Shifting the Risk

Reflections about the different risk strategies in the sequential (waterfall) and iterative software development approach

Author: DI. Andreas Nehfort

Area: Engineering Process Philosophies

Abstract

The main difference between the sequential approach (waterfall, V-Modell[1]) and the iterative software development approach (like RUP[2], XP[3] or MSF process model[4]) is not primarily based on the different sequence of development activities, it is based on fundamentally different risk management strategies.

This fact is not figured out very clearly in most of the software process literature, so many IT professionals are not fully aware of it. Consequently the chance is very high that SW development projects which want to apply a modern SW development approach will fall into this trap and take the disadvantages from both sides – the sequential and the iterative approach.

My reflections on the different risk strategies in the sequential (waterfall) and iterative approach shall help to find a clear position between these two leading software development paradigms.

Keywords

Software process models, V-Model, iterative software development, risk management

---

[1] V-Model proposed by Barry W. Boehm [1] and actually represented e.g. in the V-Model 97 [2]

[2] RUP – Rational Unified Process [3], [4]

[3] eXtreme Programming [5], [6]

[4] MSF – Microsoft Solution Framework

1 Motivation

In the last decade the iterative software development approach has become more and more popular. Consequently it more and more displaces the sequential V-Model approach.

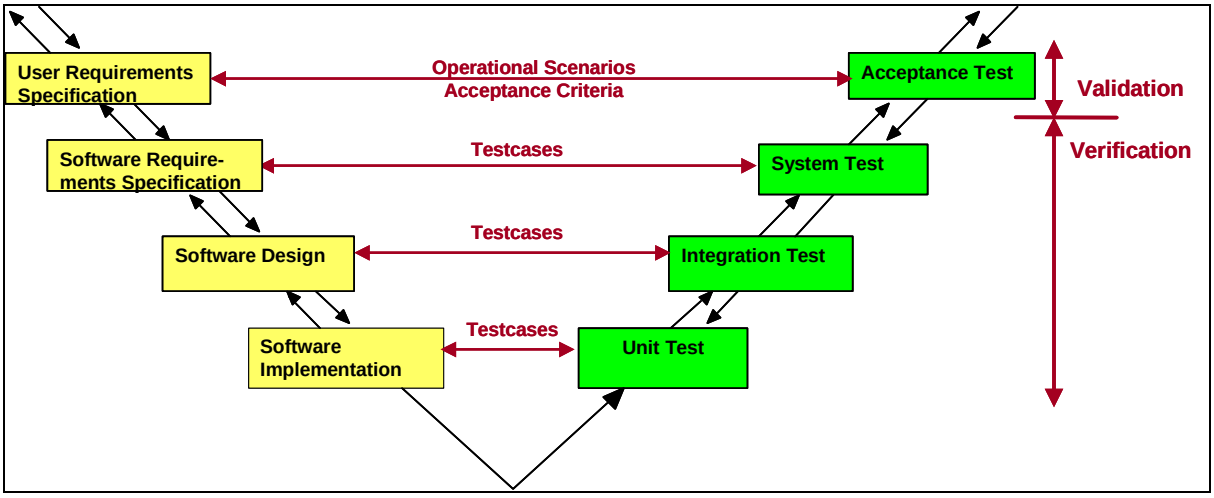

Figure 1: The sequential SW development approach represented by the V-Model

A lot of iterative Software development models have been established. Besides of all differences these models have in common:

- short iteration cycles which shall produce new versions of software with increasing functionality.
- a simplified phase model for each iteration.

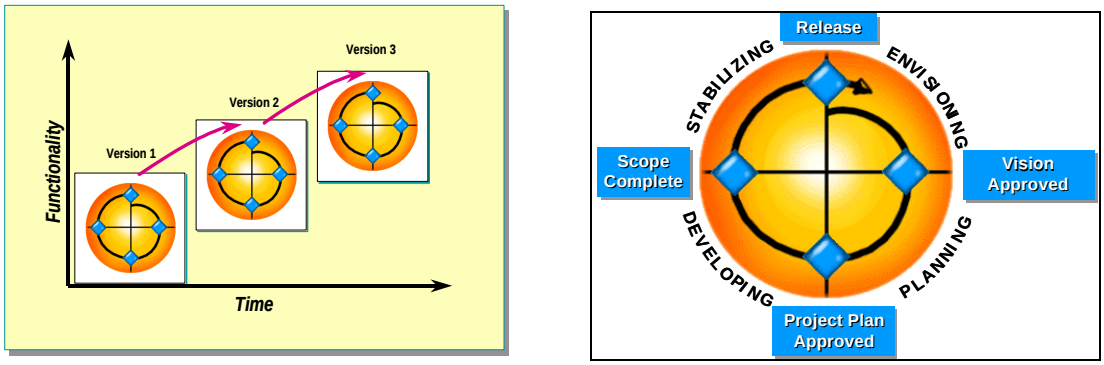

Figure 2: The iterative SW development approach represented by some MSF-charts

During my consulting work I have found out, that among many IT professionals there is a substantial lack of knowledge what iterative software development really means. So many followers of the iterative approach act quite similar compared to software engineers which follow the sequential approach with an incremental implementation.

The following dialogues are authentic[5] and shall illustrate a typical situation:

Mr. Nehfort (asks)	**Mr. Meyer (Proponent of the V- Model):**
What do you think about sequential versus iterative SW- development?	I am strictly following the V-Model!
Why?	Without clear requirements in the beginning you cannot expect a reasonable solution and there is no chance for a fixed price tender!
How do you deal with the fact, that many customers have problems to specify their requirements in an early phase?	Well, the customer has to decide what he really needs; of course we help him to make a methodical analysis and an orderly requirements specification
But there will still be a requirements creep	Therefore we implement a professional change management. We also recommend our customers to save 20% of the budget for changes after requirements freeze.
That means: In the analysis phase you typically fix about 80% of the requirements and during the project work you define or redefine the rest?	Yes, that has proved in many projects.

Mr. Nehfort (asks)	**Mr. Smith (Proponent of the RUP):**
What do you think about sequential versus iterative SW-development?	We have recently established the RUP in our SW development!
Why?	Because it fits better to our situation!
Following the RUP you do not specify all requirements in an early stage. How do you deal with requirements?	Well, we define about 80% of the requirements in the inception phase and about 20% later.

Wow! - Two different approaches – The same outcome!

Note:

Requirements engineering and requirements management have split up the term “requirement” into different classes like “user requirements”, “system requirements” “software requirements”, “functional” an “non-functional requirements”, “required behavior”, “required restrictions” and so on. In this paper I will subsume all the different kinds of requirements which influence the final solution under the term “requirement”.

[5] I have just renamed Mr. Meyer and Mr. Smith

From my experience it seems that IT professionals interpret the V-model as an idealized model and iterative SW development as a practical approach which allows them to work as they did before: with a primarily sequential approach BUT less rigid, less straight, less orderly, less disciplined → to implement incrementally those requirements which they have more or less precisely defined in the analysis phase.

This is a fundamental misunderstanding of the iterative approach!

Ivar Jacobsen, one of the RUP originators, recommends that in the inception phase (first phase of the RUP) about 50% of the requirements shall be established (these 50% should include 90% of the so called key requirements), following an agile approach like XP it is even less!

This paper shall help to find a clear position between these both leading software development paradigms. It shall clarify the different positions regarding the risk strategies in the sequential (waterfall) and iterative approach, which differs substancially regarding the handling of the following risks:

- The risk to develop the wrong product (inadequate functionality & behavior).
- The risk to develop the product wrong (inadequate design / technical faults).

2 The main difference

2.1 The risk management strategy of the sequential approach

The sequential SW development approach (waterfall, V-model) is based on the following consideration:

> "If we know all the requirements in an early stage of the project, there is a pretty good chance to develop the software right – in other words: Our risk to make technical faults during the development (SW developers call them bugs) can be minimized."

The risk management strategy behind this consideration can be called:

"Freeze requirements first"

This is the dominating motivation for the sequential SW development approach!

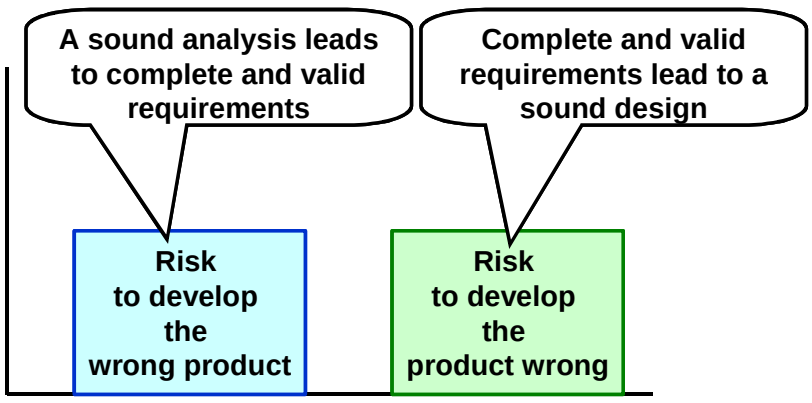

Figure 3: The intended risk characteristic of the sequential SW development approach: Low risks in both dimensions!

2.2 The risk management strategy of the iterative approach

One problem of the “freeze requirements first” strategy is the following:

> The more a SW application shall be innovative or an application field evolves dynamically, the less we know about the required functionality and behavior in an early stage of the project – if we try to fix them early, they become moving targets.

The iterative SW development approach (RUP, XP, …) tries to find a solution for this dilemma based on the following consideration:

> “If we know that we cannot fix the requirements in an early stage of the project, we should postpone detailed requirements specification to make our decisions very late. If we don’t know much about the required functionality and behavior, we should try to test implemented features early against the customers needs. With a short feedback loop between decision and practical prove we have a pretty good chance to build the right software - in other words: fixing the requirements very late can help us to minimize the risk of building the wrong software or system.”
>
> “We know that this increases our risk to make technical faults during the development, but we can handle this risk much better than years ago when the sequential SW development model was proposed.”

The progress of the last 20 years in SW architectures, SW development methods and tools, SW components and so on, has decreased our technical risks and their impact dramatically.

The risk management strategy behind this consideration can be called:

“Decide late - try out early”;

This is the dominating motivation for the iterative SW development approach!

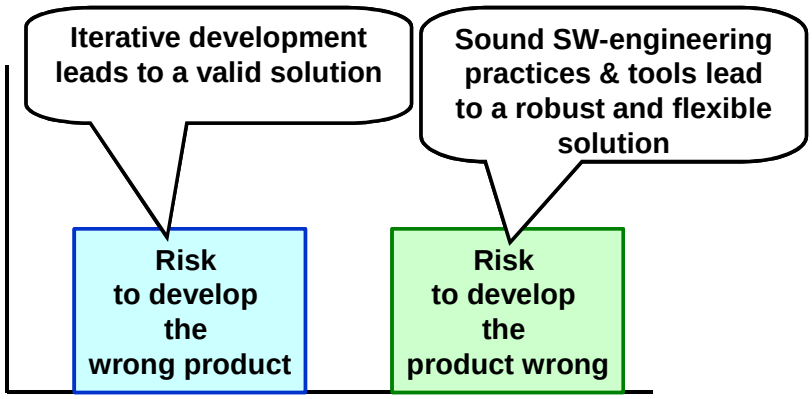

Figure 4: The intended risk characteristic of the iterative SW development approach: Low risks in both dimensions!

3 The Consequences

3.1 Some consequences of “freeze requirements first”

“Freeze requirements first” has been the dominating approach from about 1975 up to about 1995.

Our experience of the last 30 years has demonstrated that this approach was not as successful as we expected: Even if we have built the software right, in many cases we did not develop the right software:

- Our customer’s understanding about their requirements has changed.
- The business has evolved during the project.
- Consequently the software requirements have changed.

So in many projects the intended risk characteristic of the sequential SW development approach shifts:

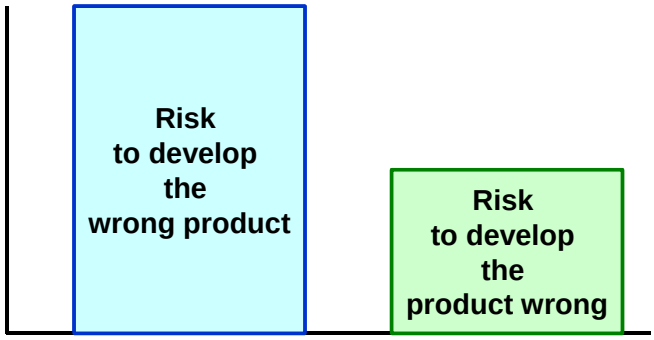

Figure 5: Typical risk characteristic of the sequential approach: High risk to develop the wrong product!

The Standish Groups Chaos Reports [7] and [8] demonstrate impressively, that SW projects are not that successful:

	Projects succeeded[6]	Projects challenged[7]	Projects failed[8]
1994 [7]	16%	53%	31%
2000 [7]	28%	49%	23%
2004 [8]	29%	53%	18%

In 1994 the “Projects challenged” had an average cost overrun of +189% and have implemented about 61% of the features & functions initially specified.

In 2000 the “Projects challenged” had an average cost overrun of +45% and have implemented about 67% of the features & functions initially specified.

---

[6] Project succeeded:
The project is completed on time, on budget with all features & functions initially specified.

[7] Project challenged:
The project is completed and operational, but over budget, over the time estimate and offers fewer features & functions than initially specified.

[8] Project: failed: The project has been cancelled at some point during the development cycle.

That is a reasonable improvement; nevertheless two of three projects have a cost per feature overrun of more than a factor of 2 or have been cancelled (did not deliver any valuable software).

These projects do not develop the right software – at least not in the first attempt!

The risk to build the wrong software can be divided into the following main categories:

- We did not have the right requirements (we have frozen invalid requirements).
- We have misunderstood the requirements.

But also “developing the product wrong” may lead in a wrong product, if we have made wrong design decisions based on requirements, which were invalid or have been misunderstood (but exactly this risk we wanted to minimize by the “freeze requirements first” approach).

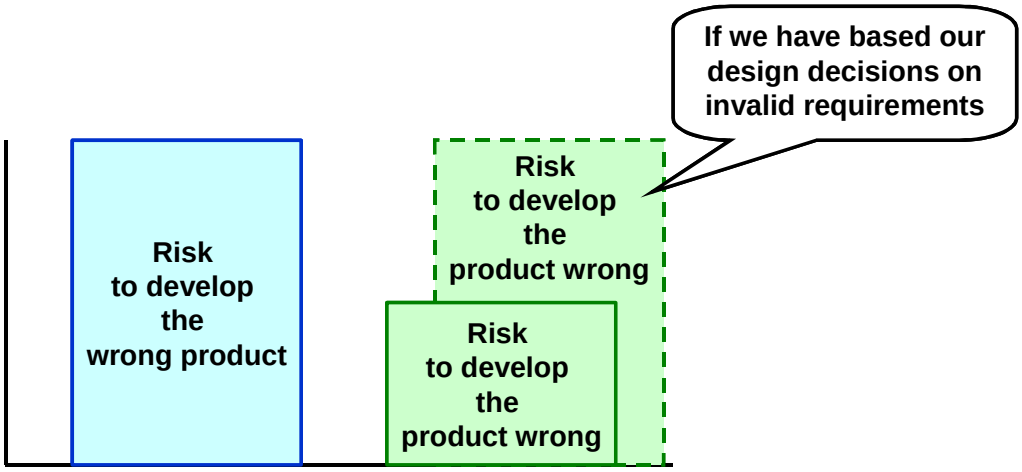

Figure 6: Risk characteristic of the sequential approach, if we have frozen invalid requirements

The SW project managers have found a way to handle their risk:

> “If we can force our customers to freeze the requirements in an early stage of the project, we can transfer the risk having the wrong requirements to our customer!”

A whole generation of SW methodologists and project managers has forced this approach against the customers resistance and at last they succeeded. Our customers have accepted the “freeze requirements first” paradigm.

By the way: This is a proven implementation of a risk management strategy which you can e.g. find in the MSF - Microsoft Solution Frameworks risk management discipline: “Change the consequences of the risk” → “If you cannot avoid or reduce a risk, transfer the risk” – e.g. to the customer.

Also the customers have found a way to handle their risk:

> “If we have to freeze the requirements in an early stage, the contractor shall freeze our costs in a fixed price contract. Via validation test we can force him to deliver valuable software.”

A fair deal: A defined solution for a fixed price! - The Standish Groups Chaos Report demonstrates that this fair deal is working just for one third of the SW projects.

There are still some tricky problems to solve:

a) The customer has the risk to define the wrong requirements
(the more innovative the solution or dynamic the business, the higher the risk).

b) The SW-vendor has the risk to misunderstand the requirements (the more innovative the solution or unfamiliar the business, the higher the risk).

c) The time between requirements specification and testing or operation is quite long – we have to spend a lot of time and money before we can prove the results and get a return on investment.

d) Requirements may change: The longer the project lasts the higher the risk.

As the chaos study proves – the risk is high not to get the defined results.

To handle these problems we have established some professional provisions, e.g.:

- A sound analysis and a precise requirements specification.
- Quality Assurance (by an independent QA-instance).
- Change Management and a Change Control Board.

But these measures increase the problems under c): They take time and are costly. And more than that - for many customers and SW-developers these measures are tedious and so they have got low acceptance, as Alistair Cockburn has pointed out:

- The people on the projects were not interested in learning our system!
- They were successfully able to ignore us, and we're still delivering software, anyway!

Beside these professional provisions a more emotional funded and seemingly clever solution has been established widely: "Let us make the specifications less precise![9]"

This saves time and money and more than that: Both sides believe/hope/expect that with a less precise specification they can reduce their own risk:

- For the customer: To be bound to the wrong or insufficient requirements, he has defined in an early stage of the project.
- For the SW developer: To be bound to requirements he has misunderstood and therefore consequently implemented an insufficient solution.

This strategy has been extremely successful – regarding its wide acceptance. It is one explanation for the results of the Standish Groups Chaos Report …

The "freeze requirements first" approach has another unwanted implication:
"At the end of the project you will get no more than you have specified at the beginning – anything else will cost extra time and extra money!" What would you do at the beginning of the project? Right - you will specify a little bit more – to be on the safe side and leave no doubt! The Chaos Report suggests that most projects can go and stay operational with about 60% – 70% of the initially defined functionality[10]. Also Kent Beck [5] argues that many requirements specified early will never be implemented because nobody misses these features at the end of the project.

---

[9] Of course they don't call it "less precise specification" they call it: "Time pressure", "competence in the application field", "trust", …

[10] The percentage of implemented functions & features in "challenged projects" stays constantly between 61% and 67%, while time- and cost overrun have improved substantially .

3.2 Some consequences of “decide late - try out early”

The first consequence is, that we make architectural decisions and start implementing based on incomplete / fragmentary requirements. This in the first cut increases the risk of wrong design decisions.

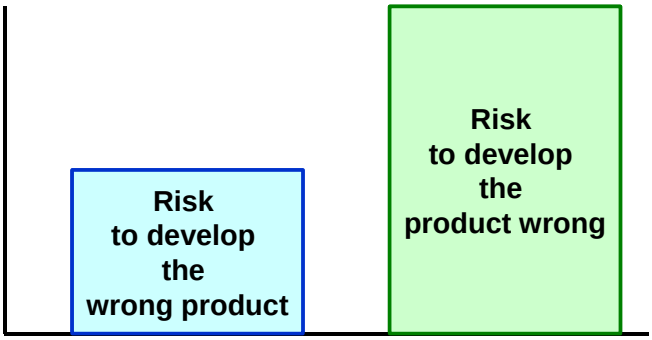

Figure 7: Basic risk characteristic of the sequential approach: Higher risk of wrong design decisions

The second consequence implementing the “decide late - try out early” approach is that we must dramatically reduce the cycle time between definition of functionality and getting feedback by testable or even applicable software.

Working faster by making shortcuts in the sequential process: Not the solution!

Working faster by reducing complexity: A reasonable way! It leads us to split the development in smaller pieces. Grady Booch [9] has visionary figured out this concept in the early 80ies of the last century: ”Analyze a little – design a little – code a little – test a little[11]”. It is not done with splitting the implementation into small pieces, we also have to split analysis and design[12]!

When requirements evolve with increasing understanding or business dynamics they have lost their function as a stabilizing fixed point for project management: The leading invariant in the project management triangle (fixed functionality, budget and schedule) is gone.

For the iterative SW development process we have to replace the “frozen requirements“ by another “lighting house” to give orientation: The project vision.

The vision shall figure out, what the customer wants to achieve with this SW solution. It is a representation of the business value the SW shall realize and justifies the invested budget. For many SW developers the vision is a new perspective. Having detailed and precise requirements, the developer is not forced to give attention to the customers business needs and goals[13]. Having no detailed and precise requirements the customers vision is the only medium- and long-term guideline for the evolution of the system. Each SW developer should take the vision statements into account as basic condition for the implementation of the defined functionality for the actual iteration.

---

[11] It has taken years to transform this nice sounding concept into a manageable SW development process.

[12] This is the main distinction between “iterative SW development” and “incremental SW development”.

[13] Some developers even are not interested in the customers business and vision.

The iterative SW development approach leads us to different best practices for the SW development process, e.g.:

- "Focus on requirements first" in the sequential process is replaced by "Focus on the architecture first".
- The detailed project plan is replaced by a consequent risk management process and planning discipline: Fixed time frame for each iteration and a consequent prioritization of desired SW features for each iteration.
- While requirements change management (with a Change Control Board) is best practice to handle the requirements creep in the sequential process, the iterative/agile process would regard requirements change management as a work around. Best practice is short iterations and a flexible (adaptive) iteration planning process, which fixes the set of features for each iteration.

Another consequence of iterative SW development is that the customer has to make decisions more often: In the sequential SW development most customer decisions are focused to both ends of the project: The analysis/planning phase and the acceptance test. In iterative SW development the customer has to decide at the beginning and end of each iteration to define and prioritize features, valuate test results, and so on.

Last but not least in iterative SW development the risks are more obvious. Consequently risk management becomes increased significance. Any iterative SW development model defines risk management as a central control process. Risk assessment documents and a risk mitigation plan are essential artifacts from an early stage of the project up to the end![14]

4 Conclusion

The big majority of SW professionals grew up with the sequential SW development paradigm. Some of its implications seem to be laws of nature for many of them. Many SW professionals learnt that the sequential SW development process does not fulfill their expectations promoted by the SW engineering theory. The iterative SW-development approach seems to be the answer to their discomfort.

Many of them interpret iterative SW development as a less disciplined variant of an incremental SW development approach (a pretty complete requirements analysis first and then a step by step implementation) and do not succeed. In the last two years I have heard at least a dozen times: "We have tried XP but it does not work". No one of them has implemented XP in a disciplined way!

Others, more traditionally oriented IT professionals clearly see the additional risks of the iterative approach (e.g.: No fixed requirements, problems with fix price tenders, and planning, …), and overlook that iterative SW development can eliminate other risks (or at least substantially mitigate the consequences) they have accepted as part of the game (e.g.: customers uncertainties about required functionality, long elapse time between project start and initial operation of the system, moving target requirements caused by a dynamic evolving business, …)

[14] The sequential SW development approach has suggested for a long time, that most risks can be mitigated by sound analysis, detailed planning and consequently following this plan.

Both approaches (sequential or iterative) may lead you either to low or to high risks, depending on your circumstances:

- An inadequate SW development approach for your situation may lead to high risks in both dimensions.
- An adequate SW development approach for your situation may lead to low risks in both dimensions.

Caution – Avoid the Traps:

The changeover to iterative SW development may be a big chance, but on the way there are a lot of traps:

- Sticking to sequential thinking and trying iterative doing eliminates the stabilizing elements of the sequential process without having the benefits of the iterative process.
- The iterative approach without clear vision, disciplined iteration planning and time boxing easily ends up with chaos.
- Picking the best of both worlds easily leads into a process without any risk strategy – you will take the risks of both sides. As Tom Gilb said:

  "You may forget some critical factors – bout they won't forget you!"

The right SW development approach for the right project:

The selection of a SW development approach is a sensitive decision, which shall take into account the projects circumstances:

- The sequential SW-development approach is considered optimal on projects, in which clear stated requirements or specific boundary conditions (e.g. safety related systems or some kinds of embedded real time systems) favor a highly plan driven (predictive) approach.
- The iterative SW-development approach is considered optimal on projects, in which there is enough uncertainty that exploration and progressive understanding of requirements favor a highly adaptive approach.

Guidelines to select the appropriate SW development approach:

Most projects can not be classified so clearly as one of the scenarios above. The following questions shall give an orientation guide to select the appropriate SW development approach:

Is the project bound to a public invitation to tender?

- Yes → No chance for an iterative process!
- No → An iterative process may be taken into account!

Do you have an internal customer?

- Yes → The iterative approach may close the gap between theory and practice you may have had in the past.
- No → The iterative approach requires intensive customer cooperation.

Do you trust, that the customer knows pretty good what he needs?

- Yes → A strong indicator for the sequential approach.
- No → The iterative approach could help to handle the problem.

Do you trust in the project teams competence in the application field?

- Yes → The sequential approach may have a good chance to succeed.
- No → The iterative approach could mitigate your problem.

Do you trust, that your customer will accompany the project with competent people to give qualified feedback?

- Yes → Good precondition for an iterative SW development approach.
- No → Precise requirements at the beginning could mitigate your problem.

Are the members of the SW development team interested in the customers vision?

- Yes → Good precondition for any SW development approach.
- No → Precise requirements at the beginning could mitigate your problem.

How big is your project/system?

- Very big system & long term project → An iterative approach can reduce complexity and speed up your project. That can help you to bring essential aspects of your system operational (→return on investment) before a change in the basic conditions may wash your project overboard.
- Small projects → an iterative/agile approach may help you to eliminate (documentation and support) overhead which has made troubles in the past and speed up your project without increasing your risks.

Beside these questions you can find a lot of indicators in your projects. You just have to keep an eye on the situations your teams succeed and on your troubles. Which risk strategy could improve your performance?

- “Freeze requirements first” or
- “decide late - try out early”

Make a clear decision, keep in mind the implications and go the way consequently!

5 Bibliography

[1] B.W. Boehm:
Software Engineering Economics, Englewood Cliffs: prentice Hall 1991

[2] V-Modell 97:
Vorgehensmodell zur Planung und Durchführung von IT-Vorhaben; Entwicklungsstandard für IT-Systeme des Bundes → www.v-modell.iabg.de

[3] Rational Software White Paper © 1998
Rational Unified Process: Best practices for Software Development Teams

[4] Rational Software White Paper; © 2000 - Leslee Probasco
The Ten Essentials of RUP: The Essence of an Effective Development Process;

[5] Kent Beck: Extreme Programming Explained
Addison Wesley © 2000, ISBN: 0-201-61641-6

[6] Kent Beck, Martin Fowler: Planning Extreme Programming
Addison Wesley © 2001, ISBN: 0-201-71091-9

[7] Chaos Report 2001: The Standish Group International Inc. 2001

[8] Chaos Demographics and Project Resolution, excerpt from 3rd Quarter 2004:
The Standish Group International Inc. 2004

[9] Grady Booch: Software Engineering with ADA; 1983

6 Authors' biography

DI. Andreas Nehfort:

- Consultant for IT & Software Process Improvement (SPI);
  Nehfort IT-Consulting → www.nehfort.at
  Focus: Software Process Models, and Software Process Improvement (Agile Processes, Assessment Based Process Improvement → CMMI / SPiCE)
- Certified SPICE Assessor
  ISO 15504 – SPICE: Software Process Improvement and Capability dEtermination
- since 1988 Trainer for IT Project Management, Software Engineering & SW Quality Assurance
- since1986 self-employed → Nehfort IT-Consulting → www.nehfort.at.
- since 1982 Project Manager, Requirements Analyst & Consultant for Software Engineering & Software Processes
- 1978: I started my professional career as Software-Developer;
- 1975 – 1979: Study of Technical Mathematics at the Technical University of Vienna – certificate Dipl.Ing. / DI.