
Project Management

Wolfgang Pree

C. Doppler Laboratory for Software Engineering
University of Linz

Process Models

Conventional vs. OO process models

Reuse process models^{*)}

Planning & Controlling^{*)}

Team structures^{*)}

Costs & Benefits of OO Technology

Avoiding Failures^{*)}

^{*)} The presentation is based on *Succeeding with Objects—Decision Frameworks for Project Management* by Adele Goldberg and Kenneth Rubin (Addison-Wesley, 1995)

Process models

Process models

When people are confronted with the need to solve a complex task, they attempt to systematically decompose the process of solving the problem, i.e., to **define an approach model**.

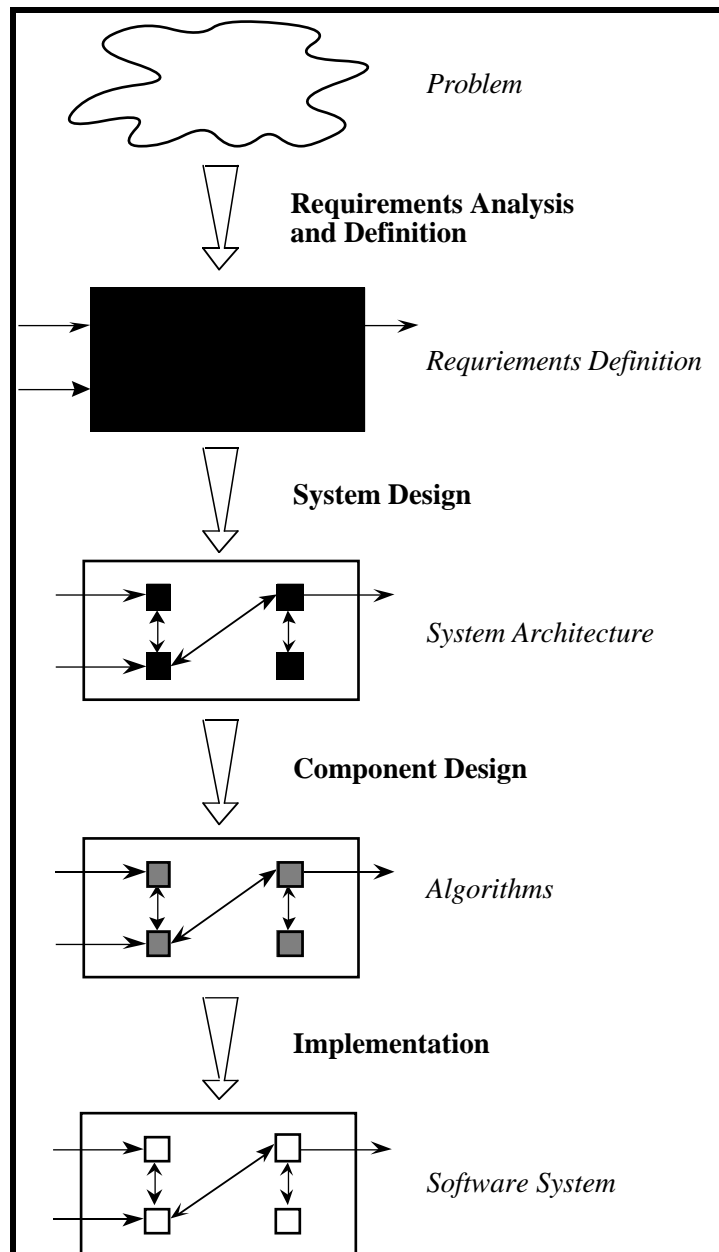
Such an approach model **regulates the chronological sequence of the solving process**. It decomposes the solving process into distinct steps that are intended to make a stepwise planning, decision and implementation possible.

We apply this basic idea from the field of systems engineering to the design process for software products. Analogous to other types of projects, software projects are thus subdivided into individual **project phases**.

The phases and their chronological sequence are collectively termed the software life cycle, which has become a classical term in the field of computer science.

Process models

Software design using the life cycle paradigm is based on the principle of **top-down decomposition of so-called black boxes**, i.e., stepwise refinement:



Process models

The prerequisites for the individual phases are well-defined inputs (usually in the form of documents). These inputs are processed in the respective phases with adequate methods and tools, and the results are passed on to the next phase.

The phases are clearly defined, and a given phase is terminated only when its outcome is validated and verified.

One characteristic of the software life cycle paradigm is that during the analysis and specification phases the **system is described from the outside**, i.e., in terms of **what** the system is to achieve. **How** the system is to achieve the requirements is left open.

Process models

The system itself is viewed as a black box whose outward effect is precisely defined and whose internal structure remains hidden.

During the **system design phase** the procedure is analogous in that **one specifies the components into which the system is to be decomposed**, what the components must do, and how they must work together. When the process is completed, i.e., the interfaces of all the system components are defined, the designers need only tend to component specifications in order to achieve the design of the algorithmic structure of the system components.

The process supports a **successive reduction in complexity** of both the design process and the product itself. The algorithms are then translated into a programming language and tested individually.

The decomposition is followed by a synthesis of system components.

Process models

Studies have shown that the software life cycle paradigm is the most widespread software development methodology used today.

Practice, however, shows the limits and the drawbacks of this paradigm.

The model is based on the (false) assumption that (as a rule) the development process is linear and iterations between phases occur only as exceptions.

The strict separation of the individual phases is an unacceptable idealization. In reality the activities of the phases overlap and interaction between phases is much more complex than exhibited in the paradigm.

Process models

The strictly sequential approach leads to **tangible products or components being available only at a late stage**. Yet experience shows that the validation process is insufficient without experiments close to reality.

=> Modifications requested by the client can only be expressed relatively late, and integrating them at that stage can lead to substantial overhead.

In short, **the activities established in this paradigm are certainly necessary**—although perhaps not sufficient—and **cannot be circumvented by any other ingenious methodology**.

The use of this methodology in practice has taught us that we cannot achieve the goal of system development completely independently of the type of solution chosen. But exactly that is assumed in the life cycle paradigm.

The strict sequential proceeding, the manner in which the prescribed activities are performed, and the nature of the intermediate products form the weak points of this methodology and leave room for improvement.

Process models

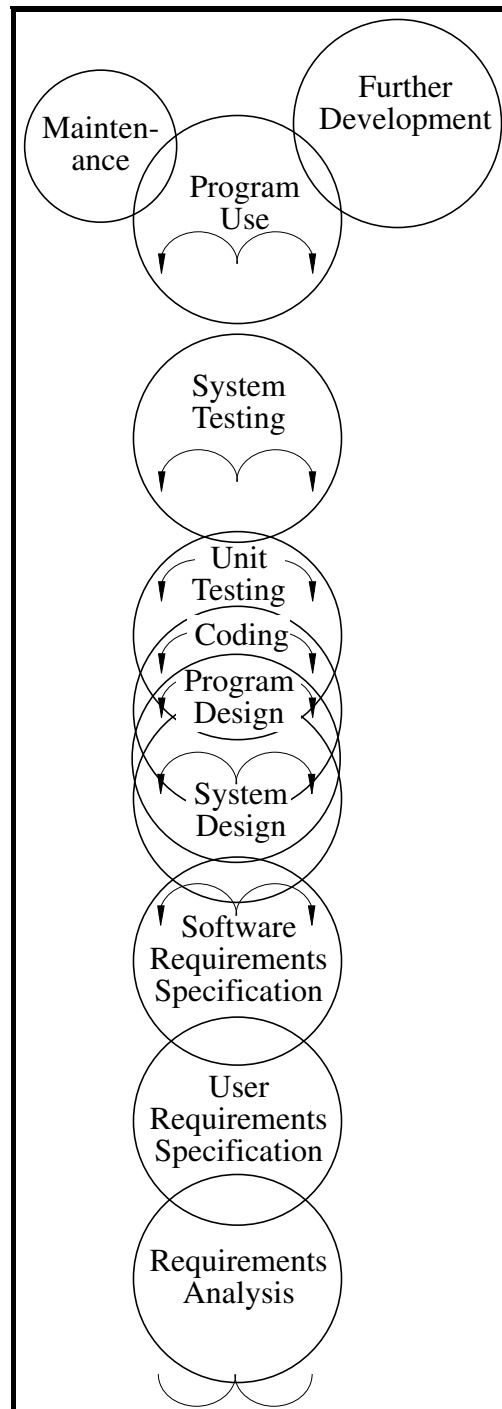
Sample enhancements of the Waterfall Model:

Spiral Model:

- **consists of a series of spirals** (identification of objectives & alternative ways how to accomplish them, evaluation of the alternatives)
- includes prototyping

Process models

Fountain Model (Brian Henderson-Sellers):



Process models

Activities of the classical software life cycle are just put one upon the other; the activities together with possible iterations form a “fountain”.

The fountain model does not significantly differ from other modifications of the classical waterfall model: It just takes iteration into account and expresses that design, implementation, and test activities overlap to a high degree in object-oriented system development.

Process models

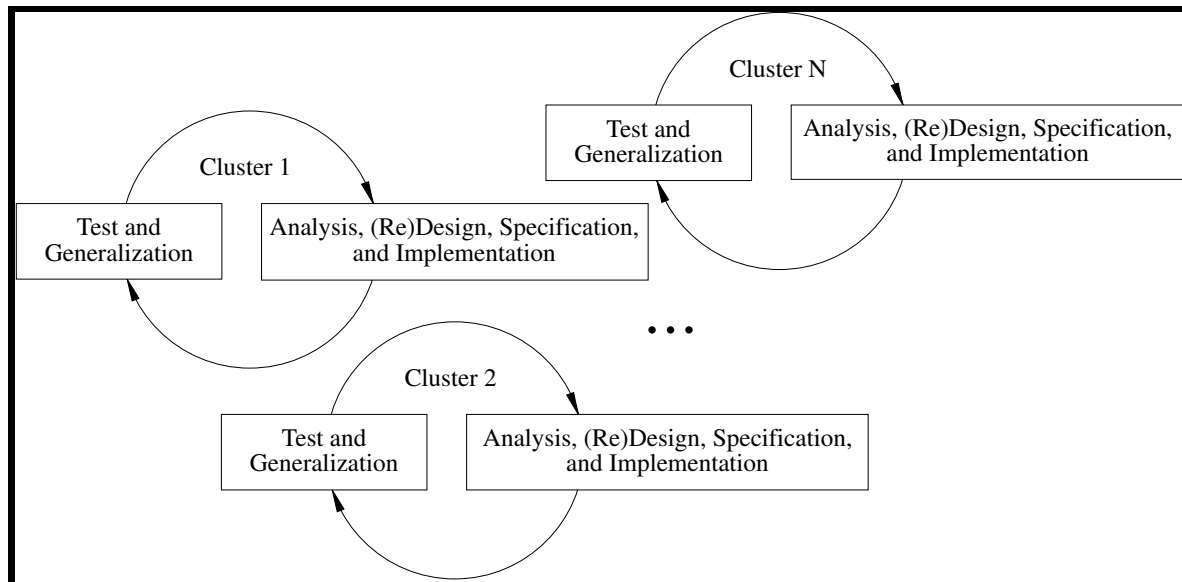
Cluster Model (Bertrand Meyer):

a cluster, in this context, is a group of related classes. The development of classes belonging to one cluster constitutes *one* life cycle. The following changes to the usual software life cycle become necessary in this model:

- Activities of the software life cycle are not applied to the system as a whole. (The all-or-nothing approach considers a system as a monolithic entity.) Instead, system development is split into several sub-life cycles overlapping in time.
- Design and implementation activities are merged into one activity—a fact that completely contradicts the usual software life cycle models. The reason why this is possible and makes sense is explained in Section 6.1.1, “Case Study: Cluster-Oriented Development of DICE”.
- A new activity—called *generalization*—is introduced in order to produce reusable software components. This activity can be merged with the test activity.

Process models

The following software life cycle results:



Note that the activities analysis, (re)design, specification and implementation are merged into one activity.

According to Booch's (1991) quote of Heinlein these recent modifications follow the admonition

“When faced with a problem you do not understand, do any part of it you do understand, then look at it again.”

Process models

Prototyping-Oriented Development:

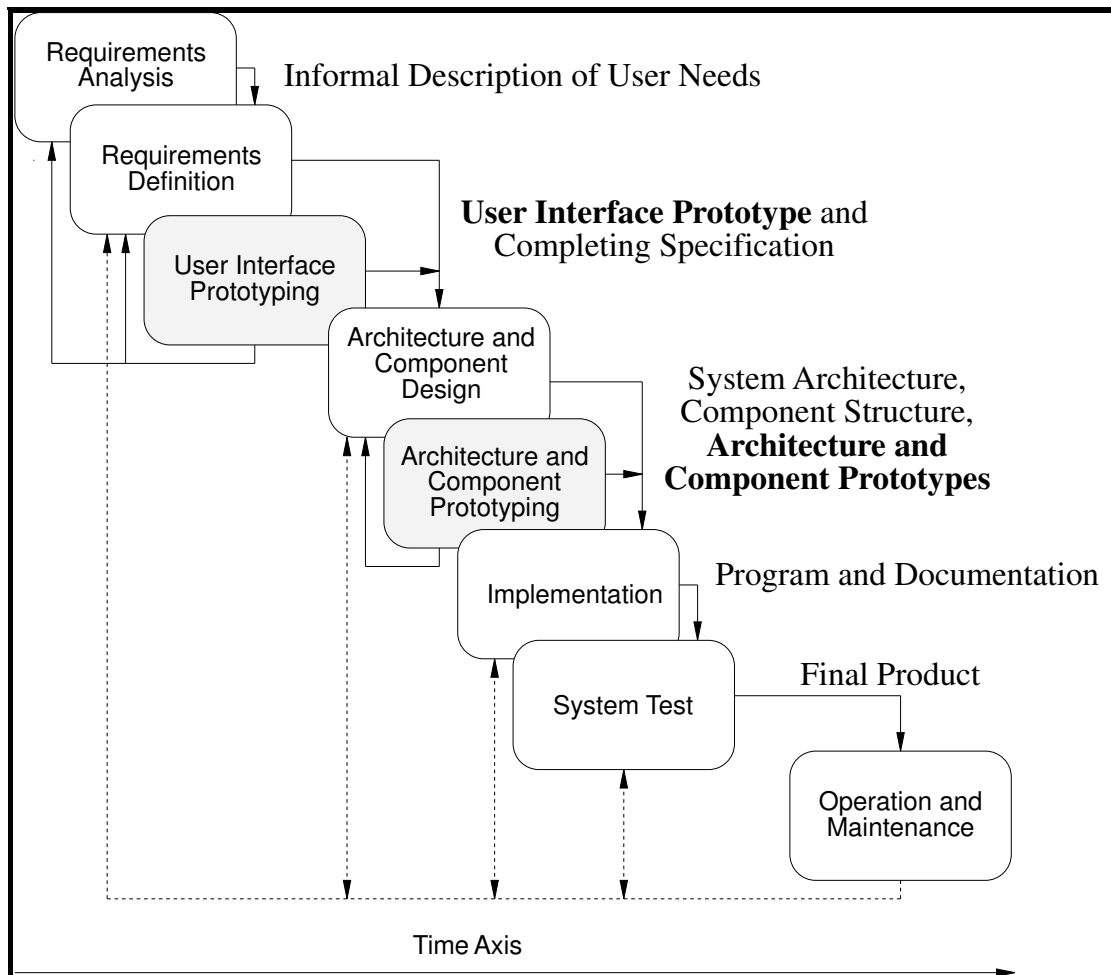
The experts are nearly unanimous that software prototypes—in contrast to other prototypes, e.g., in the realm of hardware—must be **realized quickly and cheaply**.

Connel and Shafer define a software prototype as a dynamic visual model providing a **communication tool for customer and developer** that is far more effective than either narrative prose or static visual models for portraying functionality. It should be:

- functional after a minimal amount of effort
- a means for providing users of a proposed application with a physical representation of key parts of the system before system implementation
- flexible modifications require minimal effort
- not necessarily representative of a complete system.

Process models

A prototyping-oriented development paradigm is not radically different from a purely phase-oriented one. Furthermore, the two are to be viewed more as complementary than as alternative. The new aspects are that this model is **explicitly not linear but iterative** and that it **specifies where and how this iteration is not just possible but necessary**:



Process models

Which process model is the right one for objects?

According to our experience, corroborated by Goldberg and Rubin:

There is not one process model for objects!

Goldberg and Rubin discern the following types of (OO) projects:

- First-of-Its-Kind
- Variation-on-a-Theme
- Legacy-Rewrite
- Creating Reusable-Assets
- System Enhancement or Maintenance

Process models

=> several **OO-friendly strategies** can be considered when constructing a specific process model:

- **iterative (=“opportunistic”) development**

like solving a Jigsaw Puzzle

supports the reworking of parts of a system

a more elegant name for a chaotic working style

- **incremental development**

a system is partitioned; parts are developed at different times and/or rates

many projects combine iterative and incremental development

Process models

- **prototyping**

conventional prototyping (e.g., Spiral Model): as means to reduce the risk of getting the user interface wrong

OO prototyping: prototyping of all system aspects, in particular its architecture

- **consuming/producing reusable assets**

Process models

Defect detection potential (adapted from Goldberg and Rubin):

	Analysis	Design	Code	Docu.
Prototyping	++	++	--	n.a.
Inspections	++	++	+	++
Reviews	±	±	±	+
Testing	±	±	++	n.a.
Proofs	+	+	+	n.a.

Process models

Reuse process models

Aspects that have to be considered when setting up a reuse process model (from Goldberg and Rubin):

define

decide what should be reusable and how to reuse

identify

determine the specific artifacts that are to become reusable assets of the organization

acquire

(let) build or buy reusable assets

certify

set and follow guidelines to determine acceptability of a reusable asset

classify

select and apply a method and notation for organizing and labeling the reusable assets

store

choose a method for storing assets so they can later be located and retrieved

Process models

communicate

select a method for making potential reusers aware of and interested in reuse opportunities

locate

choose a method for finding stored assets

retrieve

choose a method for taking possession of an asset

understand

choose a method for determining the purpose of the asset, and its analytic, design, or operational characteristics

use

choose a method for integrating an asset into the new context

update assets

choose methods for extending, updating, and repairing the assets in the library

update reusers

choose a method for updating the systems that use modified assets

Process models

Recommendations:

- focus on strategic assets, i.e., frameworks that represent your industry
- create these frameworks by yourself, but seek cooperation with other businesses in your industry to collaborate to create business components
- you need not expend initial efforts on creating retrieval schemes and tools when you create fewer, more powerful reusable assets

Process models

Organizational models for reuse:

Ad Hoc Model

everything goes into a repository

Supply and Demand Model

some things go out of the repository when they are not used

Expert Services Model

an expert team is responsible for identifying, acquiring, certifying, and storing reusable assets

Commercial-off-the-Shelf (COTS) Model

all reusable assets are acquired from outside vendors

Requirements for reuse success:

critical mass

availability of sufficient reusable assets

intelligibility

ability to understand code written by others

incorporation

ability to link reusable assets into new requirements

cultural transfusion

a way to transfer corporate reuse culture to subcontractors

attitude

a way to overcome the conservative views of older software engineers

Planning and Controlling

A **plan** includes a breakdown of work to be done, the time dependencies of tasks, and the allocation of resources.

Controlling is the set of activities that ensures that the project executes according to its plan.

Special planning and controlling are necessary for software projects that apply incremental and iterative development and prototyping.

Process models

OO technology allows you to manage the risk of a project with unknowns because it supports the use of prototyping, incremental and iterative development strategies.

Recommendations to plan under uncertainty:

- state clearly what you know and what you don't know
- state clearly what you'll do to eliminate unknowns
- make sure that all early milestones can be met
- plan frequent deliveries
 - => you win management trust
- plan to replan, i.e., interleave planning with execution
 - confidence comes from the planning process, not from the plan
 - => treat the plan as a living document

Process models

How you should track project progress:

- milestones
- tasks
every task should have a measurable outcome
- system capabilities
- quality based on quality-related tasks

Don't transform traditional progress measurements to OO, such as

LOC → number of classes

Process models

Team Structures

A **team** is a group of people who work together in a coordinated way to meet a clear set of goals and objectives.

Essentials aspects of how teams function:

- **roles** (e.g., analyst, prototyper, administrator, customer, technical leader, implementor, manager)
- **management**
- **communication**

Managerial roles: **technical** (regarding the technical solution), **people** (motivating team members), **administrative**

In smaller teams a single person might play all three roles.

Smaller teams (everyone has access to everyone) also communicate informally.

Process models

Classification of teams according to their structure:

Note: the selected team structure reflects the partitioning of the problem!

Hierarchical Team Model: is characterized by layers of authority for delegating responsibility and carrying out action

typical for large teams

subteams might be organized as **chief programmer teams**; precondition: availability of a guru

Egoless Team Model: group leadership is a rotating function; people share their work openly

typical for small teams

Process models

Military Team Model: as alternative to the Hierarchical Team Model it combines the notion of a hierarchy for planning and overall decision making, with full delegation of responsibility and authority when on the job

Goldberg and Rubin discern the following team structures focused on OO development that work both in small and large organizations:

Application Productization Team: It carries out analysis, design, implementation, and delivery of a software product.

The process model defines activities that map to useful roles on the team.

Objects form a common language by which the team members communicate.

In addition to the roles administrative manager, technical leader, and integrator an OO project should have an object expert to provide hands-on experience.

Process models

Application Prototyping Team: Prototypers should explore what is not known. Team members are encouraged to improvise (—> Jazz band :: marching band).

Framework Team: A special development team that creates a framework which is reused by various different Application Productization Teams.

Team members are reuse producers and have to adhere to the organization's guidelines for reusable assets.

A special role is a framework designer who can render the reusable abstractions of a problem domain as a collection of objects.

Process models

Cross-Project Team: In order to encourage reuse, a team of pollinators constantly moves from application project to application project. They identify reusable assets in one project and transfer them to other projects.

=> reusable assets are available before an application project is started

Reuse Team: manages the reuse process model (for the corresponding activities identification, acquisition, certification, ... see above). Roles: reuse manager, administrator, librarian, engineer, evaluator, maintainer.

Maintenance Team: is responsible for all incremental enhancements to a released product. If reuse is viewed as maintenance opportunity, the Maintenance Team could be the same as the Reuse Team (—> an improvement in a reusable asset could improve all systems that reuse this component).

Process models

Recommendations:

- The number and kinds of roles on a project are tied to the kinds of activities necessary to meet a project's goals.
- Successful teams enroll the end user as an insider.
- Managers serve their teams (take care that team members have the adequate information and resources).
- People cannot be successful if they are given responsibility without authority.
- (Sub-)teams should consist of a maximum of 6–8 people.
- Highly motivated but unqualified people can cause a project to fail, albeit enthusiastically.
- Every OO effort requires at least one team member who is an expert in the technology.

Costs & Benefits of OO Technology

Cost & Benefits of OO Technology

Since the software industry can hardly afford to implement large-scale software systems twice most comparisons of conventional and object-oriented programming techniques are based on small projects. This situation served as an incentive for us to compare representative software systems.

But even reimplementations have to be considered carefully. For instance, a system should not be implemented twice by the same project team. Experience gained in the first implementation will help in the reimplementation whether or not the team starts with the conventional or object-oriented solution. Thus a system has to be implemented by two different project teams which have about the same knowledge in the beginning.

Cost & Benefits of OO Technology

The type of a system to be built also has an undeniable influence on the comparison. Strongly algorithm-oriented systems won't benefit as much from object-orientation as systems that have to deal with various different and complex data structures.

This study starts with a presentation of results based on several systems which have been developed in various object-oriented programming languages:

- C++
- Smalltalk
- Object Pascal
- Eiffel and
- Oberon

and compared to conventionally implemented systems over the past five years.

Cost & Benefits of OO Technology

The overall size of these projects is about 500,000 lines of code. The projects were carried out together with partners from industry, for example, the Union Bank of Switzerland (UBS), Siemens AG Munich, and the Austrian Industries Corporation.

Application areas:

- software engineering tools: development environments (SNiFF+, ...), compiler, documentation tools, UI prototyping tools
- automation software: process control, distributed systems
- information systems

Cost & Benefits of OO Technology

Positive aspects of object-oriented software development are marked with a “+”, negative ones with a “-”. If there is no significant difference as compared to conventional software development, a “±” sign is used.

Qualitative aspects

Qualitative aspects focus on project management and the final software product. The presented results are also corroborated by other authors, e.g., Tom Loves in *Object Lessons* (SIGS Books, 1993) as well as Goldberg and Rubin.

Cost & Benefits of OO Technology

Impact on Project Management

The positive effects of object-oriented programming on several phases of the software life cycle result from

- the reduction of the semantic gap (developers can think in terms of real world objects)
- increased reusability and thus the fact that less new code has to be written.

It is no surprise that most disadvantages are related to the management of class libraries.

Cost & Benefits of OO Technology

+ **Planning** It can easier be determined in the planning phase of a project which subprojects/tasks will require most efforts. The reason for this lies in the reduced semantic gap and the better reusability of object-oriented building blocks. Because of the almost schematic way in which object-oriented systems are designed (especially if application frameworks are used), more time remains for planning.

+ **Organizational Effort** Due to the reduced design and implementation efforts, smaller project teams are sufficient. Smaller teams ease the communication among the people and the coordination of subtasks.

+ **Design and Development** Frameworks already anticipate much of a system's design. Thus elbowroom diminish which could else lead to bad design is diminished. Object-oriented programming supports a more homogeneous style so that programmers who are not involved in design decisions will be able to understand a systems's design more quickly.

Cost & Benefits of OO Technology

Furthermore, design and implementation are more intertwined so that intermediate results will be produced within a shorter time frame. Thus project team members will experience a sense of achievement which can strongly motivate them.

+ **Division of Labor** Object-oriented analysis and design produce class definitions which create a modularization of the system under development. The classes can be refined (i.e., implemented) almost independently by different teams or members of a team. Another kind of division of labor can be observed in larger organizations where a dedicated group is responsible for providing elementary classes and concepts.

+ **Prototyping** Due to a reduced implementation effort, extensible prototypes (in the sense of evolutionary prototyping) for some system components can be developed within a short time. Application frameworks for graphic user interface programming also help to come up with user interface prototypes within a reasonable time if user interface prototyping tools are missing.

Cost & Benefits of OO Technology

- **Settling-In Period** Programmers with experience in conventional programming often have a hard time getting used to the new way of thinking which object-oriented software development requires. A couple of months are usually required to accomplish this migration from conventional paradigms to the object-oriented paradigm.
- **Management of a Class Library** In order to benefit from a class library it has to be kept up-to-date. The required effort directly depends on the number of users of a particular class library, its size (i.e., number of classes) and the number of projects that are based on it.
- **Project Cost Calculations** Due to reuse of software components it becomes unclear how to split the costs among several projects. A class developed in a project causes costs which are higher if the class is designed to be reusable in other projects. But other projects benefit from this additional effort. The costs of class library management have to be treated in a similar way: they have to be shared among several projects.

Cost & Benefits of OO Technology

Implications on the Quality of the Final Product

Object-oriented software development improves the overall quality of the produced software. Quality improvements result from the reuse of already tested components.

Subsequently, implications of object-oriented software development on crucial software quality criteria are presented.

+ **Correctness and Reliability** Since the correctness of a software system depends on the correctness of its components, the probability of producing a correct system is the higher when more matured software components can be reused. Components of a class library usually have been reused a number of times so that their correctness can be regarded as high (ideally 100%).

Cost & Benefits of OO Technology

Furthermore, the usage of an application framework has a positive effect on the correctness of a software system, since cookbook recipes describe of how to combine framework components in order to achieve a certain goal. Adhering to these guidelines means avoiding typical errors.

The reliability of a software system is also tightly coupled with its correctness. So reusable object-oriented components exert a positive effect on this quality aspect.

Experience has proven that the time consumed for stabilizing a software system can be reduced to a third of the time required for conventional systems.

+ User Friendliness The ease of use is especially improved if a system is based on a GUI application framework. GUI application frameworks anticipate much communication between application and user. This provides functionality that would not have been implemented in conventional applications since many features that make GUIs easier to use are hard to implement. Most GUI application frameworks, for example, support

Cost & Benefits of OO Technology

undoing/redoing of commands. Furthermore, several different look and feel standards are supported by some GUI frameworks so that applications based on such a framework can easily be ported to other look and feel standards.

Offering too much functionality could also be a disadvantage. Since it is very simple to provide certain features just by reusing components, many details could detract users from what they really want to do.

+ Maintainability Due to the modularity of object-oriented programs, errors can easier be detected and localized: operations associated with objects are gathered in one class. Thus testing of object-oriented programs becomes easier, too.

Since frameworks define already much of a system's design, they cause a uniformity of all systems built on top of them. Thus learning details about an object-oriented system often means less effort compared to conventional ones.

Cost & Benefits of OO Technology

Finally, polymorphism and dynamic binding promote extensibility as they can help to avoid case statements spread over a software system.

± **Efficiency** Object-oriented software development might have a negative influence on a program's run-time and memory requirements (see the section about quantitative aspects for details). Taking the advances in hardware into consideration, these costs can almost be neglected.

± **Portability** Porting object-oriented systems based on class libraries that depend on a specific hardware and operating system can be considered as hard as porting such conventionally implemented systems.

Furthermore, porting an object-oriented system from one language to another is anything but trivial: a C++ program, for example, cannot be transformed into a Smalltalk program or vice versa simply by syntactical changes. The different ways of thinking in both language worlds have to be matched.

Cost & Benefits of OO Technology

Quantitative aspects

The numbers given for various quantitative aspects express the impact of object-orientation compared to conventional approaches.

These numbers have to be seen as a percentage with reference to the conventional solution. A value of 120%, for instance, means that the quantitative measure increases by a fifth if the object-oriented paradigm is used instead of a conventional paradigm.

Cost & Benefits of OO Technology

Impact on Source Code Size

In the case of object-oriented programs, we have to discern between the newly written source code and the reused source code:

+ **Newly Written Code: 25-50%** The reduction of source code that has to be written is often considered to be a very important achievement of object-oriented programming. In fact, this number directly influences the duration of a software project and its maintenance costs. Since many standard problems of a specific domain have already been solved by classes of an application framework, usage of an appropriate framework can even reduce the size of newly written source code up to 90%. Such extreme reductions are typical for small projects (about 5,000 to 15,000 lines of code) where an existing application framework has to be adjusted to specific needs. Code reductions are less in case of systems that contain many algorithm-oriented parts.

Cost & Benefits of OO Technology

Even if there is no class library at hand reductions of the size of newly written code can be observed. These reductions are proportional to the project size. The larger the project the less code has to be written. The reason for this lies in the internal reuse of components within a project. A careful design (especially factoring out commonalities into superclasses) implies that more components can be reused as compared to conventional (e.g., module-oriented) software development.

- Overall Source Code Size: 120-300% Although less code has to be written, the overall size of the source code increases. This is caused by inherited code and indirectly used classes. The given number may vary extremely depending on the size and cohesion of the used class library. Using classes only for data structures, for example, results in a minimal increase of the overall source code size and almost neglectible savings in code that has to be written newly. On the other hand, using an application framework implies importing numerous classes that do not

Cost & Benefits of OO Technology

contribute directly to the solution of a particular problem. But these additional classes also provide functionality for free which could only be achieved with enormous effort if no application framework is used. The generality of directly and indirectly used classes constitutes another factor that influences the size of imported code. In order to increase the generality of a class additional methods have to be provided that enlarge the code size even if they are not used.

Impact on Run-Time

More precise numbers can be given about the impact of object-oriented programming on run-time issues than can be given about the overall code size. The time elapsed during a procedure call and method call can be measured resulting in a factor. The same is true for accessing object components. Of course, garbage collection affects a program's run-time too, since searching for unreferenced objects consumes time. Implications on the overall run-time

Cost & Benefits of OO Technology

are more difficult to determine, since it depends on the quality of the underlying class library and the programming style.

- Method Call: 105-120% Due to dynamic binding, method calls are more expensive than procedure calls. But it has to be taken into consideration that the overhead for passing parameters and the stack-like management of procedure call chains remains the same in both cases. Thus the given numbers describe calls without parameters based on method search with indexing. So-called dynamic method searching approaches used in some pure object-oriented languages cause significantly more overhead.

Cost & Benefits of OO Technology

- **Accessing Object Components: 100-?%** We assume that pointers to dynamically allocated memory are also used in the conventionally implemented system. The required machine code does not differ in either case (conventional and object-oriented) if the object-oriented language allows access to object components from outside. This kind of component access also occurs when object components are accessed within an object's method.

Accessing object components becomes much more expensive if method calls have to be used. In that case this operation can cost up to ten times more than a direct access. But this increase also occurs in conventional programs that are based on encapsulation and abstract data types.

Cost & Benefits of OO Technology

- Automatic Garbage Collection: 105-150% Several factors determine the costs of automatic garbage collection: number and size of objects, available memory, frequency of object generation and, obviously, the method of garbage collection. Normally automatic garbage collection does not cost more than 5% of the overall run-time. This minimal overhead is made possible by sophisticated garbage collection methods that take into account that object-oriented programs generate numerous objects with short life times. Unfortunately, since many objects are generated during calculation intensive periods most garbage collection methods become active then. Thus slow downs may occur that cannot be tolerated in real-time applications. Incremental methods avoid this disadvantage. They search for unreferenced objects and dispose them quasi parallel to the actual program. This parallel garbage collection effort avoids abrupt slow downs but also retards normal execution.

Cost & Benefits of OO Technology

± Overall Run-Time: 80-120% Though some factors cause a negative impact on a program's run-time, we could not observe a significant increase of the overall run-time.

We believe that there are three reasons for this:

- Systems written in hybrid languages do not only use objects. Methods contain not only method calls but also “regular” statements. Many local variables and basic data types are used in hybrid languages. This means that no dynamic binding and access of object components occurs.
- Class libraries use highly efficient implementation techniques. For example, sets of objects are implemented by means of hash tables resulting in a constant factor required for searching an element. Usage of such classes means an increase of several magnitudes in run-time efficiency.
- Dynamic binding is less expensive than a sequence of selection statements. Conventional programs are cluttered with case statements that can be avoided in object-oriented systems.

Cost & Benefits of OO Technology

- **Memory Requirements: 120-200%** Memory requirements of object-oriented systems depend not only on the source code size but also on method tables and dynamically allocated objects. The latter two aspects are discussed below.

The used class library exerts a strong influence on the memory requirements caused by source code. As already mentioned above the overall source code size increases due to direct and indirect reuse of classes. Finally, the compiler and linker determine the overall memory requirements. Some linkers are smart enough to identify classes and methods that are not used so that this overhead can be avoided.

Cost & Benefits of OO Technology

± Method Tables: 101-110% The ratio of method table size and source code size depends on the number of methods provided by objects of a particular class as well as on the method size. If a class implements 20 methods, for example, all subclasses will cause at least 20 entries in the method table even though the methods are not overridden in subclasses. Thus many small methods lead to huge method tables. Nevertheless, the memory requirements of method tables are almost neglectable compared to the overall memory requirements.

Cost & Benefits of OO Technology

- **Dynamically Allocated Objects: 100-120%** Usage of dynamically allocated objects means increased memory requirements to manage the objects. This is especially true for automatic garbage collection which requires additional information about an object's structure. If there also exists a method table that stores references to all objects, memory requirements can increase up to 20%. In case of simple memory management without garbage collection, objects are not more expensive than other dynamically allocated data.

Avoiding Failures

Avoiding failures

Goldberg and Rubin:

You fail because you choose to fail.

Good project management leads to success,
poor project management leads to failure.

Template for presenting typical failures related to OO
technology:

Kind of failure:

Failure-related decisions

Failures related to goals and objectives:

Never make a real management commitment to the use
of OO technology.

Position OO technology as the goal, rather than as the
means to the goal.

Avoiding failures

Failures related to product process models:

Fail to select a product process model.

Expect one process model to work for all projects.

Treat a prototype as a product.

Failures related to planning:

Insist on detailed schedules that do not have the developer's concurrence.

Report progress on lines of code and/or numbers of classes completed and/or consumed resources.

Failures related to reuse process model:

Ignore the need to reuse systematically.

Expect the first project to provide reusable assets.

Believe that objects magically make producing and consuming reuse easy.

Avoiding failures

Failures related to team structures:

Choose a team structure that does not mimic the project's process model and the target system's structure.

Choose people for team roles because they happen to be available.

Leave the role of object technology expert vacant.

Force developers to use OO technology.

Failures related to software development environments:

Treat C++ as a better C.

View the language as **the** technology.

Treat a notation as a method.

Select an environment that is OO in name alone.

Avoiding failures

Failures related to training:

Ask people to learn too much at once, without sufficient time to assimilate.

Fail to train management, at least up to the level where their jobs are affected.

Failures related to measurement:

Don't collect any measurement data.

Collect data about everything.

Forget to inform team why they are being asked to collect specific data.

Report effective reuse as a single number.