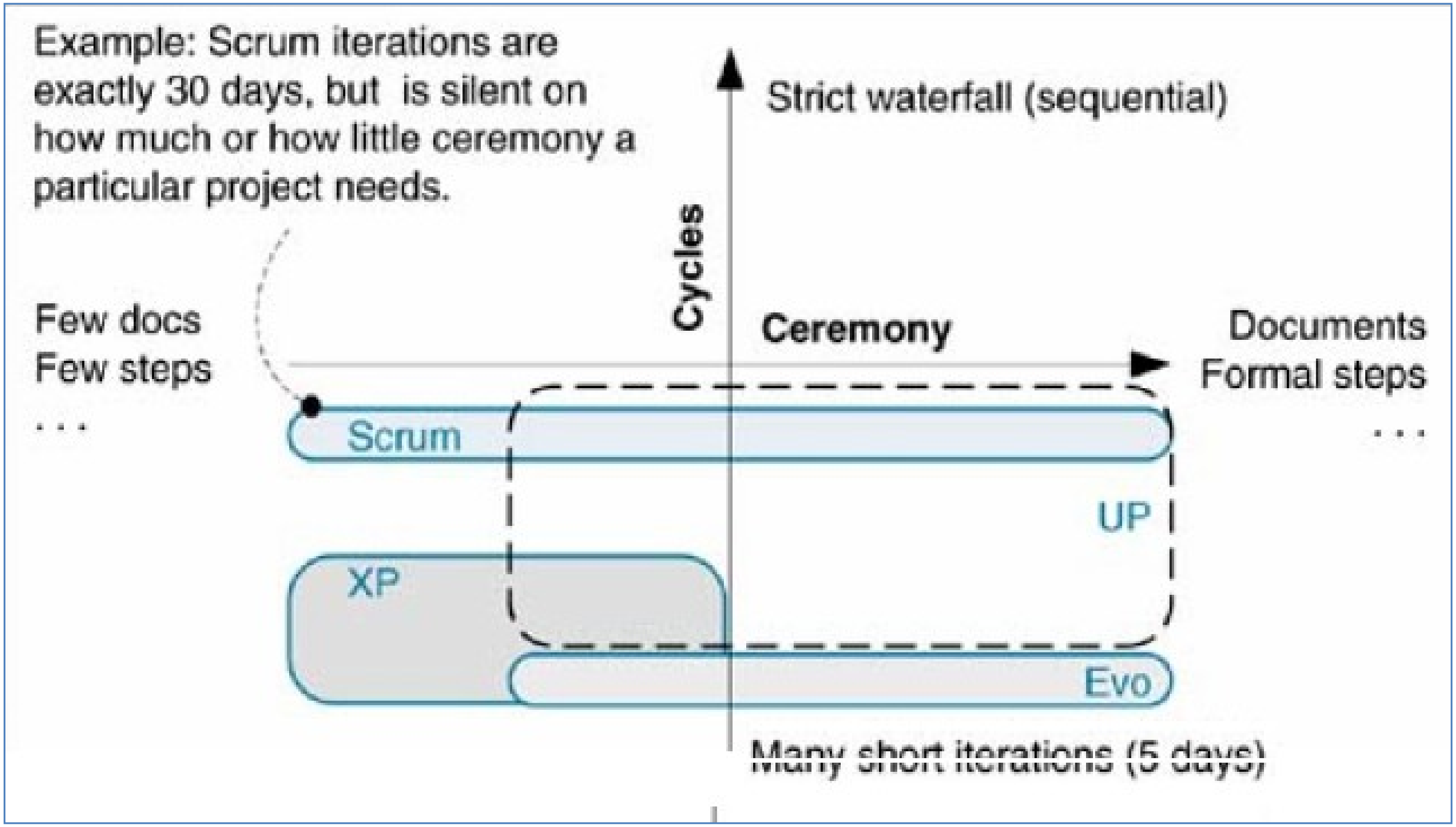
Agile Development

Agile development methods apply time boxed iterative and evolutionary development, adaptive planning, promote evolutionary delivery, and include other values and practices that encourage agility.

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

Classification Methods


Example: Scrum iterations are exactly 30 days, but is silent on how much or how little ceremony a particular project needs.
Strict waterfall (sequential)
Cycles
Ceremony
Few docs
Few steps
. . .
Documents
Formal steps
. . .
Scrum
UP
XP
Evo
Many short iterations (5 days)

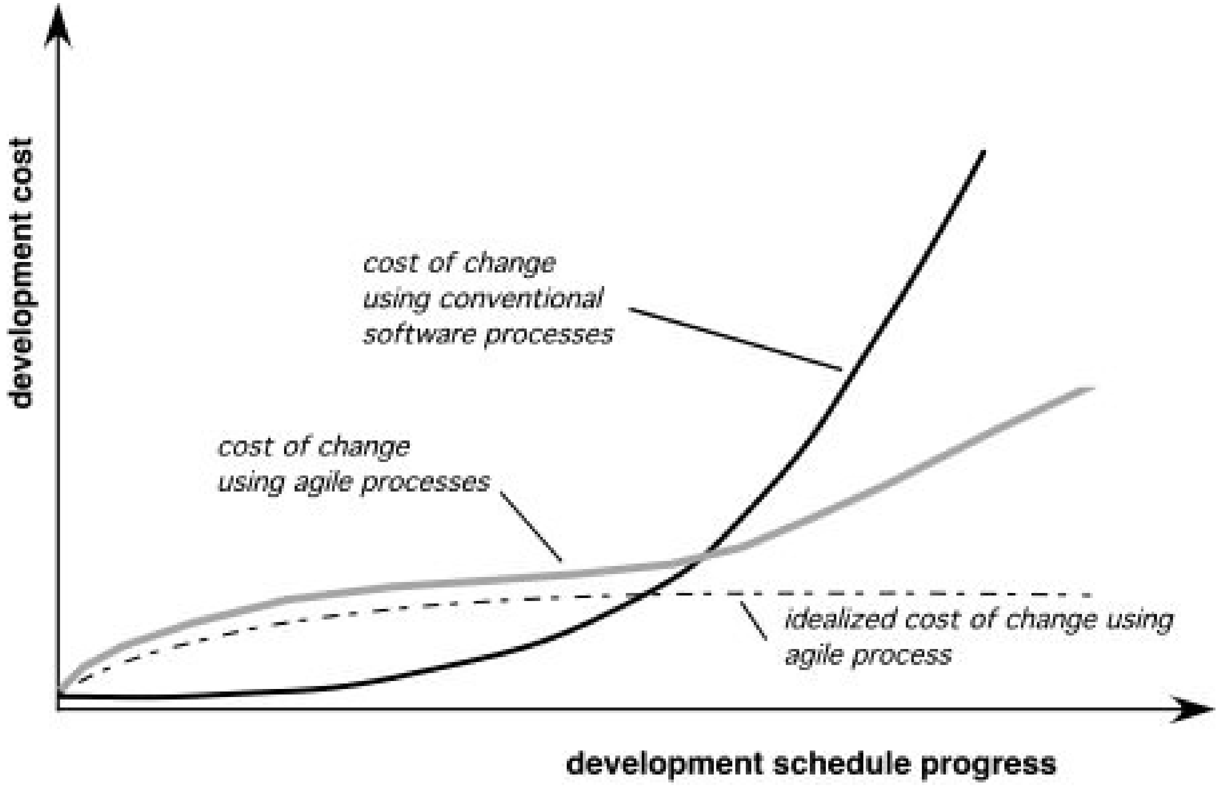
The Manifesto for
Agile Software Development

"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."

Agility and the Cost of Change



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.

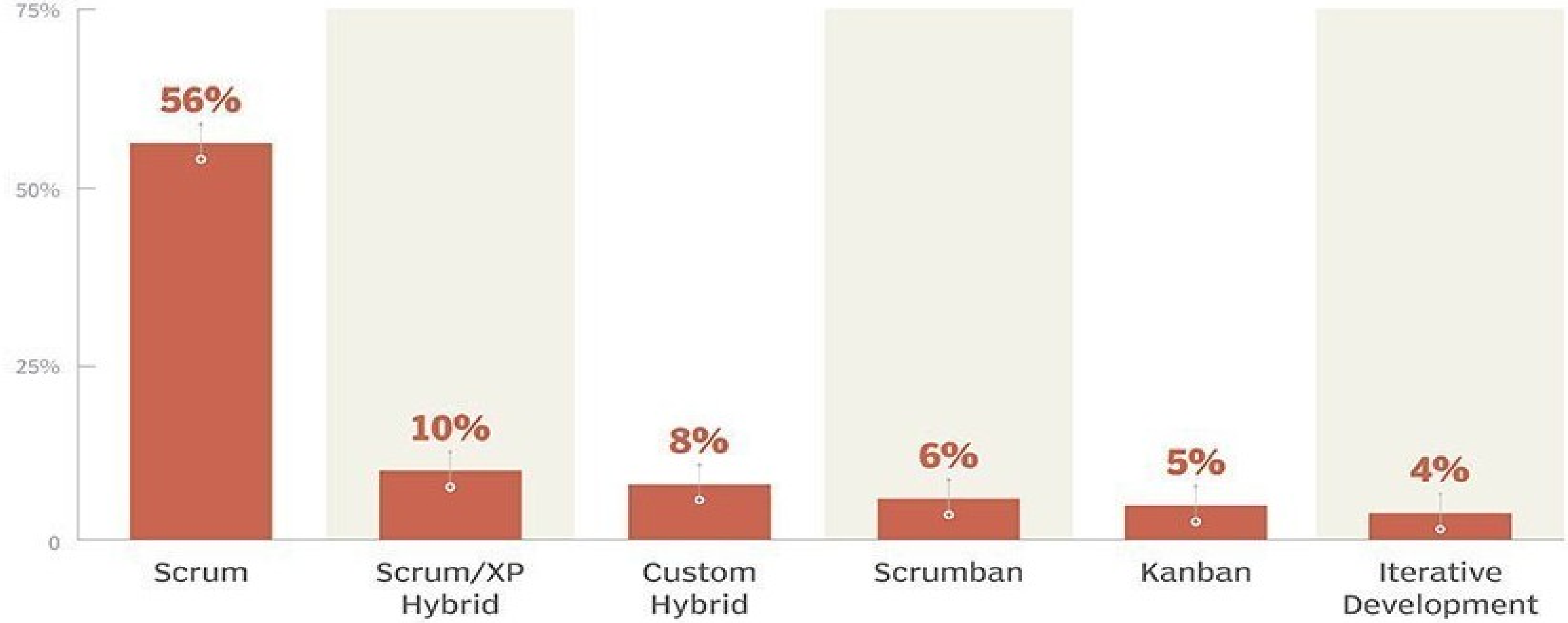
Agility Principles - II

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.

Agile Project Management

Agile Project Management (APM) is an iterative approach to planning and guiding project processes.

What is the most popular Agile process?



Agile Project Management

Jim Highsmith, an Agile Alliance founder and creator of the Adaptive Software Development method, summarizes nine principles for the agile project manager [Highsmith02]:

1. Deliver something useful to the client; check what they value.
2. Cultivate committed stakeholders.
3. Employ a leadership-collaboration style.
4. Build competent, collaborative teams.
5. Enable team decision making.
6. Use short time boxed iterations to quickly deliver features.
7. Encourage adaptability.
8. Champion technical excellence.
9. Focus on delivery activities, not process-compliance activities.

Questions

1. Define Agility? Explain Agile process.
2. Explain Classification of methods in agile development.
3. List out Agile principles.
4. Illustrate the agile manifesto.
5. explain agile project management with help of Safe Home application

Session – 10

Agile

Topics

- Simple Practices and Project Tools
- Empirical vs. Defined & Prescriptive Process
- Principle-Based versus Rule-Based
- Agile Hype?
- Specific Agile Methods

Simple Practices and Project Tools

Simplicity—the art of maximizing the amount of work not done—is essential.

List Of The Best Agile Project Management Tools

Monday.com
Nifty
Wrike
SpiraTeam
Atlassian Jira
Active Collab
Agilo for Scrum
SpiraTeam
Pivotal Tracker
VSTS
Icescrum
Gravity
SprintGround
VersionOne
Taiga

Empirical
vs.
Defined & Prescriptive Process

In general, agile methods promote empirical rather than defined processes.

A defined process (also known as a **prescriptive process**) are suitable for predictable manufacturing domains.

Empirical processes are used for high-change and unstable domains.

Principle-Based
vs.
Rule-Based

Agile methods are more principle-based than rule-based.

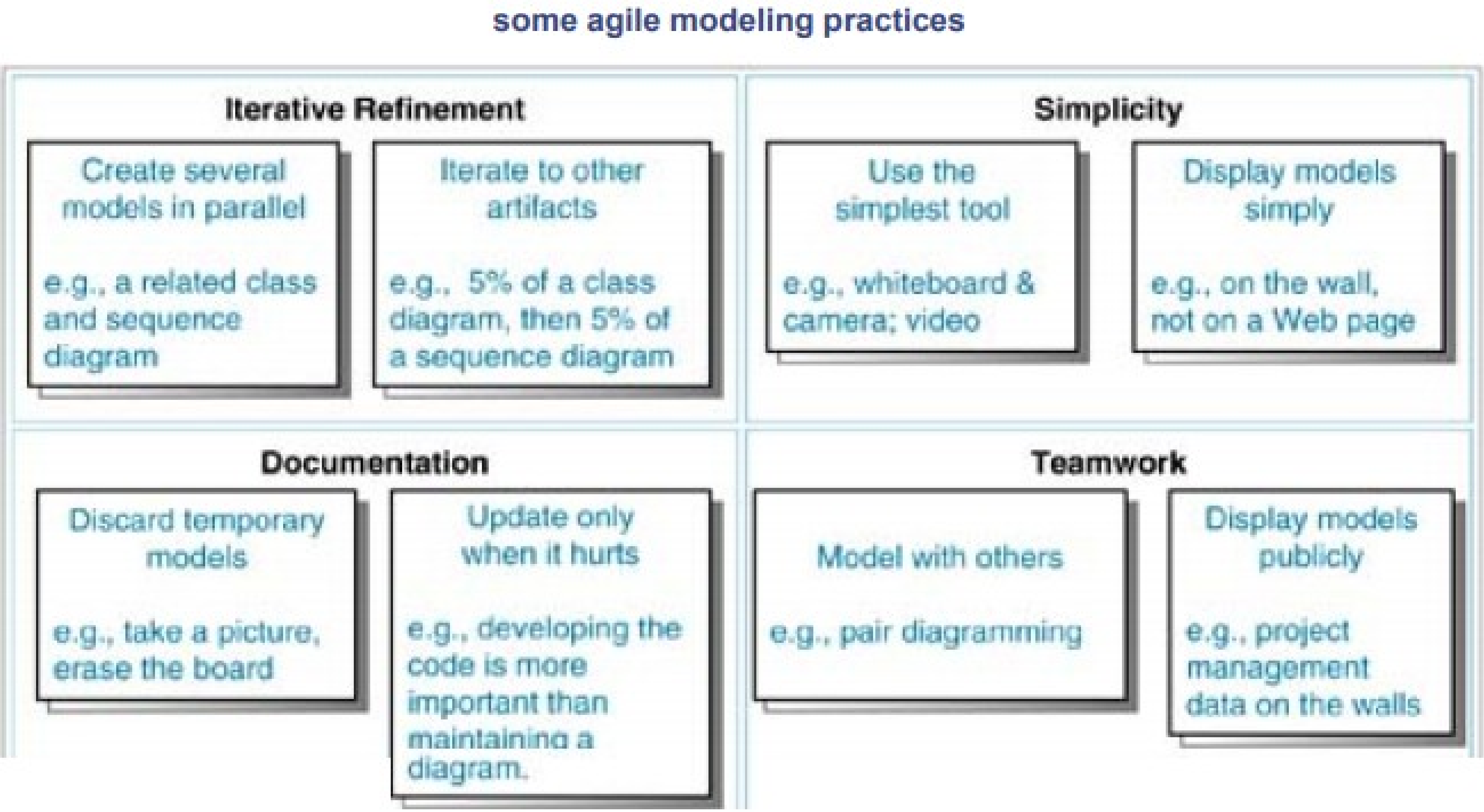
Agile Methods

- Scrum
- XP
- Crystal Methods

Other Methods and Practices

- Adaptive Software Development
- Dynamic Solution Delivery Model
- Feature Driven Development
- Lean Development
- Pragmatic Programming

Agile Modeling



some agile modeling practices

Iterative Refinement

- Create several models in parallel

  e.g., a related class and sequence diagram

- Iterate to other artifacts

  e.g., 5% of a class diagram, then 5% of a sequence diagram

Simplicity

- Use the simplest tool

  e.g., whiteboard & camera; video

- Display models simply

  e.g., on the wall, not on a Web page

Documentation

- Discard temporary models

  e.g., take a picture, erase the board

- Update only when it hurts

  e.g., developing the code is more important than maintaining a diagram.

Teamwork

- Model with others

  e.g., pair diagramming

- Display models publicly

  e.g., project management data on the walls

Questions

1. List out Various Agile project Management Tools and explain working functionality.
2. Differentiate between Empirical processes and defined process
3. Define principle based approach and rule based approach?
4. Agile Methods are Invent Iterative Development or not? Justify.
5. Write Short notes about Agile methods.
6. Compare and contrast Scrum and XP.

Session – 11

Extreme Programming

Extreme Programming

Extreme Programming (XP) is a well-known agile method; it emphasizes collaboration, quick and early software creation, and skillful development practices.

Extreme Programming

XP founded on four values:

communication, simplicity, feedback, and courage.

In addition to IID, it recommends **12 core practices**:

1. **Planning Game**
2. **small, frequent releases**
3. **system metaphors**
4. **simple design**
5. **testing**
6. **frequent refactoring**
7. **pair programming**
8. **team code ownership**
9. **continuous integration**
10. **sustainable pace**
11. **whole team together**
12. **coding standards**

Extreme Programming (XP) Model

- XP Planning
  - – Begins with the creation of “user stories”
  - – 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

Extreme Programming (XP)

XP Design

- Follows the KISS(Keep it Simple, Stupid) principle
- Encourage the use of CRC cards (Class Responsibility Collaboration)
- For difficult design problems, suggests the creation of "spike solutions"—a design prototype(A Working Model)
- Encourages "refactoring"—an iterative refinement of the internal program design

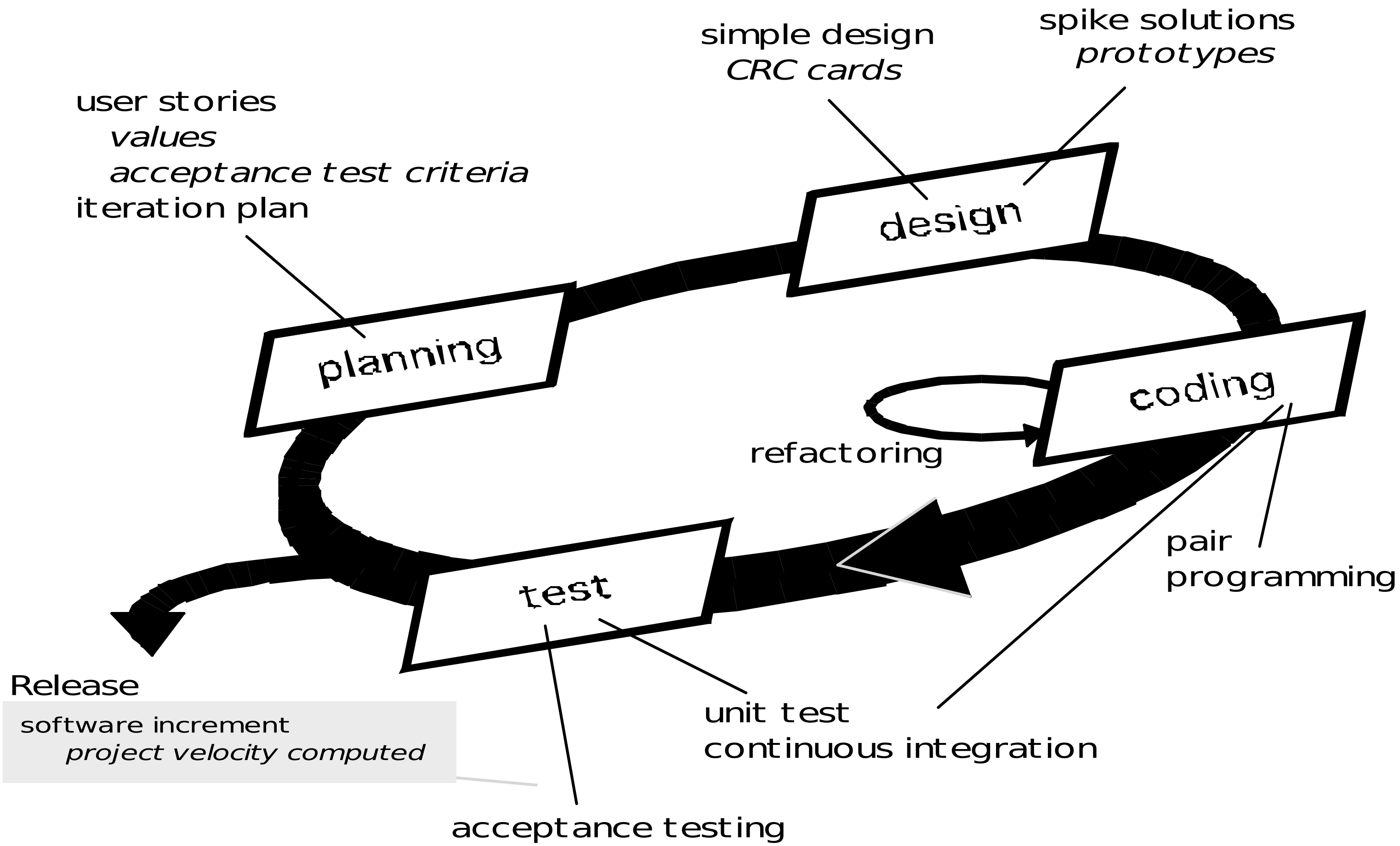
XP Coding

- Recommends the construction of a unit test for a store *before* coding commences
- Encourages "pair programming"

XP Testing

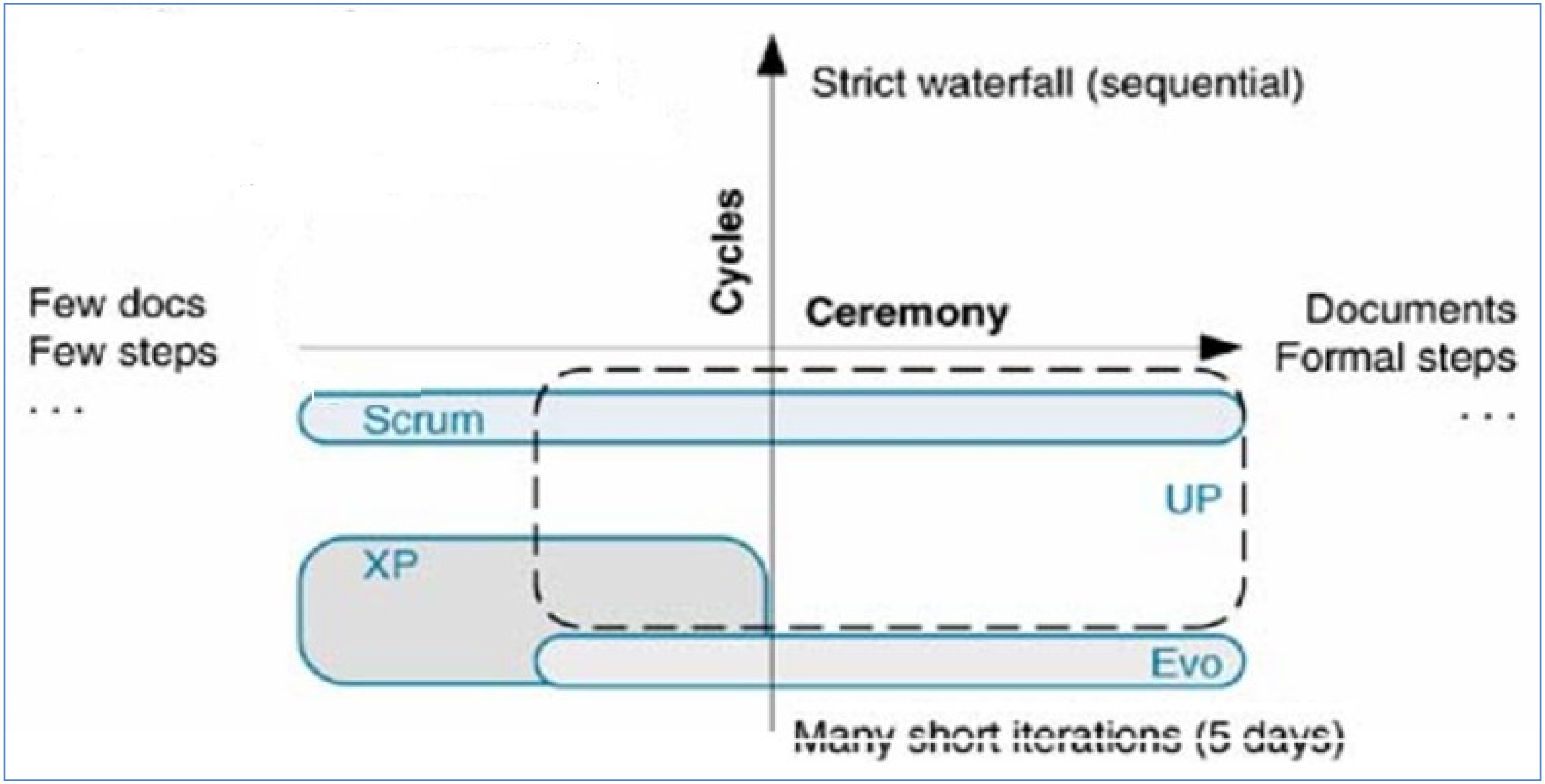
- All unit tests are executed daily
- "Acceptance tests" are defined by the customer and executed to assess customer visible functionality

Extreme Programming (XP)



Extreme Programming (XP)

Method Overview



Work products Roles and Practices

Work Products (Non Software):

-Requirements – Story Cards

-Design- CRC Cards, Sketches

-Project Management- Task list, Visible graphs

Roles:

-Custumer

-Development

-Management

-Other

Practices:

-Requirements

-Design

-Implementation

-Test and Verification

-Project Management

-Configuration & Change management environment

Questions

1. List out Core practices in Extreme Programming.
2. Explain Extreme programming Life cycle.
3. Illustrate XP Work products Roles and Practices for safe home application.
4. Write extreme programming advantages and disadvantages?
5. How User requirements are expressed in Extreme Programming.

Session – 12

Extreme Programming

Extreme Programming

Adoption Strategies: XP recommends adoption like this

- Pick the worst project or problem.
- Apply XP until solved.
- Repeat.

If all the XP practices can't be swallowed at once, Beck recommends starting with:

- whole team together in a common project room
- test-first development
- acceptance tests written/owned by customers
- Planning Game
- pair programming

Extreme Programming

Fact versus Fantasy

Process is only a second-order effect. The unique people, their feelings and qualities, are more influential.

Extreme Programming

Strengths versus "Other"

Strengths:

- Practical, high-impact development techniques, many of which are easily and sustainably adopted by developers (e.g., continuous integration, test-driven development).
- Emphasizes customer participation and steering.
- Evolutionary and incremental requirements and development, and adaptive behavior.
- Programmers estimate the tasks they have chosen, and the schedule follows this, not vice versa (i.e., scheduling is rational).
- Emphasizes communication between all stakeholders.

- Emphasizes quality through many practices.

  Test-first development, continuous integration, and

  team code ownership are examples.
- Clarifies what is an acceptable system by requiring the customer to define the acceptance tests.
- Daily measurement, and developers are involved in measuring and defining what to measure.
- Every iteration, developers get practice (during the Planning Game) identifying tasks and estimating them, leading to improvement in these vital skills.
- Frequent, detailed reviews and inspections, as all significant work is done in pairs. Inspection is strongly correlated with reduced defect levels.

Extreme Programming

Strengths versus "Other"

Other:

- Requires the presence of onsite customers (or proxies).
- Relies on oral history for knowing the design and requirements details.
- The XP practices are interdependent and mutually supporting.
- No standard way to describe or document the software design as a learning aid.
- Some developers do not want to do pair programming.
- Many projects will need a set of documents other than code. XP does not define what these may be, and thus each project may create ones with similar intent, but varying names and content.
- Lack of architecture-oriented emphasis in the early iterations. Lack of architectural design methods. XP advocates claim simple design and refactoring lead to the same goal.

Questions

1. Write History of Extreme Programming.
2. List out Strengths of the Extreme Programming.
3. Identify activities involved in Extreme Programming.
4. Differentiate pair programming with normal programming.
5. List out Adoption Strategies in XP