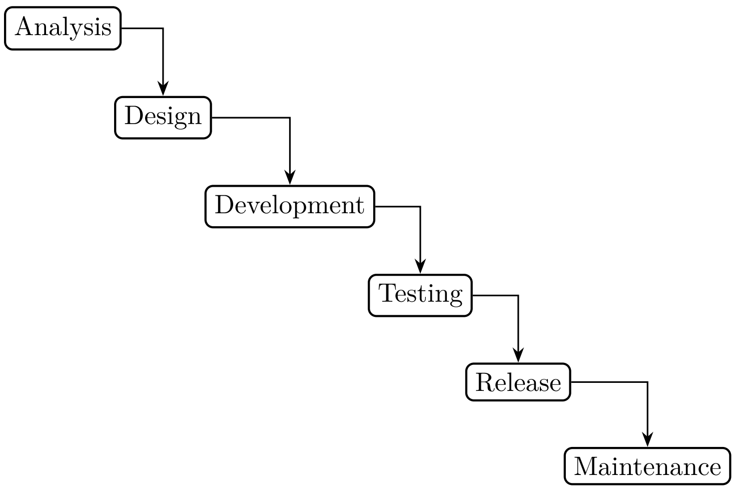
Software Development

Contents

Methodologies

Waterfall

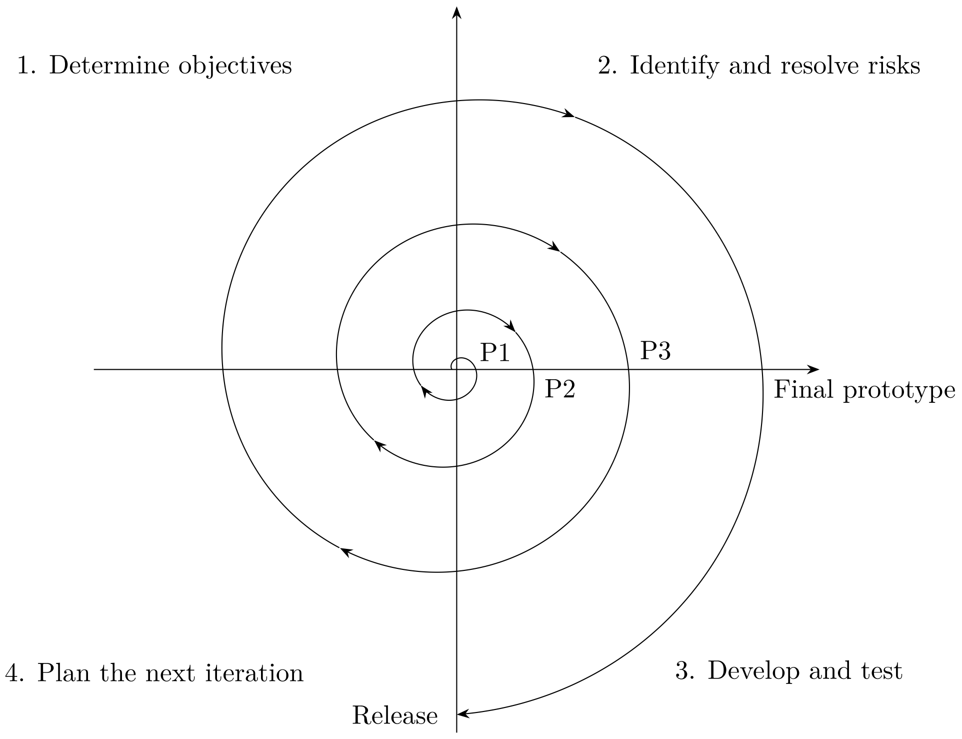


In the waterfall method, each stage is carried out sequentially and must be completed and signed off on before the next stage can begin. This method comes from engineering and construction, where the whole building has to be meticulously planned out before any construction can begin. The developers must perform analysis and write detailed descriptions of the project requirements before starting development.

This method is well suited for projects where all the requirements can be known at the beginning and won't change over the course of development. It also makes the whole project easier to plan, and different people and teams have distinct and clear responsibilities for the project.

However, the waterfall method produces large amounts of documentation and has a lack of user involvement in the development process. It is heavily dependent on the initial specification of project requirements; if the client doesn't know what they want or how to express it, or if their requirements change, then the final product will not properly meet their needs.

Spiral



The spiral model is much like the waterfall model, but it uses **project cycles**, each producing a prototype which is used to inform the next cycle. This process is repeated until the final product is completed and released.

Each cycle is split into 4 stages:

1. Determine objectives
2. Identify and resolve risks
3. Develop and test
4. Plan the next iteration

The risk management step is a key difference between the spiral method and other approaches. In any iteration, if the risks cannot be successfully managed, then the project can be stopped.

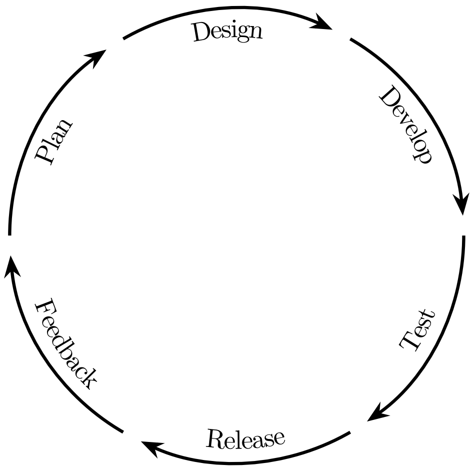
The emphasis on risk management makes the spiral approach suitable for large-scale, high-risk projects. These might be projects where technologies are unproven or multiple organisations are involved. However, project teams will often need to include a risk management expert (someone who has been trained to identify and quantify risk), which will add to the cost of the project.

Agile

The agile approach advocates building prototypes, testing, and incorporating feedback as soon as possible. Once a prototype is built, it is reviewed by the team and the client. At the next stage, the prototype can be modified and new features can be added if required.

The agile approach is suitable for many projects and is widely used. Large projects may need to be decomposed so that work can be spread across multiple teams and managed separately.

However, the use of prototypes is often criticised for giving rise to *scope creep*. This is where the client continually adds to the requirements when each prototype is evaluated, as they think of new features or functionality.



The Agile Manifesto describes 12 principles:

1. Prioritise customer satisfaction and provide early and continuous delivery
2. Welcome changing requirements, even late in development
3. Deliver working software frequently
4. Business people and developers should work together daily throughout the project
5. Build projects around motivated individuals — give them the environment and support they need, and trust them to get the job done
6. Face-to-face conversation is the most effective way of communicating within a team
7. Working software is the primary measure of progress
8. Development should be sustainable — everyone should be able to maintain a constant pace indefinitely
9. Continuous attention to technical excellence and good design enhances agility
10. Simplicity — the art of maximizing the amount of work *not* done — is essential
11. The best architectures, requirements, and designs emerge from self-organizing teams
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly

Rapid application development

Rapid application development (RAD) is an agile approach based around prototyping. It puts less emphasis on rigid planning and more emphasis on an adaptive process.

RAD is especially well suited for developing software where the requirements are unclear, or for projects where the user interface is an important part of the project. It is good for smaller projects where there are smaller development teams that include users or their representatives.

Because of its emphasis on continuous feedback, it is not suitable for projects where the users are hard to contact or do not want to be actively involved.

However, it can encourage sloppiness — a 'try-and-see' approach — and ignores the importance of making sure that the overall system architecture is right. Projects can also sometimes take far longer than anticipated, where users make constant requests for minor changes.

Extreme programming

Extreme programming (XP) is another agile approach. Where RAD focuses on the refinement of the objectives, XP focuses on the refinement of the code.

The core principle is that there should be frequent releases in short development cycles, and continual communication with the customer. In XP, the user (or a representative) is part of the development team. They are there to focus on the 'user journey' and to provide feedback at every stage of the project.

There are several key differences between XP and RAD. The importance of code quality is paramount in XP. The development and feedback loops are much shorter, and the releases are not called 'prototypes', but are working versions of a system component.

At the start of each release, there is a planning phase that determines what will be developed and how it will be tested. Programming is done in pairs, with two programmers working in collaboration to develop, test, and review code. There is a heavy emphasis on development standards and strict version control. At the end of each release, there is a feedback stage that will inform the next release.

The main benefit of the extreme programming approach is that code quality is high. This means that there are fewer bugs, and therefore less time needs to be spent fixing them. Also, having a user on each team means that there is a strong focus on getting the user requirements right.

The main criticism of XP — as with all forms of iterative development — is that of scope creep, due to a lack of firm requirements at the outset of the project. It also requires a high degree of communication and collaboration, which can sometimes be difficult or even impossible if team members do not work in the same office. As with RAD, the client has to commit to having one or more users actively involved throughout the project.

Testing

Types of testing

Black box testing is **integration** testing - independent of code. It looks at the program specification and creates a set of test data that covers all reasonable inputs, outputs, and program functions.

White box testing is **unit** testing - it depends on the details of the code logic. Unit tests should hit every code path at least once

Alpha testing is carried out by the software developer's in-house team. It can reveal errors or omissions in the definition of the system requirements and find bugs as early as possible. Alpha testing can also be done by calling the user in and asking them to evaluate the application at that stage of development.

Beta testing is used when commercial software is being developed. The software is given to a subset of potential users, who agree to use the software and report any faults. Real users will do things that the developers never anticipated.

Acceptance testing is **evaluation**. The users needs to test every aspect of the software to make sure it does what it is supposed to do. It will be evaluated against the original specification document.

Types of maintenance

Corrective maintenance: Bugs will usually be found when the software is used in the field, no matter how thoroughly the software was tested.

Adaptive maintenance: Over time, requirements will change and the software will have to be adapted to meet new needs.

Perfective maintenance: Even if the software works as intended, there are often ways of improving it. Perhaps by making it easier to use, faster, or adding quality-of-life improvements.