



N•CYCLES

software solutions

Development Methodologies Compared

Why different projects require different development
methodologies.

December 2002

Dan Marks

65 Germantown Court
Suite 205
Cordova, TN 38125
Phone 901-756-2705
Fax 901-758-1597

1616 West Gate Circle
Nashville, TN 37027
Phone 615-403-3168
Fax 630-214-4909

www.ncycles.com

Table of Contents

Table of Contents	1
Executive Summary	2
Problem Examined	2
Waterfall Methodologies Summarized	3
Waterfall Strengths	5
Waterfall Weaknesses	5
Iterative Methodologies Summarized	6
Iterative Strengths	8
Iterative Weaknesses	9
Conclusion	10

Executive Summary

Software development organizations are always asked what methodology they use. The implication is that one particular methodology must be superior to all others. This is particularly common among consulting companies, who usually claim some sort of competitive advantage based on their unique methodology. Other software tool companies or academics promote their own methodology as the best choice for achieving the highest quality, lowest cost, and fastest development time. Sometimes, proponents will argue for the benefits of their own particular choice with almost religious fervor. However, most development methodologies can be basically categorized as either a waterfall or iterative.

The waterfall approach emphasizes a structured progression between defined phases. Each phase consists of a definite set of activities and deliverables that must be accomplished before the following phase can begin. In addition, different people are usually involved during each phase. Strengths of the waterfall approach include the ease of analyzing potential changes, the ability to coordinate large distributed teams, predictable dollar budgets, and the relatively small amount of time required from subject matter experts. On the other hand, the weaknesses of the waterfall methodologies include: lack of flexibility, difficulty in predicting actual needs for the software, the loss of intangible knowledge between phases, discouragement of team cohesion, and the tendency to not discover design flaws until the testing phase.

Iterative methodologies include Extreme Programming and RAD approaches. The emphasis is on building a highly skilled and tightly knit team that stays with the project from beginning to end. The only formal project deliverables is the actual working software and the essential system documentation that is completed at the end of the project. This approach delivers the following benefits: rapid feedback from users increases the usability and quality of the application, early discovery of design flaws, an ability to easily roll-out functionality in stages, a more motivated and productive team, and finally knowledge retention for the duration of the project. Offsetting the strengths are the following drawbacks to iterative methodologies: difficulty in coordinating large projects, the possibility for the project to never end, a tendency to not thoroughly document the system after completion, and the difficulty in predicting exactly what features will be possible within a fixed time or dollar budget.

The trade-offs between the two basic methodologies suggests that no cure-all exists and in choosing a methodology, one should consider the particular goals and constraints to determine whether just one or a mix will be best the given situation.

Problem Examined

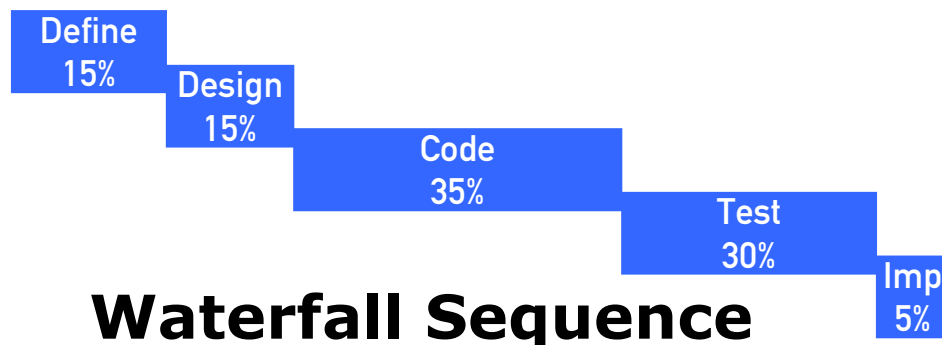
Proponents of a specific software development methodology typically argue passionately for its unique merits. Traditionalists will argue that predictable results are only possible with the type of structured approach written about in formal software engineering textbooks. They say that separating out the planning, programming, testing, and implementation activities into distinct phases is the only way to control costs, work efficiently, and ensure quality work. There are many different names for this type of approach but they are all basically variations of the classic waterfall methodology. More

recently, the classical approach has been challenged by a variety of iterative techniques. More popular examples include Rapid Application Development (RAD) and Extreme Programming. This family of methodologies is not as cohesive, but I group these approaches together based on the emphasis on minimalist structure and design by prototyping. This group argues that spending one to six months on analysis and design is counter-productive because so much time is spent collecting only part of the information necessary to develop software that truly meets the desired objective. Supporters of iterative methodologies point to the fact that the Subject Matter Experts have an excellent understanding of some particular task to be automated, but have a hard time articulating the information necessary to adequately architect software. The experts in software analysis and design are then forced to make assumptions based on their experience. The problem is compounded by the fact that typically the architects and analysts are only partially involved in the actual programming stage. This only makes the information loss problem more severe. By the time the software reaches testing and deployment, the needs may have shifted and users will realize that they did not completely communicate their needs in the first place. In fact, some proponents have argued that Extreme Programming is to software development what Just in Time was to manufacturing.

Which methodology is the best one? Throughout the debate, there seems to be an underlying assumption that there must be a winner. Both sets of arguments seem compelling and definitely have merit, so picking a winner would be very difficult. However, what if both camps were correct? What if both types of development methodologies are the best? To the rational mind, this must seem like a logical impossibility. The rest of this paper will argue that it is very consistent because the best methodology to use will change based on the particular project. Depending on the situation and the objectives, one methodology may be a better choice than the other. It would be overly simplistic to say that one methodology is better for *all* situations.

Waterfall Methodologies Summarized

Rather than try to give an all-encompassing definition for methodologies that should be classified as waterfall approaches, it is easier to describe some common characteristics. Primarily, a waterfall methodology structures a project into distinct phases with defined deliverables from each phase. The phases are always named something different, depending on which company is trying to differentiate its own particular flavor, but the basic idea is that the first phase tries to capture *What* the system will do (its requirements), the second determines *How* it will be designed, in the middle is the actual programming, the fourth phase is the full system *Testing*, and the final phase is focused on *Implementation* tasks such as go-live, training, and documentation.



Waterfall Sequence

Waterfall Deliverables

Define	Design	Code	Test	Imp
Requirements.	Screens Database Objects Test Plan	UI Logic Reports	Test Scripts Defect Report User Feedback	Training Documentation
Project Management				
Project Charter, Status Reports, Change Requests				

Typically waterfall methodologies result in a project schedule with 20-40% of the time budgeted for the first two phases, 30-40% of the time to the programming, and the rest allocated to testing and implementation time. The actual project organization tends to be highly structured. Most medium to large size projects will include a rigidly detailed set of procedures and controls to cover everything from the types of communications to use in various situations, to authorizing and tracking change orders, to the specific ways that defects are logged, communicated, resolved, and re-tested.

Perhaps most importantly, waterfall methodologies also call for an evolution of project staffing throughout the various phases. While typical consulting companies will refer to the differences in staffing as simply “roles,” which imply that the same people could remain on the project and simply switch roles, the reality is that the project staff constantly changes as it progresses. Reasons for the change include economics, mentoring, and expertise - economics in the sense that the project budget encourages the replacement of a relatively highly paid architect with a lower paid staff programmer as soon as possible. On the other hand, an architect with a particular skill set or an analyst with valuable subject area knowledge may be demanded on another project. A fundamental assumption is that the extensive project documentation and control procedures enable relatively easy knowledge transfer to new project staff.

Waterfall Resources

Define	Design	Code	Test	Imp
SMEs	SMEs	Architects	Testers	Analysts
Analysts	Analysts	Coders	Coders	Coders
Proj Mgr.	Architects Proj. Mgr.	Proj. Mgr.	Proj. Mgr.	Proj. Mgr.

Waterfall Strengths

Most of the benefits from using a waterfall methodology are directly related to its underlying principles of structure. These strengths include:

- Ease in analyzing potential changes
- Ability to coordinate larger teams, even if geographically distributed
- Can enable precise dollar budget
- Less total time required from Subject Matter Experts

Because the requirements and design documents contain an abstract of the complete system, the project manager can relatively quickly analyze what impact a change will have on the entire system. An example might be if one developer wanted to modify the fields in a database table or a class. The project manager could look up what other components of the system rely on that particular table or class and determine what side effects the change may have.

The same documents that take so much time to assemble at the front-end also make dividing up and subsequently coordinating the work easier. Because the design produced in a waterfall approach is so detailed, it is easier to ensure that the pieces will be easier to integrate when the project nears the end of the programming phase.

Even with a waterfall approach, the only way to ensure a precise up-front budget cost is to have an external vendor submit a fixed bid for the remaining phases of the project after one or two of the initial phases have been completed. In this way, financial risk is contained even if the project takes longer than expected.

Perhaps one of the most compelling reasons for using the waterfall approach is simply the relatively small amount of time required of the subject matter experts. Because the SMEs in a project typically have other primary responsibilities, their time is limited. The waterfall approach ensures that a significant involvement from them is only required during the initial requirements phase as well as part of the design, testing, and implementation phases. They have very little involvement during the entire programming phase.

Waterfall Weaknesses

While waterfall has advantages, its highly structured approach also leads to disadvantages such as the following:

- Lack of flexibility
- Hard to predict all needs in advance
- Intangible knowledge lost between hand-offs
- Lack of team cohesion
- Design flaws not discovered until the Testing phase

Even though the impact of changes can be analyzed more effectively using a waterfall approach, the time required to analyze and implement each change can be significant. This is simply due to the structured nature of the project, and this is particularly acute when the changes are frequent or large.

Even with improvements in providing wire frame screen mockups and more detailed flowcharts, the front-end planning process has a hard time effectively predicting the most effective system design on the front-end. One of the most challenging factors that causes this difficulty is the unfamiliarity most Subject Matter Experts are with formal system design techniques. One complaint that I have heard a lot is “the document looks impressive, but I don’t know if the system will meet my needs until I see the actual screens.”

Even more disturbing is the inevitable loss of knowledge between the planning and programming phases. Even with the most detailed documents, the analysts and architects always have an implicit understanding of the project needs that are very hard to transfer via paper documents. The information loss is particularly harmful to the project when it is developing a relatively new system as opposed to modifying an existing system.

Closely linked to the knowledge loss effect, is the fact that the waterfall methodology discourages team cohesion. Many studies have found that truly effective teams begin a project with a common goal and stay together to the end. The tendency to switch out project staff from phase to phase weakens this overall team cohesion. In fact, it is common for the project manager be the only person that sees a project from beginning to end. The effect on team productivity is very hard to quantify, but may be illustrated with the following question: Would you have a passion for quality if you knew that someone else would be responsible for fixing your document or code in the next phase?

The most significant weakness is the possibilities that a poor design choice will not be discovered until the final phases of testing or implementation. The risk of this occurring increase as project size and duration goes up. Even dedicated and competent people make simple mistakes. In the context of the rigid waterfall timetable, mistakes made in the master design may not be discovered until six or nine months of programming have been completed and the entire system is being tested.

Iterative Methodologies Summarized

The iterative family of methodologies shares a common emphasis on highly concentrated teams with minimal structure and access to constant feedback from the Subject Matter Experts. While it may appear that they lack design or testing, these types of methodologies actually place a great deal of emphasis on them. They just do it in a different way. Typically, a project will begin with the integrated project team being assembled and briefed on the project objectives. The team will consist of all of the essential roles from the very beginning and each member may actually play multiple roles. Rather than a distinct progression through phases, the iterative methodology emphasizes creating a series of working prototypes for evaluation by the SMEs until the objectives are accomplished and the system is ready for final release. During the process, it is critical for the actual project lead as well as the senior members of the team to balance the SME requests against the overall system constraints and platform limitations to ensure that quality and performance objectives can be met. It is the development team’s responsibility to offer constructive feedback to the SMEs in order to suggest alternatives and work together to the best mutual solution. Where possible, individual team members will be given complete ownership of a particular component and charged with ensuring its

usability, quality, and performance. The senior team members are responsible for enforcing quality and consistency standards. Even on large projects, the initial planning will consist of only a broad outline of the business objectives and creating a framework for the overall project components. In some cases, a set of core features for the entire system will be built initially and subsequent features added as the project progresses. In other cases, just certain modules will be built entirely during the early part of the project and other components added over time.

Iterative Sequence

Iteration 1	Iteration 2	Iteration 3	Iteration 4	Release
20%	20%	30%	25%	5%

Iteration 1	Iteration 2	Iteration 3	Iteration 4	Release
25%	20%	25%	25%	5%

Iterative Resources

Iteration 1	Iteration 2	Iteration 3	Iteration 4	Release
SMEs	SMEs	SMEs	SMEs	SMEs
Analysts	Analysts	Analysts	Analysts	Analysts
Architect	Architect	Architect	Architect	Architect
Coders	Coders	Coders	Coders	Coders
Proj Mgr.	Proj Mgr.	Proj Mgr.	Proj Mgr.	Proj Mgr.

Iterative Deliverables

Iteration 1	Iteration 2	Iteration 3	Iteration 4	Release
Tangible Software	Tangible Software	Tangible Software	Tangible Software	Improved Software
				Training
				Document
Project Management				
Project Charter, Status Reports, Budget Control, Timeline Control				

Iterative Strengths

Many of the strengths of the iterative system are listed below:

- Rapid feedback from actual users
- Flexibility to address evolving requirements
- Design flaws discovered quickly
- Easy to roll-out new functionality in stages
- Higher motivation and great productivity
- Very little knowledge loss between phases

Feedback from the Subject Matter Experts or users can be based on an actual working prototype of the system relatively early in the project life cycle. This enables the SME to base his or her feedback on actually working with a limited version of final product. Much like it is easier to decide if a product meets your needs if you can examine it in the store than if someone were to just describe it to you, the SME is instantly able to identify potential problems with the application as the developer is interpreting his requirements before too much time has passed.

Since the development team receives feedback at early stages in the overall development process, changes in requirements can be more easily incorporated into the finished product. More importantly, if the SME determines that a feature would not be as valuable, it can be omitted before too much development time has been spent or integrating the particular component into the overall system.

In a similar way, since the team is deploying actual working prototype versions of the application along the way, a flaw in the design should become more apparent earlier in the project schedule. Instead of discovering a potential problem only after the system goes to full-scale testing, more design flaws can be addressed before they impact other features and require significant effort to correct.

Because each iteration actually functions (sometimes to a limited degree), deploying parts of the system in a staged roll-out becomes much easier. Using an iterative methodology, the team simply stabilizes an earlier iteration of the component, collaborates with the SME to ensure it is stable and rolls it out. Another advantage of doing a staged roll-out in this way is that actual production use will generate more improvement suggestions to be incorporated in subsequent iterations of the same component and/or other components.

The team approach stressed in the iterative methodology increases overall motivation and productivity. Because the same people are involved from beginning to end, they know that the design choices made will ultimately affect their ability to successfully complete the project. Productivity will be enhanced because of the sense of ownership the project team has in the eventual result. While it may seem like the “empowerment” fad, many studies have found that a team charged with a common goal tends to be much more productive than groups of people with individual incentives and shifting assignments. One example of such a study is *Groups that Work* by Gerard Blair.

The fact that an integrated team maintains a thorough understanding of the project is a more tangible benefit. This effect arises simply by having the same individuals involved from the very beginning and listening first hand to the Subject Matter Experts describe their needs and objectives. The subsequent feedback during each iteration of the project builds upon the initial understanding. Since the same person is listening to the needs and writing the code, less time needs to be spent authoring documents to describe those requirements for eventual hand-off. This translates into more time spent writing and testing the actual software.

Iterative Weaknesses

The drawbacks to using an iterative approach are worth considering and should be weighed carefully when deciding on a methodology for a new project. Some of the more serious weaknesses include:

- Difficulty coordinating larger teams
- Can result in a never-ending project if not managed properly
- Tendency to not document thoroughly
- Predicting the precise features to be accomplished in a fixed time/budget

Iterative projects tend to be most effective with small, highly skilled teams. It is much more difficult to ensure that the components mesh together smoothly across larger, geographically distributed projects. While steps can be taken to minimize the chances of failure, coordinating large iterative development efforts is typically very hard to accomplish effectively because of the lack of detailed planning documents.

Because there are no specific cut-off milestones for new features, an iterative project runs the risk of continuing into perpetuity. Even though one of the strengths is the ability to react to changing business needs, the project leader must determine when the major business needs have been met. Otherwise, the project will continue to adapt to ever changing business needs and the software will never end. This will result in never really deploying a finished product to full production use. This is a risk even in a staged roll-out situation because there are always improvements possible to any software.

In any software project, there is always the tendency to borrow time from the final system documentation tasks to resolve more defects or polish certain features more. This risk increases on iterative projects because there is usually no scheduled documentation period. The result is a system that is very hard to maintain or enhance.

In a similar way, in an iterative project it is much easier to fix a definite project schedule or dollar budget than determine exactly what features will be able to be built within that timeline. This is simply due to the fact that the features change based on user feedback and the evolution of design.

Conclusion

In this brief paper, I have clearly presented the tradeoffs between two basic approaches to software development in order to show that no methodology is universally superior. Instead, the approach that you should take on your next project should depend on its particular needs and the constraints that you have to work with. While certainly not an exhaustive reference on the subject of how a particular methodology is structured, my purpose was to help you become more familiar with the strengths and weaknesses inherent in each of the large schools of thought prevalent in the software development community. Based on the issues discussed, a few basic guidelines that may help point you in the right decision are listed below. Keep in mind that no methodology should ever be considered a substitute for ensuring project members have the proper experience and skillset for the task to be accomplished.

- The iterative methodology is usually better for new concepts

- Waterfall is usually better for modifications to existing systems or building large scale systems after proof-of concept prototypes have been established

- However, some situations will require a hybrid approach