Module - III

AGILE DEVELOPMENT: What is Agility?, Agility and the cost of change. What is an agile Process?, Extreme Programming (XP), Other Agile Process Models, A tool set for Agile process
Principles that guide practice: Software Engineering Knowledge, Core principles, Principles that guide each framework activity
Textbook 1: Chater 3: 3.1 to 3.6, Chapter 4: 4.1 to 4.4

WHAT IS AGILITY?

An agile team is a nimble team able to appropriately respond to changes. Change is what software development is very much about. Changes in the software being built, changes to the team members, changes because of new technology, changes of all kinds that may have an impact on the product they build or the project that creates the product. Support for changes should be built-in everything we do in software, something we embrace because it is the heart and soul of software. An agile team recognizes that software is developed by individuals working in teams and that the skills of thesepeople, their ability to collaborate is at the core for the success of the project.

It encourages team structures and attitudes that make communication (among team members, between technologists and business people, between software engineersand their managers) more facile. It emphasizes rapid delivery of operational software and de-emphasizes the importance of intermediate work products.

Agility can be applied to any software process. However, to accomplish this, it is essential that the process be designed in a way that allows the project team to adapt tasks and to streamline them, conduct planning in a way that understands the fluidity of an agile development approach, eliminate all but the most essential work products and keep them lean, and emphasize an incremental delivery strategy that gets working software to the customer as rapidly as feasible for the product type and operational environment.

AGILITY AND THE COST OF CHANGE

The conventional wisdom in software development is that the cost of change increases nonlinearly as a project progresses (Figure: solid black curve). It is relatively easy to accommodate a change when a software team is gathering requirements (early in a project). A usage scenario might have to be modified, a list of functions may be extended, or a written specification can be edited. The costs of doing this work are minimal, and the time required will not adversely affect the outcome of the project.

The change requires a modification to the architectural design of the software, the design and construction of three new components, modifications to another five components, the design of new tests, and so on. Costs escalate quickly, and the time and cost required to ensure that the change is made without unintended side effects is nontrivial. Although debate about the degree to which the cost curve flattens is ongoing, there is evidence to suggest that a significant reduction in the cost of change can be achieved.

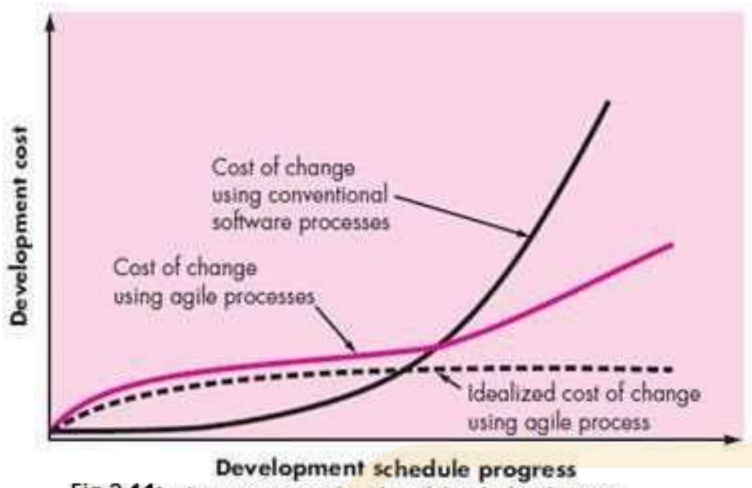

Fig 2.11: Change Costs as a function of time in development

WHAT IS AN AGILE PROCESS?

Any agile software process is characterized in a manner that addresses a number of key assumptions about the majority of software projects:

1. It is difficult to predict in advance which software requirements will persist and which will change. It is equally difficult to predict how customer priorities will change as the project proceeds.

2. For many types of software, design and construction are interleaved. That is, both activities should be performed in tandem so that design models are proven as they are created. It is difficult to predict how much design is necessary before construction is used to prove the design.

3. Analysis, design, construction, and testing are not as predictable as we might like. Given these three assumptions, an important question arises: How do we create aprocess that can manage unpredictability? The answer, as I have already noted, lies inprocess adaptability. An agile process, therefore, must be adaptable. But continualadaptation without forward progress accomplishes little. Therefore, an agile software process must adapt incrementally. To accomplish incremental adaptation, an agile teamrequires customer feedback. An effective catalyst for customer feedback is an operational prototype or a portion of an operational system. Hence, an incremental development strategy should be instituted. Software increments must be delivered in short time periods so that adaptation keeps pace with change. This iterative approach enables the customer to evaluate the software increment regularly, provide necessary feedback to the software team, and influence the process adaptations that are made to accommodate the feedback.

Agility Principles: The Agile Alliance defines 12 agility principles for those whowant to achieve agility:

- ✓ Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- ✓ Welcome changing requirements, even late in development. Agile processes harness

- ✓ change for the customer's competitive advantage.
- ✓ Deliver working software frequently, from a couple of weeks to a couple of months,with a preference to the shorter timescale.
- ✓ Business people and developers must work together daily throughout the project.
- ✓ Build projects around motivated individuals. Give them the environment and supportthey need, and trust them to get the job done.
- ✓ The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- ✓ Working software is the primary measure of progress.
- ✓ Agile processes promote sustainable development. The sponsors, developers, andusers should be able to maintain a constant pace indefinitely.
- ✓ Continuous attention to technical excellence and good design enhances agility.
- ✓ Simplicity—the art of maximizing the amount of work not done—is essential.
- ✓ The best architectures, requirements, and designs emerge from self–organizingteams.
- ✓ At regular intervals, the team reflects on how to become more effective, then tunesand adjusts its behavior accordingly.

The Politics of Agile Development: There is considerable debate about the benefits and applicability of agile software development as opposed to more conventional software engineering processes. No one is against agility. The real question is: What is the best way to achieve it? As important, how do you build software that meets customers' needs today and exhibits the quality characteristics that will enable it to be extended and scaled to meet customers' needs over the long term?

There are no absolute answers to either of these questions. Even within the agile schoolitself, there are many proposed process models, each with a subtly different approachto the agility problem. Within each model there is a set of "ideas" that represent a significant departure from traditional software engineering. And yet, many agile concepts are simply adaptations of good software engineering concepts. Bottom line: there is much that can be gained by considering the best of both schools and virtually nothing to be gained by denigrating either approach.

Human Factors: Proponents of agile software development take great pains to emphasize the importance of "people factors." If members of the software team are to drive the characteristics of the process that is applied to build software, a number of key traits must exist among the people on an agile team and the team itself:
Competence: In an agile development (as well as software engineering) context, "competence" encompasses innate talent, specific software-related skills, and overall knowledge of the process that the team has chosen to apply. Skill and knowledge of process can and should be taught to all people who serve as agile team members.
Common focus:. Although members of the agile team may perform different tasks and bring different skills to the project, all should be focused on one goal—to deliver a working software

increment to the customer within the time promised. To achieve this goal, the team will also focus on continual adaptations (small and large) that will make the process fit the needs of the team.

Collaboration: Software engineerinG is about assessing, analyzing, and using information that is communicated to the software team; creating information that will help all stakeholders understand the work of the team; and building information that provides business value for the customer. To accomplish these tasks, team members must collaborate—with one another and all other stakeholders.

Decision-making ability. Any good software team must be allowed the freedom to control its own destiny. This implies that the team is given autonomy—decision-making authority for both technical and project issues.

Fuzzy problem-solving ability. Software managers must recognize that the agile team will continually have to deal with ambiguity and will continually be buffeted by change. Insome cases, the team must accept the fact that the problem they are solving today may not be the problem that needs to be solved tomorrow. However, lessons learned from any problem-solving. activity (including those that solve the wrong problem) may be of benefit to the team later in the project.

Mutual trust and respect. The agile team must become what DeMarco and Lister calla "jelled" team. A jelled team exhibits the trust and respect that are necessary to make them "so strongly knit that the whole is greater than the sum of the parts."

Self-organization. In the context of agile development, self-organization implies three things: (1) the agile team organizes itself for the work to be done, (2) the team organizes the process to best accommodate its local environment, (3) the team organizes the work schedule to best achieve delivery of the software increment. Self- organization has a number of technical benefits, but more importantly, it serves to improve collaboration and boost team morale. In essence, the team serves as its own management.

EXTREME PROGRAMMING (XP)

Extreme Programming (XP), the most widely used approach to agile software development.

XP Values: Beck defines a set of *ive values* that establish a foundation for all work performed as part of XP—**communication**, **simplicity**, **feedback**, **courage**, and **respect**. Each of these values is used as a driver for specific XP activities, actions, andtasks.

Communication: In order to achieve effective communication between software engineers and other stakeholders (e.g., to establish required features and functions for the software), XP emphasizes close, yet informal (verbal) collaboration between customers and developers, the establishment of effective metaphors for communicating important concepts, continuous feedback, and the avoidance of voluminous documentation as a communication medium.

Simplicity: To achieve simplicity, XP restricts developers to design only for immediate needs, rather than consider future needs. The intent is to create a simple design thatcan be easily implemented in code. If the design must be improved, it can be refactored at a later time.

Feedback: It is derived from three sources: the implemented software itself, the customer, and other software team members. By designing and implementing an effective testing strategy, the software provides the agile team with feedback The degree to which the software implements the output, function, and behavior of the use case is a form of feedback. Finally, as new requirements are derived as part of iterative planning, the team provides the customer with rapid feedback regarding cost and schedule impact.
Courage: Beck argues that strict adherence to certain XP practices demands courage.A better word might be discipline. An agile XP team must have the discipline (courage) to design for today, recognizing that future requirements may change dramatically, thereby demanding substantial rework of the design and implemented code.
Respect: By following each of these values, the agile team inculcates respect amongits members, between other stakeholders and team members, and indirectly, for the software itself. As they achieve successful delivery of software increments, the team develops growing respect for the XP process.

The XP Process: 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. Figure 3.2 illustrates the XP process .

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 get a broad feel for required output and major features and functionality.

Design: XP design rigorously follows the **KIS** (keep it simple) principle. A simple design is always preferred over a more complex representation. In addition, the design provides implementation guidance for a story as it is written—nothing less, nothingmore. If a difficult

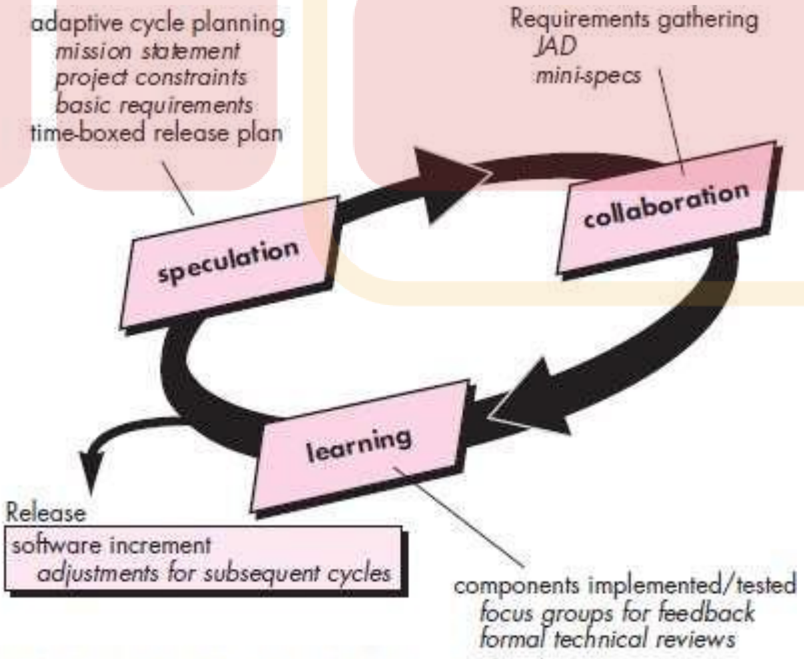

Fig 2.12: Adaptive software development

design problem is encountered as part of the design of a story, XP recommends the immediate creation of an operational prototype of that portion of the design. Called a spike solution, the design prototype is implemented and evaluated. A central notion in XP is that design occurs both before and after coding commences. Refactoring means that design occurs continuously as the system is constructed. In fact, the construction activity itself will provide the XP team with guidance on how to improve the design.

Coding. After stories are developed and preliminary design work is done, the teamdoes 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. Once the unit test has been created, the developer is better able to focus on what must be implemented to pass the test. Nothing extraneous is added (KIS). 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. Problem solving (two heads are often better than one) and real-time quality assurance.

Testing: I have already noted that the creation of unit tests before coding commences

is a key element of the XP approach. 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. 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.

Industrial XP: IXP is an organic evolution of XP. It is imbued with XP's minimalist, customer-centric, test-driven spirit. IXP incorporates six new practices that are designed to help ensure that an XP project works successfully for significant projects within a large organization.

Readiness assessment: Prior to the initiation of an IXP project, the organization shouldconduct a readiness assessment. The assessment ascertains whether (1) an appropriate development environment exists to support IXP, (2) the team will be populated by the proper set of stakeholders, (3) the organization has a distinct quality program and supports continuous improvement, (4) the organizational culture will support the new values of an agile team, and (5) the broader project community will be populated appropriately.

Project community. Classic XP suggests that the right people be used to populate the agile team to ensure success. The implication is that people on the team must be well- trained, adaptable and skilled, and have the proper temperament to contribute to a self- organizing team. When XP is to be applied for a significant project in a large organization, the concept of the "team" should morph into that of a community. Acommunity may have a technologist and customers who are central to the success of a project as well as many other stakeholders (e.g., legal staff, quality auditors, manufacturing or sales types) who "are often at the periphery of an IXP project yet they may play important roles on the project". In IXP, the community members and their roles should be explicitly defined and mechanisms for communication and coordination between

community members should be established.
Project chartering. The IXP team assesses the project itself to determine whether an appropriate business justification for the project exists and whether the project will further the overall goals and objectives of the organization. Chartering also examines the context of the project to determine how it complements, extends, or replaces existing systems or processes.
Test-driven management. An IXP project requires measurable criteria for assessing the state of the project and the progress that has been made to date. Test-driven management establishes a series of measurable “destinations” and then defines mechanisms for determining whether or not these destinations have been reached.
Retrospectives. An IXP team conducts a specialized technical review after a software increment is delivered. Called a retrospective, the review examines “issues, events, and lessons-learned” across a software increment and/or the entire software release. The intent is to improve the IXP process. Continuous learning. Because learning is a vitalpart of continuous process improvement, members of the XP team are encouraged to learn new methods and techniques that can lead to a higher quality product.

The XP Debate: Issues that continue to trouble some critics of XP are:
• ***Requirements volatility***. Because the customer is an active member of the XP team, changes to requirements are requested informally. As a consequence, the scope of the project can change and earlier work may have to be modified to accommodate current needs.
• ***Con licting customer needs***. Many projects have multiple customers, each with his own set of needs. In XP, the team itself is tasked with assimilating the needs of different customers, a job that may be beyond their scope of authority.
• ***Requirements are expressed informally.*** User stories and acceptance tests are the only explicit manifestation of requirements in XP. Critics argue that a more formal model or specification is often needed to ensure that omissions, inconsistencies, and errorsare uncovered before the system is built. Proponents counter that the changing nature of requirements makes such models and specification obsolete almost as soon as they are developed.
• ***Lack of formal design.*** XP deemphasizes the need for architectural design and in many instances, suggests that design of all kinds should be relatively informal.

OTHER AGILE PROCESS MODELS

The most widely used of all agile process models is Extreme Programming (XP). But many other agile process models have been proposed and are in use across theindustry. Among the most common are:
• Adaptive Software Development (ASD)
• Scrum
• Dynamic Systems Development Method (DSDM)
• Crystal
• Feature Drive Development (FDD)

- Lean Software Development (LSD)
- Agile Modeling (AM)
- Agile Unified Process (AUP)

Adaptive Software Development (ASD): Adaptive Software Development (ASD) has been proposed by *Jim Highsmith* as a technique for building complex software and systems. The philosophical underpinnings of ASD focus on human collaboration and team self-organization.

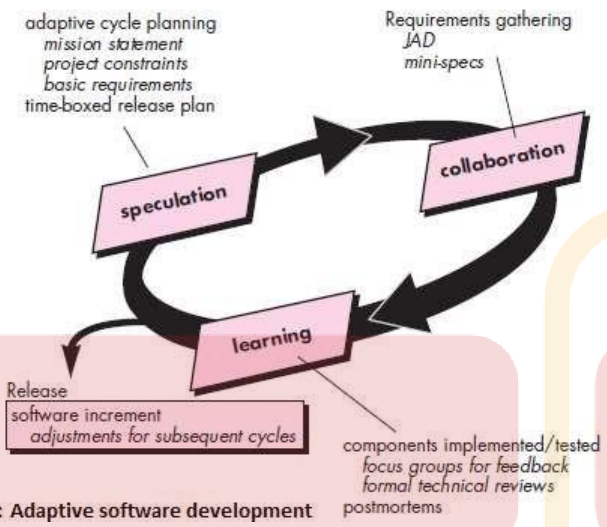

Fig 2.13: Adaptive software development

Scrum: Scrum is an agile software development method that was conceived by Jeff Sutherland 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 is adapted to the problem at hand and is defined and often modified in real time by the Scrum team. The overall flow of the Scrum process is illustrated in Figure

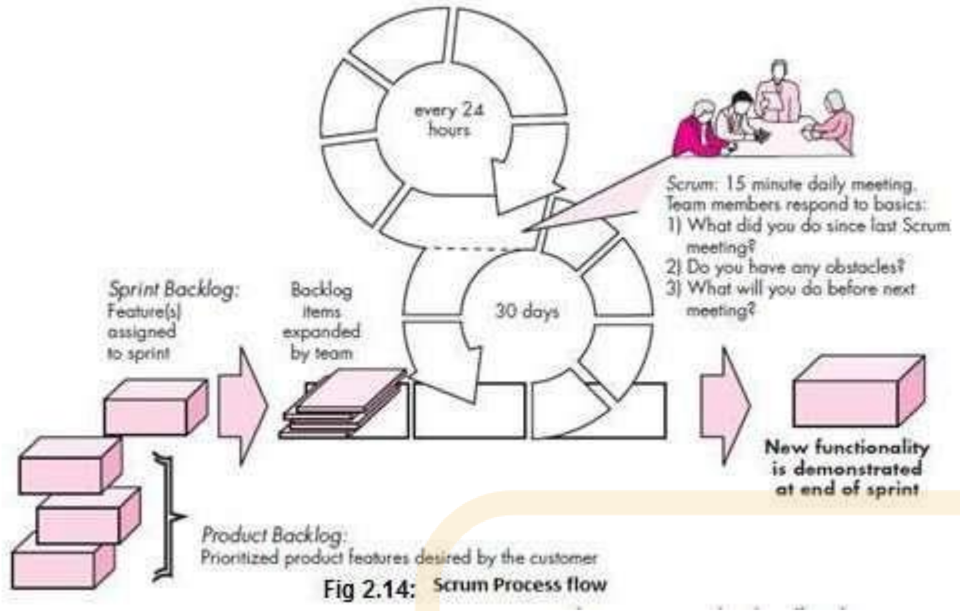
every 24 hours
30 days
Scrum: 15 minute daily meeting.
Team members respond to basics:
1) What did you do since last Scrum meeting?
2) Do you have any obstacles?
3) What will you do before next meeting?
Sprint Backlog: Feature(s) assigned to sprint
Backlog items expanded by team
New functionality is demonstrated at end of sprint
Product Backlog: Prioritized product features desired by the customer

Fig 2.14: Scrum Process flow

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. 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.

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.

Dynamic Systems Development Method (DSDM): The Dynamic Systems Development Method (DSDM) 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". The DSDM philosophy is borrowed from a modified version of the ***Pareto principle***—80 percent of an application can be delivered in 20 percent of the time it would take to deliver the complete (100 percent) application.

DSDM is an iterative software process in which each iteration follows the 80 percentrule. 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.
DSDM consortium has defined an agile process model, called the DSDM life cycle thatdefines 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 isa 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 completeor (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.

Crystal: 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 Crystalfamily 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 project and environment.

Feature Driven Development (FDD): Like other agile approaches, FDD adoptsa 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 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

In the context of FDD, a feature "is a client-valued function that can be implemented in two weeks or less". The emphasis on the definition of features provides the following benefits:

- Because features are small blocks of deliverable functionality, users can describethem 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 toinspect effectively.
- Project planning, scheduling, and tracking are driven by the feature hierarchy,rather than an arbitrarily adopted software engineering task set.

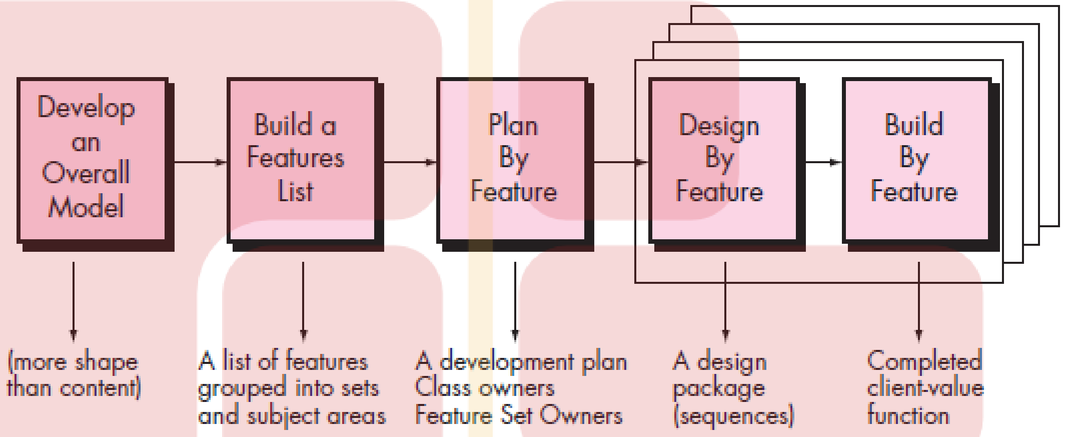
Develop an Overall Model
Build a Features List
Plan By Feature
Design By Feature
Build By Feature
(more shape than content)
A list of features grouped into sets and subject areas
A development plan Class owners Feature Set Owners
A design package (sequences)
Completed client-value function

Fig 2.14: Feature Driven Development

Lean Software Development (LSD): The lean principles that inspire the LSD process can be summarized as eliminate waste, build quality in, create knowledge, defer commitment, deliver fast, respect people, and 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.

Agile Modeling (AM): There are many situations in which software engineers must build large, business critical 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 whomust solve it, and (3) quality can be assessed as the system is being engineered and built.

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 re factor. The team must also have the humility to recognize that technologists do not have all the answers and that business expert and other stakeholders should be respected and embraced. Although AM suggests a wide array of "core" and "supplementary" modeling principles, those that make AM unique are:

- Model with a purpose
- Use multiple models
- Travel light
- Content is more important than representation
- Know the models and the tools you use to create them
- Adapt locally

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 UP phased activities—inception, elaboration, construction, and transition—AUP provides a serial overlay that enables a team to visualize the overall process flow for a software project. However, 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 iteration 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.
- **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.