# BUSINESS ANALYST'S VIEW OF SOFTWARE DEVELOPMENT LIFE CYCLE MODELS

## Table of Contents



## List of Figures

## General Approach

Regardless of the time an activity takes whether they are done simultaneously or in long planned phases fraught with documentation and approvals, the software development life cycle (SDLC) must answer certain questions about the product being developed.

- What is the business problem being solved? Concept Phase
- What is the solution to that problem? Requirements
- How are we going to affect the solution? Logical Design
- What are the elements of that solution? Physical Design and Coding
- How do we know our solution is right? Unit, Integration and System Testing
- How do we know we have the right solution? Acceptance Testing
- Will it work in the environment with the actual users? Implementation

Each of the life cycle models includes activities, tasks, or phases that answer these questions, although not necessarily in the direct format given above.

## Linear or Phased Approaches

### *Waterfall*

While the Waterfall Model presents a straight-forward view of the software life cycle, this view is only appropriate for certain classes of software development. Specifically, the Waterfall Model works well when the software requirements are well understood (e.g., software such as compilers or operating systems) and the nature of the software development involves contractual agreements. The Waterfall Model is a natural fit for contract-based software development since this model is document driven; that is, many of the products such as the requirements specification and the design are documents. These documents then become the basis for the software development contract.

There have been many waterfall variations since the initial model was introduced by Winston Royce in 1970 in a paper entitled: "Managing the Development of Large Software Systems: Concepts and Techniques". Barry Boehm, developer of the spiral model (see below), modified the waterfall model in his book Software Engineering Economics [Prentice-Hall, 1987]. The basic differences in the various models are in the naming and/or order of the phases.

The basic waterfall approach looks like the illustration below. Each phase is done in a specific order with its own entry and exit criteria and provides the maximum in separation of skills, an important factor in government contracting.

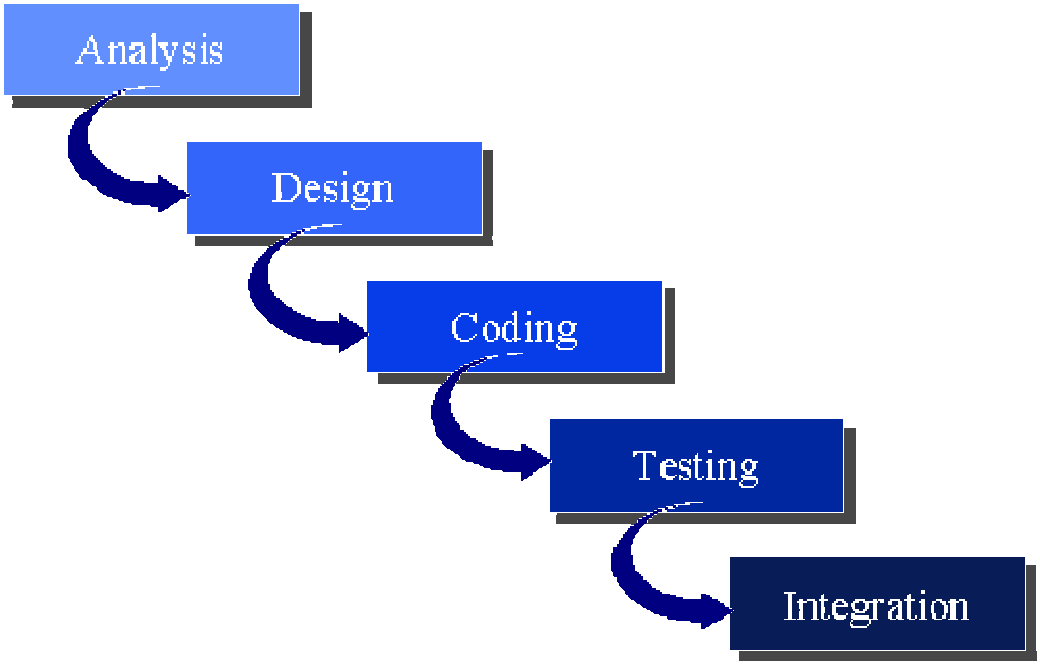


**Figure 1. Example of a Typical Waterfall Approach**

While some variations on the waterfall theme allow for iterations back to the previous phase, “In practice most waterfall projects are managed with the assumption that once the phase is completed, the result of that activity is cast in concrete. For example, at the end of the design phase, a design document is delivered. It is expected that this document will not be updated throughout the rest of the development. You cannot climb up a waterfall.” [Murray Cantor, Object-Oriented Project Management with UML, John Wiley, 1998.]

The waterfall is the easiest of the approaches for a business analyst to understand and work with and it is still, in its various forms, the operational SDLC in the majority of US IT shops. The business analyst is directly involved in the requirements definition and/or analysis phases and peripherally involved in the succeeding phases until the end of the testing phase. The business analyst is heavily involved in the last stages of testing when the product is determined to solve the business problem. The solution is defined by the business analyst in the business case and requirements documents. The business analyst is also involved in the integration or transition phase assisting the business community to accept and incorporate the new system and processes.

## *V Model*

The "V" Model (sometimes known as the "U" Model) reflects the approach to systems development wherein the definition side of the model is linked directly to the confirmation side. It specifies early testing and preparation of testing scenarios and cases before the build stage to simultaneously validate the definitions and prepare for the test stages.

It is the standard for German federal government projects and is considered as much a project management method as a software development approach.

“The V Model, while admittedly obscure, gives equal weight to testing rather than treating it as an afterthought. Initially defined by the late Paul Rook in the late 1980s, the V was included in the U.K.'s National Computing Centre publications in the 1990s with the aim of improving the efficiency and effectiveness of software development. It's accepted in Europe and the U.K. as a superior alternative to the waterfall model; yet in the U.S., the V Model is often mistaken for the waterfall…”

“In fact, the V Model emerged in reaction to some waterfall models that showed testing as a single phase following the traditional development phases of requirements analysis, high-level design, detailed design and coding. The waterfall model did considerable damage by supporting the common impression that testing is merely a brief detour after most of the mileage has been gained by mainline development activities. Many managers still believe this, even though testing usually takes up half of the project time.” [Goldsmith and Graham, “The Forgotten Phase”, *Software Development*, July 2002.]

 

As shown below, the model is the shape of the development cycle (a waterfall wrapped around) and the concept of flow down and across the phases. The V shows the typical sequence of development activities on the left-hand (downhill) side and the corresponding sequence of test execution activities on the right-hand (uphill) side.

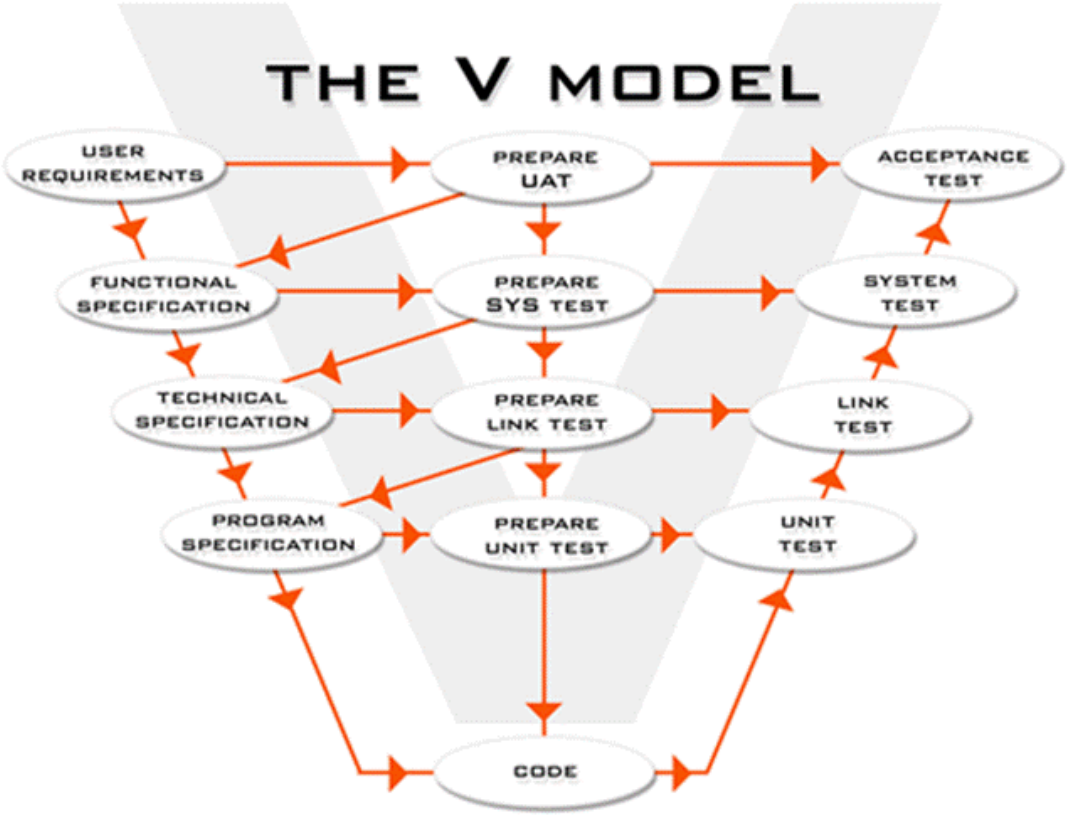


**Figure 2. Example of a Typical V Model (IEEE)**

The primary contribution the V Model makes is this alignment of testing and specification. This is also an advantage to the business analyst who can use the Model and approach to enforce early consideration of later testing. The V Model emphasizes that testing is done throughout the SDLC rather than just at the end of the cycle and reminds the business analyst to prepare the test cases and scenarios in advance while the solution is being defined.

The business analyst's role in the V Model is essentially the same as the waterfall. The business analyst is involved full time in the specification of the business problem and the confirmation and validation that the business problem has been solved. This is done at the acceptance test stage. The business analyst is also involved in the requirements phases and provides advice during the system test stage which is typically performed by independent testers – the

quality assurance group or someone other than the development team. The primary business analyst involvement in the system test stage is keeping the requirements updated as changes occur and providing “voice of the customer” to the testers and development team. The rest of the test stages on the right side of the Model are done by the development team to ensure they have developed the product correctly. It is the business analyst’s job to ensure they have developed the correct product.

## *Incremental Development*

The incremental approach is a generic method of delivering a working part of a total product or solution. It is the basis of most agile and iterative methods including Rational Unified Process (RUP). The agile concept of “timeboxing” is an incremental delivery approach.

Similar to the V Model, incremental development is as much a management approach to software development as a development approach, probably more so. “Incremental development is a scheduling and staging technique nicely suited to projects using technology and techniques new to an organization...[It is] a scheduling and staging strategy that allows pieces of the system to be developed at different times or rates and integrated as they are completed. Specifically intended is that between increments, additions could be made to the requirements, process changes could be incorporated, or the schedule could be improved and revised. Incremental is distinguished from iterative development in that the latter supports "predicted rework" of parts of the system. A good grasp of incremental development helps in applying iterative development...” [Alistair Cockburn, “Unraveling Incremental Development”]

Staged Delivery

The staged delivery model initially advanced by Tom Gilb and later championed by Steve McConnell applies the principles of incremental delivery to the waterfall model. It is a development approach that emphasizes project planning and risk reduction by using multiple software releases.

During the first phases of the staged delivery process, the overall problem is defined and the solution specified. The second phase consists of creating an architecture for the overall solution. After that, the product is partitioned into interim or successive deliveries and each delivery goes through its own development life cycle as shown in the following diagram. The goal of each stage is to advance toward a complete and robust product on time and within budget. Typically each stage has its own budget and deliverable schedule that may be refined based on previous deliverables. The duration of each successive stage is generally similar. The number of planned stages for achieving a final release is dependent upon the functionality, application complexity, and so forth.

 

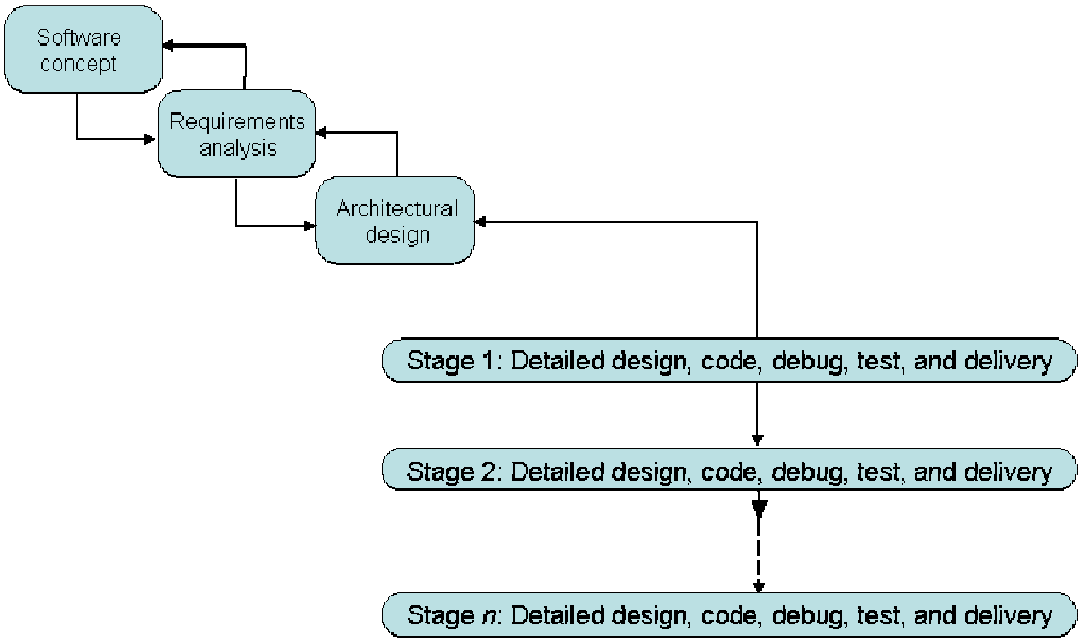


**Figure 3. Example of the Staged Approach**
**[From Steve McConnell, Software Project Survival Guide, Microsoft Press]**

At the core of successful staged delivery is the customer review that occurs between stages and is typically based on using the software in the business environment. This is what Alistair Cockburn calls the "breathing space". It allows for customer feedback and adjustments to the architect, requirements, and process for successive stages.

The staged and any incremental delivery approach presents extra challenges for the business analyst. After the problem has been defined and the product requirements specified, the business analyst then may be handling requirements changes for multiple stages, or addressing the user or stakeholder feedback from previously installed software while assisting with the development and testing of succeeding stages. Especially when multiple development teams are involved, the business analyst activities in a staged or incremental delivery life cycle are best handled in a team effort with multiple business analysts assigned to different stages and product deliverables.

 

## Iterative Approaches

### *Spiral*

The spiral model was defined by Barry Boehm, now at the University of Southern California, as an early example of an iterative approach to software development. It was not the first model to discuss iteration, but it was the first model to explain why the iteration matters. As originally envisioned, the iterations were typically six months to two years long.

Each phase of the spiral starts with a design goal (such as a user interface prototype as an early phase) and ends with the client (which may be internal) reviewing the progress and determining if the next spiral will take place.

The primary difference between the spiral model and the other life cycle models is the emphasis of risk assessment as evidenced by the following illustration. Each cycle through the phases ends with an evaluation by the customer and an assessment of the risk of going ahead with the next iteration or stopping and completing the delivery of the product.

Once the product is deemed ready for development based on customer feedback and risk assessment, the process becomes a standard waterfall approach. The difference is that the high-level design and specifications have been completed and the construction phase has a prototype to follow. This makes the implementation faster and of higher quality.

While the spiral model in its pure form is not used much outside the US Military, it sets the pattern for most of the agile and iterative approaches that follow it. There are also a multitude of “spiral” variations for all or parts of the solution life cycle, such that spiral has almost become synonymous with iterative.

 

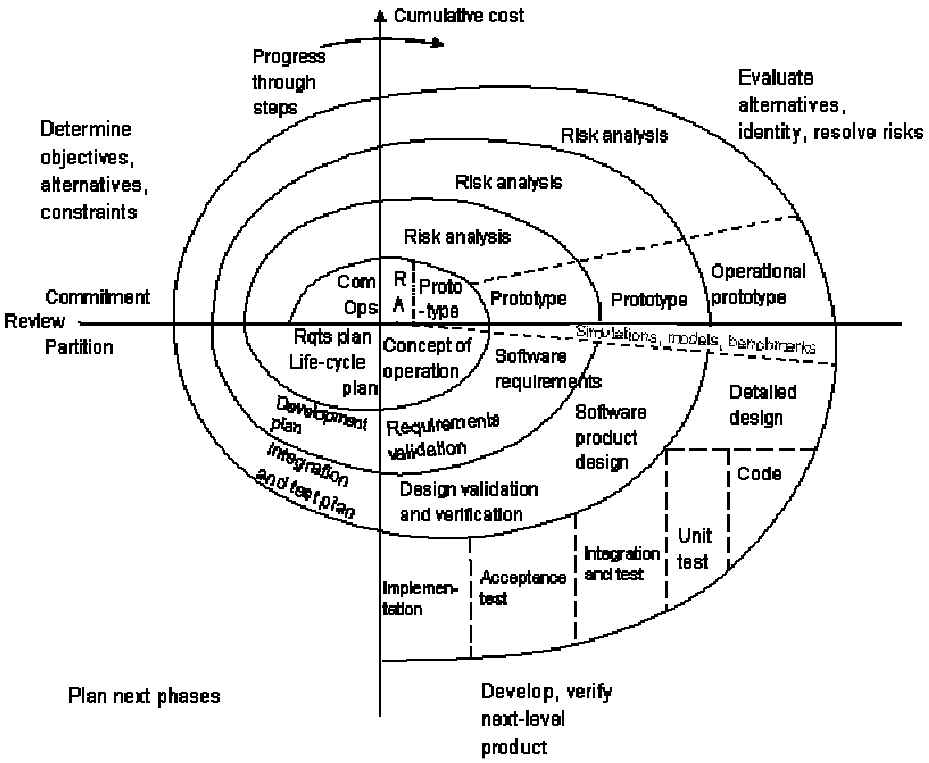


**Figure 4. Example of the Spiral Model**
**[From Boehm, B.W. "A Spiral Model of Software Development and Enhancement." *IEEE Software Engineer and Project Management*, 1987]**

The steps in the spiral model can be generalized as follows:

1. The new system requirements are defined in as much detail as possible. This usually involves interviewing a number of users representing all the external or internal users and other aspects of the existing system.
2. A preliminary design is created for the new system.
3. A first prototype of the new system is constructed from the preliminary design. This is usually a scaled-down system and represents an approximation of the characteristics of the final product.
4. A second prototype is evolved by a fourfold procedure: (a) evaluating the first prototype in terms of its strengths, weaknesses, and risks; (b) defining the requirements of the second prototype; (c) planning and designing the second prototype; and, (d) constructing and testing the second prototype.
5. At the customer's option, the entire project can be aborted if the risk is deemed too great. Risk factors might involve development cost overruns, operating-cost miscalculation, or any other factor that could, in the customer's judgment, result in a less-than-satisfactory final product.
6. The existing prototype is evaluated in the same manner as was the previous prototype, and, if necessary, another prototype is developed from it according to the fourfold procedure outlined above.

7. The preceding steps are iterated until the customer is satisfied that the refined prototype represents the final product desired.
8. The final system is constructed based on the refined prototype.
9. The final system is thoroughly evaluated and tested. Routine maintenance is carried out on a continuing basis to prevent large-scale failures and to minimize downtime.

The business analyst's role in the spiral is further extended into the development life cycle. In addition to the requirements definition and liaison activities, the business analyst is typically quite involved with the prototyping efforts and the risk assessment stages. The business analyst also works with the customer during the evaluations and sometimes represents the customer in evaluating the applicability of the product at that point.

## *Rational Unified Process*

The Rational Unified Process (RUP) was developed by Ivar Jacobson, Grady Booch, and Jim Rumbaugh at Rational Software Corporation after they had defined the Unified Modeling Language (UML). Both the UML and the Unified Process were handed over to the Object Management Group (OMG) to be established as object-oriented development standards. The approach assumes the development will be done using object-oriented analysis, design, and programming as a basis.

"RUP is a software engineering process. It provides a disciplined approach to assigning tasks and responsibilities within a development organization. Its goal is to ensure the production of high-quality software that meets the needs of its end users within a predictable schedule and budget…The Rational Unified Process is also a process framework that can be adapted and extended to suit the needs of an adopting organization." [Philippe Krutchen, RUP: An Introduction, 3rd edition, Addison-Wesley, 2003.]

The RUP is organized around phases and disciplines in a matrix (see the illustration that follows) that emphasizes the iterative nature of the process. Each of the disciplines iterates a number of times through each of the phases until the exit criteria for that phase are achieved. The phases are not defined so much by activities, but by goals and outcomes:

- Inception: agreement among the team and customer as to what will be built,
- Elaboration: agreement within the team as to the architecture and design needed to deliver the agreed system behavior,
- Construction: the iterative implementation of a fully functional system,
- Transition: delivery, defect correction, and tuning to ensure customer acceptance.

 

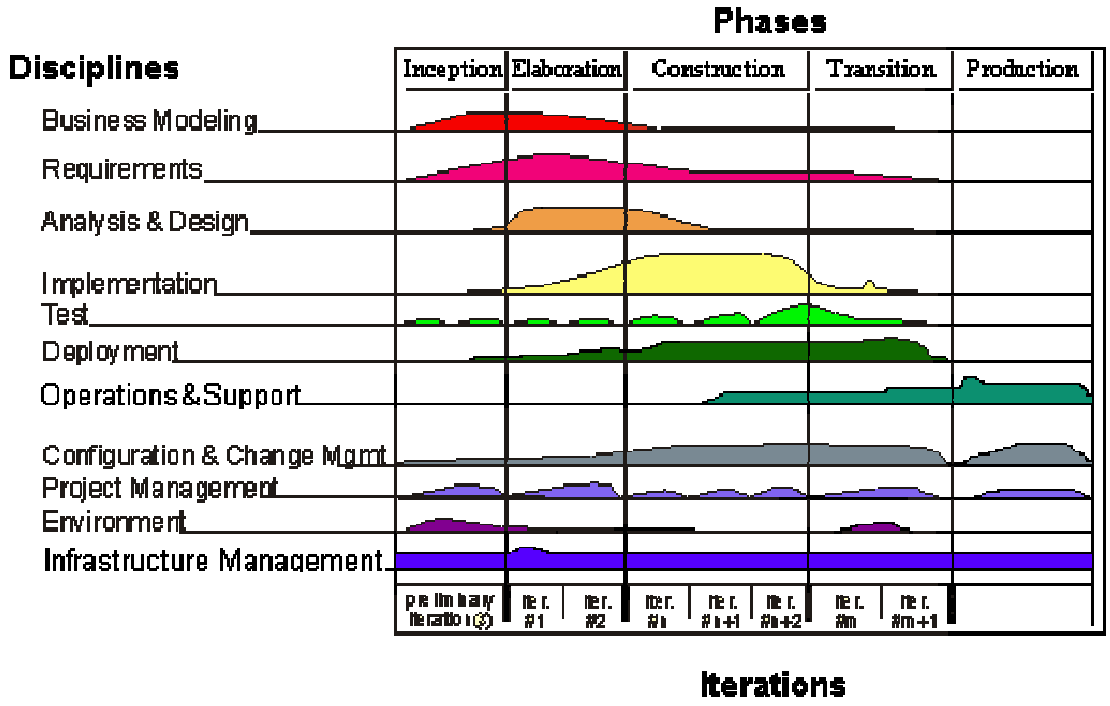


**Figure 5. Example of the Rational Unified Process**
**[From Rational Software Corporation]**

The RUP is a use-case driven, UML-based iterative development approach that delivers software incrementally.

The business analyst is usually deeply involved as the “voice of the customer” throughout all the iterations and phases, more so, of course, in the inception and elaboration phases.

# Agile Approaches

## *Rapid Application Development*

Rapid Application Development (RAD) is also a generic term for a range of evolutionary prototyping approaches. Many of the approaches have been developed over the years and are used within organizations and by consulting firms to produce software results on a more rapid basis than the structured linear methods. RAD is now in vogue due to the increasing demand for web-based software to be developed at high speed.

Most RAD approaches nowadays are object-oriented because of the speed object-oriented languages and tools provide. However, RAD approaches have been successfully applied to speed up structured analysis, design and programming efforts.

## *DSDM*

The Dynamic Systems Development Method (DSDM) is a framework of controls for the development of IT systems to tight timescales. It is independent of any particular set of tools and techniques. It can be used with object-oriented, structured analysis, and design approaches in environments ranging from the individual PC to global distributed systems. DSDM has been used successfully by organizations in both the public and private sectors.

DSDM provides a generic process which must be tailored for use in a particular organization dependent on the business and technical constraints. DSDM outlines a five-phase process:

1. Feasibility Study
   The feasibility study assesses the suitability of the application for a RAD approach and checks that certain technical and managerial conditions are likely to be met. The feasibility study typically lasts a matter of weeks.

2. Business Study
   The business study scopes the overall activity of the project and provides a sound business and technical basis for all future work. The high-level functional and non-functional requirements are baselined, a high-level model of the business functionality and information requirements is produced, the system architecture is outlined and the maintainability objectives are agreed upon.

 

3. Functional Model Iteration
   The bulk of development work is in the two iteration phases where prototypes are incrementally built towards the tested system which is placed in the user environment during the implementation phase.

4. Design and Build Iteration
   During the design and build iteration, the focus is on ensuring that the prototypes are sufficiently well engineered for use in the operational environment.

5. Implementation
   Implementation consists of putting the latest increment into the operational environment and training the users. The final step in implementation is a review of what has been achieved.

Principles of DSDM are:

- Active user involvement is imperative,
- DSDM teams must be empowered to make decisions,
- The focus is on frequent delivery of products,
- Fitness for business purpose is the essential criterion for acceptance of deliverables,
- Iterative and incremental development is necessary to converge on an accurate business solution,
- All changes during development are reversible,
- Requirements are baselined at a high level,
- Testing is integrated throughout the life-cycle,
- A collaborative and co-operative approach between all stakeholders is essential.

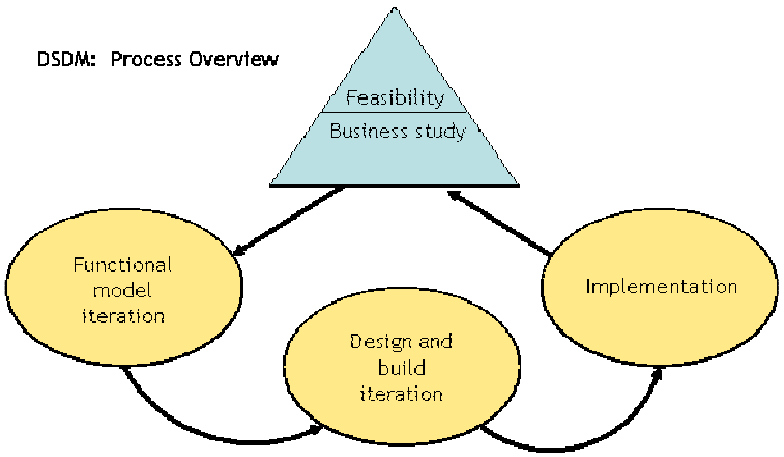


**Figure 6. Example of DSDM Process**
**[From Stapleton, Jennifer. DSDM: The Method in Practice. Addison-Wesley, 1997]**

## *Extreme Programming*

Extreme Programming (XP) is the poster child for the agile approaches. The process was developed by Kent Beck, Ward Cunningham, and Ron Jeffries. When people think of agile they usually think in terms of XP. Extreme Programming is a software development methodology that represents a throwback to the very earliest and purest days of software development when technicians ran the show. Then, software engineers enjoyed greater autonomy largely because there were few in the management ranks who understood the task at hand.

XP is the highest risk approach and requires the greatest skill, knowledge and experience to run it successfully. According to the founders, all the principles of XP must be adhered to be successful; adapting several of the principles and trying to do XP will not work.

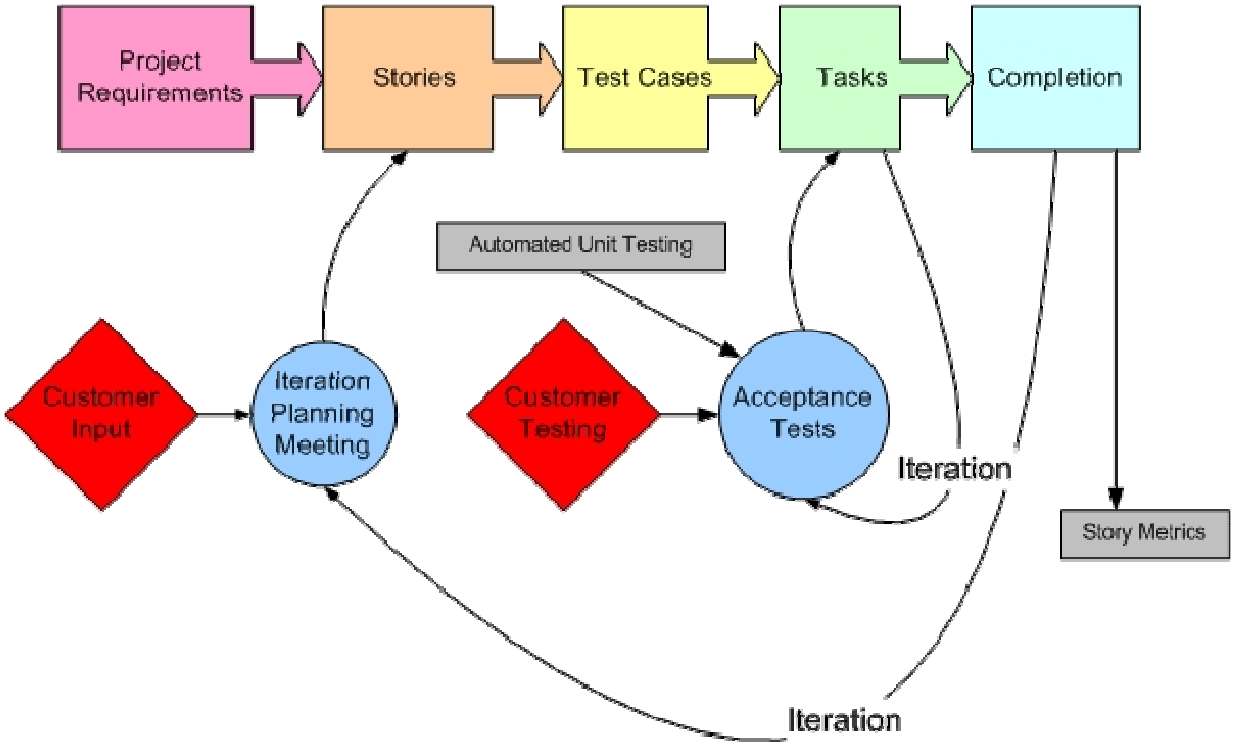


**Figure 7. Example XP Process**

The XP process as shown above starts with a project's requirements which are laid out as individual stories typically written on 3x5 index cards. These stories have test cases which are written by the customer and developer working as a team. A test case includes the test and the desired results which, if met, indicate that that story is complete.

The project is done in iterations. At the end of an iteration, the next iteration is planned out in an iteration planning meeting called the Planning Game. The customer (with the developers' guidance, of course) decides what stories are to be implemented in the next iteration, based on what is most important to them, and what is most practical to implement next.

It is important that all iterations take the same amount of time. This is so that metrics can be retained that let the XP team know what their velocity (speed of development) is. The time that it took to complete a given story is recorded so that the programmers get a good feel for how much effort it takes to do that type of work.

The stories are broken down into tasks, which are the steps needed to implement a given user story. Developers then sign up to do those tasks.

On the completion of an iteration, acceptance tests are done to ensure that the test cases defined in that story have been met. The acceptance tests include both automated unit testing and customer testing. This ensures that all requirements are reached since the development is based around those requirements for which there are definite indicators that tell whether a given requirement is met. It also breaks down the project into reachable goals rather than having programmers work away forever on an ever-expanding scope.

Some XP Precepts:

- XP calls for short release cycles, a few months at most, so the scope of any slip is limited. Within a release, XP uses one- to four-week iterations of customer-requested features for fine-grained feedback about progress. Within an iteration, XP plans with one- to three-day tasks, so the team can solve problems even during an iteration.

- XP asks the customer to choose the smallest release that makes the most business sense, so there is less to go wrong before going into production and the value of the software is greatest.

- XP creates and maintains a comprehensive suite of tests, which are run and re-run after every change (several times a day), to ensure a quality baseline.

- XP calls for the customer to be an integral part of the team. The specification of the project is continuously refined during development, so learning by the customer and the team can be reflected in the software.

- XP asks programmers to accept responsibility for estimating and completing their own work, gives them feedback about the actual time taken so their estimates can improve, and respects those estimates. [Kent Beck, Extreme Programming Explained, Addison-Wesley, 2004]

XP Principles

The following principles from Kent Beck's book Extreme Programming Explained constitute the essence of XP. Many are the same as principles already described for other agile processes or in general. Some were initiated by XP and some were adopted by XP.

1. The Planning Game: Business and development cooperate to produce the maximum business value as rapidly as possible. The Planning Game happens at various scales but the basic rules are the same:

   – Business comes up with a list of desired features for the system. Each feature is written out as a user story, which gives the feature a name, and describes in broad strokes what is required. User stories are typically written on 3x5 cards.
   – Development estimates how much effort each story will take and how much effort the team can produce in a given time interval.
   – Business then decides which stories to implement in what order, as well as when and how often to produce a production release of the system.

2. Small Releases: XP teams practice small releases in two important ways:

   – First, the team releases running, tested software, delivering business value chosen by the customer, [at] every iteration. The customer can use this software for any purpose, whether evaluation or even release to end users. The most important aspect is that the software is visible, and given to the customer, at the end of every iteration.
   – Second, XP teams release to their end users frequently as well. XP web projects release as often as daily, in-house projects monthly or more frequently.

3. Simple Design: XP uses the simplest possible design that gets the job done. The requirements will change tomorrow, so only do what's needed to meet today's requirements. Design in XP is not a one-time thing but an all-the-time thing. There are design steps in release planning and iteration planning, plus teams engage in quick design sessions and design revisions through refactoring throughout the course of the entire project.

 

4. Metaphor: Extreme Programming teams develop a common vision of how the program works which we call the "metaphor". At best, the metaphor is a simple description of how the program works. XP teams use a common system of names to be sure that everyone understands how the system works and where to look to find the correct functionality you're, or to find the right place to put the functionality you're about to add.

5. Continuous Testing: XP teams focus on validation of the software at all times. Programmers develop software by writing tests first, and then writing code that fulfills the requirements reflected in the tests. Customers provide acceptance tests that enable them to be certain that the features they need are provided.

6. Refactoring: XP Team refactors out any duplicate code generated in a coding session. Refactoring is simplified due to extensive use of automated test cases.

7. Pair Programming: All production code is written by two programmers sitting at one machine. This practice ensures that all code is reviewed as it is written and results in better design, testing, and code.

   – Some programmers object to pair programming without ever trying it. It does take some practice to do well, and you need to do it well for a few weeks to see the results. Ninety percent of programmers who learn pair programming prefer it, so it is recommended to all teams. Pairing, in addition to providing better code and tests, also serves to communicate knowledge throughout the team.

8. Collective Code Ownership: No single person "owns" a module. Any developer is expected to be able to work on any part of the codebase at any time.

9. Continuous Integration: All changes are integrated into the codebase at least daily. The unit tests have to run 100% both before and after integration. Infrequent integration leads to serious problems on a software project. First of all, although integration is critical to shipping good working code, the team is not practiced at it, and often it is delegated to people who are not familiar with the whole system. Problems creep in at integration time that are not detected by any of the testing that takes place on an unintegrated system. Also weak integration process leads to long code freezes. Code freezes mean that you have long time periods when the programmers could be working on important shippable features but that those features must be held back.

 

10. 40-Hour Work Week: Programmers go home on time. In crunch mode, up to one week of overtime is allowed. But multiple consecutive weeks of overtime are treated as a sign that something is very wrong with the process and/or schedule.

11. On-Site Customer: Development team has continuous access to the customer who will actually be using the system. For initiatives with lots of customers, a customer representative (i.e., Product Manager) will be designated for Development Team access.

12. Coding Standards: Everyone codes to the same standards. The specifics of the standard are not important: what is important is that all the code looks familiar, in support of collective ownership.

The business analyst has a place in the solution development effort regardless of the development approach or methodology used by the solution team.

 