DF-16a

DATA TEKNISK FORUM

Iterative Software Development

- A Practical View

Abridged version

Morten Korsaa
Robert Olesen
Otto Vinter

February 2002

Secretariat:
DELTA
Venlighedsvej 4
DK-2970 Hørsholm
Tel.: (+45) 72 19 40 00
Fax: (+45) 72 19 40 01
www.datatekniskforum.dk

Contents

Iterative Software Development
A Practical View, Abridged version

Abridged

This is an abridged version of the full report. The following descriptions on this page pertain to the full report.

Purpose

The purpose of this report is to give you sufficient information to be able to:

- Understand and evaluate if iterative development will support your business.
- Choose among the most common iterative methods the one that is most suitable for you.

The audiences of this report are:

- Primarily developers, project managers, and line managers in development.
- Secondarily method/architecture groups and process/quality departments.

Abstract

This report contains a general description of the preconditions for iterative software development. A number of principles, which characterise iterative development, are presented along with a set of typical techniques to be used.

The main elements of the most common iterative methods are presented and organised into a common context for iterative development.

The most important issues and processes related to iterative software development are then discussed, as well as the effect it may have on obtaining or maintaining a software process maturity level or quality certificate.

Finally, experiences from the use of iterative software development in practice in a number of Danish companies are presented.

This report has been written by the following DELTA consultants on behalf of-Datateknisk Forum: Morten Korsaa, Robert Olesen and Otto Vinter

Contents

This report contains the following chapters.

Introduction

Introduction

This chapter introduces you to iterative development in general and to the report.

An executive summary is supplied for those who only need a quick glance at the essential messages of the report.

A Reader's Guide will tell you more about how the report is organised. This is followed by a clarification of some terms that we have chosen to use throughout the report. Please take the time to study these two sections. You can save yourself some time and become more focused in your further reading.

The final section in this introduction gives a short history of iterative software development.

Enjoy your reading.

Contents

This chapter contains the following topics.

Executive Summary

Overview

Iterative Software Development (ISD) may seem as yet another promising new technique that will save the software development world. What makes it different from other software development paradigms is that it addresses the fundamental problem: The uncertainty that we base our planning upon.

ISD is closer to solve this problem, than any other paradigm that has been around and hence more promising.

But it is also shaking the foundation of beliefs in companies, because it alters most of the ways that planning is performed and how plans are executed.

This report will provide you with:

- a framework of principles that you can use to appraise your own environment
- some inspiration from Danish case stories
- a short presentation of four of the most used methodologies
- an explanation of how ISD affects other processes in software development

After reading this report, we hope that you will know if ISD will support your business.

Business Need

Implementing ISD in an organisation will definitely affect the bottom-line of your business. If you are in charge of software development you may have a natural interest to know if and how ISD can improve your business.

The urgency by which your organisation needs ISD is determined by:

- How volatile your requirements are
- Your relationship with your customer

The importance of ISD for your business is determined by the sum of the many small benefits that the specific techniques bring to your projects.

Managers Responsibility

If you are in charge of software development, and you find that ISD can support your business, it is your responsibility that processes are designed based on the techniques that are available, and that the change is implemented. Get help if you like, but never forget that it is only you who have the power to drive the implementation process.

Maybe you will find one of the described methods in this report as suitable for your business. They all represent a good starting point. And if you favour another method that is not described here, it is probably the right for you. Use what seems to be the best, and improve it.

The Root Cause

Many problems in software development can be traced back to the uncertainty in the planning phase. Conventional planning is based on the assumption, that it is actually possible to predict what activities the project needs to do in order to complete the job. That rarely turns out to be the truth, and change becomes the rule in-

stead of the exception. This assumption spins off many activities that seem clever and necessary at the moment, but that never should have appeared in the first place. ISD is fundamentally dealing with the fact that change is the rule. Initially this may seem as a small difference, but it turns out to be a radical new paradigm.

Requirements Volatility

The volatility of requirements in your environment determines your need for ISD. You may benefit from ISD in any case, but you really need it if conditions around your project change fast and frequently.

Changing the planning paradigm to ISD may not be worth the effort if

- your customer delivers a complete detailed requirement specification
- your customer is happy if he receives exactly what is actually written in the specification
- your customer at the end will accept that the delivery is late, because getting the full functionality is more important

However, if you can foresee that;

- the requirements will change considerably
- the environment is not stable
- delivering on time is essential
- it is not possible to specify the requirements in enough detail up front

then you may find that there really is no other way than to change the planning paradigm to one that is originally constructed to deal with your daily reality, as opposed to one that is constructed to deal with a situation that rarely appears in reality.

Customer Relationship

The relationship with your customer determines how much of the ISD business benefit you can achieve. ISD is a new type of co-operation between the development department and the customer. Both parties have to agree to the use of ISD and appreciate the possibilities that ISD may provide. It takes two to tango.

If your customer can only deal with standard contracts that specify everything, and if you don’t deliver, it is a court matter, then you have to play that game. Some techniques may still be good for your business, but you won't get close to the full benefit.

But for every step your customer is willing to accept the responsibility (for his own business by the way!), and take active part in the development activities as described in various ISD methods, your mutual benefit from ISD will increase.

Are You Ready?

If you decide to adopt an ISD approach, you embark on an adventure that can bring you surprisingly new achievements. A new mindset will rule and you have to be ready for that. If you need the comfort of big Gantt charts, then stay where you are! The potential of applying ISD is huge, but getting there is tough. What you are about to do, is to wipe out all that you and your team/organisation have ever learned about planning. You have to be ready to reconsider your basic thinking. Be ready to burn the wall-size Gantt charts, and step out into planning and replanning on the basis of need. You have to trust:

- That you will get to your target faster, by accepting that you don’t know exactly how to get there.
- That you can and will utilise whatever you learn on your way, and that it will lead you to a better result.

Without this trust, you can implement some ISD techniques and with a reasonable result, but you won't see the full benefit.

Implementing

There is no winner or looser among the methods supporting the ISD paradigm. It all depends on your business situation. You need the directions from your organisational software development strategy to guide you. In the strategy you should find values that are important for your business, and quality targets that need to be met to achieve the company goals. If no strategy exist, then start to make one. Then:

- go through the principles described in this report, and determine which will support your strategy most
- check the methods for the best fit, and carefully design your processes to support your business using the principles as guidelines
- plan the deployment with great respect for the fact that what you are about to change are the beliefs of real people, both as professionals and as humans, and the way they have been used to work for many years. Most will find this exercise by far the most problematic

Remember that it is your and your colleague's competencies that determine the degree of success. No matter how full of endless wisdom the books are, they will never save your world more than a cookbook will cook your food. A good chef can make wonders out of simple ingredients, but a poor one can destroy the best ingredients. Be aware of your responsibility. Then go ahead and develop your organisation:

- in small steps
- always with feedback
- the most important first
- in empowered teams
- involving the customers

… but if you have the spirit, there really is no need to mention these last bullet points.

Reader's Guide

Introduction

This report will introduce you to the many aspects of iterative software development (ISD).

The metaphor that we think best describes our intentions for the report is: A Traveller's Guide. We have had this metaphor in mind in our planning and layout of the report. We hope that you will find many interesting places to visit in the report.

A Traveller's Guide

What characterises a traveller's guide? It is generally not read from end-to-end. Rather the reader will look for interesting topics to explore, which will lead to reading of other parts, gradually expanding the reader's knowledge of the area.

The chapters found in most traveller guides, comprise the following subjects:

- Introduction to the area
- The history
- The people and the culture
- Tours in the area
- Description of places and sights
- Practical hints and tips
- Commonly used phrases
- References to more information

A Guide to Iterative Software Development

What would characterise a traveller's guide to iterative software development? It should tell the reader about the following topics:

- Introduction – what is it
- The history – where does it come from
- Its purpose – why, when, and when not to use it
- Principles – what are the characteristics
- Methods – what frameworks exist
- Related activities – what is important to know
- Experiences – what have others done
- References to more information – where to find more detail

Structure of the Report

The structure of the report follows the metaphor of a traveller's guide, meaning that most of the sections can be read independently of each other. E.g. the reader can read about only one method or case study.

And if the reader chooses to compare methods or cases, the sections are structured in the same way, so that it is easy to find the same type of information across subjects in the report.

Level of Detail

We have deliberately chosen not to include a lot of the detailed descriptions, which can be found elsewhere e.g. in the reference literature. Our intent has been to describe the characteristics and separating practices of the most prominent methods (frameworks) and related activities in the area of iterative development.

Our focus has also been on the practical experiences that companies and people have had when using these. We expect that most readers will know about the experiences from abroad, which are referenced in papers and books. We have therefore concentrated on Danish experiences and cases, which are not very well known, and which would be particularly relevant to a Danish audience.

Principles

We have defined a number of principles which seem characteristic to iterative software development (ISD). They have been extracted from the literature we have studied, or based on our own experience. To our knowledge they cannot be found in other text-books, they carry no other authority than ours, we don't proclaim these to be a comprehensive or orthogonal set of characteristic principles of ISD, and we don't intend any prioritisation in the following list:

- Adaptability
- Time has priority over functionality
- Most bang for the buck
- Power to the customer
- Empowered teams
- Feedback
- Short cycles

Each of the principles will be elaborated in separate sections later in the report.

We have defined these principles in order to be able to demonstrate to what extent the iterative methods and practical experiences with iterative development comprise these characteristics. By relating to this common set of principles in our descriptions we aim to improve the structure of the information in this report.

Our Reader

The audiences of this report are:

- Primarily developers, project managers, and line managers in development.
- Secondarily method/architecture groups and process/quality departments.

This means that we have focused on practical advice on how to decide on and use the characteristics and activities in iterative development, rather than on the introduction and implementation of these in an organisation.
We expect to give our audience the basis to:

- Understand and evaluate when iterative development can be used instead of conventional waterfall models.
- Choose among well-known iterative development methods and customise those that are best suited in their project/organisation.

Start Your Travel Now

Pick your subject, get on the way, and enjoy yourself.

We will always be happy to hear about your adventures and experiences with iterative development. Just contact us at [DELTA].

Clarification of Terminology

ISD

Throughout this report we shall use the acronym ISD for the term: Iterative Software Development.

Iteration or Increment?

There is much confusion in the industry as to whether the basic element in evolutionary development is called an “iteration” or an “increment”. In fact some use the two terms in combination e.g: iterative incremental development, which seems to indicate that they don’t know what is meant by these terms.

The following definitions by Goldberg and Rubin [Goldberg e.a.] seem to explain the difference:

- Iteration: The controlled reworking of part of a system to remove mistakes or make improvements.
- Increment: Making progress in small steps to get early tangible results.

As the former term is more generic, we have chosen in this report to use “iteration” throughout, unless an increment is specifically intended. This allows us to use the same term for activities leading to a throw-away prototype as well as a full production product release.

Evolutionary or Iterative Model?

Today the most common term for the alternative to the waterfall model is called an iterative model. However, when this approach was first defined, the term evolutionary model was more frequent.

The term "evolutionary" carries a nice flavour of progress in it. However, it also gives the impression that all iterations are increments. As we have chosen to use the term iteration (see above), we have kept the same term when we talk about the model. The only exception is in the history section, where we use the term evolutionary, which was common at that time.

Customer or User?

Throughout this report we shall use the term "customer" to signify the person, who is affected by the developed software product or system. We could have distinguished between a "customer", who buys or commits the product, and a "user", who uses the product in his/her daily work, but we have found it difficult to maintain this distinction in the text.

Product or System?

Some refer to the result of a development effort as a "product." Others refer to it as a "system." The former is typical of companies that produce concrete products like embedded software, shrink-wrap software, and COTS software (Custom off the shelf). The term "system" is mostly used by companies that develop IT systems for their use in-house. In this report we have not decided on a specific term, so you will see both the term "product" and "system" used whenever we refer to the result of a development effort.

Short History of Iterative Software Development

Introduction

Software projects have always been late and over budget. Consequently, management has been searching for ways to plan and control the development. Many types of software development models have been tried, often with limited success.

Most of the models originate from the quality movement of the mass production industry, characterised by repetitive processes and entry and exit controls. Exemplified through the ISO 9000 standards [DS/EN ISO 9001].

Software development, on the other hand, has always exhibited signs of being a one-of-a-kind activity more characteristic of crafting, or even a form of creative activity. At its worst, pure guesswork and hacking.

Combining these contrary viewpoints into one model, which fits all, has not been achieved yet.

Basics of Software Development Models

The reason for using a software development model is to produce better quality software faster. That goal is equal for all models. However the means turn out to be very different.

Barry Boehm [Boehm] states that the primary functions of a software development model are to determine the order of the stages involved, and to establish the transition criteria for progressing from one stage to the next. These transitions include completion criteria for the current stage plus entrance criteria for the next stage.

Development models are important because they provide guidance on the order (phases, increments, prototypes, validation tasks etc.) in which a project should carry out its major tasks, and define the conditions for progressing to the next stage.

Many software projects have experienced serious problems because they pursued their various development activities without proper regard for the stages and transition criteria. And naturally, using any model is better than using no model.

A number of software development models have been employed throughout the industry. However, they can usually be grouped according to one of the following basic models: the waterfall model, or the evolutionary model.

Waterfall Models

The by far most popular model is the waterfall model, which is characterized by confining feedback loops to successive stages in order to avoid expensive rework over many stages.

The assumptions for waterfall model are:

- The customer knows what he wants
- The requirements are frozen (changes are exceptions).
- Phase reviews are used as control and feedback points

The characteristics of a successful waterfall project are:

- Stable requirements

- Stable environments
- Focus is on the big picture
- One, monolithic delivery

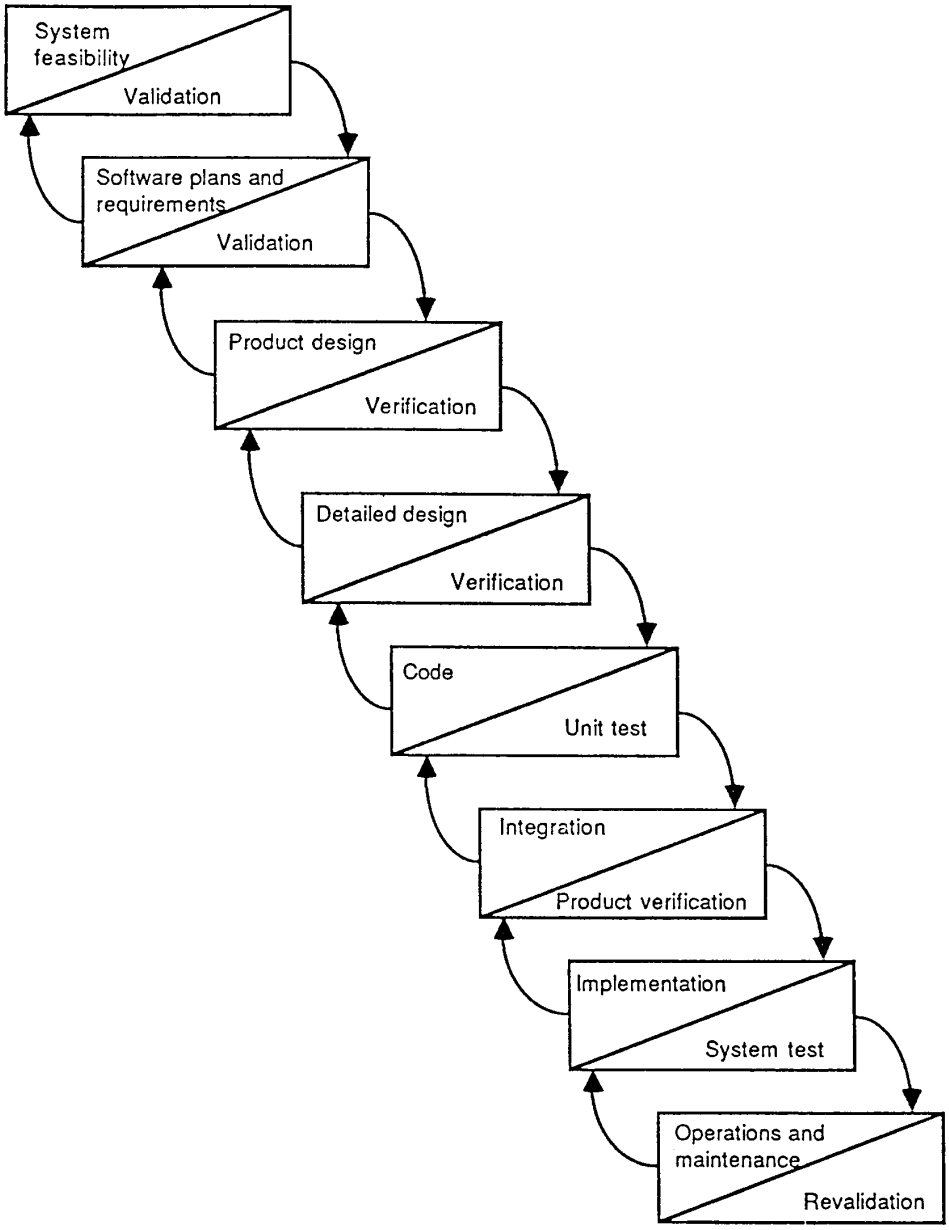

Figure 1 Stages of the Waterfall Model [Boehm]

The goals of the waterfall model are achieved by enforcing fully elaborated documents as stage completion criteria, and formal approval of these (signatures) as entry criteria for the next. For some classes of software (e.g. safety critical systems, or fixed price contracts) this may be the proper way to proceed. However, it does not work well for many classes of software, particularly interactive end-user applications where requirements must adjust to the needs of the customer.

To illustrate this issue, Hasan Savani [Savani] states that: “The waterfall model works well for salmon, not people. While trying to achieve their goal, salmon tend to move intractably upstream. When people strive for a goal they tend to meander,

sometimes venturing forward and sometimes retracting steps. While following a general course from the problem to the solution, people often look ahead (such as by prototyping or simulation) and then retreat to accommodate their current understanding of the problem to what they learned in their look ahead."

Evolutionary Models

The evolutionary model is built on a philosophy of interaction and change. It is characterised by frequent feedback loops and customer participation. The model defines its stages to consist of expanding increments of the operational software product. The directions of evolution are determined by the operational experience from these increments.

The assumptions for the evolutionary model are:

- The customer cannot express exactly what he wants
- The requirements will change.
- Reviews are done continuously for control and feedback

The characteristics of a successful evolutionary model project are:

- Fast and continuous customer feedback
- Floating targets for the product
- Focus is on most important features
- Frequent releases

The evolutionary model matches situations in which the customers say: "I can't tell you what I want, but I'll know it when I see it" (IKIWISI). Evolutionary models are also frequently called RAD models (Rapid Application Development), because they are driven by the need for rapid reactions to changes in the market.
The evolutionary model is perfectly suited for a class of applications where there is a close and direct contact with the end-user, and where requirements can only be established through actual operational experience. For applications that are sold as a product for a wider market with a diversity of customers, the constant upgrading with new increments is a problem. The model also represents a problem because there is no known limit to the number of increments, and the constant changes to the software may require special increments to stabilize the product.

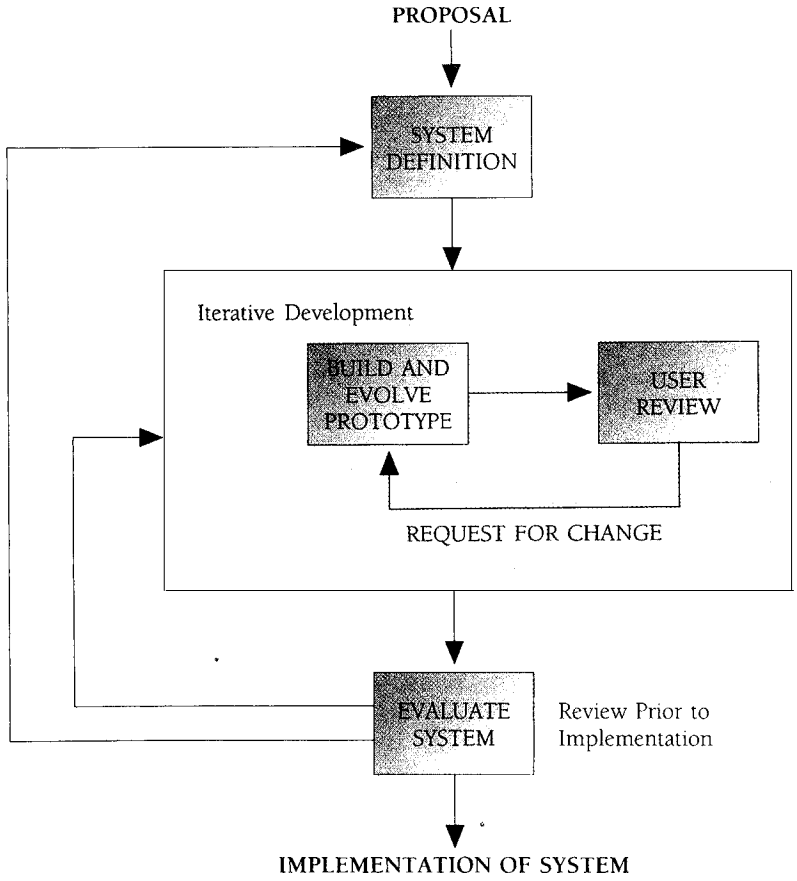

Figure 2 Stages in an Evolutionary Model [Martin]

The goals of an evolutionary model are achieved through various prototypes. These prototypes are developed and validated continuously throughout the development until a meaningful set of features are complete. This set of features (an increment) is then released and put in operation. Based on the experience gained throughout the development and use of the product, the next set of features are planned, developed, and deployed.

Combined Models

Several models have aimed at combining the stages of the waterfall model with an evolutionary approach. The two most prominent have been:

- The Spiral model by Barry Boehm
- The RAD (Rapid Application Development) model by James Martin

Both of these are presented below.

In the recent years, the term RAD has almost become synonymous with any evolutionary model.

The Spiral Model

Barry Boehm [Boehm] has defined a so-called Spiral Model, which aims at accommodating both a waterfall and an evolutionary model. It uses a set of full cy-

cles of development, which successively refines the knowledge about the future product. The order of the cycles is risk driven.
The first cycles use simulations and prototypes to evaluate alternatives and resolve risks, but conclude with reviews and approvals of fully elaborated documents before the next cycle is initiated. The last cycle when all risks have been uncovered, consists of a conventional waterfall development of the product.

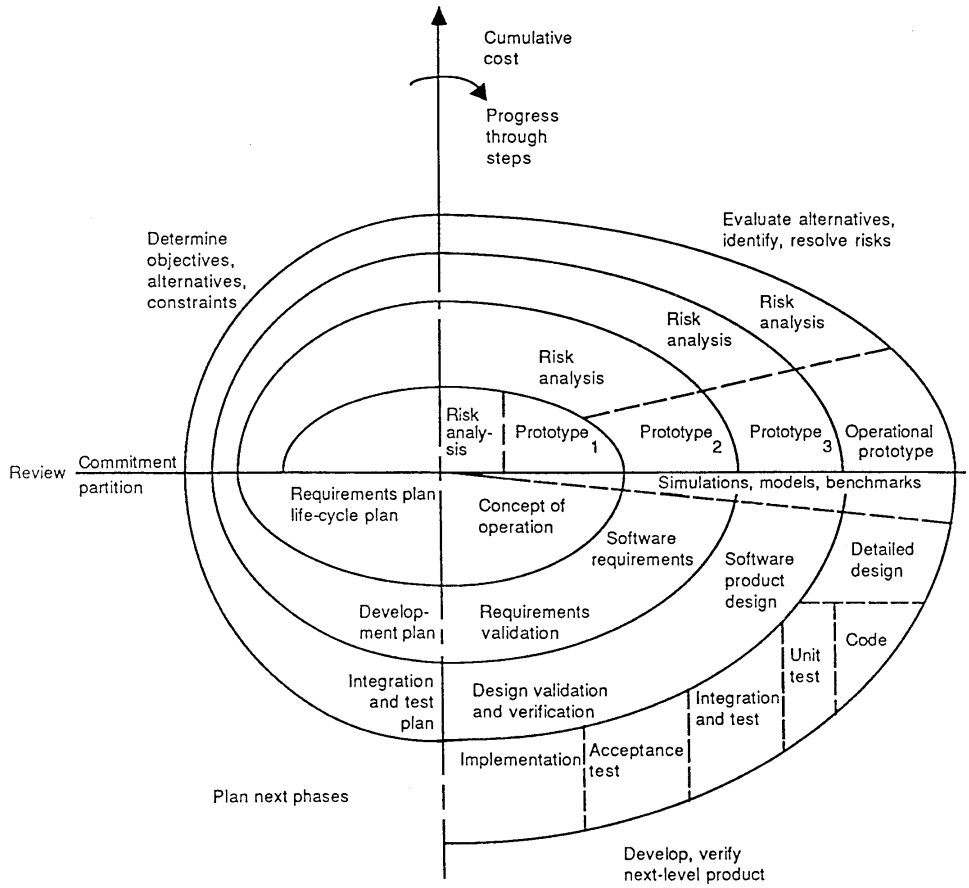

Figure 3 The Spiral Model [Boehm]

The RAD Model

James Martin [Martin] was the first to coin the term RAD (Rapid Application Development). Since then the term RAD has become a generic term for many different types of evolutionary models.
The original RAD model by James Martin is based on incremental releases on a short cycle basis determined by business priorities, and close involvement of the customer in the development process. However, the stages in each increment are basically waterfall, only much shorter.

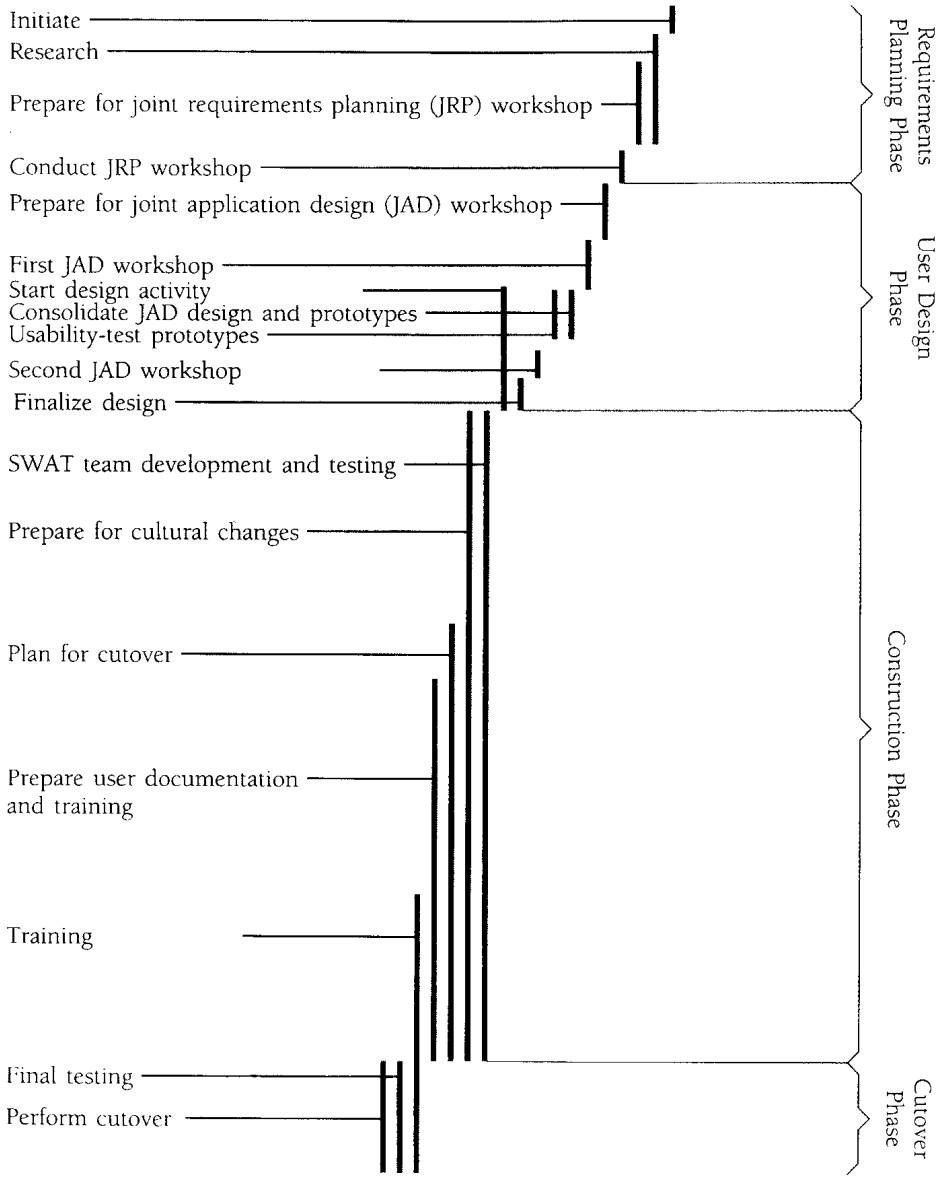

Figure 4 Phases and Activities During a RAD Lifecycle [Martin]

Agile Models

Recently a number of evolutionary models have appeared, which put emphasis on so-called values and principles, as described in the “Manifesto of Software Development” [Agile Alliance]:

- Individuals and interactions are valued over processes and tools
- Working software is valued over comprehensive documentation
- Customer collaboration is valued over contract negotiation
- Responding to change is valued over following a plan

One popular example of this is the so-called eXtreme Pogramming (XP) model. The models are based on a loosely structured small-team style of product development. The objective is to get small teams (3-8 persons) to work together to build large products relatively quickly while still allowing individual programmers and teams freedom to evolve their designs and operate nearly autonomously.
These small teams evolve features and whole products incrementally while introducing new concepts and technologies. However, because developers are free to innovate as they go along, they must synchronise frequently so product components all work together.

Principles of Iterative Software Development

Introduction

The purpose of defining a set of principles, which characterise ISD, is to help the reader to decide whether a given practice is supporting ISD or not.

Take a given practice in your organisation, and compare it to the principles. Then judge for yourself whether this practice is iterative. But much more important - judge if the practice makes business sense. Is your software development practice supporting your business?

The principles are listed as slogans or, if you like, value based guidelines. Based on literature and experience we have derived these principles. We don't expect them to be comprehensive or complete, only that they will help you as they helped us.

Contents

This chapter contains the following principles:

Time has Priority over Functionality

Floating Target

In the classic way of planning, we operate with three variables:

- Time
- Resources
- Functionality

This ISD principle is about keeping time fixed and functionality variable, for the simple reason that one of them has to be flexible, and time and resources are in reality fixed in nearly all software projects.

Time

The deadline for the next delivery is fixed. You will deliver something that works at that deadline. The complete delivery chain is dependent on this deadline, and if one party is not delivering, the process is suffering a poor performance.

The focus is changing from: "We have to complete everything into this release" to: "We have to optimise the complete delivery process, and use it to produce as fast as possible." Or "We want to maximise the benefit from the development process."

The good thing is that because of the short cycles delivering all the functionality is not as important as in a waterfall project. There is a new delivery just around the corner.

On an iterative project you may not precisely know what is in a given release. But you know for certain, that it is what the customer considers to be of most value to his business (power to the customer), and you know that it has the intended quality (short life cycles).

Resources

The resources are theoretically negotiable. In reality they are as fixed as the deadline. It might seem to be a good solution to add a new resource. But you always need to allow a certain amount of time before a new resource is productive. That time usually exceeds the iteration you are in. Add to that the "noise" it creates for the rest of the team when you add new resources and the time the project manager has to spend on different practical issues. Often this will lead to the conclusion that in reality - resources are fixed.

Functionality

This leaves only one variable left, and that is functionality. The functionality is the only real variable. Of course you plan to have a certain functionality in the next release, but if it fails, you accept to deliver less than you planned. You do not attempt to push the deadline, and adding resources does not make sense. This means that you cannot know exactly what you will deliver at a given deadline just a few iterations away. But everybody knows that a delivery is coming, and can plan accordingly, and the customer has decided what is in it, e.g. that which is of most business value to him (power to the customer).

What's New?

Just because a project is said to have a fixed deadline, it is not necessarily compliant with this principle. If it has frequent deliveries and those are met, it may comply.

A fixed deadline sounds trivial. Everybody has experienced a fixed deadline. Well, at least the project manager said, it was fixed. But in reality it turned out that the quality was not good enough, or the full functionality was not delivered. As a result the project was delayed. That means that the deadline was not fixed in reality! The functionality was fixed.

What makes ISD different from waterfall projects, is that the duration of an iteration is much shorter. From many month or even years in waterfall projects, to few weeks.

The deadline in an ISD project is predetermined to be e.g. every second week or maybe every second month based on a fixed delivery cycle. As opposed to the deadlines in waterfall projects, which are often set by a market requirement, and not as a result of serious engineering estimates based on reality.

The principle works for the "final deadline" as well. Many small iterations, with a floating target, adds up to one big. You have two choices.

- If you want to have full functionality, you have to accept that you don’t know at what time it will be finished.
- If you want functionality delivered at a fixed point in time, you don’t know how much functionality you will get. But what you do get will be the most important for you, and it will be working.

Both may sound unacceptable. However, it has always been true! This ISD principle covers this reality.

Quote

"You can't stop the clock."

Get the Most Bang for the Buck

First Things First!

People involved in an accident will be more satisfied, if the doctor fixes their open chest wound before the broken fingernail. Customers feel the same way about the order in which you deliver the features to them. Most likely they expect all their requirements to be met, but if not, there are certainly some issues that are the most important.

Whatever you deliver first, should be what the customer believes to be the most important to his business. If you do that, you can stop the project anytime and leave the customer with a useful working system reflecting the investment to date.

This way you will find the customer more committed, because he can see the progress, and the progress is according to his needs.

Quote

"Things which matter the most must never be at the mercy of things which matter least" - Goethe

Power to the Customer

Overview

This principle is about the sharing of power and responsibility.

In ISD, the customer is very strong, and has full visible responsibility for his own business. The representatives for the customer must prioritise what comes first in the next iterations of the project. If the customer wants the development to go in a different direction, he can do that because he is responsible for his own business result.

Responsibilities

The software development people are responsible for:

- the efficiency of the development
- pointing out technical risks and consequences
- producing the quality as the customer specifies

They are responsible for the transformation of information from the customer to a software system that the customer can use.

The customer is responsible for delivering the information the development team needs.

He is also responsible for presenting the result to other customers/users, validating that it supports the business, and bringing the feedback from these activities into the project.

Prioritisation

Prioritisation of tasks may become a joint responsibility between the developers and the customer. Normally it is the customer who drives this, but there can be two conflicting interests.

- The customer may require a feature, that needs one month of architectural work. But he wants the feature next week.
- The development team wants to spend the next month on architectural stuff, that the customer does not see the immediate benefit from.

Since this is the classic situation where business reality does not match engineering reality, there has to be some decision guidelines.

First priority is the customer's immediate requirement. If that is technically possible the project will do that.

If it is not doable because it needs some architectural work, the developers and the customer have to decide together if the customer will pick another feature or the development team should spend time on the architecture to prepare for the customer's immediate requirement.

Fixed Scope and Price

If the scope and price of the project is fixed, this principle cannot be applied fully, as the development team has an additional conflicting responsibility: To ensure that the scope is completed within the specified budget. This limits the power of the customer to accept or reject specified milestones. Still, most of the responsibilities listed above can be applied to good advantage.

Quote "The customer is always right "

Empowered Teams

Empowered Teams

An empowered team is characterised by having a clear target and being able to make all needed decisions by themselves inside a given framework.

- A good example is the team on a sailing boat during a regatta. The target is to win. The framework is set by the conditions of the sea and the racing rules. Inside the framework the team makes all decisions, and is responsible for the end result.
- A professional volleyball team is less empowered. They are not able to make a big decision without consulting the coach during a timeout (steering committee meeting), and the team is in general expecting guidance from the coach during the game.

Empowered teams are important in ISD because decisions need to be taken very fast in a fast changing environment.

- On a racing course you need to make a decision e.g. every time the wind changes just a little bit. There are many factors out of the team's control they have to react upon. That is why they have to be empowered, and not depending on anyone.
- In volleyball the environment is stable. There are just the two teams that concentrate on the game. If any change that is outside the team's control should happen, the coach and the officials are supposed to handle it. That is why they don’t need to be empowered the same way as the sailing team.

Decision Speed

To ensure fast decision making, authority and responsibility must go hand in hand. First of all, the team has to be staffed to make the decisions. If not this will have a delaying effect on the release.

Proper staffing includes:

- Customer representatives must be present to make decisions on behalf of the business.
- Developers must be present to make decisions about technical issues and task estimates.
- Responsibility for test and quality assurance must be allocated to ensure that the products and processes fulfil the quality criteria.
- Production/implementation representatives must be present to bring every release to the target audience.
- Logistics representatives must be present to handle supply channel and product support issues

The management above the project obviously has the go/no-go power at predefined milestones. But between two milestones, the team rules. When you are making e.g. deliveries every second week, it is not possible to wait two months for a decision at the next steering committee meeting. Neither can you wait two months for a delivery to go into production just because that is the "next available timeslot"!

Balance Between Political and Creative Work

This principle deals with the balance of political work and creative work. A certain amount of political work is always needed, but the tendency is to look more on how much value a creative process is adding to the product, than to how much value a political process is adding to the product.

There is no upper limit on the time it can take to make a given decision. The duration depends on the culture of the company. In some organisations a developer will just decide in a given situation, and if it turns out to be wrong, it will be changed without a big hassle. In other organisations a similar decision may require the signature from the CEO, and if it turns out to be wrong, a big blame-storming process take place and it may be more difficult and expensive to change.

ISD requires the best possible set-up regarding decision speed. To meet this requirement, you need truly empowered teams. You also need a supporting management that actively builds the needed environment around the project.

Quote

"A little practice beats a lot of steering committee meetings"

Adaptability

Adaptability

The adaptability principle in ISD is about learning as the project progresses. It is the principle of Evolution as opposed to Definition (which leads to revolution).

The adaptability principle deals with the fact that your customer rarely knows what he wants in the first place, and that the market changes while you develop your product.

If your practices support this principle, you will effectively learn together with your customer as the project progresses. This way adaptability addresses the uncertainty, which is present in most development projects.

In ISD, changes are the reality, stability is the exception. When changes come frequently, your ability to absorb the changes become a success factor.

This means that you should acknowledge that not knowing everything from the start is not bad. Assuming that you do is!

Why build a project on an assumption of a truth, which is not valid!

Learning

Learning is an important aspect of adaptability. The learning situation can be quite different depending on how you approach the situation.

If you are willing to accept that you don't know exactly where you are going, then you are most likely willing to learn. If you are willing to learn, it is smart to design your development process in a way that you are able to absorb and utilise whatever you learn. If you learn and react, you adapt to the changing environment.

On the other hand, why bother about learning, if you have made a specification that you believe is true? The specification then becomes the law, instead of reality. And when project managers spend time making the reality look like the specification they are not adapting to a changing environment.

Quote

"If the map and the landscape disagree, trust the landscape!"

Winston Churchill

Short Life Cycles

Short Life Cycles

Short life cycles are used in order to gain experience faster and reduce risk.

Experience Capability

If you participate in a waterfall project that lasts two years, you will only practice each of the development disciplines once, and you will only get the life cycle experience once. This means that if you learn something smart about e.g. how to migrate to the production environment, you have to wait two years to use that experience again. And it does not help you that you learned the same thing from all the four thousand objects that you had to migrate.

If you join an ISD project with e.g. biweekly deliveries, you will experience the full life cycle fifty times in two years. After a few iterations, you will have optimised your migration process, and master the discipline. The same goes for all the other disciplines that are inevitable parts of a professional software development environment.

If you were boiling potatoes, this principle would mean that you would take each potato, peel it, boil it and eat it. Obviously a bad choice, since you master the potato boiling skills. Software development is very different. And regarding e.g. migration skills, chances are that a developer with ten years experience may have done it five times. And experience that is maybe eight years old is not worth much today because the environment has changed significantly.

Risk

Every new design or component represents a possible risk. If all components are taken to the target environment as soon as possible, you will learn as much as possible about the component and the environment, as early as possible. This will minimise the risk of defects in each component and therefore significantly reduce the risk for the project.

The alternative may seem familiar to some. The test phase is scheduled late in the project, and during the test phase, the number of defects turn out to be somewhat surprising, and a threat to the project.

If you where asked to cook French truffles to a party of eight (cost of truffles approx. $1000) instead of potatoes, would you cook them all at once just before the main course, or test the cooking process on a small sample first?

Quote

"Deal with small problems before they become big."

Fast Feedback

Fast Feedback

Fast feedback is applied at many levels with the main purpose of reducing risk and identifying achievements.

Risk Management

The earlier you learn how your development work affects the customers' business, the better.

The sooner your customer either acknowledges or rejects your creative work, the better.

The more frequently you test your ideas, the smaller is the risk you run. If you work for one week on a design, and then ask the customer if he likes it or not, your stake is only one week's work. If you plan to deliver the complete system one year from now, your stake is one year’s work.

Hence - the fast feedback principle in ISD is a risk management technique.

Levels

This principle operates at all levels.

- At developer level, you test all new code as fast as possible to see how the environment reacts.
- At feature level you test new features with the customer to see if it meets his expectations.
- At system level you test whether the system can operate in the expected environment.
- At product level you test whether it improves the customers business.

Always utilise the fast feedback principle, and establish a dialogue with the interested parties as early as possible, to get their feedback.

Identifying Achievements

Fast feedback is an essential mechanism for measuring progress, enhancing the satisfaction of the development team and allowing the project to move on.

- Fast feedback from a test will allow the developers to identify whether their code works.
- Fast feedback from the customer will allow the project to identify whether a feature is completed.
- Fast feedback from the customer will ensure that the project is not hung up on decisions regarding required changes.

Customer Adaptation

A system becomes useful, only when it improves the customer’s business. Seen from a business development point of view, the new system is just one brick in the wall. Another brick is how the customers adapt the system. Fast feedback ensures that the user community gets used to the new system in small bits, and that they are not overwhelmed by a completely new system. They need time to digest a new

system, and they will get it if your development practices support the Fast Feedback principle.

Quotes

"It is better to light a candle, than to curse the dark."

Methods of Iterative Software Development

Introduction

This chapter contains a short description of four commonly used iterative methods. Practical use of these methods is documented in the Case story chapter.

Other methods may be equally relevant for your specific situation. Comparing methods is very relevant, but only if the purpose is to determine what will help you the most.

Contents

This chapter contains the following topics, which can be found in the full report:

Related Aspects

Introduction

There are a number of aspects and processes that are specifically related to iterative development, or that must be dealt with differently when iterative development is adopted.

This chapter describes, for the most important of these topics, what you should be specifically aware of when you are using iterative development.

Contents

This chapter contains the following topics, which can be found in the full report:

Quality and Maturity Models

Introduction

What will happen to your maturity level or quality certificate if you switch to iterative software development? Are there specific issues that need attention?

This chapter is intended to answer that question for the most common quality and maturity models.

Contents

This chapter contains the following topics, which can be found in the full report:

Case Studies

Introduction

This chapter contains a number of case studies from the Danish industry and public sector.

The case studies are of a different nature. Some are project cases and some relate experiences of working with iterative development in general.

Contents

This chapter contains the following topics, which can be found in the full report:

Datateknisk Forum

The Association

Datateknisk Forum (www.datatekniskforum.dk) is an association of electronics and software companies, who develop and strengthen the many possibilities of information technology through a focused cooperation.

The members are businesses, who develop and exploit information technologies as essential elements in their production or service, ranging from micro electronics to telecommunications, from electronic device manufacturing to administrative data processing.

The activities of Datateknisk Forum are focused towards three main areas

- Exchange of experiences in ERFA-groups
- Definition and execution of projects in state-of-the-art fields.
- Cooperation with major industry support programs in Denmark and abroad.

Membership of Datateknisk Forum provides the employees of a company with the opportunity to stay up-to-date on a high technical level in untraditional and efficient manner.

Datateknisk Forum forms the framework of a technology community, where the technicians and leaders of the member businesses meet to discuss their own development as well as their mutual development.

ERFA-groups

One of the most important tasks of Datateknisk Forum is to provide a framework, within which the members can gather the inspiration and knowledge needed to progress their software development and information technology.

Datateknisk Forum continually forms new ERFA-groups, responding to the demands of the members.

The groups usually have 15-25 member companies. At the meetings, the participants exchange knowledge and experiences within the field covered by the group and discuss relevant problems. This happens in an informal atmosphere and often takes the form of case stories.

Many companies find that experience exchange groups are the most efficient and quickest way to keep up-to-date within a specific field. The ERFA groups of Datateknisk Forum are very popular.

Right now there are ERFA groups within the following fields:

- Software Quality Assurance and Quality Management.
- Object Oriented Methods.
- Software Project Management in Practice
- Software Process Improvement.
- User Centred Design and Development
- Software Test and Test Management.

- Communication Technology.
- Pervasive Computing – Devices on the Internet
- Internet applications
- eXtreme Programming (XP)

Each ERFA-group is managed by an experienced consultant from DELTA.

Reports

Datateknisk Forum defines and commissions DELTA to execute projects. The scope of each project is determined by the members through a survey, asking them in which fields they wish to improve their knowledge. This report is the result of such a project.

It is essential that these projects provide new and relevant information to the members, and that the knowledge gained can be used effectively.

The reports are published both on paper and on the internet. The internet version is available to the members on Datateknisk Forum's website.

These projects are anchored within the association through the following means:

- Each project is associated with an ERFA-group, in order to provide ongoing review and discussion between the project group and the ERFA-group participants.
- The published report is distributed to all member companies.
- A one day seminar concerning the theme of the report is arranged.

Contact

Datateknisk Forum (www.datatekniskforum.dk) has an independent, elected board with participation from Danish Industry. A yearly general assembly is held in November.

The secretariat of Datateknisk Forum is located at:
DELTA, Venlighedsvej 4, 2970 Hørsholm. Tlf.: +45 45 86 77 22.

Author Biographies

Morten Korsaa

Morten Korsaa has 11 years of SW development experience from development, project management to global management, including three years at Nokia Mobile Phones as Global Software Process Improvement manager, responsible for all SPI activities for 16 sites in 9 countries.

Today, Morten is supporting software companies to optimise their development capacity, quality and efficiency. In his toolbox he carries all the well-known models/methods/techniques and especially the knowledge of which will be useful for whom, and how they can be implemented in the daily work to support the business.

More resources can be found on http://www.korsaa.com/ and on http://www.it-sirius.dk/ for consultancy services.

Robert Olesen

Robert Olesen is cand. scient. in statistics and has 17 years experience within the IT field.

Robert has worked with technical software – first as developer and project manager and later as quality manager. This has given him a wide range of experiences within software development: Specification, analysis, design, programming, test, support and software process improvement.

Robert is a BOOTSTRAP assessor and his main field within DELTA besides assessments is software process improvement.

Otto Vinter

Otto Vinter (otv@delta.dk) is a project manager with DELTA specialising in software process improvements. Previously, he was responsible for software process improvements at Brüel & Kjær. He has been the driving force in that company's improvement activities in testing, requirements engineering, development models, and configuration management. He has managed software development projects for more than 30 years. He received his Masters Degree in Computer Science from the Danish Technical University in 1968.

He is a regular speaker at conferences and seminars, performs mentoring activities for clients, and is an expert evaluator on the framework programmes of the CEC.

More information on his software process improvement results can be found at: http://inet.uni2.dk/~vinter.

References

[Agile Alliance] www.agilealliance.org
"Manifesto of Software Development"

[Boehm] B.W. Boehm, "A Spiral Model of Software Development and Enhancement", *IEEE Computer*, pp. 61-72, May 1988.

[Datateknisk Forum] www.datatekniskforum.dk
The assosiation that requested and financed this report.

[DELTA] www.delta.dk/iterativ
Contains the web contact point at DELTA for this report and its authors.

[DS/EN ISO 9001] DS/EN ISO 9001, 3rd Edition, 2000-12-21
Dansk Standard, Kollegievej 6, 2920 Charlottenlund,
The home page for Dansk Standard is: www.ds.dk

[Goldberg e.a.] A. Goldberg and K.S. Rubin, *Succeeding with Objects*, Addison-Wesley, 1995.

[Martin] J. Martin, *Rapid Application Development*, Maxwell Macmillan International Edition, 1991.

[Savani] H. Savani, "IEEE Software Manager Exchange", *IEEE Software*, page 108, July 1989.