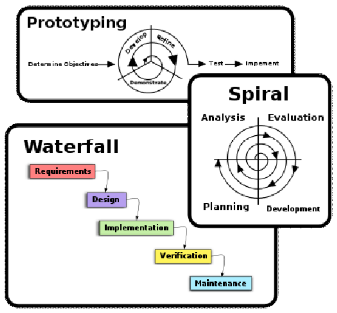
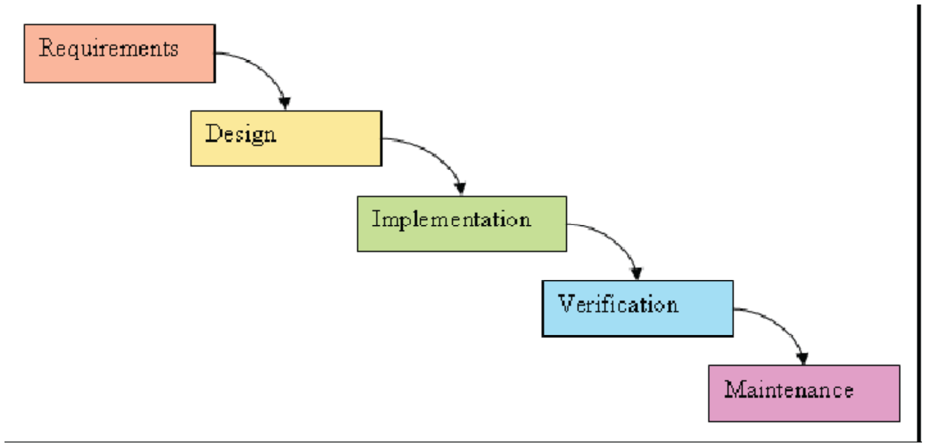
Software Development Life Cycles (SDLC)

- Waterfall
- Prototyping
- Spiral
- Incremental
- Agile development methodologies
    - Extreme Programming (XP)
    - Scrum
    - Dynamic Systems Development Method (DSDM)
    - Adaptive Software Development
    - Crystal



Waterfall



Classical Waterfall model

The classic waterfall model is a popular version of the SDLC for software engineering, which was introduced in the 1970s by Win Royce at Lockheed. It is so named because it can be represented or graphically modelled as a cascade from establishing requirements, to design creation, to program implementation, to system test, to release to customer, as shown in Figure 1.

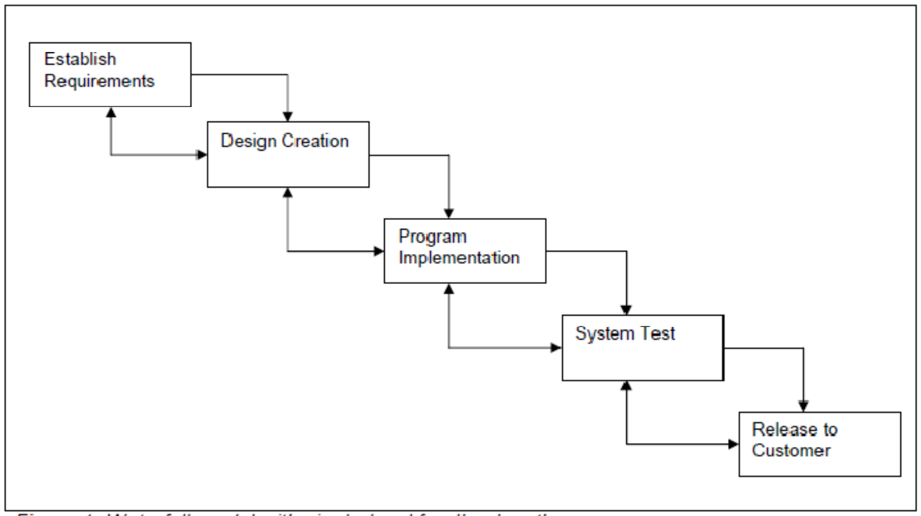


Figure 1. Waterfall model with single-level feedback paths

It was a great step forward in software development as an engineering discipline. The figure also depicts the single-level feedback paths that were not part of the original model but that have been added to all subsequent improvements of the model. The original waterfall model had little or no feedback between stages; it describes a development method that is linear and sequential.

Waterfall development has distinct goals for each phase of development, just as water does not reverse or flow uphill. Imagine a waterfall on the cliff of a steep mountain. Once the water has flowed over the edge of the cliff and has begun its journey down the side of the mountain, it can't turn back. It is the same with waterfall development. Once a phase of development is completed, the development proceeds to the next phase and there is no turning back.

This method might work satisfactorily if design requirements could be perfectly addressed before flowing down to design creation, and if the design were perfect when program implementation began, and if the code were perfect before testing began, and if testing guaranteed that no bugs remained in the code before the users applied it, and of course if the users never changed their minds about requirements. Unfortunately, from experience, none of these things is ever true.

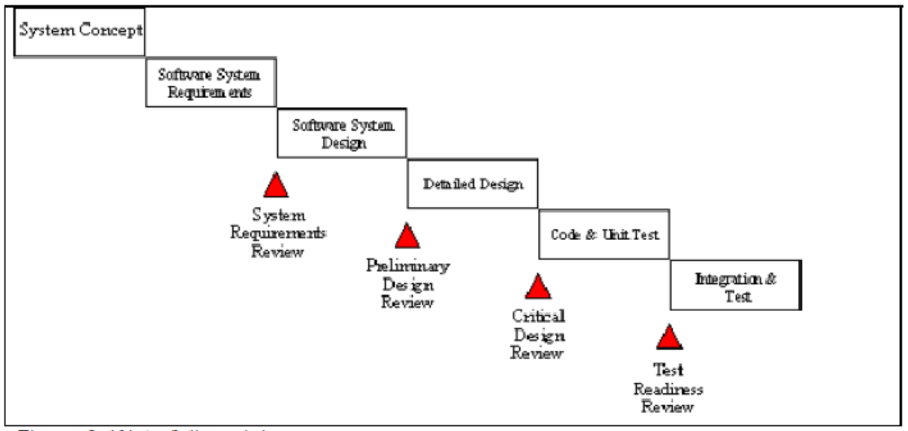


Figure 2. Waterfall model

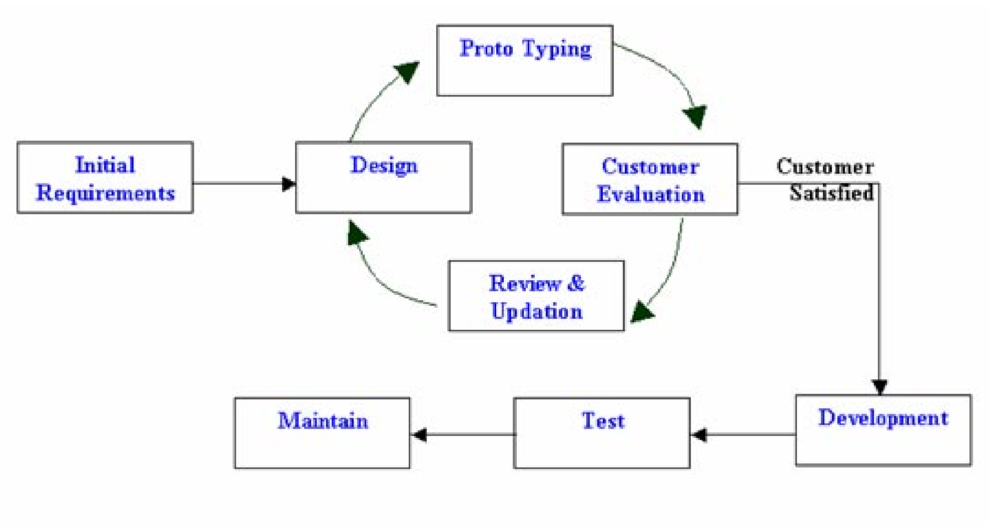
In figure 2 the approach is used in high risk projects, particularly large defense contracts. The problems in waterfall do not arise from "immature engineering practices, particularly in requirements analysis and requirements management." Studies of the failure rate of the DODSTD-2167 specification, which enforced waterfall, have shown that the more closely a project follows its process, specifically in up front requirements gathering, the more likely the project is to release features that are not used in their current form. In terms software sizes and complexity and millions lines of code in current systems, it is hardly for waterfall handle such projects.

The advantage of waterfall devolvement method is that, it allows for departmentalization and managerial control. A schedule with deadlines for each stage of the development can be set and a product can proceed through the development process, theoretically, to be delivered on time. In more simplicity, development moves from concept, through design, implementation, testing, installation and ends up in production and maintenance, each phase of development proceeds in strict order without any overlapping or iterative steps.

The disadvantage of waterfall development method/model is that it does not allow for much reflection or revision. Once an application/system is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.

Prototyping

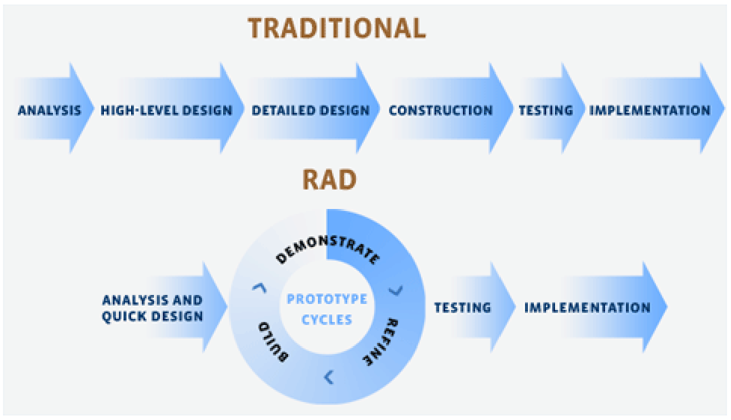
Software prototyping is the development approach of activities during software development, the creation of prototypes, i.e., incomplete versions of the software program being developed.



Proto Type Model

The basic principles are:

- Not a standalone, complete development methodology, but rather an approach to handle selected parts of a larger, more traditional development methodology (i.e. incremental, spiral, or rapid application development (RAD)).
- Attempts to reduce inherent project risk by breaking a project into smaller segments and providing more ease-of-change during the development process.
- User is involved throughout the development process, which increases the likelihood of user acceptance of the final implementation.
- Small-scale mock-ups of the system are developed following an iterative modification process until the prototype evolves to meet the users' requirements.
- While most prototypes are developed with the expectation that they will be discarded, it is possible in some cases to evolve from prototype to working system.
- A basic understanding of the fundamental business problem is necessary to avoid solving the wrong problem.



Spiral

While the waterfall methodology offers an orderly structure for software development, demands for reduced time-to-market make its series steps inappropriate. The next evolutionary step from the waterfall is where the various steps are staged for multiple deliveries or handoffs, as shown in figure 3.

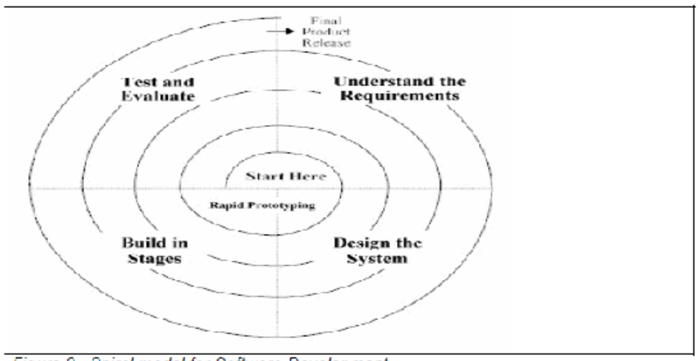


Figure 3. Spiral model for Software Development

The ultimate evolution from the waterfall is the spiral, taking advantage of the fact that development projects work best when they are both incremental and iterative, where the development team is able to start with a small team and benefit/learn from progressive trial and error along the way.

The spiral methodology reflects the relationship of tasks with rapid prototyping, increased parallelism and concurrency in design and builds activities. The spiral method should still be planned methodically, with tasks and deliverables identified for each step in the spiral.

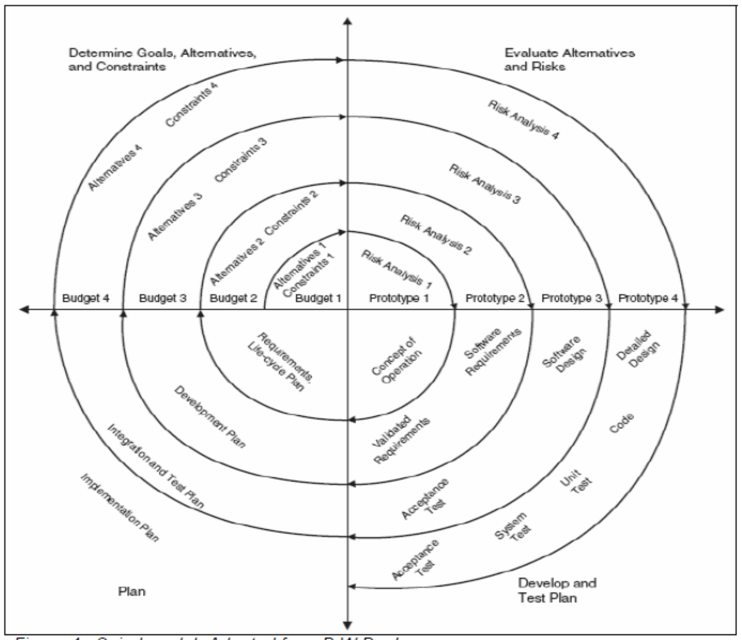


Figure 4. Spiral model- Adapted from B.W.Boehm

The spiral model, developed by Dr. Barry Boehm at TRW, is an enhancement of the waterfall/rapid prototype model, with risk analysis preceding each phase of the cascade. You can imagine the rapid prototyping model drawn in the form of a spiral, as shown in Figure 4. This model has been successfully used for the internal development of large systems and is especially useful when software reuse is a goal and when specific quality objectives can be incorporated. It does depend on being able to accurately assess risks during development. This depends on controlling all factors and eliminating or at least minimizing external influences. Like the other extensions of and improvements to the waterfall model, it adds feedback to earlier stages. This model has seen service in the development of major programming projects over a number of years, and is well documented in publications by Boehm and others.

Incremental

This model combines the elements of the waterfall model with the iterative philosophy of prototyping. The incremental recognizes that software development steps are not discrete. Instead, it consists of builds of prototypes, so Build 1 (a prototype) is improved and functionality/features are added until it becomes Build 2, which becomes Build 3, and so on. These builds are not the versions released of the product to the customer, but are merely staged compilations of the developing system at a new level of functionality or completeness. As a major system nears completion, the project manager may schedule a new build every week. It is intended to deliver an operational system with high quality that meets customer requirements at each build, but it does not yet complete the functional specification.

The advantages of the incremental are:
It is flexible enough to respond to critical specification changes as development progresses. Also the analysts and developers can tackle smaller chunks of complexity. Users and developers both learn from a new system's development process, and any model that allows them to incorporate this learning into the product is advantageous.

The disadvantage:
Learning exceeds productivity and the development project becomes a research project exceeding time and budget or even worse, it never delivers the product at all. However, learning need not exceed productivity if the development team remains aware of the risks and focused on customer requirements.

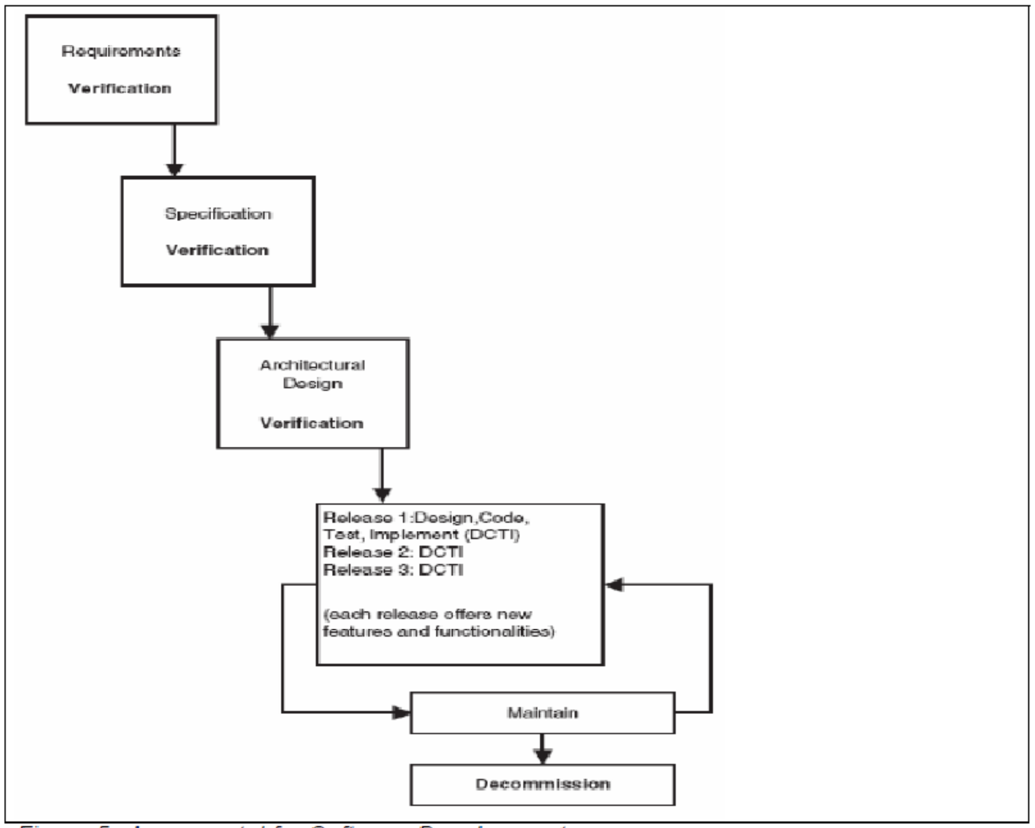


Figure 5. Incremental for Software Development

Agile development methodologies

Extreme Programming (XP)

Extreme programming is one of the agile development methodologies. It is the development of the incremental model that puts the client in the right path. Each feature or feature set of the final product envisioned by the client and the development team is individually scoped for cost and development time (project resources). The client then selects features that will be included in the next build (again, a build is an operational system at some level of functionally) based on a cost benefit analysis.

Extreme Programming (XP) is an agile software development method that was created by Kent Beck and was first used in 1996 in a project at Chrysler. Figure 6 below, shows the lifecycle of the XP process. The XP lifecycle consists of six phases:

1. Exploration phase.
2. Planning phase
3. Iterations to release phase
4. Product-ionizing phase
5. Maintenance phase
6. Death phase.

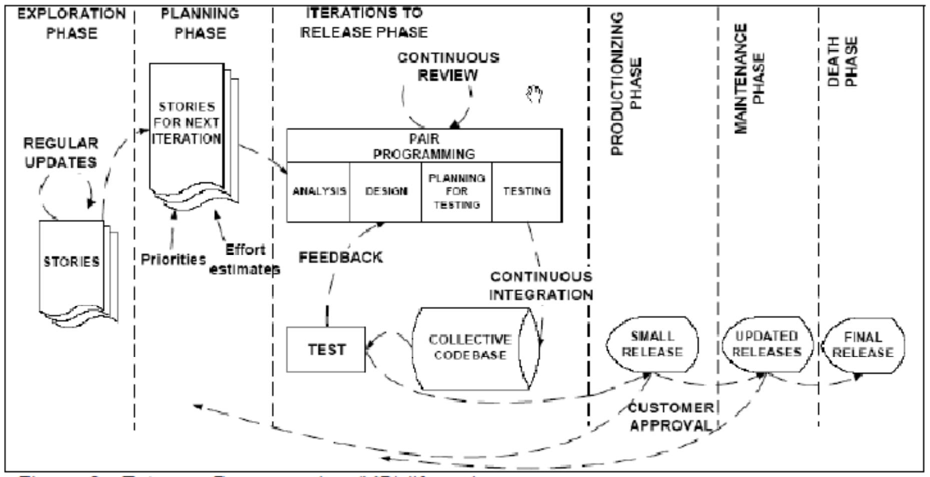


Figure 6. Extreme Programming (XP) lifecycle

During the exploration phase, the customer writes his requirements on so-called story cards (cards which contain a feature to be implemented) and the developers work on technologies to be used in the project. Based on this preparation, a prototype of the system might be built.
Within the planning phase, also known as the “planning game”, the developers estimate for each card how long it would take to implement this card and based on this estimations, customer and developers decide together about the prioritization of each card. According to this prioritization, a release plan/schedule is finally set up which says which feature will be implemented in each release.
In the iterations to release phase the prioritized story cards are implemented in iterations of one to four weeks. In these iterations, the design as well as the coding is done, but before any line of code is written, first a unit test to test these lines has to be developed by the programmers.
Finally, as soon as the developed features are tested by the developers (probably by automated unit tests), they are given to the customer and thereby, the next phase is entered.

The product- ionizing phase mainly concerns performance and system testing. In this phase, the customer performs functional tests and validates if the product works as in-tended. If new requirements are elicited, they are either included directly or a new story card is created which will be considered in a following release planning.

Throughout the maintenance phase, the customer is supported by (probably new) team members whose task is to ensure that certain customer requests as e.g. improvement suggestions are considered. In the death phase, the XP project is either successfully finished or finished without the expected results, e.g. if the budget is overrun. In both cases, no changes to the sys-tem are made anymore. The only action that might still happen in this phase is writing documentation.

Scrum

The Scrum methodology was invented in 1993 and developed in 1996 by Jeff Sutherland and Ken Schwaber. It is worth to mention that, XP and Scrum are an iterative software development method. Scrum process is divided into 3 phases:

1. Pregame,
2. Development,
3. Postgame (Figure 7)

The pregame phase is sub-divided into planning and architecture/high level design. During the planning, a so-called product backlog list (a requirements specification) is written, the requirements are prioritized and the development effort is estimated. Moreover, the project resources (time, budget and staff) are fixed. Throughout architecture/high level design, the high level design based on the product backlog is defined and reviewed in a meeting where implementation details are decided.

Secondly, the development phase consists of so-called sprints (iterative software development cycles that last from one to four weeks) which result in a release. The sprints are based on a sprint backlog list (a document that contains the features to be implemented in the sprint). Each sprint includes activities of traditional software development approaches (e.g. requirements engineering, design, coding etc.) and can be performed by several teams.

Finally, the postgame phase is characterized by documentation, integration testing and system testing. The postgame phase ends with the final release of the software.

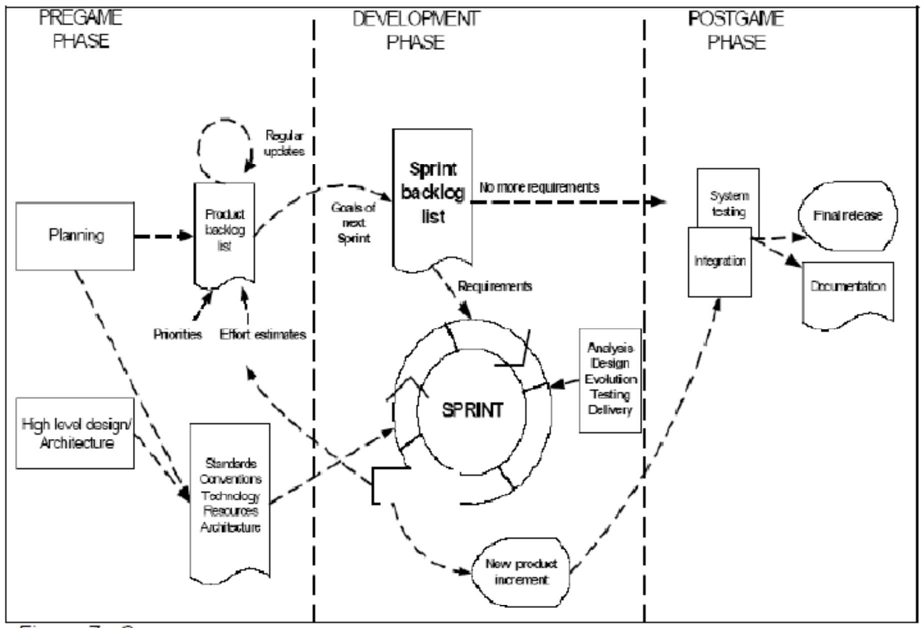


Figure 7. Scrum process

Dynamic Systems Development Method (DSDM)

DSDM has its origin in 1994 and was established by a group of 16 people, the DSDM consortium. It is a Rapid Application Development (RAD) framework with the basic idea of fixing the project resources before fixing the functionality. In figure 8, the DSDM process is divided into five phases:

1. Feasibility study
2. Business study
3. Functional model iteration
4. Design and build iteration and
5. Implementation phase

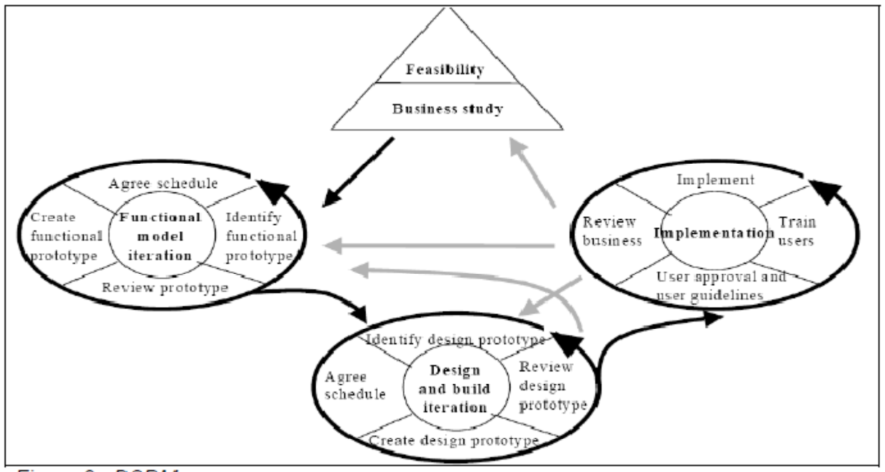

Feasibility
Business study
Agree schedule
Create functional prototype
Functional model iteration
Identify functional prototype
Review prototype
Implement
Review business
Implementation
Train users
User approval and user guidelines
Identify design prototype
Agree schedule
Design and build iteration
Review design prototype
Create design prototype

Figure 8. DSDM

1. Feasibility study
The DSDM process starts with the *feasibility study*. The feasibility study is utilized to evaluate whether the planned project is technically feasible and whether to use or not use DSDM. The feasibility study results in the feasibility report (describes the feasibility of the project) and the development outline (general outline of the system to be developed). If the technologies to be used are not well known, a prototype can be built within the feasibility study.

2. Business study
During the *business study*, the technological and business requirements are specified and prioritized with the help of the customer. The business study results in the business area definition (process descriptions with the help of e.g. entity-relationship diagrams and specification of user classes to be involved), the system architecture definition (a first architecture draft), and the prototyping plan (a plan including the prototyping strategy and configuration management).

3. Functional model iteration
After the business study, in the *functional model iteration*, several iterations are used to find a functional model for the system that is to be developed. Finding this model also includes the development of prototypes and depending on the quality of these prototypes, it might even occur that parts of these prototypes are later integrated in the final system. The functional model

iteration results in a functional model, including prototype code and analysis models. Besides, the functional model iteration produces a list of prioritized functions, functional prototyping review documents including user comments, a list of non-functional requirements to be included, and a document about risk analysis of further development.

4. Design and build iteration
In the *design and build iteration phase* the system is completed based on the most important requirements. This completion is also done by several iterations. As well as in the preceding phase, prototypes are built, tested, and reviewed by the users.

5. Implementation
The *implementation phase* (which is also done in iterations) is utilized to transfer the system into its real application field at the customer. If it was not possible to complete the system in the given time, the DSDM process might be run again in order to include remaining features. The implementation phase results in two things, first, a user manual which explains to the users how to use the system, and second, in a project review document in which the project's outcome is specified.

DSDM roles are:

- *Executive sponsor* (customer representative with financial authority)
- *Visionary* (customer representative with high knowledge of the business do-main)
- *Ambassador user* (spreads information about the project progress & user in-formation)
- *Project Manager*
- *Technical coordinator* (responsibility for the system architecture, technical quality, and configuration management)
- *Team Leader* (ensures team work)
- *Developer* (transfers requirements into code)
- *Tester* (responsibility for the non-user testing)
- *Scribe* (makes notes of requirements in meetings and workshops) and
- *Facilitator* (manages workshops).

Adaptive Software Development

Adaptive Software Development (ASD) was developed by Jim Highsmith and was first published in 1997. The methodology is focused on iterations and constant prototyping and is basically supposed to be used for the development of large software systems. The ASD process is divided into three phases:

1. Speculate
2. Collaborate, and
3. Learn

During the *speculate phase*, the objectives, vision, goals, and requirements of the sys-tem to be developed are specified. These artifacts are also called the mission of the project and they are identified with Joint Application Development (JAD) sessions, i.e. workshops with customer representatives and developers. ASD uses the terms speculate due to Highsmith's opinion that in large environments outcomes are unpredictable.

Secondly, the *collaborate phase* mainly deals with the iterative development of product components. ASD rather emphasizes the quality of the components which is reached by teamwork. At the end of each iteration, quality reviews are held with the help of the customer.

Finally, the *learn phase* concerns final quality assurance and lessons learned sessions. The lessons learned sessions are important for future projects because the employees shall avoid mistakes they made before.

ASD roles are:

• *Executive sponsor* (has the whole project responsibility)
• *Facilitator* (organizes the JAD sessions)
• *Scribe* (makes notes during the JAD sessions)
• *Project Manager*
• *Customer*, and
• *Developer*

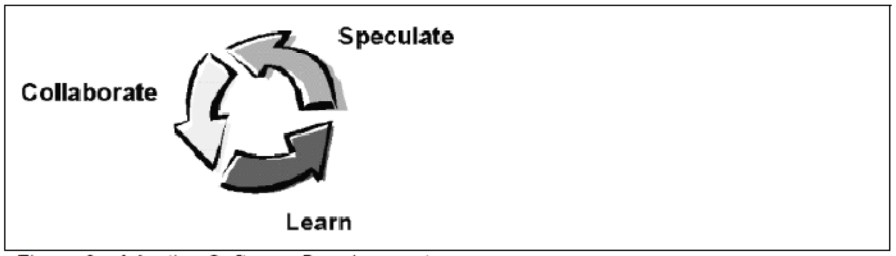


Figure 9. Adaptive Software Development

Crystal

The Crystal Family of Methodologies (Crystal) was developed by Alistair Cockburn. Crystal is based on several software development methods in order to choose the most appropriate technique for a project and is a typical people-centric methodology since it focuses on improving the human power. The methods are shown in figure 10, where they are categorized by the size of the project by the number of participating employees (x-axis) and the criticality of the system (y-axis).

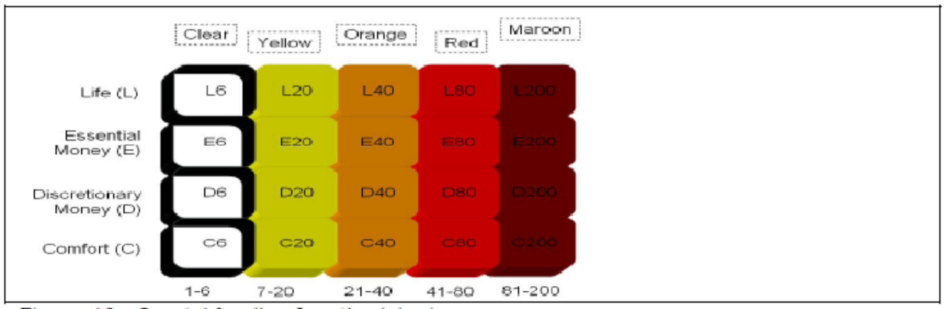


Figure 10: Crystal family of methodologies