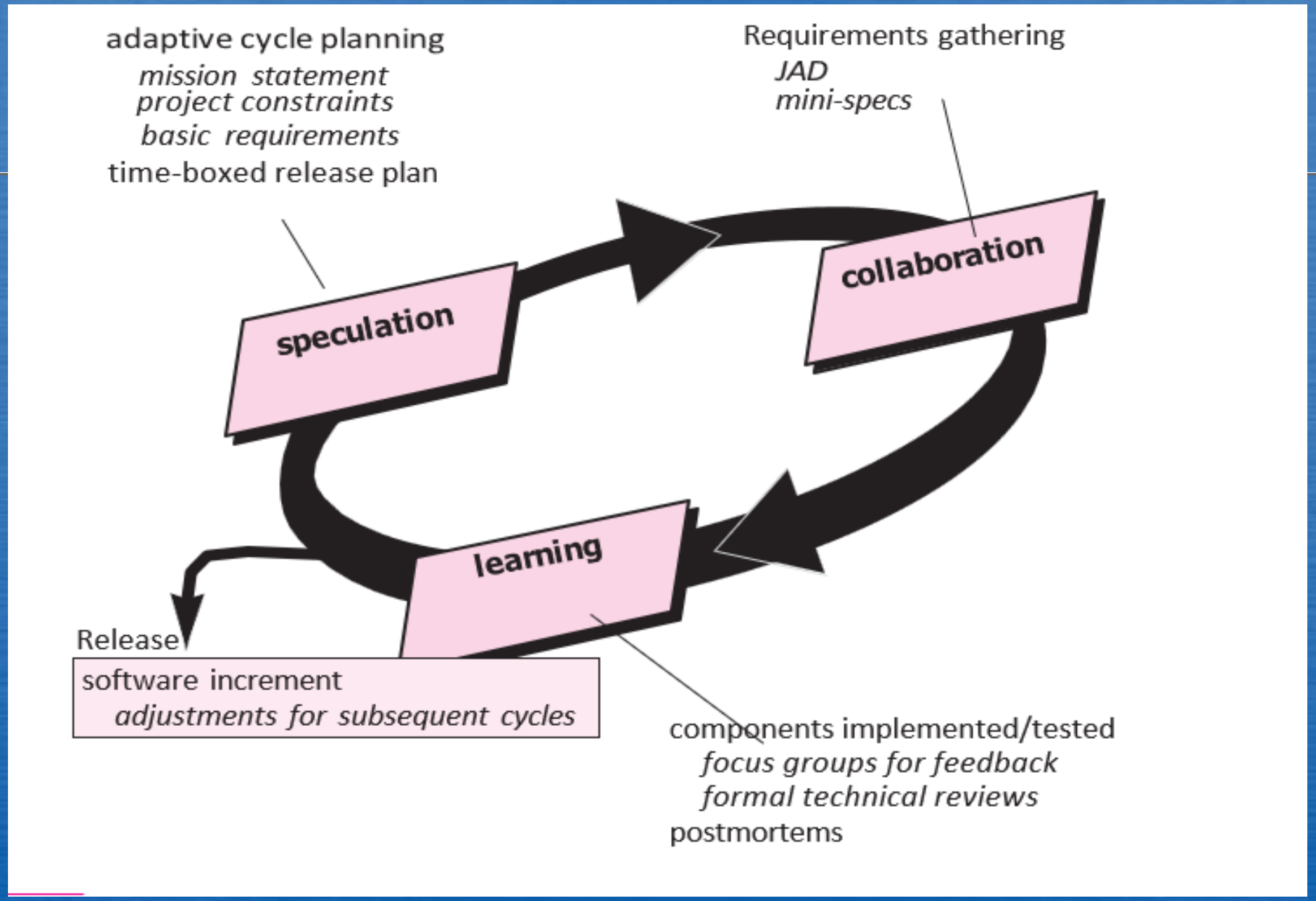
Agile Process Models

- 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)

ASD focus on human collaboration and team self-organization. He defines an ASD "life cycle" that incorporates three phases, **speculation, collaboration, and learning.**

- Originally proposed by Jim Highsmith
- ASD — distinguishing features:-
  - Mission-driven planning
  - Component-based focus
  - Uses "time-boxing"
  - Explicit consideration of risks
  - Emphasizes collaboration for requirements gathering
  - Emphasizes "learning" throughout the process


adaptive cycle planning
mission statement
project constraints
basic requirements
time-boxed release plan
Requirements gathering
JAD
mini-specs
speculation
collaboration
learning
Release
software increment
adjustments for subsequent cycles
components implemented/tested
focus groups for feedback
formal technical reviews
postmortems

Speculation-

- 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 (e.g., delivery dates or user descriptions), and basic requirements
- Based on information obtained at the completion of the first cycle, the plan is re- viewed and adjusted so that planned work better fits the reality in which an ASD team is working.

Collaboration- Motivated people use *collaboration* in a way that multiplies their talent and creative output beyond their absolute numbers. But collaboration is not easy. It encompasses communication and teamwork, but it also emphasizes 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

(5) communicate problems or concerns in a way that leads to effective action.

Learning

- Challenges all stakeholders.
- Examine their assumptions and use the results of each development cycle to learn the direction of the next.
- Customer focus groups, technical reviews, beta testing, and postmortems are all practices that expose results to scrutiny.

Advantages and disadvantages

Advantages

- Software incremental adjustment
- Rapid and complex software development

Disadvantages

- There is lack of emphasis on necessary designing and documentation.
- it is difficult to assess the effort required at the beginning of the software_development_life_cycle.

Dynamic Systems Development Method (DSDM)

provides a framework for building and maintaining systems which meet tight time constraints through the use of incremental prototyping in a controlled project environment"

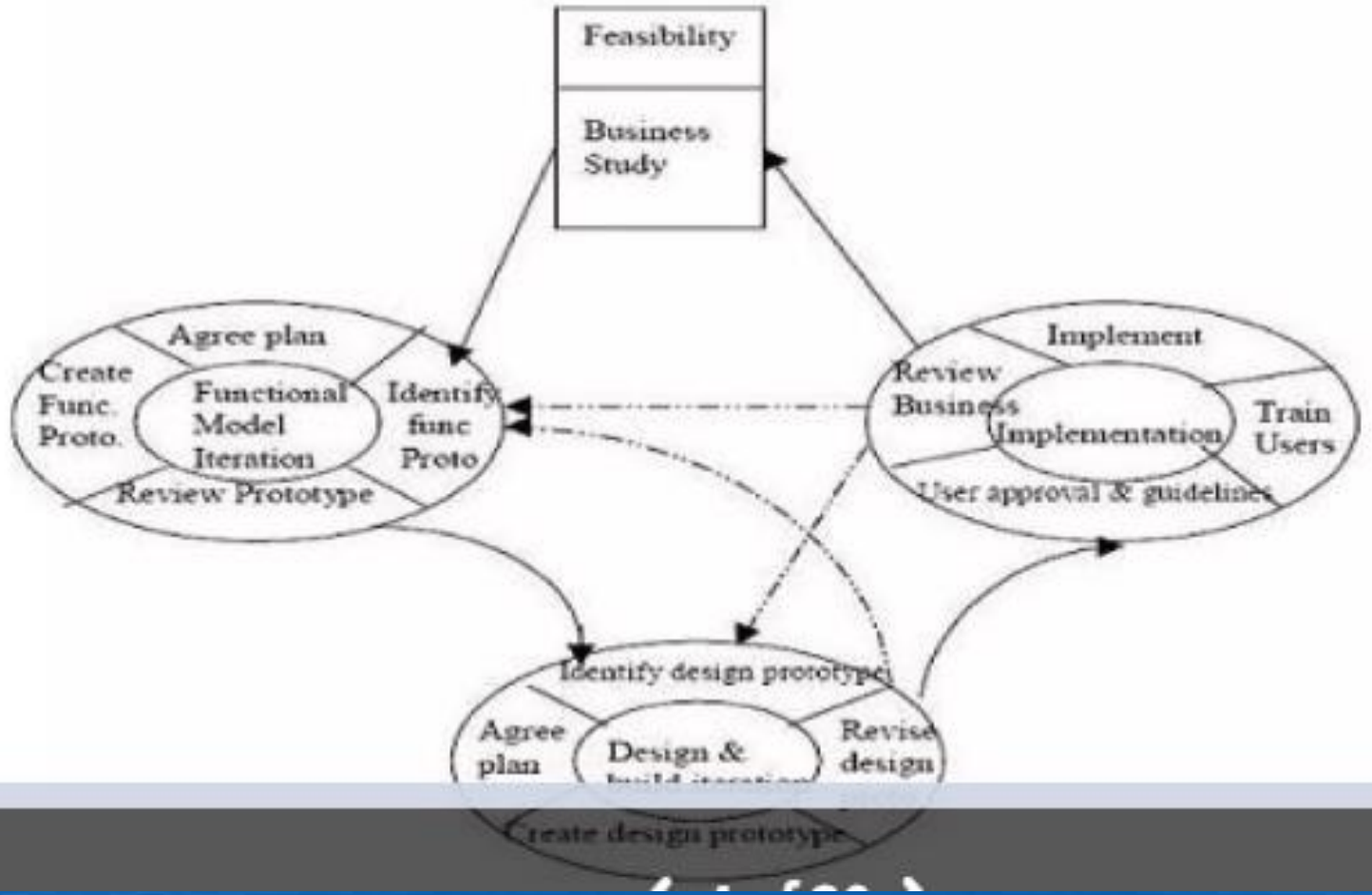
- DSDM is an iterative software process in which each iteration follows the 80 per- cent rule(80 percent of an application can be delivered in 20 percent of the time). That is, only enough work is required for each increment to facilitate movement to the next increment. The remaining detail can be completed later when more business requirements are known or changes have been requested and accommodated.

- Dynamic System Development Method is approach to system development, which, as the name suggests, develops the system dynamically.
- The Dynamic System Development Method (DSDM) is dynamic as it is a Rapid Application Development method that uses incremental prototyping.
- DSDM is an iterative and incremental approach that emphasizes continuous user/customer involvement.

- Its goal is to deliver projects on time and on budget while adjusting for changing requirements along the way.
- DSDM is one of a number of Agile methods for developing software and non-I.T. solutions.
- This method is particularly useful for the systems to be developed in short time span and where the requirements cannot be frozen at the start of the application building.

- Promoted by the DSDM Consortium
- DSDM—distinguishing features
  - Similar in most respects to XP and/or ASD
  - Eight guiding principles
    1. Active user involvement is imperative.
    2. DSDM teams must be empowered to make decisions.
    3. The focus is on frequent delivery of products.
    4. Fitness for business purpose is the essential criterion for acceptance of deliverables.
    5. Iterative and incremental development is necessary to converge on an accurate business solution.
    6. All changes during development are reversible.
    7. Requirements are baselined at a high level
    8. Testing is integrated throughout the life-cycle.

Dynamic System Development Method (DSDM) has a five-phase life cycle as given the following figure:



- **Feasibility and Business Study**
  - In this phase the problem is defined and the technical feasibility of the desired application is verified. Apart from these routine tasks, it is also checked whether the application is suitable for Rapid Application Development (RAD) approach or not.
  - Only if the RAD is found as a justified approach for the desired system, the development continues.
- The overall business study of the desired system is done.
- The business requirements are specified at a high level and the information requirements out of the system are identified.
- Once this is done, the basic architectural framework of the desired system is prepared.
- The systems designed using Rapid Application Development (RAD) should be highly maintainable, as they are based on the incremental development process.

- **Functional Model Iteration:**
  - The main focus in this phase is on building the prototype iteratively and getting it reviewed from the users to bring out the requirements of the desired system.
  - The prototype is improved through demonstration to the user, taking the feedback and incorporating the changes.
  - This cycle is repeated generally twice or thrice until a part of functional model is agreed upon.
  - The end product of this phase is a functional model consisting of analysis model and some software components containing the major functionality.

- **Design and Build Iteration:**
  - This phase stresses upon ensuring that the prototypes are satisfactorily and properly engineered to suit their operational environment
  - The software components designed during the functional modeling are further refined till they achieve a satisfactory standard.
  - The product of this phase is a tested system ready for implementation.
  - There is no clear line between these two phases and there may be cases where while some component has flown from the functional modeling to the design and build modeling while the other component has not yet been started.

- **Implementation**
  - Implementation is the last and final development stage in this methodology.
  - In this phase the users are trained and the system is actually put into the operational environment.
  - At the end of this phase, there are four possibilities, as depicted by figure:
    - Everything was delivered as per the user demand, so no further development required.
    - A new functional area was discovered, so return to business study phase and repeat the whole process.
    - A less essential part of the project was missed out due to time constraint and so development returns to the functional model iteration.
    - Some non-functional requirement was not satisfied, so development returns to the design and build iterations phase.

- Dynamic System Development Method (DSDM) assumes that all previous steps may be revisited as part of its iterative approach.
- Therefore, the current step need be completed only enough to move to the next step, since it can be finished in a later iteration.
- According to this approach, the time is taken as a constraint i.e. the time is fixed, resources are fixed while the requirements are allowed to change.

Nine Principles of DSDM

- Active user Involvement is Imperative:
  - The first principle is considered the most important, because user involvement through out the project effectively reduces errors in terms of user perception, and therefore reduces error costs.
  - Instead of working with a large set of users a DSDM project guidelines recommend working with a small, select set of users continually, rather then in periodic workshops or review sessions.

- Teams Must be Empowered to Make Decisions:
  - To proceed as quickly as possible transaction costs, as resulting from friction in communications of project participants and managers, need to be avoided.
  - Addressing these inefficiencies users and other DSDM participants should be given limited authority to make decisions related to:
    - Requirements in practice
    - Which functionality needs to be in a given increment
    - Prioritization of requirements and features
    - Fine details of the technical solution

- Focus on Frequent Delivery:
  - Frequent deliveries of results ensure that errors are detected quickly, are easily reversed and closer at the source of the error.
  - This applies both to program code as well as to documents like requirements or data models.

- Fitness for Business is Criterion for Accepted Deliverables:
  - As the name of the DSDM framework suggest, its primary endeavor is to deliver software which is good enough to solve the business need and bother with any enhancements in a later iteration.
  - DSDM does not promote to write ad-hoc software, but suggest satisfying the business needs first and related activities in a later iteration.

- Iterative and Incremental Development is Mandatory:
  - In order to keep the complexity of the project manageable, it needs to be decomposed into small feature packages; with each release adding new features until the complete set of business requirements are fulfilled.
  - This principle requires accepting the fact that any software system is subject to change.
  - This principle can easily be introduced even at the beginning of a project, since specifications and other results can be produced in an iterative manner as well.

- All Changes During Development Must Be Reversible:
  - Being responsive to change requires that system configurations are changing during the development of any one increment due to changed priorities in the requirements.
  - Modern software tools support a dynamic configuration of projects as required by this principle.
  - It's often feared that reversing a development process will result in loosing precious previous work, but since DSDM advises to iterate though small increments, the total loss of work is very limited.

- Requirements are Base lined at High-Level:
  - To limit the degree of freedom to which requirements can be altered during the development process, some high-level requirements need to be established.
  - This baseline which is to be interpreted as a requirements "freeze" is agreed upon during the business study phase of the process.

- Testing is Integrated Throughout the Lifecycle:
  - Many development methods ask for testing as late as the design or implementation phase.
  - DSDM requires testing early in the development process.
  - Even testing interview documents by cross checking them with a control group, or similar techniques.

- Collaborative and Co-operative Approach:
  - Avoiding separation and encouraging collaboration of technical staff and business staff in a project is mandatory during DSDM projects, because co-operation is crucial to succeed in a DSDM project.
  - Without an atmosphere of trust and honesty it will be hard to gather requirements, and later getting honest feedback on the resulting products.

Crystal

- Proposed by Cockburn and Highsmith
- Crystal—distinguishing features
  - Actually a family of process models that allow
  - "maneuverability" based on problem characteristics.
  - Face-to-face communication is emphasized.
  - Suggests the use of "reflection workshops" to review the work habits of the team.

The crystal method was developed by an American scientist named Alistair Cockburn who worked at IBM. He decided not to focus on step-by-step developmental strategies, but to develop team collaboration and communication. Some of the traits of Cockburn's Crystal method were:

•**Human**-powered i.e. the project should be flexible and people involved in preferred work.

•**Adaptive** i.e. approaches don't have any fixed tools but can be changed anytime to meet the team's specific needs.

•**Ultra**-light i.e. this methodology doesn't require much documentation.

Properties of Crystal Agile Framework:

1.Frequent Delivery- It allows you regularly deliver the products and test code to real users. Without this, you might build a product that nobody needs.

2.Reflective Improvement- No matter how good you have done or how bad you have done. Since there are always areas where the product can be improved, so the teams can implement to improve their future practices.

3.Osmotic Communication- Alistair stated that having the teams in the same physical phase is very much important as it allows information to flow in between members of a team as in osmosis.

4. **Personal Safety-** There are no bad suggestions in a crystal team, **team members should feel safe to discuss ideas openly without any fear.**
5. **Focus-** Each member of the team knows exactly what to do, which enables them to focus their attention. This boosts team interaction and works towards the same goal.
6. **Easy access to expert users-** It enhances team communication with users and gets regular feedback from real users.
7. **Technical tooling-** It contains very specific technical tools which to be used by the software development team during testing, management, and configuration. These tools make it enable the team to identify any error within less time.
8. **Continuous learning** – The framework emphasizes on continuous learning, **enabling team members to acquire new skills and knowledge, and apply them in their work.**
9. **Teamwork –** The framework stresses on the importance of teamwork, promoting collaboration, and mutual support among team members.
10. **Timeboxing –** The framework adopts timeboxing to manage project deadlines, ensuring that the team delivers within set timelines.

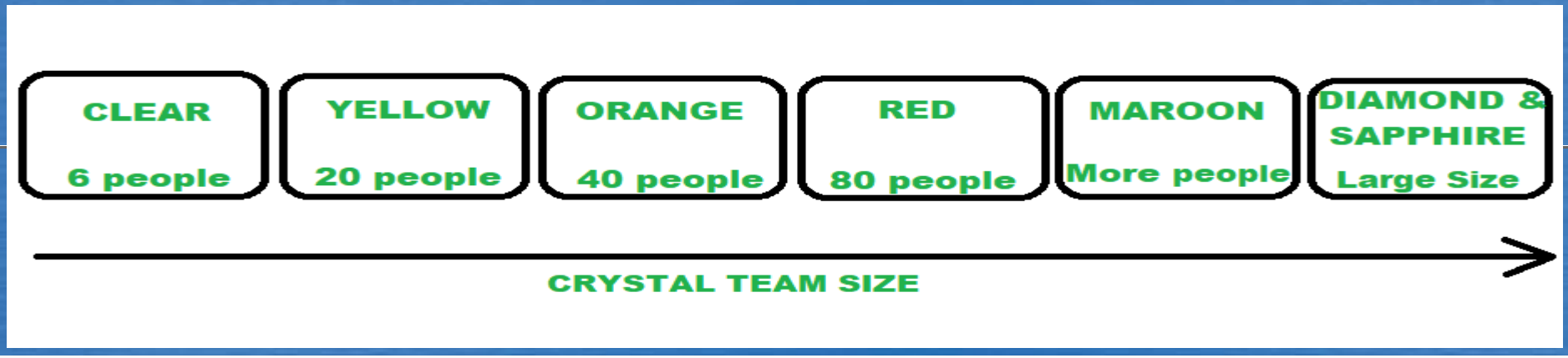
11. **Incremental development –** The framework promotes incremental development, enabling the team to deliver working software frequently, and adapt to changes as they arise.

12. **Automated testing –** The framework emphasizes on automated testing, enabling the team to detect and fix bugs early, reducing the cost of fixing errors at later stages.

13. **Customer involvement –** The framework emphasizes on involving customers in the development process, promoting customer satisfaction, and delivering products that meet their needs.

14. **Leadership** – The framework encourages leadership, enabling team members to take ownership of their work and make decisions that contribute to the success of the project.

How does Crystal function?



1. **Crystal Clear-** The team consists of only 1-6 members that is suitable for short-term projects where members work out in a single workspace.
2. **Crystal Yellow-** It has a small team size of 7-20 members, where feedback is taken from Real Users. This variant involves automated testing which resolves bugs faster and reduces the use of too much documentation.
3. **Crystal Orange-** It has a team size of 21-40 members, where the team is split according to their functional skills. Here the project generally lasts for 1-2 years and the release is required every 3 to 4 months.
4. **Crystal Orange Web-** It has also a team size of 21-40 members were the projects that have a continually evolving code base that is being used by the public. It is also similar to Crystal Orange but here they do not deal with a single project but a series of initiatives that required programming.
5. **Crystal Red-** The software development is led by 40-80 members where the teams can be formed and divided according to requirements.
6. **Crystal Maroon-** It involves large-sized projects where the team size is 80-200 members and where methods are different and as per the requirement of the software.
7. **Crystal Diamond & Sapphire-** This variant is used in large projects where there is a potential risk to human life.

Advantages:

- Effective team communication and this facilitates learning amongst team members from each other
- This methodology can be adjusted as per project type and team size
- Crystal Clear is an agile methodology for projects with small teams, less than about 10 people in size.
- It supports fixed price contracts
- Crystal Clear is not mutually exclusive with other methodologies

Disadvantage:

- The planning & development are not depended on requirements
- May not work well for distributed teams.
- Moving from one flavor of Crystal to another in mid project doesn't work
- Crystal was not designed to be upward or downward compatible.
- varying by project size and criticality

Feature Driven Development

- Originally proposed by Peter Coad
- FDD—distinguishing features
  - Emphasis is on defining "features"
    - a feature "is a client-valued function that can be implemented in two weeks or less."
  - Uses a feature template
    - <action> the <result> <by | for | of | to> a(n) <object>
  - A features list is created and "plan by feature" is conducted
  - Design and construction merge in FDD

FDD adopts a philosophy that

(1) emphasizes collaboration among people on an FDD team

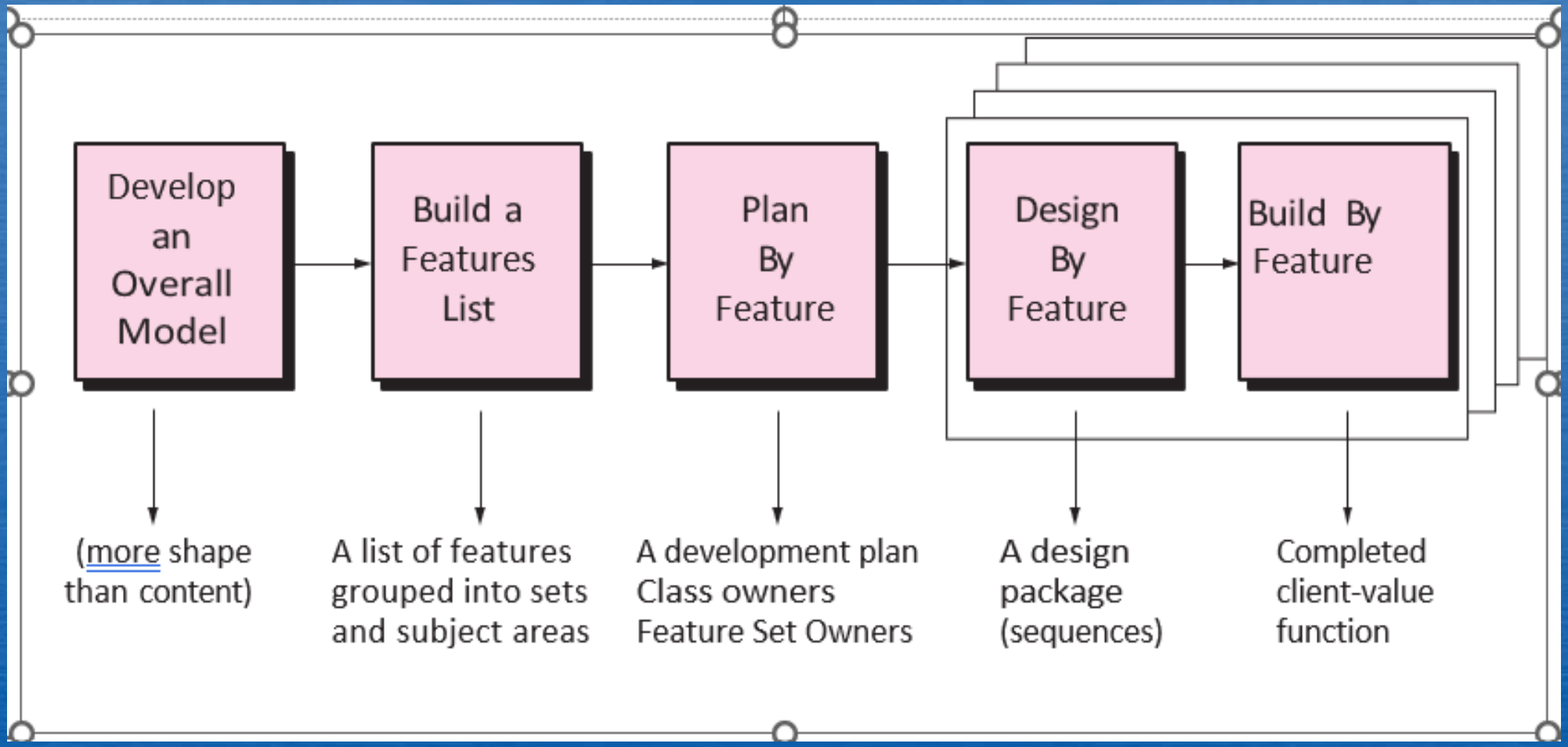
(2) manages problem and project complexity using feature-based decomposition followed by the integration of software increments

(3) communication of technical detail using verbal, graphical, and text-based means.

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 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.

The FDD approach defines five “collaborating” framework activities (in FDD these are called “processes”)



- Process one – develop an overall model: The FDD model insists that teams exert the adequate amount of effort at the start of the project in order to build an object model highlighting the domain problem. Modelling with feature driven development is time-boxed and collaborative. Domain models should be created in detail by small groups and then presented for peers to review. It is hoped that a proposed model – or potentially a combination of them – will then be used for each area of the domain. They will then be merged over time to produce an overall model.

- Process two – build a feature list: From the knowledge that is obtained during the modelling process, a list of features is established by dividing domains into subject areas that contain information on business activities. The steps that are used for each business activity represent a categorised list of features. Features are expressed in the form of: “action, result, object”. The expectation is that they will not take more than two weeks to complete: if they do, they should be broken into smaller sections.

- Process three – plan by features: Once the feature list has been established, the following process involves assigning the various feature sets to the programmers.
- Process four – design by feature: Programmers then produce a design package for each feature with a chief programmer selecting a group of features that should be developed within a two-week period. The chief programmer will also establish detailed diagrams for each feature while refining the model. When this is complete, prologues are produced and a design inspection is carried out.

- Process five – build by feature: Once design inspections are complete, designers plan an activity for each feature and develop the code for their respective classes. When the inspection is complete and a unit test carried out, the feature is then pushed to the main build.

Feature driven development: best practices

At the heart of FDD are a number of best practises designed for software engineering: all of which are formed from a client's perspective. Some of the best practices that should be followed by developers include:

- Domain object modelling: Explores and explains the problem that needs to be sold and provides a framework from which features can be added.

- Developing by feature: Functions that cannot be implemented within two weeks should be divided into smaller functions: with each sub-problem to be small enough to be labelled as a feature. This should make it easier to modify the system and ensure that the correct functions are offered.
- Individual class ownership: Groups of code should be assigned to individual owners.

- Feature teams: A dynamic small team that focuses on individual activities. This ensures that multiple minds are used on each design decision so that several options are always explored. Typically roles will include – a project manager, chief architect, development manager, domain expert, class owner and chief programmer.
- Inspections: They should be carried out regularly to ensure there are no defects.

- Configuration management: Identifies source code for all features and maintains a history of the changes that are made so that the feature teams can enhance them.
- Regular builds: Ensures that the system is always up to date.
- Visibility of progress: Progress reporting should be carried out on a regular basis to ensure that managers can steer the project in the right direction.

FDD’s strengths include:

- Simple five-step process allows for more rapid development
- Allows larger teams to move products forward with continuous success
- Leverages pre-defined development standards, so teams are able to move quickly

FDD’s weaknesses include:

- Does not work efficiently for smaller projects
- Less written documentation, which can lead to confusion
- Highly dependent on lead developers or programmers

Lean Software Development(LSD)

History of Lean Thinking and Lean Software Development

- Toyota has started in the 1980s to revolutionize the automobile industry with their approach of "Lean Manufacturing“
  - to eliminate waste
  - to streamline the value chain (even across enterprises)
  - to produce on request (just in time), and
  - to focus on the people who add value.
- Lean Thinking capitalizes on the intelligence of frontline workers, believing that they are the ones who should determine and continually improve the way they do their jobs.
- Mary and Tom Poppendieck have transferred principles and practices from the manufacturing environment to the software development environment.

The 7 principles of Lean Thinking

1. Eliminate Waste
2. Amplify Learning
3. Decide as Late as possible
4. Deliver as Fast as possible
5. Empower the Team
6. Build Integrity In
7. See the Whole

Principle #1: Eliminate Waste

- Does not mean to throw away all documentation, but to spend time only on what adds real customer value.
- Thus, the first step to implementing lean development is learning to see waste.

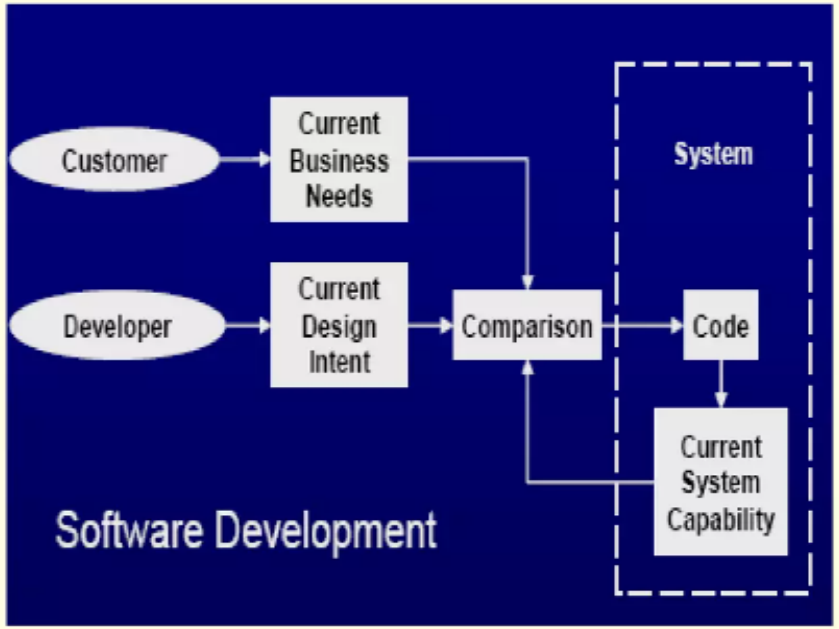
- The second step is to uncover the biggest sources of waste and eliminate them.

Seeing Waste

Wastes of Manufacturing	Wastes of Software Development
Inventory	Partially work done
Extra processing	Paperwork or excess documentation
Overproduction	Extra features
Transportation	Building the wrong thing
Waiting	Waiting for the information
Motion	Task switching
Defects	Defects

Principle #2: Amplify Learning

- Does not mean to keep on changing your mind, but to increase feedback, when you have tough problems.
- When a problem develops…
  - the first thing to do is to make sure the feedback loops are all in place,
  - the next thing to do is to increase the frequency of the feedback loops in the problem areas.



Amplify Learning with Synchronization

- Whenever several individuals are working on the same thing, a need for synchronization occurs. The need for synchronization is fundamental to any complex development process.

- Synchronize / integrate technically:
  - Integrate daily within a team (i.e. check-in at least daily into local repository)
  - Integrate weekly across multiple teams (i.e. check-in at least weekly into the central repository)

Principle #3: Decide as Late as Possible

- Does not mean to procrastinate, but to keep your options open as long as practical, but no longer.
- We usually don't give our customers the option to change their minds. And yet, almost everyone resists making irrevocable decisions in the face of uncertainty. Options allow fact-based decisions based on learning rather than speculation.
- Premature design commitment restricts learning, exacerbates the impact of defects, limits the usefulness of the product, and increases the cost of change.
- But, options are not free and it takes expertise to know which options to keep open.

Principle #4: Deliver as Fast as Possible

- Does not mean to rush and do sloppy work, but to deliver value to customers as soon as they ask for it.
- Customers like rapid delivery, which often translates to increased business flexibility. Companies can deliver faster than customers can change their minds. Companies have fewer resources tied up in work-in progress.
- The principle *Deliver as Fast as Possible* complements *Decide as Late as Possible*: The faster you can deliver, the longer you can delay decisions.

Real world examples

- The most disciplined organizations are those that respond to customer requests
  - Rapidly
  - Reliably
  - Repeatedly

- Software Development Maturity
  - The speed at which you reliably and repeatedly convert customer requests to deployed software

Principles of Speed

- Pull from customer demand
  - Pull with an order
  - Don't push with a schedule

- Make work self-directing
  - Visual Workplace
- Rely on local signaling and commitment
  - Kanban
  - Scrum Meetings
- Use Small Batches
  - Limit the amount of work in the pipeline

Principle #5: Empower the Team

- Does not mean to abandon leadership, but to let the people who add value use their full potential.
- While software development cannot be successful without disciplined, motivated people, experimentation and feedback are more effective than trying to getting things right the first time.
- The critical factor in motivation is not measurement, but empowerment: moving decisions to the lowest possible level in an organization while developing the capacity of those people to make decisions wisely.

Empower the Team with..

- Motivation
  - Intrinsic motivation requires a feeling of belonging, a feeling of safety, a sense of competence, and sense of progress
- Leadership

Managers	Leaders
Cope with Complexity ▪ Plan and Budget ▪ Organize and Staff ▪ Track and Control	*Cope with Change* ▪ Set Direction ▪ Align People ▪ Enable Motivation

- Expertise

6. Build Integrity In

- Integrity refers to the quality of the software code, which is why this principle is sometimes referred to as “Build quality in.” Products have two forms of integrity: perceived and conceptual.

- **Perceived integrity** – the elements of the software that are obvious to the end-user (can see or interact with them). E.g. User Interface. This form of integrity prioritizes the understanding of customer needs and effective communication of requirements.

- **Conceptual integrity –** the elements of the software that users never see or engage with, ensuring the vision of the product reflects a consistent whole that is balanced for “flexibility, maintainability, efficiency, and responsiveness.” The product should work on consistent entities or operations to avoid duplication and ensure the system as a whole has integrity.

- There is a fine balance in Lean between building quality and creating waste: you need enough testing to ensure a quality product free of major defects, but too much testing can be a source of waste.

7. See the Whole

- Lean recognizes that software is more than just the sum of its parts, but also the interactions between the parts. Optimizing the whole looks at dependencies and builds for collaboration to ensure that every employee is incentivized to optimize for the whole – not just for their part.

Conclusion

- The lean production metaphor is a good one for software development, if it is applied in keeping with the underlying spirit of lean thinking.
- The underlying principles of eliminating waste, empowering front line workers, responding immediately to customer requests, and optimizing across the value chain are fundamental to lean thinking
- When applied to software development, these concepts provide a broad framework for improving software development.

Agile Modeling

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 who must solve it

(3) quality can be assessed as the system is being engineered and built.

The Official Agile Modeling Site," Scott Ambler describes *agile modeling* (AM) in the following manner:

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.

Agile Modeling

- Originally proposed by Scott Ambler
- Suggests a set of agile modeling principles
  - Model with a purpose
  - Use multiple models
  - Travel light
  - Know the models and the tools you use to create them
  - Adapt locally
  - Content is more important than representation

Agile modeling principles

AM suggests a wide array of “core” and “supplementary” modeling principles, those that make AM unique are :

Model with a purpose- A developer who uses AM should have a specific goal (e.g., to communicate information to the customer or to help better understand some aspect of the software) in mind before creating the model. Once the goal for the model is identified, the type of notation to be used and level of detail required will be more obvious.

Use multiple models. There are many different models and notations that can be used to describe software. AM suggests that to provide needed insight, each model should present a different aspect of the system and only those models that provide value to their intended audience should be used

- **Travel light. -**As software engineering work proceeds, keep only those models that will provide long-term value. Every work product that is kept must be maintained as changes occur.. Ambler notes that “Every time you decide to keep a model you trade-off agility for the convenience of having that information available to your team in an abstract manner (hence potentially enhancing communication within your team as well as with project stakeholders).”

- **Content is more important than representation-** Modeling should impart information to its intended audience. A syntactically perfect model that imparts little useful content is not as valuable as a model with flawed notation that nevertheless provides valuable content for its audience.

- **Know the models and the tools you use to create them**. Understand the strengths and weaknesses of each model and the tools that are used to create it.
- **Adapt locally** - modeling approach should be adapted to the needs of the agile team.

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 (i.e., a linear sequence of software engineering activities) that enables a team to visualize the overall process flow for a software project.

- 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-**

1. **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.

2. **Implementation-** Models are translated into source code.
3. **Testing-** Like XP, the team designs and executes a series of tests to uncover errors and ensure that the source code meets its requirements.
4. **Deployment.-**deployment in this context focuses on the delivery of a software increment and the acquisition of feedback from end users.
5. **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.
6. **Environment management-** Environment management coordinates a process infrastructure that includes **standards, tools, and other support technology available to the team.**

The Agile UP is based on the following philosophies

1. **Your staff know what they're doing-** People are not going to read detailed process documentation, but they will want some high-level guidance and/or training from time to time. The AUP product provides links to many of the details, if you are interested, but doesn't force them upon you.
2. **Simplicity-** Everything is described concisely using a handful of pages, not thousands of them.
3. **Agility--** The Agile UP conforms to the values and principles of the agile software development and the Agile Alliance.
4. **Focus on high-value activities-**The focus is on the activities which actually count, not every possible thing that could happen to you on a project.
5. **Tool independence-**You can use any toolset that you want with the Agile UP. The recommendation is that you use the tools which are best suited for the job, which are often simple tools.
6. You'll want to tailor the AUP to meet your own needs.

THE END