SAFe Diaries

Es ist ja nicht alles schlecht, wo „SAFe“ drauf steht…

Sebastian Bauer – agilschmiede.de
Mein Scrum ist kaputt – meinscrumistkaputt.de

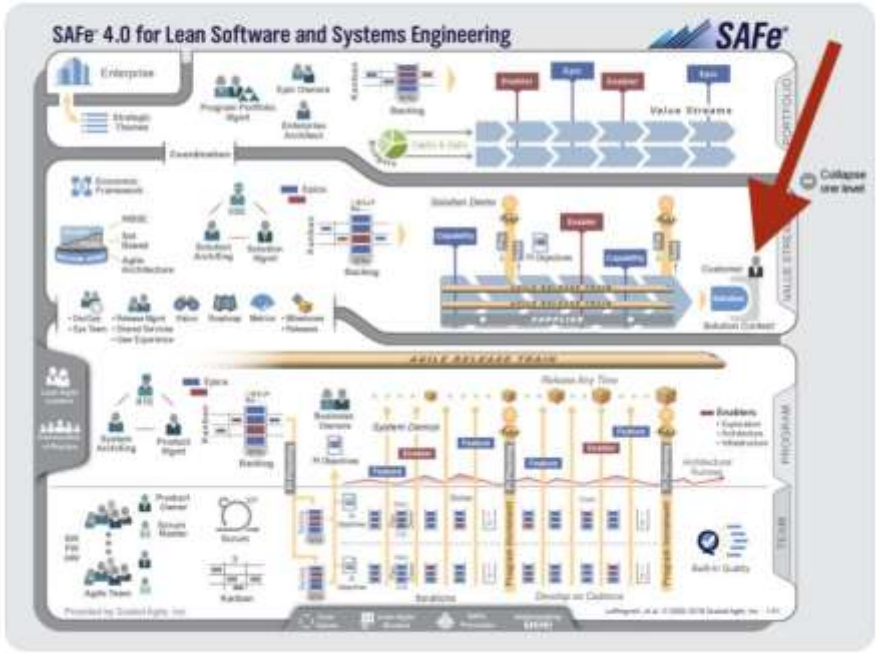
SAFe

Scaled Agile Framework (for Enterprise)

John Cutler
@johncutlefish

I found it! I found it! I found the CUSTOMER!

#agile ?

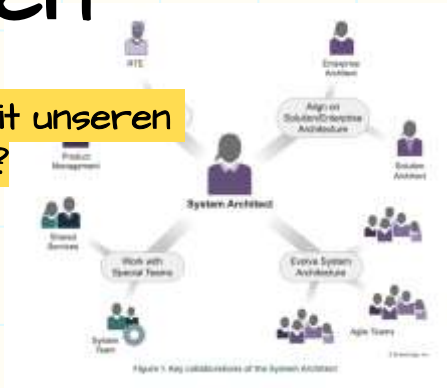


3:58 AM · Mar 18, 2019

SAFe bietet Antworten

Wie machen wir das jetzt mit dem Produktmanagement?

Was machen wir mit unseren Architekt:innen?



Wie läuft denn der ganze Anforderungsprozess außerhalb der Teams?

Sind unsere Teamleiter:innen/Abteilungs-leiter:innen jetzt arbeitslos?

Wie teilen wir denn unsere Teams auf?

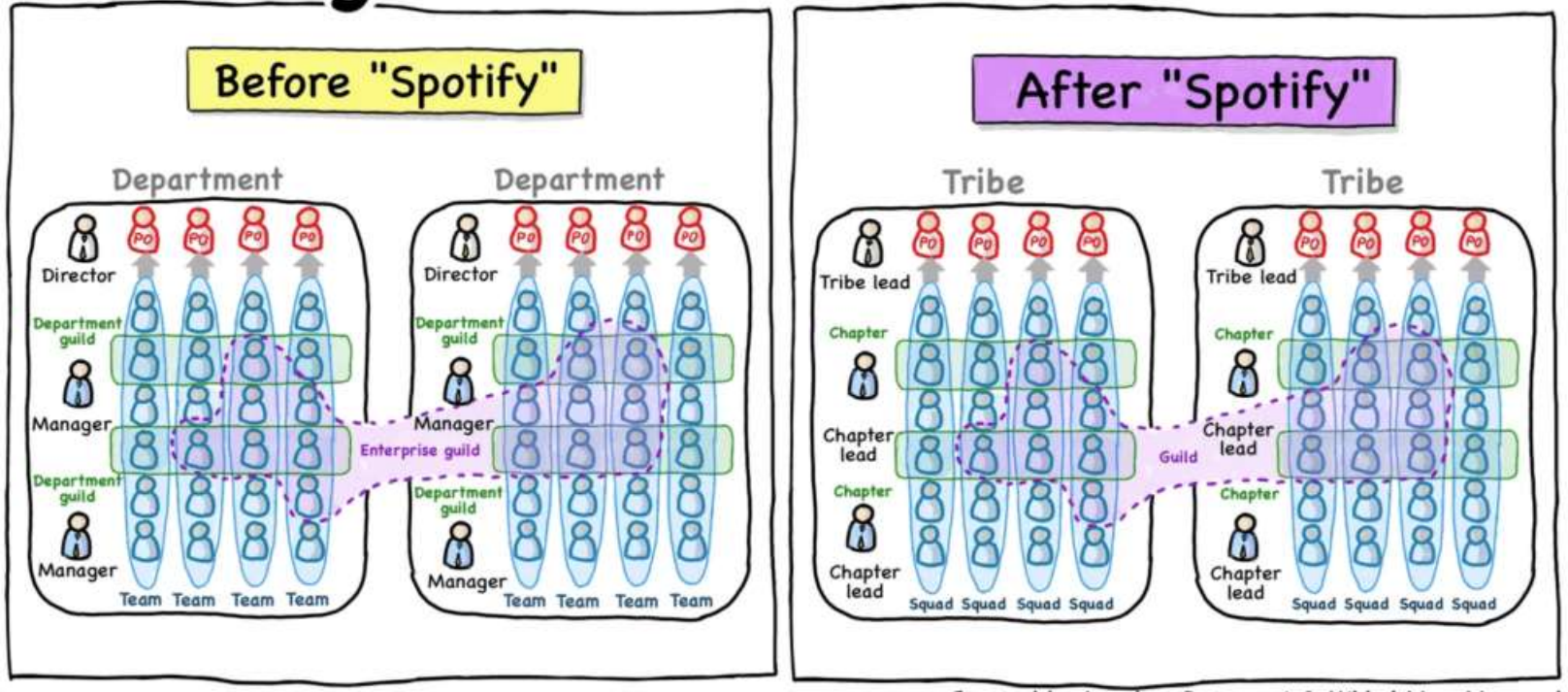
- **Stream-aligned teams** are end customer-aligned and are capable of performing all the steps needed to build end-to-end customer value
- **Complicated subsystem teams** are organized around critical solution subsystems. They focus on areas of high technical specialization, which limits the cognitive load on all the teams
- **Platform teams** provide application services and APIs for stream-aligned teams to be able to leverage common platform services
- **Enabling teams** provides tools, services, and short-term expertise to other teams

Das Problem?

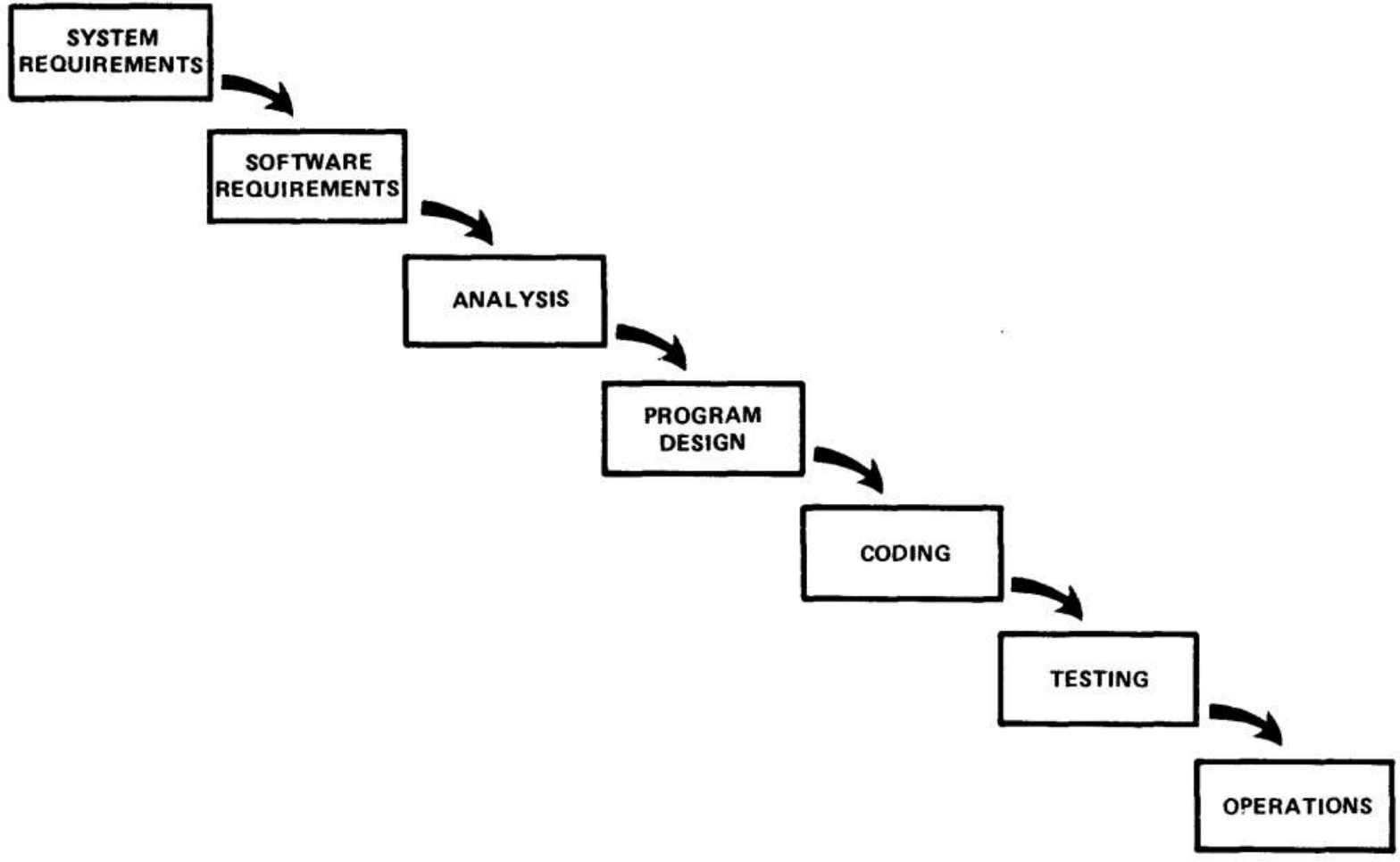
Unternehmen lieben Blueprints!

Comic Agilé

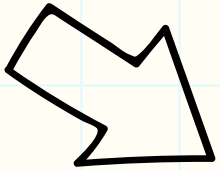
www.comicagile.net



Created by Luxshan Ratnaravi & Mikkel Noe-Nygaard



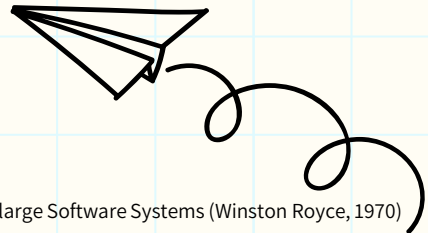
Managing the Development of large Software Systems (Winston Royce, 1970)



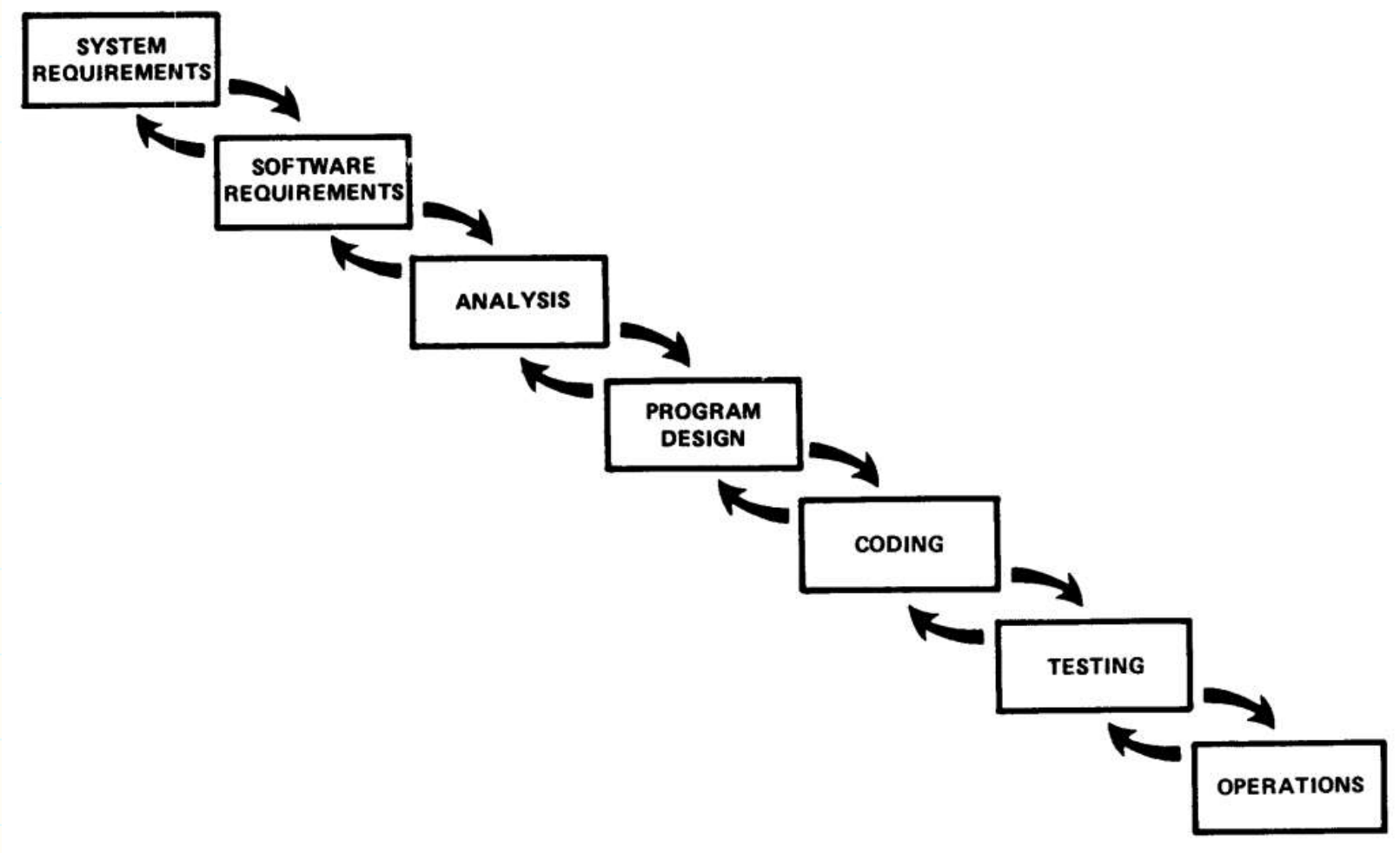
[…] but the implementation described above is risky and invites failure. […] The testing phase which occurs at the end of the development cycle is the first event for which timing, storage, input/output transfers, etc., are experienced as distinguished from analyzed. […] if these phenomena fail to satisfy the various external constraints […], the required design changes are likely to be so disruptive that the software requirements upon which the design is based […] are violated.

Either the requirements must be modified, or a substantial change in the design is required. In effect the development process has returned to the origin and one can expect up to a 100-percent overrun in schedule and/or costs.

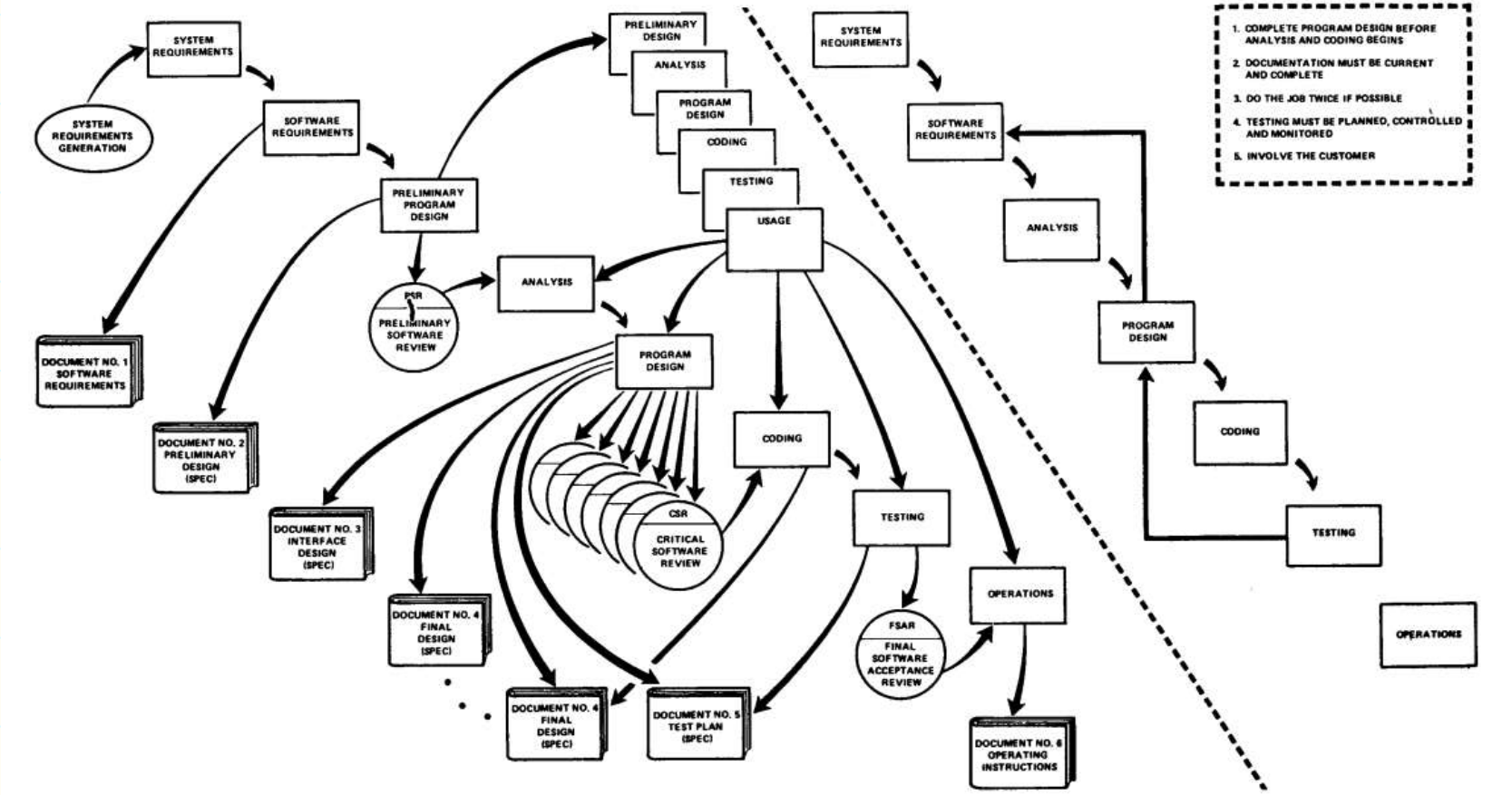
-Winston Royce, 1970



Managing the Development of large Software Systems (Winston Royce, 1970)



Managing the Development of large Software Systems (Winston Royce, 1970)



Managing the Development of large Software Systems (Winston Royce, 1970)

MANAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS

Dr. Winston W. Royce

INTRODUCTION

I am going to describe my personal views about managing large software developments. I have had various assignments during the past nine years, mostly concerned with the development of software packages for spacecraft mission planning, commanding and post flight analysis. In these assignments I have experienced different degrees of success with respect to arriving at an operational state, on time, and within costs. I have become prejudiced by my experiences and I am going to relate some of these prejudices in this presentation.

COMPUTER PROGRAM DEVELOPMENT FUNCTIONS

There are two essential steps common to all computer program developments, regardless of size or complexity. There is first an analysis step, followed second by a coding step as depicted in Figure 1. This sort of very simple implementation concept is in fact all that is required if the effort is sufficiently small and if the final product is to be operated by those who built it — as is typically done with computer programs for internal use. It is also the kind of development effort for which most customers are happy to pay, since both steps involve genuinely creative work which directly contributes to the usefulness of the final product. An implementation plan to manufacture larger software systems, and keyed only to these steps, however, is doomed to failure. Many additional development steps are required, none contribute as directly to the final product as analysis and coding, and all drive up the development costs. Customer personnel typically would rather not pay for them, and development personnel would rather not implement them. The prime function of management is to sell these concepts to both groups and then enforce compliance on the part of development personnel.

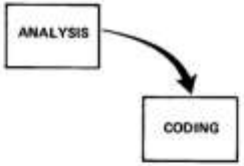


Figure 1. Implementation steps to deliver a small computer program for internal operations.

A more grandiose approach to software development is illustrated in Figure 2. The analysis and coding steps are still in the picture, but they are preceded by two levels of requirements analysis, are separated by a program design step, and followed by a testing step. These additions are treated separately from analysis and coding because they are distinctly different in the way they are executed. They must be planned and staffed differently for best utilization of program resources.

Figure 3 portrays the iterative relationships between successive development phases for this scheme. The ordering of steps is based on the following concept: that as each step progresses and the design is further detailed, there is an iteration with the preceding and succeeding steps but rarely with the more remote steps in the sequence. The virtue of all of this is that as the design proceeds the change process is scoped down to manageable limits. At any point in the design process after the requirements analysis is completed there exists a firm and closely moving baseline to which to return in the event of unforeseen design difficulties. What we have is an effective fallback position that tends to maximize the extent of early work that is salvageable and preserved.

Reprinted from Proceedings, IEEE WESCON, August 1970, pages 1-9.

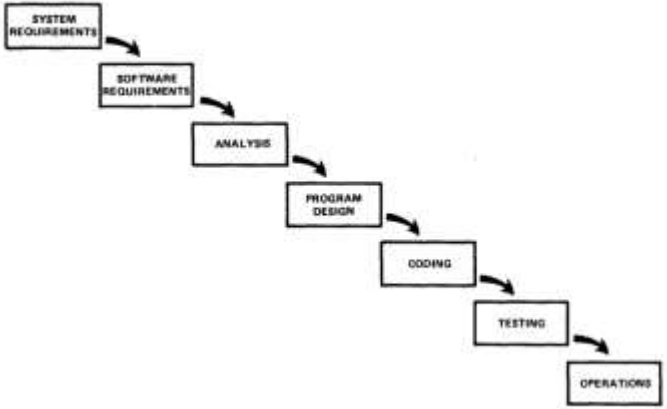


Figure 2. Implementation steps to develop a large computer program for delivery to a customer.

I believe in this concept, but the implementation described above is risky and invites failure. The problem is illustrated in Figure 4. The testing phase which occurs at the end of the development cycle is the first event for which timing, storage, input/output transfers, etc., are experienced as distinguished from analyzed. These phenomena are not precisely analyzable. They are not the solutions to the standard partial differential equations of mathematical physics for instance. Yet if these phenomena fail to satisfy the various external constraints, then invariably a major redesign is required. A simple octal patch or redo of some isolated code will not fix these kinds of difficulties. The required design changes are likely to be so disruptive that the software requirements upon which the design is based and which provides the rationale for everything are violated. Either the requirements must be modified, or a substantial change in the design is required. In effect the development process has returned to the origin and one can expect up to a 100-percent overrun in schedule and/or costs.

One might note that there has been a skipping-over of the analysis and code phases. One cannot, of course, produce software without these steps, but generally these phases are managed with relative ease and have little impact on requirements, design, and testing. In my experience there are whole departments consumed with the analysis of orbit mechanics, spacecraft attitude determination, mathematical optimization of payload activity and so forth, but when these departments have completed their difficult and complex work, the resultant program steps involve a few lines of serial arithmetic code. If in the execution of their difficult and complex work the analysts have made a mistake, the correction is invariably implemented by a minor change in the code with no disruptive feedback into the other development bases.

However, I believe the illustrated approach to be fundamentally sound. The remainder of this discussion presents five additional features that must be added to this base approach to eliminate most of the development risks.

Managing the Development of large Software Systems (Winston Royce, 1970)

Kritik an SAFe (ein Auszug)

SAFe's Planning Intervalle (PI) sorgen für einen manifestierten x-Monatsplan.

Die Autonomie der Teams wird regelmäßig durch allerlei hierarchisch verstandene Rollen verletzt.

Teams haben keine Luft mehr für technische Innovation oder Verbesserungen.

Product Owner sind in SAFe nur noch Proxys.

SAFe baut so viele (teils hierarchische) Strukturen auf - von agilem Wandel bleibt nichts übrig.

SAFe ist zu komplex

"Wie ich lernte SAFe ~~zu lieben~~."
zu akzeptieren.

Deutscher Premium-Automobilhersteller

16 Teams; 130 Developers zzgl. PMs, RTE, POs, Architekten, Team Coaches & Co.

1 große Online-Plattform für Händler:innen und Kund:innen



Warum funktioniert SAFe dort so gut?

Welchen Beitrag leistet SAFe dabei?

Naja…

Was wirklich hilft:

Rollen

PMs, Architekten & Co leben Agile Prinzipien

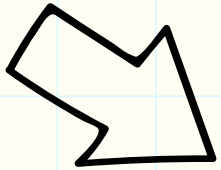
Reifegrad der Teams

(E2E-)Verantwortung, Feedback und Einstehen für agile Prinzipien

Team Coaches sind sehr erfahren

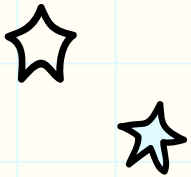
… und selbst keine SAFe-Fans

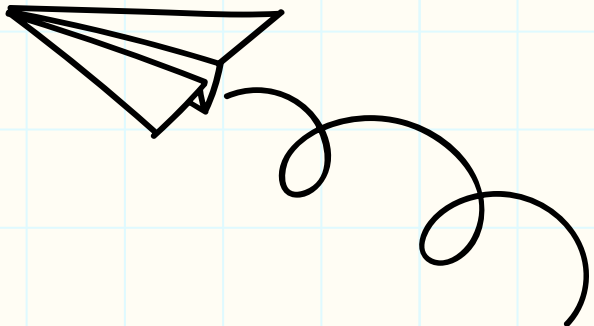
Der RTE mag kein SAFe

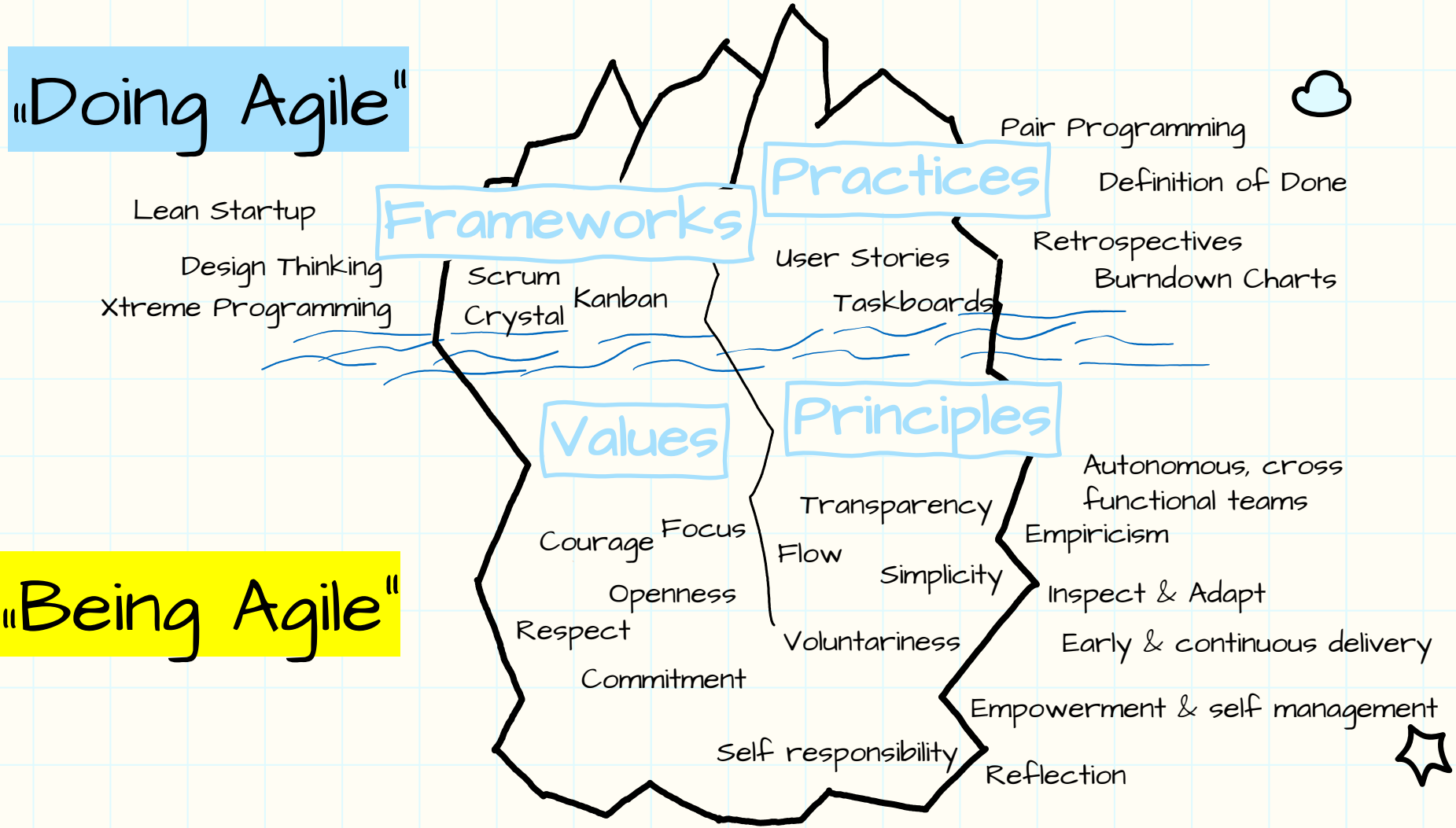


Anders gesagt:

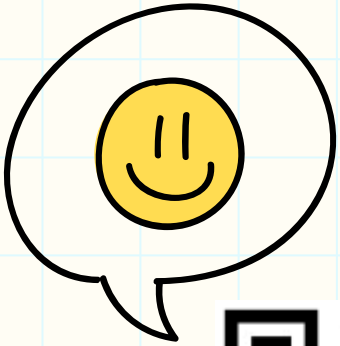
Der agile Reifegrad der Organisation ist so hoch, dass das Framework eigentlich egal ist.






„Doing Agile"
„Being Agile"
Frameworks
Practices
Values
Principles
Lean Startup
Design Thinking
Xtreme Programming
Scrum
Crystal
Kanban
User Stories
Taskboards
Pair Programming
Definition of Done
Retrospectives
Burndown Charts
Courage
Focus
Openness
Respect
Commitment
Transparency
Flow
Simplicity
Voluntariness
Autonomous, cross functional teams
Empiricism
Inspect & Adapt
Early & continuous delivery
Empowerment & self management
Reflection
Self responsibility

Ist das Framework schuld, wenn es ständig falsch implementiert wird?



Stay „SAFe"!

… but please think agile!

Sebastian Bauer

sebastian@agilschmiede.de

agilschmiede.de

Mein Scrum ist kaputt – meinscrumistkaputt.de

QR-Code für euer Feedback