5. Waterfall versus cyclical project management

The six-phase model is a waterfall model. In other words, the phases take place in succession. Just as it is impossible to swim upstream against a waterfall, the pure waterfall method does not allow returning to a phase after it has been completed. During the implementation phase, it is not desirable to decide to adapt the design, thereby bringing implementation to a standstill. For a number of reasons (see e.g. McConnell, 1996; Kroll, 2004; Chromatic, 2003; Stapleton, 2002), the waterfall method is usually less suited to software-development projects.

- Software development is a creative process.
- It is nearly impossible to identify all of the requirements (functionalities) beforehand.
- Estimating the amount of time that will be necessary to implement a functionality is quite difficult.
- It should be possible for all intermediate results to be tested by users throughout the entire trajectory of the project.

Software development is a creative process

To outsiders, software development appears to have more to do with engineering than it does with inventing. It does, however, correspond to inventing in a number of ways. In the process of invention, one never knows in advance precisely what is going to happen.

The designers and programmers who write new software must conceive of solutions for the problems that are set before them. Regardless of the many methods and prescriptions for work, writing programming code remains largely new and therefore largely uncertain. Programmers can choose their solutions out of millions of possibilities that are written in dozens of programming languages, and which operate according to thousands of hardware configurations and in dozens of software platforms. Although this freedom does offer a number of advantages, it also makes the project more difficult to manage than traditional projects (e.g. construction or building projects).

It is nearly impossible to identify all of the requirements (functionalities) beforehand

The waterfall method prescribes the formulation of requirements as the project result of the definition phase. Ideally, there should be little further deviation from these requirements throughout the rest of the project. The development of software according to the waterfall method requires the investment of considerable time in the beginning of the project in analysing the necessary functionalities (requirements). This leads to a functional design (for more details on functional designs, see [i]). Using the functional design as a guide, the software architect makes a technical design in the design phase. The technical design includes a description of how the desired functionalities should be implemented.

Although this may appear quite straightforward, consider the following situation (example adapted from McConnell, 1996, p. 138). There is a project to produce a new automobile. As an active automobile driver, you are asked to formulate the requirements for an automobile. You consult with other drivers and create an extensive list:

- The automobile must accommodate four people.
- The automobile must have a steering wheel in the front on the driver's left-hand side, a brake pedal, a gas pedal and a hand brake.
- The automobile must have four wheels.
- The automobile must have white headlamps and red tail lamps.
- et cetera (the actual list would obviously be enormous)

Using your list as a guide, the designers develop a new design, which is subsequently built several months later. One day, as you are walking down the street, you see an automobile stopping. You realise that you neglected to include brake lamps in your list! You immediately telephone the engineer of the automobile manufacturer, who nearly explodes in reaction to your call. You maintain that it is only a small change. For the automobile manufacturers, however, it is a disaster. The automobiles that have already been made must be completely disassembled, cables must be stretched from the brake pedals to the rear, the rear of the automobile must be completely re-designed in order to accommodate the brake lamps, the boot hatches that had already been produced would have to be discarded and the list goes on.

While the example above is somewhat absurd, it reflects an almost daily reality in software development. Programmers erroneously assume that the clients, customers or users of the software that is to be developed know precisely what they want Clients erroneously assume that the software builders can make (and adapt) everything in the shortest possible time. This problem is the greatest source of conflict between customers and software builders.

The following anecdote illustrates the fact that there are many conflicts between customers and software suppliers. A beginning entrepreneur wanted to obtain legal insurance for his business. He asked about the possibilities. When the broker asked him to identify the sector in which his business was active, the entrepreneur replied, 'IT'. 'Too bad', replied the broker, 'there are so many lawsuits between IT suppliers and customers that there are no insurance companies that will write legal-insurance policies for IT companies'.

Estimating the amount of time that will be necessary to implement a functionality is quite difficult

The waterfall method assumes a number of phases. In their project plans, project leaders must include estimates of the amount of time (and therefore money) that will be needed for each phase. We have already seen that time estimates are generally difficult for any project, particularly if they must be made in the early stages of a project. For software projects, it is simply impossible. Imagine that it were feasible to compile a qualitatively adequate list of functionalities in the definition phase. In theory, the project team should then be able to provide a reasonable estimate of how much time will be necessary to implement each functionality. In practice, however, there are too many uncertainties to allow a reasonable estimate (e.g. McConnell, 1996):

- Once a functionality has been made, it is often discovered that the customer does not need it after all. The hours that were used in implementing this functionality can therefore be regarded as useless.
- Requirements change during the project.
- Should the functionality be implemented expensively or inexpensively? There are many possible methods of implementation and realisation.
- How will the functionality be designed technically? The design largely determines the amount of time that will be needed to make it.

- How high must the quality of the functionality be? For example, should a web application always remain completely available, or can it be offline for a few hours each year?
- The time needed to identify and repair errors in software can vary from minutes to weeks.
- How long will it take to install and integrate the new software into the customer's existing systems?
- The lack of knowledge concerning possible solutions also complicates the estimation of time. Further, it is difficult to estimate how long it will take to acquire the necessary knowledge.

The list above shows that many factors can affect the amount of time that will ultimately prove necessary to implement a new piece of software. Research (McConnell, 1996, p. 168) has shown that the estimates of the time needed to implement a functionality at the beginning of a project varies between four times too little time and four times too much time. This means that the amount of time necessary can turn out to be either four times higher or four times lower than a faulty estimate. These estimates become more accurate as the project progresses. After the functional design has been made, the margin is reduced to twenty-five per cent too much or too little. Only when the functionality is implemented can an estimate be provided with a high level of certainty (see Figure 8).

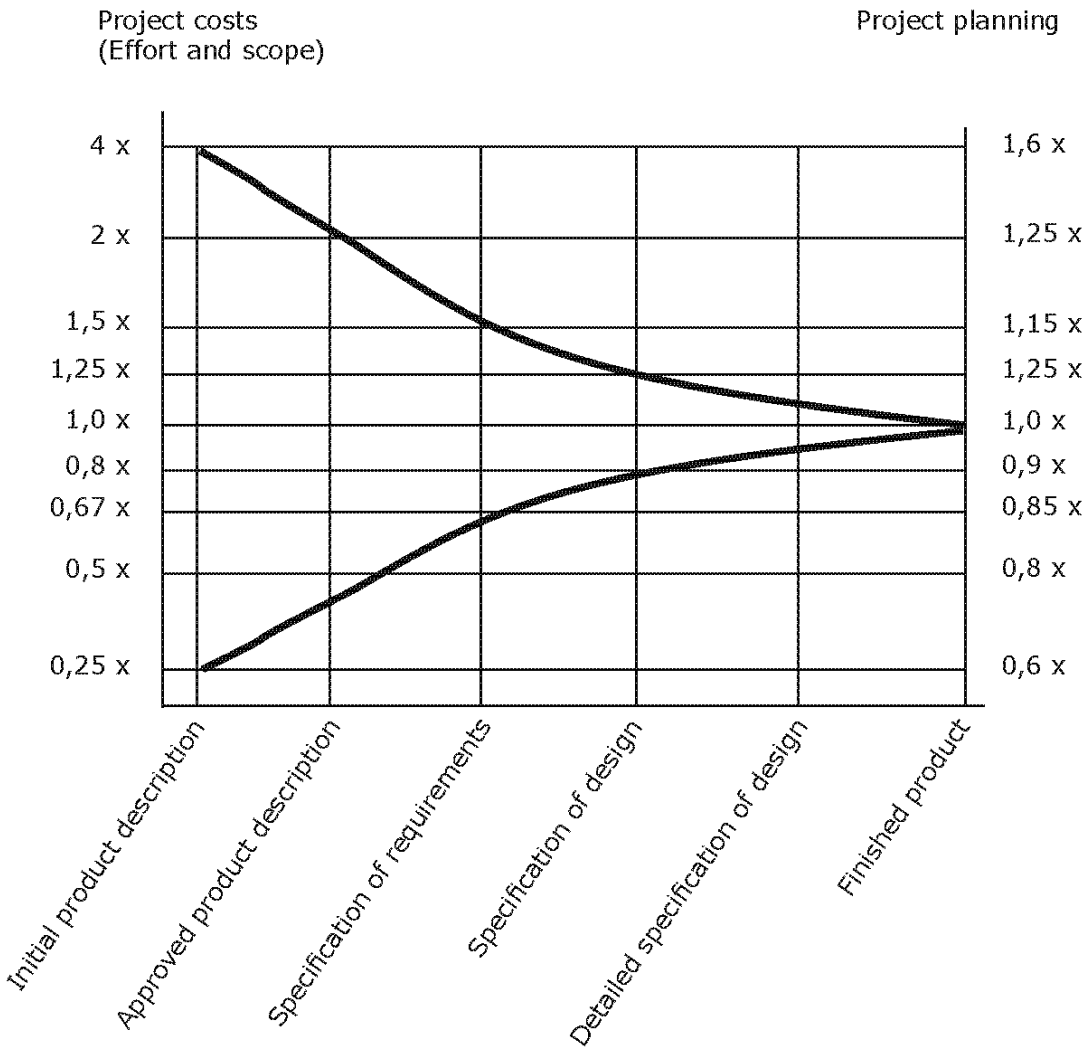

Figure 8: Accuracy of estimates of time necessary to implement a functionality (Source: McConnell, 1996, p. 168).

Software is never completely free of errors. Even for the well-known packages that are used by many (e.g. Word, Excel, Explorer, OSX, PHP, Flash), there are lists of known bugs available on the Internet. New releases regularly appear on the market, in which software errors have been removed. Customers sometimes expect error-free software. In practice, however, such a goal would make it impossible to complete a piece of software. Moreover, software errors are often not inherent in the software itself.

An educational game was made in Flash. It was agreed that the game would be delivered as a file and that the customer would install it himself on his own server. During the project, the customer requested that a high score feature be included in the game to increase competition between players. This would require a piece of script code in PHP. The game builders made and tested the script code on their own server, which worked in Linux. It worked. The game and script code were delivered to the customer. The customer, however, had a Windows server and, for some reason, the script no longer worked well. The high scores were not saved. The programmer eventually needed a week to resolve the problem. As it turned out, the combination of PHP and Windows does not always work well. The script itself

contained absolutely no errors! By using a hack, he was able to get it to work, and everyone was satisfied – but who should pay for that extra week of work?

Another software-development project also involved educational software. The problem was that the software worked for the programmers, but it did not work well for the customer. After trying nearly everything, the programmer decided to pay a visit to the customer. As it turned out, the customer was using an old Pentium III system. The slowness of the computer caused the poor performance of the software. The programmers had also tested the software on a Pentium III, but it had worked fine. Further investigation revealed that the customer's computer was so slow because it was full of viruses and spyware.

The uncertainty that is illustrated by the examples above does not simplify the writing of project plans. It also complicates agreements between the parties involved. Someone must assume the risks for extra work that must be done. If payment is on an hourly basis, the customer assumes the risk. If a fixed price has been agreed (as in grant-funded projects), the software builder assumes the risk. When more than two parties are involved, it becomes even more complicated. In such a case, who should pay for the extra hours in the project?

Discussions often arise concerning who should be responsible for delays. In many cases, the guilty party is difficult to identify. It is quite possible that none of the parties involved is at fault, as the 'delay' is actually the result of a faulty initial estimate of the number of hours that would be needed. Exceeding the number of project hours and the question of who should pay are frequent sources of conflict in the IT world.

It should be possible for all intermediate results to be tested by users throughout the entire trajectory of the project

In the definition phase and the design phase, customers are asked to formulate their requirements as well as possible. This is difficult for two reasons. First, customers have only a limited conception of the possibilities or impossibilities of IT. They do not have a good idea of what is or should be possible or what they should or should not want. Second, customers often have only limited knowledge of their own business processes. Many IT projects involve the computerisation of a customer's existing business practices. Even though customers may have worked with the processes for many years, they are often not able to define their own business processes. They can work fine in their own way, but cannot say exactly what that way is. The precise definition of processes is a precondition for making software that will drive computerisation. The complexity for customers thus increases if they must describe existing processes.

The list of requirements that is developed in the definition phase is often incomplete. In the implementation phase, programmers make software according to this incomplete list. When users are confronted with the initial versions of the new software, additional requirements are identified. 'It looks good, but now can you make it so that I don't have to keep entering my password?' Programmers often complain that customers do not know what they want. Customers appeal to the 'fact' that, because software developers are professionals, they should have determined earlier in the process what the customers wanted.

In a software project that involved the automatic processing of applications for a web-based service, an extensive functional design was made. Long lists of requirements from the customer were compiled. A number of screen designs and flow charts were added, after which the programmers could get started.
Probably because the project was under extreme time pressure or perhaps because the customer's organisation was rather chaotic, the designers had neglected to

include one important component: manual administration. The applications were processed by the software. Because the processing of the applications was to be automatic, the programmers thought that no manual administration would be desired. This requirement also did not appear in the functional design. When the software was delivered for testing, the customer realised that there were exceptions in some of the applications. These applications could not be processed automatically, but would have to be handled manually. The software, however, worked only automatically.

In the waterfall method, the actual project result is tested at the end of the implementation phase. This is late in the development process. The time between the definition phase, design phase and implementation phase sometimes amounts to months or even more than a year. If design errors are discovered in a late stage of the project, it can be expensive or sometimes even impossible to adapt software without beginning an entirely new project. Because we have seen that it is practically impossible to specify all requirements beforehand, a working method that allows the possibility of testing (intermediate) results earlier is desirable.

Comparing a number of prospective software houses, a customer asked the competing parties what was possible. One party was somewhat reserved and doubted whether some of the customer's requests would be feasible. The other party had an aggressive sales representative. When the customer asked the sales representative if a particular request would be possible, the sales representative telephoned his programmers. 'Can we build a functionality that can do X?' The programmer, who thought primarily in technical terms, answered that, in principal, anything was possible. Neither the programmer nor the sales representative worried about feasibility in terms of project management (e.g. time, money, complexity, risk).

The sales representative's enthusiasm was more effective than the other party's reserved attitude had been. The customer chose the aggressive sales representative's offer. The newly acquired project subsequently came into the hands of a project leader and a group of programmers. After a time, it became apparent that the project did not fulfil the customer's expectations. This had partially to do with the fact that the processes were much more complicated for the customer than they had originally appeared. During a heated discussion between the two parties, the customer referred to the fact that the sales representative 'had said that functionality X would not be a problem'.

Cyclical methods of project management

Because of the issues that have been sketched above, a number of other methods of project management have emerged in recent years. These methods are particularly suited for IT-development projects. Examples of these relatively new streams within project management include DSDM, RUP, eXtreme Programming (XP), RAD and agile project management (McConnell, 1996; Kroll, 2004; Chromatic, 2003; Stapleton, 2002, [ii], [iii])

Although the above-mentioned methods of project management differ according to a number of aspects, they are essentially the same. Because the path toward the final goal of IT projects has proved so uncertain, these methods assume that the goal will be achieved in a number of short cycles. This is the background for the term 'cyclical' project management for these methods.

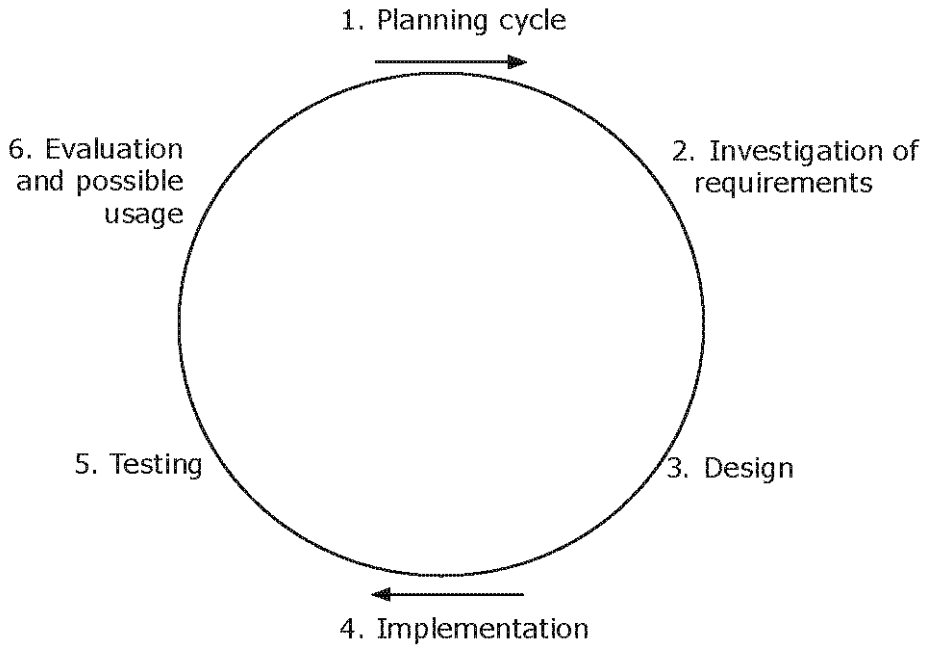

Figure 9: The activities in a cycle

In cyclical project management, the project goal is pursued in several short, successive consecutive cycles. Each cycle is relatively short, preferably lasting less than one month. Within each cycle, a portion of the project is carried out. Analysis, design, implementation and testing occur within each cycle. This is fundamentally different from the waterfall method, in which these activities all take place within their own separate phases. In addition, the waterfall method prescribes only single moments for definition, design, implementation and testing. In the cyclical method, this occurs many times in sequence.

Various components of the software are implemented during the cycles, which are therefore independent of each other. Evaluation takes place after each cycle is completed. Should advancing insight lead to new of different requirements for the project, the activities of the subsequent cycles are adapted to take them into account.

A cycle begins with the making or adjusting of the schedule. Next, the requirements of the result of the cycle are examined. A design is made, programmed and tested. Evaluation subsequently occurs and, in some methods, the new software is brought into use. Thereafter, the following cycle can begin, in order to carry out the following component of the project. (For a more extensive description of cyclical methods and the differences between the methods, see e.g. Kroll, 2004; Chromatic, 2003; Stapleton 2002.)

The most important advantages of working with the cyclical method are as follows:

- Higher product quality and improved implementation of functionalities,
- More realistic estimates of time and money,
- Project team is under less pressure,
- Higher quality.

The previous chapters have shown that it is difficult or impossible to determine of the desired functionalities accurately in the early stages of a project. In cyclical methods, the desired functionalities are implemented in several short cycles. In each cycle, a small portion of the desired functionality is not only investigated; it is designed, implemented and tested as well. The short succession of design, implementation and testing makes a particularly important contribution to quality improvement. Teams are thereby in state to make adjustments. If a design does not turn out to be good in practice, it becomes obvious during the cycle, thereby allowing adjustment. This way of working also allows customers to request adjustments.

Another reason that cyclical project management leads to better quality is that a cycle involves intensive collaboration between customer, designers and programmers. A multi-disciplinary team jointly conceives of and implements solutions. In the waterfall method, the customer is involved primarily in the beginning, in the formulation of the requirements; thereafter, the designers make a design and then the programmers programme the software. The project leader serves as the link between all of the various parties and must ensure that information that is exchanged is delivered to the right place. In practice, many programmers and designers never see the customer, who re-emerges in the process only after the software has been completed.

Cyclical methods of project management are particularly suited to projects in which the goal that is to be achieved cannot be clearly established beforehand, as in creative projects or research projects. Working in a number of cycles with a multi-disciplinary team in which the end-users are also represented allows the team to discover the real goal of the project and how it can best be achieved. Each cycle contains a point for reflection and an opportunity for adjustment.

In waterfall projects, a goal is formulated beforehand. Reflection on the original goal is less of a possibility. The originally formulated goal is never (fully) achieved, and it is likely that neither the originally formulated goal nor the goal that is achieved is exactly what was originally desired.

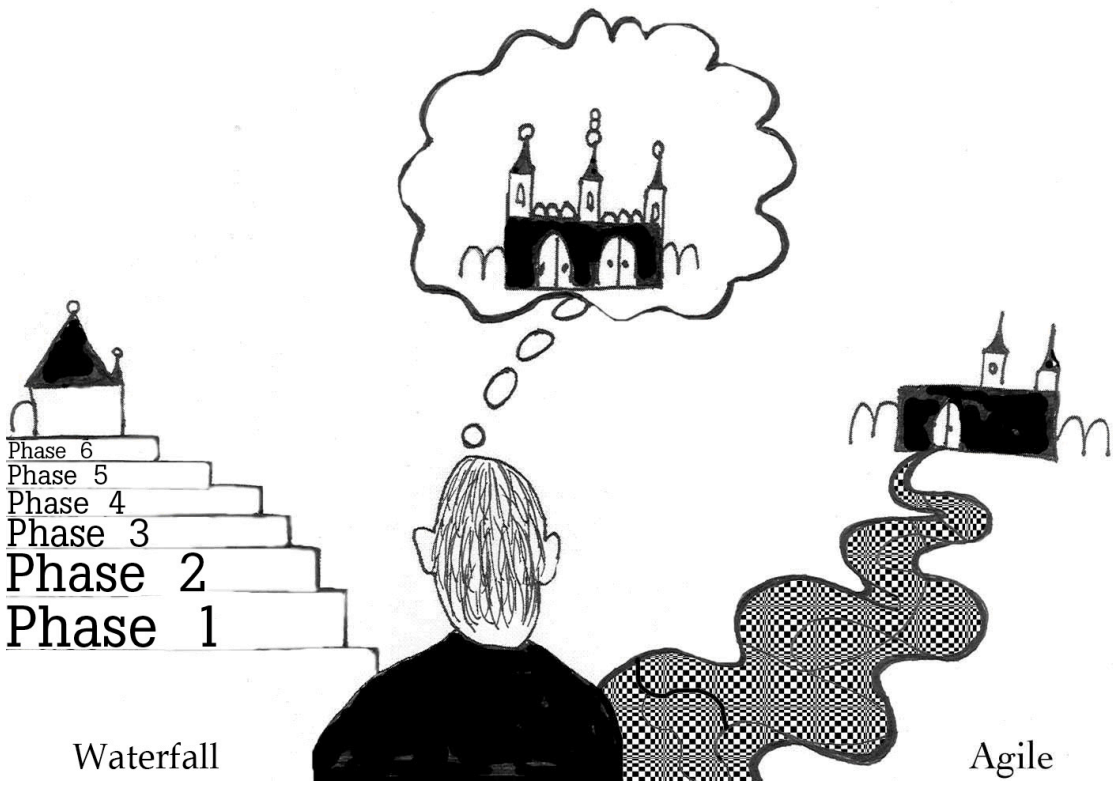

Figure 10: Better results are achieved by working in cycles. (Illustration: Rachèl Harmsen)

The last reason why cyclical project techniques generate better results is that the customer performs acceptance tests. Quality is further strengthened by using tests as criteria for well-performing functionality from the very beginning. Programmers must therefore define 'failing tests' (or unit tests) before they write their code. The code is considered acceptable only if it passes the failing test.

Test-oriented work requires programmers to prove that there are no bugs in the new code before they write new code. They do this by developing a test (failing test) that would detect any possible bugs before they begin coding. For example, imagine that a programme must be written for calculating the correct amount of change that you must receive from a candy machine. First, the availability of a function to give change must be tested. This function could be called 'give_change'. A simple test could be made and, when it is performed, it is determined that a 'give_change' function does not yet exist. If the programmer then makes the function but does not yet give it any substance, the test will succeed.

The next step would be to test whether the machine gives the right amount of money back when an item is purchased. Insert sixty cents into the machine and buy an item that costs fifty cents. The test will not succeed, because the function is still empty. The programmer then writes the code. In the 'give_change' function, he writes that the amount of change to be returned is the amount of money that was inserted in the machine, less the price of the chosen candy. The test should now succeed. The process continues in this way; for each component of the software, a test must be devised beforehand (example translated and adapted from Chromatic, 2003, p. 27 ff).

The code is not the only component that must be technically tested; the functionalities must also be tested (i.e. acceptance tests). For these tests, the customer determines the conditions under which the functionalities that are to be built can be accepted, before coding begins (e.g. how quickly a computer must respond to a certain action of a user). The prior specification of the conditions under which the new functionality can be accepted has proven particularly difficult and time-consuming. Nonetheless, the active role of customers in testing is an important determinant in the success of a project.

More realistic estimation of time and money
With cyclical project techniques, the functionalities that are to be implemented are not written in stone at the beginning of the project. The available time is specified. Agreements are made concerning the direction in which the project will proceed, and it is determined in the process what is actually needed, useful and realistic with regard to the software that is to be made. In cyclical projects, the functionalities that are to be implemented are not established as fixed goals, and they thus avoid the risk that the necessary hours, and therefore money, can get entirely out of hand. To prevent such a situation, the available time is used as a starting point, and it is determined during the process what is realistic to expect in that amount of time. Also for this reason, cyclical methods of project management are much friendlier for the project team. The team does what it can do within the specified time, but is not pressured to meet unrealistic schedules or to work within an inadequate budget.

Cyclical methods also facilitate the management of external suppliers. With the waterfall method, a project can become bound to a single supplier until the end of the project, regardless of the performance of that supplier. In the cyclical working method, is theoretically possible to make new agreements for each cycle or even for each component to be delivered and, if necessary, to change suppliers.

Conditions for cyclical project management

To apply cyclical project management, a number of conditions must be met:

1. Users/customers are actively involved.
In cyclical project management, the formulation of requirements, design, implementation and testing take place in each cycle. This means that many decisions must be taken in a cycle. If the software is to provide a good reflection of the wishes of the customer, the customer *must* participate actively in the project team. Customers must present their wishes as clearly as possible to the programmers and designers. This generally involves weekly (or at least bi-weekly) presence in the project team.

Within a project, customers are involved with determining the desired functionalities and the planning of the cycles. They collaborate on the acceptance tests, approve or reject intermediate results and collaborate on the general direction of the project. The active involvement of the customer also leads to better results in the waterfall method.

2. The team is authorised to take decisions.
Within a cycle, the project team must be authorised to do what they think is best. If the project team does not have this power, the cyclical model of project management will not work. If constant authorisation from superiors is necessary during a cycle, this can lead to stagnation. Moreover, outsiders are often poorly informed about what is going on, because they do not actively participate in the project team; this makes it difficult for them to make sensible decisions.

3. Project result (software) can be broken down into smaller parts.

With cyclical project management, parts of the project are performed in a number of cycles. This is possible only if the software that is to be created is divisible into a number of more or less separate components.

4. The requirements that are imposed by management with regard to the software are primarily global; management does not impose direct, concrete or specific requirements. One of the strengths of cyclical project management is that the customer, designers, programmers and any testers work closely together within the cycles. If there are already specific and concrete requirements at the beginning of a project, this impedes the freedom of the project team to use their better judgment to make design choices. Many requirements on a project are revealed during the process to be in need of adaptation and should therefore not be (too) firmly established in the beginning.

5. The activities are intelligible for the customer.
If considerable technical work that is difficult for the customer to understand must take place within a cycle, a risk emerges that the customer will not be in state to participate well in the team. In such a situation, the customer has very little to contribute to the necessary design choices.

A similar risk arises when the progress is not visible to the customer. For example, much of the work may involve coding, with very little being done on the user interface. It is important that customers have sufficient insight into the substance and progress of a cycle in order to ensure that they are not pushed into the background.

6. It should be possible to take a step backwards.
Even in cyclical project management, teams sometimes pursue paths that later prove to have been wrong. In such a case, it should be possible to take a step backwards. If a new module that is created in a cycle proves inadequate, it must be possible to resume working with the old module. This imposes demands, particularly with regard to the archival and documentation of the project materials. Concurrent Versioning System (CVS) and Subversion are two helpful tools for these tasks (see Appendix 3 for a list of tools).

7. In addition to programming, programmers should be able to communicate with customers, and vice versa. Team members must be in state to think conceptually. Discipline is necessary in order to persevere with the work.

8. The organisation in which the project takes place must also offer sufficient support for this method of working. Systems for time reporting, archiving and scheduling are necessary to support the projects. These registration systems ensure the transparency that is necessary to ensure the adequate distribution of resources across projects and time.

9. Projects should have sufficient priority, team members must be released for projects. Requiring team members to work in too many projects at the same time does not work. If an organisation is insufficiently adjusted to working with projects, the flexibility of cyclical project management is likely to lead to chaos. The waterfall method also benefits from an organisation that is arranged for working with projects (see Wijnen, 2004, p. 111).

The director of a software house, who was more of a visionary than he was a manager, had a brilliant idea nearly every month, and he was continually starting new projects in his company. Older projects were consequently never completely

finished, and the employees were sometimes working on as many as five projects at once. The charismatic director had once read a book on rapid application development (RAD) and was quite enthusiastic about it – particularly about the aspect of 'rapid'. He posted the basic concepts of RAD over the copier and subsequently assumed that everyone would start working according to these concepts. After all, it was a wonderful method.

Risks of cyclical project management

Cyclical methods of project management sometimes allow insufficient time to implement the desired functionalities. Because the amount of time is pre-determined, fewer functionalities will probably be made than were originally assumed.

This is indeed a real risk, but it is also inherent in the waterfall method. In the waterfall method, the definition phase includes an extensive analysis of requirements. This analysis could be expected to generate better time planning. This is often not the case in practice, however, for the reasons that are mentioned above. Functionalities are left out in this method as well, as there is not enough money to implement them.

In cyclical methods, requirements are handled pragmatically. For example, the requirements in cycles can be divided according to the MoSCoW rules (Stapleton, 2003):

- Must Have: requirements that are essential for the system
- Should Have: important requirements that people really want
- Could Have: requirements that are desirable, but which can be easily omitted
- Want to Have: but will not have this time round: requirements that can wait until later

Regardless of the fact that there may be no more time for particular functionalities, the DANS project managers agree that cyclical work yields more customer satisfaction than the waterfall method does. At any rate, the expectations are consistently better managed, and the projects deliver results that actually work, even if they are less elaborate than originally hoped.

One frequently mentioned disadvantage of XP is that considerable power comes to rest with the programmers. If they misuse this power, they can hide behind the customer's lack of technical knowledge. Preventing this situation requires a project leader who can serve the interests of both the programmers and the customer. Such project leaders assist their customers in choosing and planning the cycles, making the technical background of choices intelligible and providing for administration and reporting.

Finally, another disadvantage of XP is that there is a steep learning curve for programmers, managers and customers with regard to the introduction of the method.