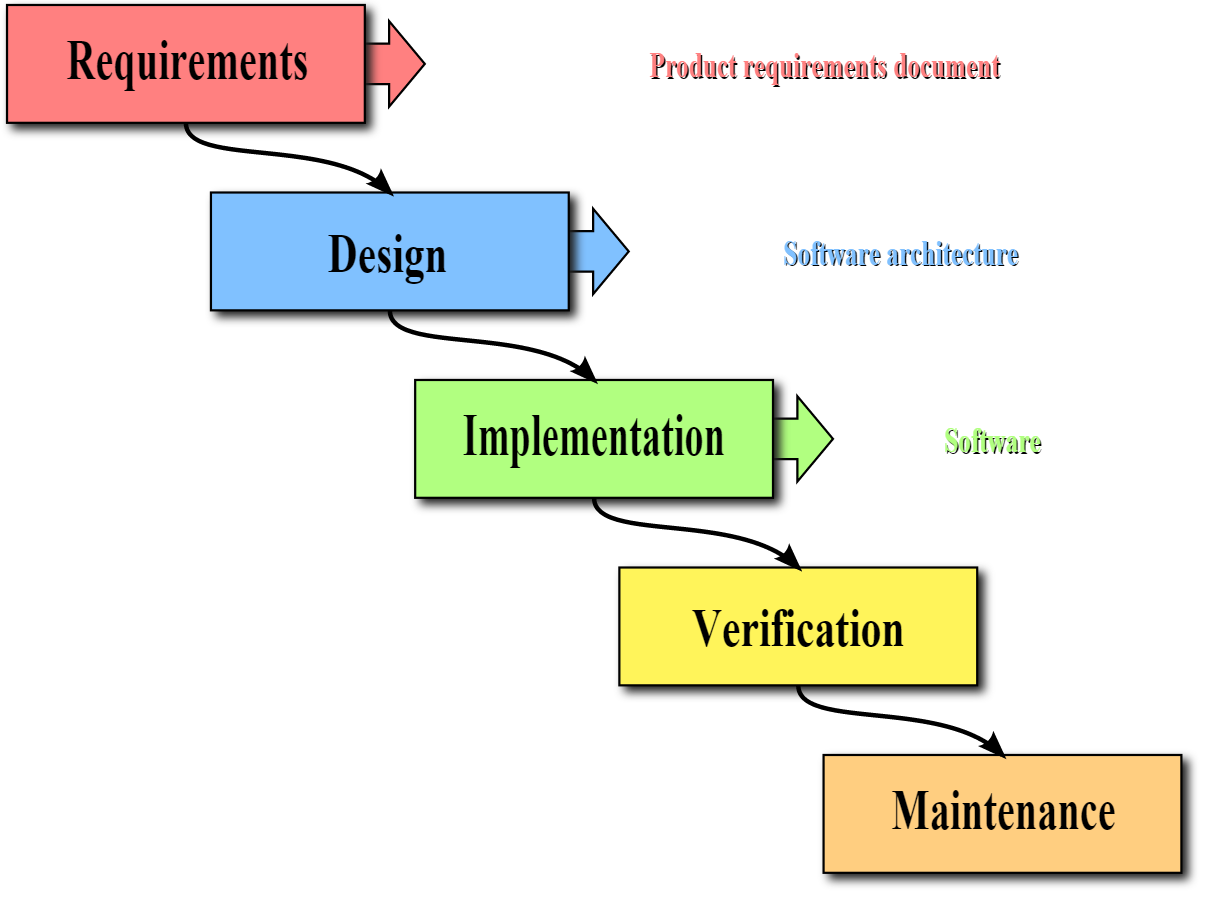
**Waterfall model**



**Summary**

The waterfall model is a sequential design process, used in software development processes, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Conception, Initiation, Analysis, Design, Construction, Testing, Production/Implementation and Maintenance.

The waterfall development model originates in the manufacturing and construction industries; highly structured physical environments in which after-the-fact changes are prohibitively costly, if not impossible. Since no formal software development methodologies existed at the time, this hardware-oriented model was simply adapted for software development.

**Pro**

Some waterfall proponents prefer the waterfall model for its simple approach and argue that it is more disciplined. The waterfall model provides a structured approach; the model itself progresses linearly through discrete, easily understandable and explainable phases and thus is easy to understand; it also provides easily identifiable milestones in the development process. It is perhaps for this reason that the waterfall model is used as a beginning example of a development model in many software engineering texts and courses.

**Cons**

It impossible for any non-trivial project to finish a phase of a software product's lifecycle perfectly before moving to the next phases and learning from them. For example, clients may not know exactly what requirements they need before reviewing a working prototype and commenting on it. They may change their requirements constantly. Designers and programmers may have little control over this.

Designers may not be aware of future implementation difficulties when writing a design for an unimplemented software product. That is, it may become clear in the implementation phase that a particular area of program functionality is extraordinarily difficult to implement.

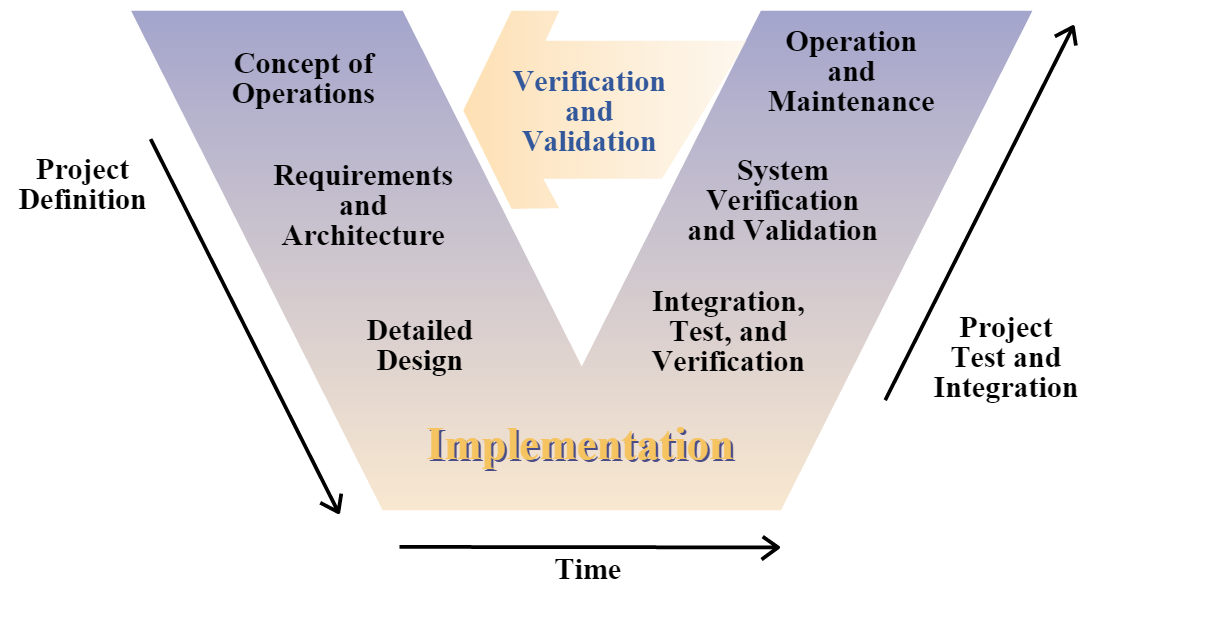
A further argument for the waterfall model is that it places emphasis on documentation (such as requirements documents and design documents) as well as source code. In less thoroughly designed and documented methodologies, knowledge is lost if team members leave before the project is completed, and it may be difficult for a project to recover from the loss.

**Feasibility**

This is the central idea behind the waterfall model: time spent early on making sure requirements and design are correct saves much time and effort later. Thus, the thinking of those who follow the waterfall process goes, make sure each phase is 100% complete and absolutely correct before proceeding to the next phase. Program requirements should be set in stone before design begins (otherwise work put into a design based on incorrect requirements is wasted). The program's design should be perfect before people begin to implement the design (otherwise they implement the wrong design and their work is wasted), etc.

So, it is argued that the waterfall model in general can be suited to software projects that are stable (especially those projects with unchanging requirements, such as with shrink wrap software) and where it is possible and likely that designers will be able to fully predict problem areas of the system and produce a correct design before implementation is started. The waterfall model also requires that implementers follow the well-made, complete design accurately, ensuring that the integration of the system proceeds smoothly.

**v-model**

****

**Summary**

The V-model represents a software development process which may be considered an extension of the waterfall model. Instead of moving down in a linear way, the process steps are bent upwards after the coding phase, to form the typical V shape. The V-Model demonstrates the relationships between each phase of the development life cycle and its associated phase of testing. The horizontal and vertical axes represents time or project completeness (left-to-right) and level of abstraction (coarsest-grain abstraction uppermost), respectively.

**Pro**

It provides a variant on the waterfall model, so it has advantages that most waterfall model has.

And has fix some disadvantages of waterfall. In each phases, there are some tests to find whether you building a right product and building a product right. So it can always meet client need.

**Cons**

It is consistent with, and therefore implicitly encourages, inefficient and ineffective testing methodologies. It implicitly promotes writing test scripts in advance rather than exploratory testing; it encourages testers to look for what they expect to find, rather than discover what is truly there. It also encourages a rigid link between the equivalent levels of either leg (e.g. user acceptance test plans being derived from user requirements documents), rather than encouraging testers to select the most effective and efficient way to plan and execute testing.

It provides only a slight variant on the waterfall model and is therefore subject to the same criticisms as that model. It provides greater emphasis on testing, and particularly the importance of early test planning. However, a common practical criticism of the V-Model is that it leads to testing being squeezed into tight windows at the end of development when earlier stages have overrun but the implementation date remains fixed.

It is inflexible and encourages a rigid and linear view of software development and has no inherent ability to respond to change.

**Feasibility**

V- Model application is almost same as waterfall model, as both the models are of sequential type. Requirements have to be very clear before the project starts, because it is usually expensive to go back and make changes. This model is used in the medical development field, as it is strictly disciplined domain. Following are the suitable scenarios to use V-Model:

Requirements are well defined, clearly documented and fixed.

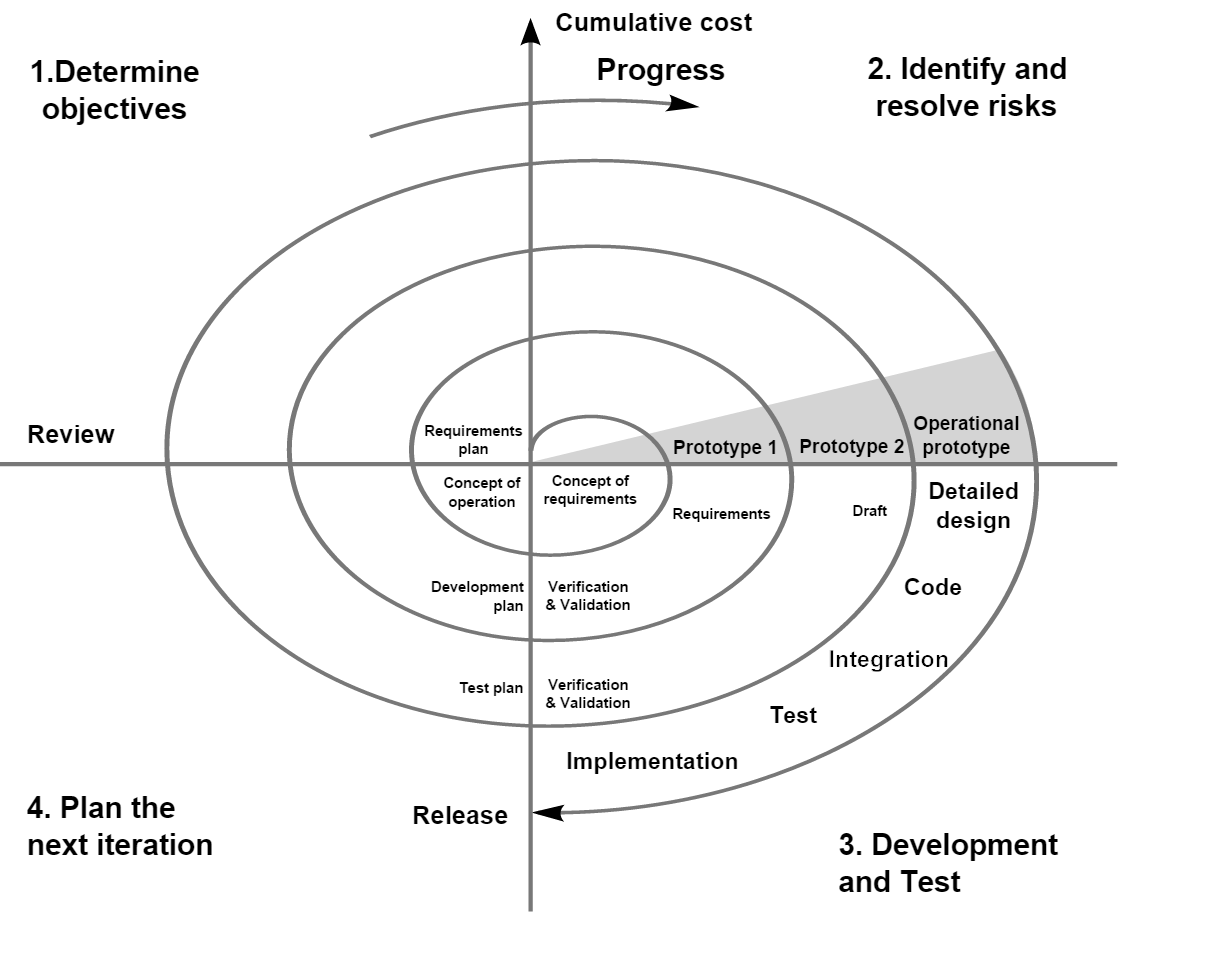
Product definition is stable.

Technology is not dynamic and is well understood by the project team.

There are no ambiguous or undefined requirements.

The project is short.

Spiral model



Summary

The spiral model is a risk-driven process model generator for software projects. Based on the unique risk patterns of a given project, the spiral model guides a team to adopt elements of one or more process models, such as incremental, waterfall, or evolutionary prototyping.

The spiral model has four phases. A software project repeatedly passes through these phases in iterations called Spirals.

Identification: This phase starts with gathering the business requirements in the baseline spiral. In the subsequent spirals as the product matures, identification of system requirements, subsystem requirements and unit requirements are all done in this phase.

Design: Design phase starts with the conceptual design in the baseline spiral and involves architectural design, logical design of modules, physical product design and final design in the subsequent spirals.

Construct or Build: Construct phase refers to production of the actual software product at every spiral. In the baseline spiral when the product is just thought of and the design is being developed a POC (Proof of Concept) is developed in this phase to get customer feedback.

Then in the subsequent spirals with higher clarity on requirements and design details a working model of the software called build is produced with a version number. These builds are sent to customer for feedback.

Evaluation and Risk Analysis: Risk Analysis includes identifying, estimating, and monitoring technical feasibility and management risks, such as schedule slippage and cost overrun. After testing the build, at the end of first iteration, the customer evaluates the software and provides feedback.

Pro

The advantage of spiral lifecycle model is that it allows for elements of the product to be added in when they become available or known. This assures that there is no conflict with previous requirements and design.

This method is consistent with approaches that have multiple software builds and releases and allows for making an orderly transition to a maintenance activity. Another positive aspect is that the spiral model forces early user involvement in the system development effort.

There is no difference between software maintenance and development. And can reduce unnecessary testing.

Cons

It takes very strict management to complete such products and there is a risk of running the spiral in indefinite loop. So the discipline of change and the extent of taking change requests is very important to develop and deploy the product successfully.

**Feasibility**

Spiral Model is very widely used in the software industry as it is in synch with the natural development process of any product i.e. learning with maturity and also involves minimum risk for the customer as well as the development firms. Following are the typical uses of Spiral model:

When costs there is a budget constraint and risk evaluation is important.

For medium to high-risk projects.

Long-term project commitment because of potential changes to economic priorities as the requirements change with time.

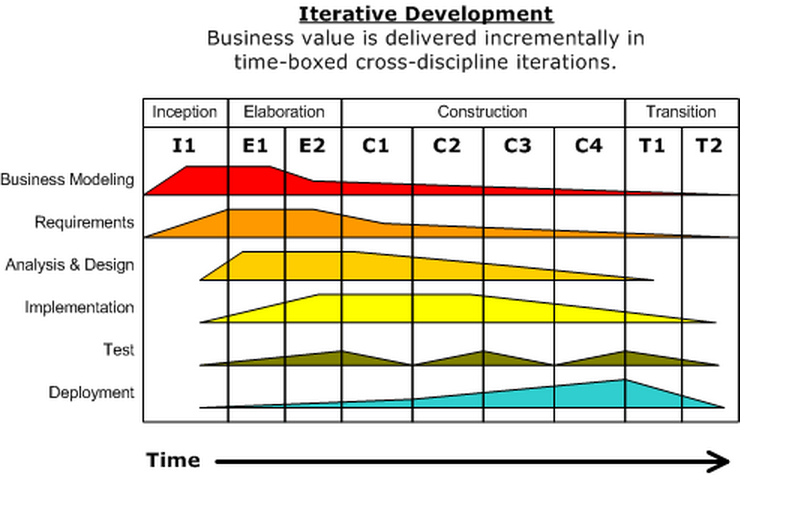
Customer is not sure of their requirements which is usually the case.

Requirements are complex and need evaluation to get clarity.

New product line which should be released in phases to get enough customer feedback.

Significant changes are expected in the product during the development cycle.

URP model



Summary

The Rational Unified Process (RUP) is an iterative software development process framework created by the Rational Software Corporation, a division of IBM since 2003. RUP is not a single concrete prescriptive process, but rather an adaptable process framework, intended to be tailored by the development organizations and software project teams that will select the elements of the process that are appropriate for their needs. RUP is a specific implementation of the unified process.

RUP is based on a set of building blocks and content elements, describing what is to be produced, the necessary skills required and the step-by-step explanation describing how specific development goals are to be achieved. The main building blocks, or content elements, are the following:

Roles (who) – A role defines a set of related skills, competencies and responsibilities.

Work products (what) – A work product represents something resulting from a task, including all the documents and models produced while working through the process.

Tasks (how) – A task describes a unit of work assigned to a Role that provides a meaningful result.

Within each iteration, the tasks are categorized into nine disciplines:

Six "engineering disciplines"

Business modelling

Requirements

Analysis and design

Implementation

Test

Deployment

Three supporting disciplines

Configuration and change management

Project management

Environment

Pro

RUP is similar in concept to Extreme Programming in that only what is useful and required is produced and the development plan is updated throughout the process. Both methods seek to develop a system of best practices in software development.

The RUP development methodology provides a structured way for companies to envision create software programs. Since it provides a specific plan for each step of the development process, it helps prevent resources from being wasted and reduces unexpected development costs.