What is agility?

Agility means characteristics of being dynamic, content specific, aggressively change embracing and growth oriented.

Agile Software engineering

Agile software engineering combines a philosophy and a set of development guidelines. The philosophy encourages customer satisfaction and early incremental delivery of software, small highly motivated project teams, informal methods, minimal software engineering work products and overall development simplicity. The development guidelines stress delivery over analysis and design and active continuous communication between developers and customers. The team of software engineers and other project stakeholders work together as an agile team (a team that is self organizing and in control of its own destiny. An agile team fosters communication and collaboration among all who serve on it.

Agile Process

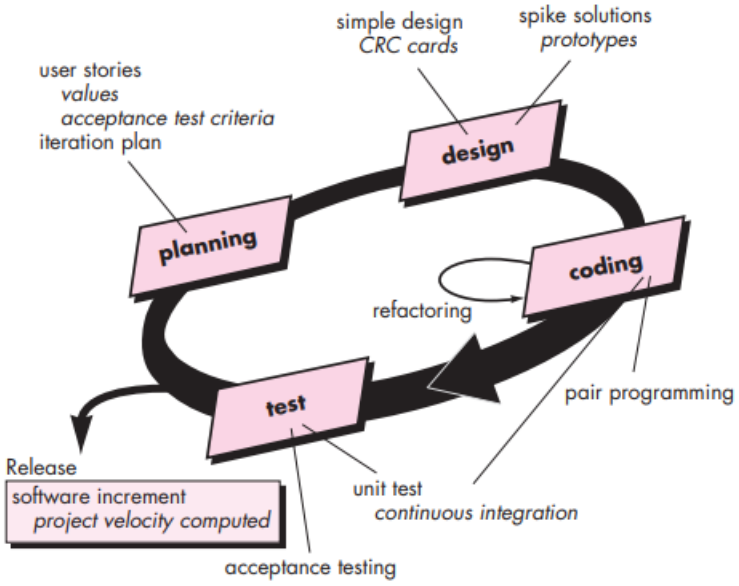
The Processes which are adaptable of changes in requirements, which have incrementality and work on unpredictability. These processes are based on three assumptions which all do refer to the unpredictability in different stages of software process development such unpredictability at time requirements, at analysis and design or at time construction.So these processes are adaptable at all stages on SDLC.

.

Agile Process models

1. Extreme Programming(XP)
2. Adaptive Software development(ASD)
3. Dynamic software Development Method(DSDM)
4. Scrum
5. Crystal
6. Lean software Development(LSD)
7. Feature Driven development (FDD)
8. Agile Modeling(AM)
9. (AUP)Agile unified Process

1. Extreme Programming(XP)



Extreme Programming uses an object-oriented approach as its preferred development paradigm and encompasses a set of rules and practices that occur within the context of four framework activities: planning, design, coding, and testing. Key XP activities are summarized in the paragraphs that follow

Planning: The planning activity begins with listening—a requirements gathering activity that enables the technical members of the XP team to understand the business context for the software and to understand the scope of required output and major features and functionality. Listening user stories describe required output, features, and functionality for software to be built. Each story is written by the customer and is placed on an index card. The customer assigns a value (which is also called priority) to the story based on the overall business value of the feature or function. Members of the XP team then assess each story and assign a cost—measured in development weeks—to it. If the story is estimated to require more than three development weeks, the customer is asked to split the story into smaller stories and the assignment of value and cost occurs again. It is important to note that new stories can be written at any time. Customers and developers work together to decide how to group stories into the next release (the next software increment) to be developed by the XP team. Once a basic commitment is made for a release, the XP team orders the stories that will be developed in one of three ways: (1) all stories will be implemented immediately (within a few weeks), (2) the stories with highest value will be moved up in the schedule and implemented first, or (3) the riskiest stories will be moved up in the schedule and implemented first.

After the first project release (also called a software increment) has been delivered, the XP team computes project velocity. In simple words, project velocity is the number of requirements implemented during the first release. Project velocity can then be used to (1) help estimate delivery dates and schedule for subsequent releases and (2) determine whether an over commitment has been made for all stories across the entire development project. As development work proceeds, the customer can add stories, change the value of an existing story, split stories, or eliminate them. The XP team then reconsiders all remaining releases and modifies its plans accordingly.

Design: XP design follows the KIS (keep it simple) principle. A simple design is always preferred over a more complex representation. XP encourages the use of CRC cards as an effective mechanism for thinking about the software in an object-oriented context. CRC (class-responsibility collaborator) cards identify and organize the object-oriented classes that are relevant to the current software increment. The CRC cards are the only design work product produced as part of the XP process.
If a difficult design problem is encountered, XP recommends the immediate creation of an operational prototype of that portion of the design which is called a spike solution. The spike solution or design prototype is implemented and evaluated. This lowers the risk when true implementation starts and validates the original estimates for the story containing the design problem

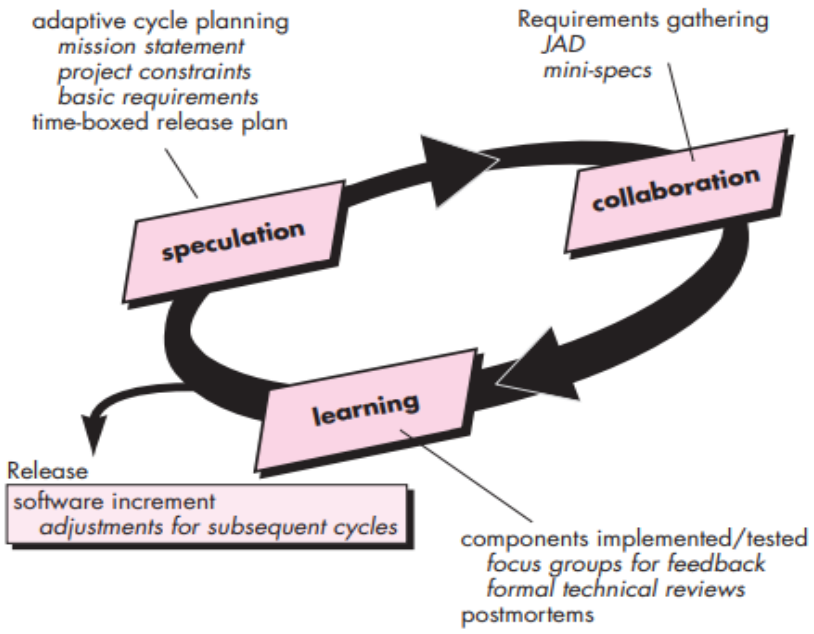
Coding: After stories are developed and preliminary design work is done, the team does not move to code, but rather develops a series of unit tests that will exercise each of the stories that is to be included in the current release (software increment). Once the unit test has been created, the developer is better able to focus on what must be implemented to pass the test. Complexity is eliminated here also. Once the code is complete, it can be unit-tested immediately, thereby providing instantaneous feedback to the developers. A key concept during the coding activity is pair programming. XP recommends that two people work together at one computer workstation to create code for a story. This provides a mechanism for realtime problem solving and real-time quality assurance. It also keeps the developers focused on the problem at hand.
As pair programmers complete their work, the code they develop is integrated with the work of others. In some cases this is performed on a daily basis by an integration team. In other cases, the pair programmers have integration responsibility. This “continuous integration” strategy helps to avoid compatibility and interfacing problems and provides a “smoke testing” environment that helps to uncover errors early.

Testing: The unit tests that are created should be implemented using a framework that enables them to be automated. This encourages a regression testing strategy whenever code is modified As the individual unit tests are organized into a “universal testing suite” integration and validation testing of the system can occur on a daily basis. This provides the XP team with a continual indication of progress and also warns if things go wrong. XP acceptance tests, also

called customer tests, are specified by the customer and focus on overall system features and functionality that are visible and reviewable by the customer. Acceptance tests are derived from user stories that have been implemented as part of a software release.

2. **Adaptive Software development(ASD):**

Adaptive Software Development (ASD) has been proposed by Jim Highsmith, as a technique for building complex software and systems. ASD focus on human collaboration and team self-organization. The ASD "life cycle" incorporates three phases, speculation, collaboration, and learning.



During speculation, the project is initiated and adaptive cycle planning is conducted. Adaptive cycle planning uses project initiation information—the customer's mission statement, project constraints , and basic requirements—to define the set of release cycles that will be required for the project.

Motivated people use collaboration in a way that multiplies their talent and creative output beyond their absolute numbers. This approach is a recurring theme in all agile methods. But collaboration is not easy. It encompasses communication and teamwork, it also emphasizes on individualism, because individual creativity plays an important role in collaborative thinking. It is, above all, a matter of trust. People working together must trust one another to (1) criticize without animosity, (2) assist without resentment, (3) work as hard as or harder than they do, (4) have the skill set to contribute to the work at hand, and (5) communicate problems or concerns in a way that leads to effective action.

As members of an ASD team begin to develop the components that are part of an adaptive cycle, the emphasis is on "learning" as much as it is on progress toward a completed cycle.

Software developers often overestimate their own understanding (of the technology, the process, and the project) and that learning will help them to improve their level of real understanding. ASD teams learn in three ways: focus groups, technical reviews, and project postmortems.
The ASD philosophy has merit regardless of the process model that is used. ASD's overall emphasis is on the dynamics of self-organizing teams, interpersonal collaboration, and individual and team learning. That yields software project teams that have a much higher likelihood of success.

3. Dynamic software Development Method(DSDM) :

The Dynamic Systems Development Method is an agile software development approach that "provides a framework for building and maintaining systems which meet tight time constraints through the use of incremental prototyping in a controlled project environment".
DSDM is an iterative software process in which each iteration follows the 80 percent rule. That is, only enough work is required for each increment to facilitate movement to the next increment. The remaining detail can be completed later when more business requirements are known or changes have been requested and accommodated.
The DSDM Consortium is a worldwide group of member companies that collectively take on the role of "keeper" of the method. The consortium has defined an agile process model, called the DSDM life cycle that defines three different iterative cycles, preceded by two additional life cycle activities:

Feasibility study—establishes the basic business requirements and constraints associated with the application to be built and then assesses whether the application is a viable candidate for the DSDM process.
Business study—establishes the functional and information requirements that will allow the application to provide business value; also, defines the basic application architecture and identifies the maintainability requirements for the application.

Functional model iteration—produces a set of incremental prototypes that demonstrate functionality for the customer. The intent during this iterative cycle is to gather additional requirements by eliciting feedback from users as they exercise the prototype.

Design and build iteration—revisits prototypes built during functional model iteration to ensure that each has been engineered in a manner that will enable it to provide operational business value for end users. In some cases, functional model iteration and design and build iteration occur concurrently.

Implementation—places the latest software increment into the operational environment. It should be noted that (1) the increment may not be 100 percent complete or (2) changes may be

requested as the increment is put into place. In either case, DSDM development work continues by returning to the functional model iteration activity.

DSDM can be combined with XP to provide a combination approach that defines a solid process model with the nuts and bolts practices (XP) that are required to build software increments. In addition, the ASD concepts of collaboration and self-organizing teams can be adapted to a combined process model.

4. Scrum :

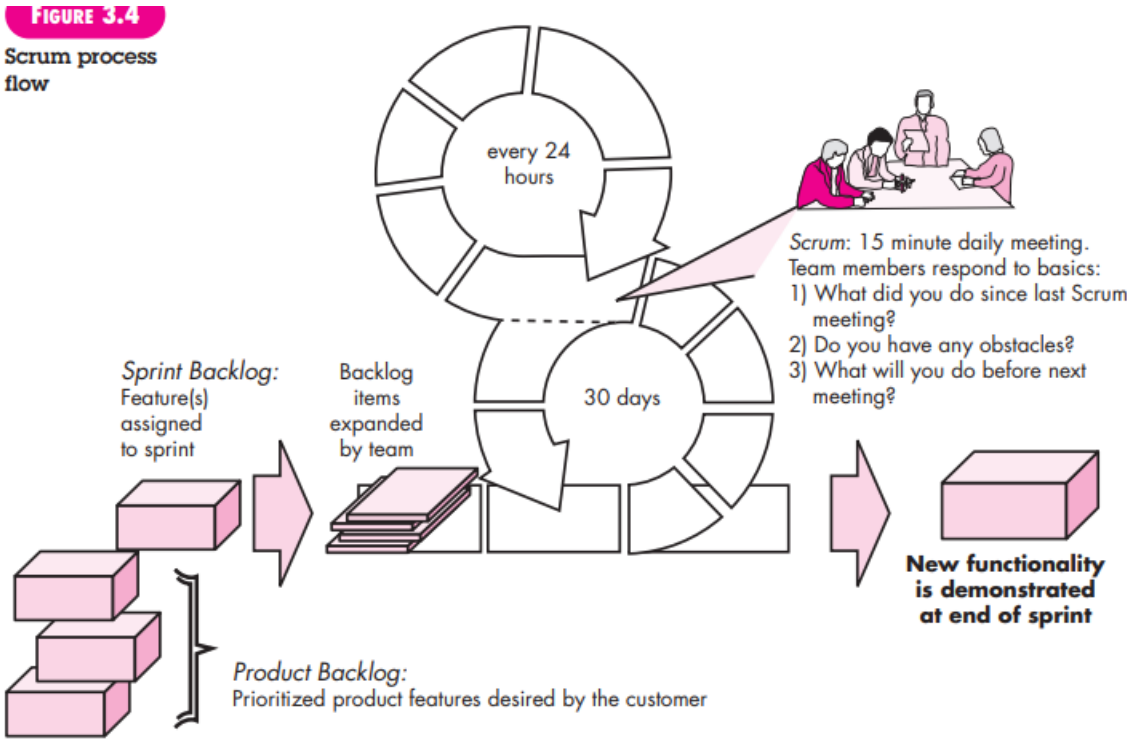


FIGURE 3.4 Scrum process flow

Scrum (the name is derived from an activity that occurs during a rugby match) is an agile software development method that was conceived by Jeff Sutherland and his development team in the early 1990s.

Scrum principles are consistent with the agile manifesto and are used to guide development activities within a process that incorporates the following framework activities:

requirements, analysis, design, evolution, and delivery.

Within each framework activity, work tasks occur within a process pattern called a sprint. The work conducted within a sprint (the number of sprints required for each framework activity will vary

depending on product complexity and size) is adapted to the problem at hand and is defined and often modified in real time by the Scrum team.

Scrum emphasizes the use of a set of software process patterns that have proven effective for projects with tight timelines, changing requirements, and business criticality. Each of these process patterns defines a set of development actions:

Backlog—a prioritized list of project requirements or features that provide business value for the customer. Items can be added to the backlog at any time (this is how changes are introduced). The product manager assesses the backlog and updates priorities as required.

Sprints—consist of work units that are required to achieve a requirement defined in the backlog that must be fit into a predefined time-box (typically 30 days).

Changes (e.g., backlog work items) are not introduced during the sprint. Hence, the sprint allows team members to work in a short-term, but stable environment. Scrum meetings—are short (typically 15 minutes) meetings held daily by the Scrum team. Three key questions are asked and answered by all team members.

• What did you do since the last team meeting?

• What obstacles are you encountering?

• What do you plan to accomplish by the next team meeting?

A team leader, called a Scrum master, leads the meeting and assesses the responses from each person. The Scrum meeting helps the team to uncover potential problems as early as possible. Also, these daily meetings lead to "knowledge socialization" and thereby promote a self-organizing team structure.

Demos—deliver the software increment to the customer so that functionality that has been implemented can be demonstrated and evaluated by the customer. It is important to note that the demo may not contain all planned functionality, but rather those functions that can be delivered within the time-box that was established.

The Scrum process patterns enable a software team to work successfully in a world where the elimination of uncertainty is impossible.

5. **Crystal :**

   Alistair Cockburn and Jim Highsmith created the Crystal family of agile methods in order to achieve a software development approach that puts a premium on "maneuverability" during what Cockburn characterizes as "a resource limited, cooperative game of invention and communication, with a primary goal of delivering useful, working software and a secondary goal of setting up for the next game".

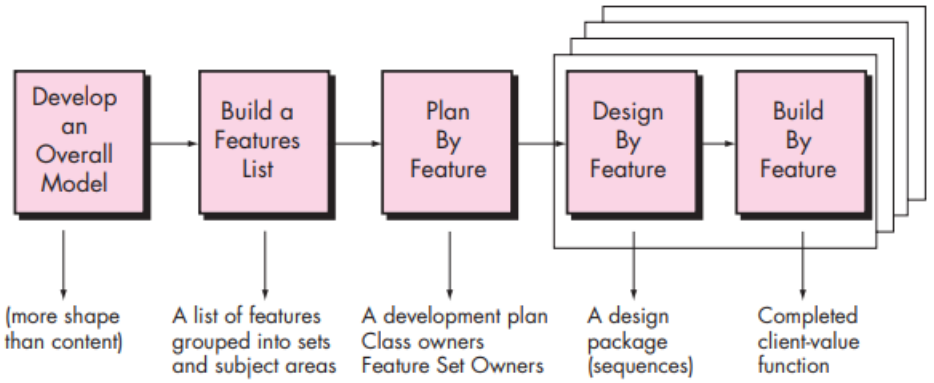
   To achieve maneuverability, Cockburn and Highsmith have defined a set of methodologies, each with core elements that are common to all, and roles, process patterns, work products, and practice that are unique to each. The Crystal family is actually a set of example agile processes

that have been proven effective for different types of projects. The intent is to allow agile teams to select the member of the crystal family that is most appropriate for their project and environment.

6. **Lean Software Development(LSD) :** Lean Software Development (LSD) has adapted the principles of lean manufacturing to the world of software engineering. The lean principles can be summarized as
   1) eliminate waste,
   2) build quality in,
   3) create knowledge,
   4) defer commitment,
   5) deliver fast,
   6) respect people,
   7) optimize the whole.

   Each of these principles can be adapted to the software process. For example, eliminate waste within the context of an agile software project can be interpreted to mean (1) adding no extraneous features or functions, (2) assessing the cost and schedule impact of any newly requested requirement, (3) removing any superfluous process steps, (4) establishing mechanisms to improve the way team members find information, (5) ensuring the testing finds as many errors as possible (6) reducing the time required to request and get a decision that affects the software or the process that is applied to create it, and (7) streamlining the manner in which information is transmitted to all stakeholders involved in the process.
7. **Feature Driven Development (FDD):**

FIGURE 3.5
Feature Driven Development [Coa99] (with permission)



The FDD approach defines five “collaborating” framework activities (in FDD these are called “processes”). FDD provides greater emphasis on project management guidelines and techniques than many other agile methods. As projects grow in size and complexity, ad hoc project management is often inadequate. It is essential for developers, their managers, and other stakeholders to understand project status—what accomplishments have been made and problems have been encountered. If deadline pressure is significant, it is critical to determine if

software increments (features) are properly scheduled. To accomplish this, FDD defines six milestones during the design and implementation of a feature: “design walkthrough, design, design inspection, code, code inspection, promote to build”.
Feature Driven Development (FDD) was originally conceived by Peter Coad and his colleagues as a practical process model for object-oriented software engineering. Stephen Palmer and John Felsing have extended and improved Coad’s work, describing an adaptive, agile process that can be applied to moderately sized and larger software projects.

Like other agile approaches, FDD adopts a philosophy that (1) emphasizes collaboration among people on an FDD team; (2) manages problem and project complexity using feature-based decomposition followed by the integration of software increments, and (3) communication of technical detail using verbal, graphical, and text-based means.
FDD emphasizes on software quality assurance activities by encouraging an incremental development strategy, the use of design and code inspections, the application of software quality assurance audits , the collection of metrics, and the use of patterns (for analysis, design, and construction).
FDD has following benefits:
• Because features are small blocks of deliverable functionality, users can describe them more easily; understand how they relate to one another more readily; and better review them for ambiguity, error, or omissions.
• Features can be organized into a hierarchical business-related grouping.
• Since a feature is the FDD deliverable software increment, the team develops operational features every two weeks.
• Because features are small, their design and code representations are easier to inspect effectively.
• Project planning, scheduling, and tracking are driven by the feature hierarchy, rather than an arbitrarily adopted software engineering task set.

8. Agile Modeling: (AM) :

There are many situations in which software engineers must build large, businesscritical systems. The scope and complexity of such systems must be modeled so that
(1) all constituencies can better understand what needs to be accomplished,
(2) the problem can be partitioned effectively among the people who must solve it, and
(3) quality can be assessed as the system is being engineered and built.

Agile Modeling (AM) is a practice-based methodology for effective modeling and documentation of software-based systems. Simply put, Agile Modeling (AM) is a collection of values, principles, and practices for modeling software that can be applied on a software development project in an effective and light-weight manner. Agile models are more effective than traditional models because they are just barely good, they don’t have to be perfect.

Agile modeling adopts all of the values that are consistent with the agile manifesto. The agile modeling philosophy recognizes that an agile team must have the courage to make decisions

that may cause it to reject a design and refactor. The team must also have the humility to recognize that technologists do not have all the answers and that business experts and other stakeholders should be respected and embraced.

There are a few features which makes the AM unique. They are

1. Model with a purpose
2. Use multiple models
3. Travel light
4. Content is more important than representation
5. Know the models and the tools you use to create them
6. Adapt locally

8 Agile Unified Process(AUP) : The Agile Unified Process (AUP) adopts a "serial in the large" and "iterative in the small" philosophy for building computer-based systems. By adopting the classic Unified Process phased activities—inception, elaboration, construction, and transition—AUP provides a serial overlay (i.e., a linear sequence of software engineering activities) that enables a team to visualize the overall process flow for a software project.

How ever, within each of the activities, the team iterates to achieve agility and to deliver meaningful software increments to end users as rapidly as possible. Each AUP iter ation addresses the following activities :

Modeling. UML representations of the business and problem domains are created. However, to stay agile, these models should be "just barely good enough" to allow the team to proceed. "Traveling light" is an appropriate philosophy for all software engg work. Build only those models that provide value … no more, no less.

• **Implementation.** Models are translated into source code.

• **Testing**. Like XP, the team designs and executes a series of tests to uncover errors and ensure that the source code meets its requirements.

• **Deployment**. Like the generic process activity ,deployment in this context focuses on the delivery of a software increment and the acquisition of feedback from end users.

• **Configuration and project management**. In the context of AUP, configuration management addresses change management, risk management and the control of any persistent work products that are produced by the team. Project management tracks and controls the progress of the team and coordinates team activities.

• **Environment management.** Environment management coordinates a process infrastructure that includes standards, tools, and other support technology available to the team