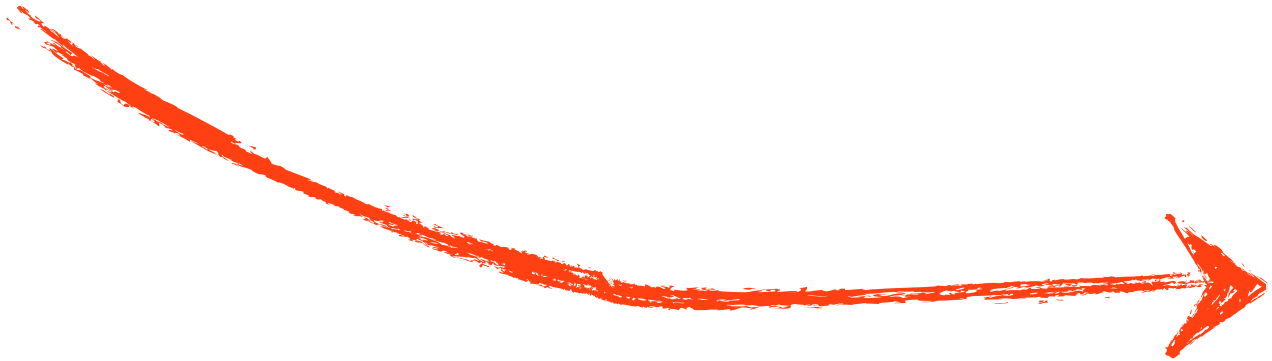
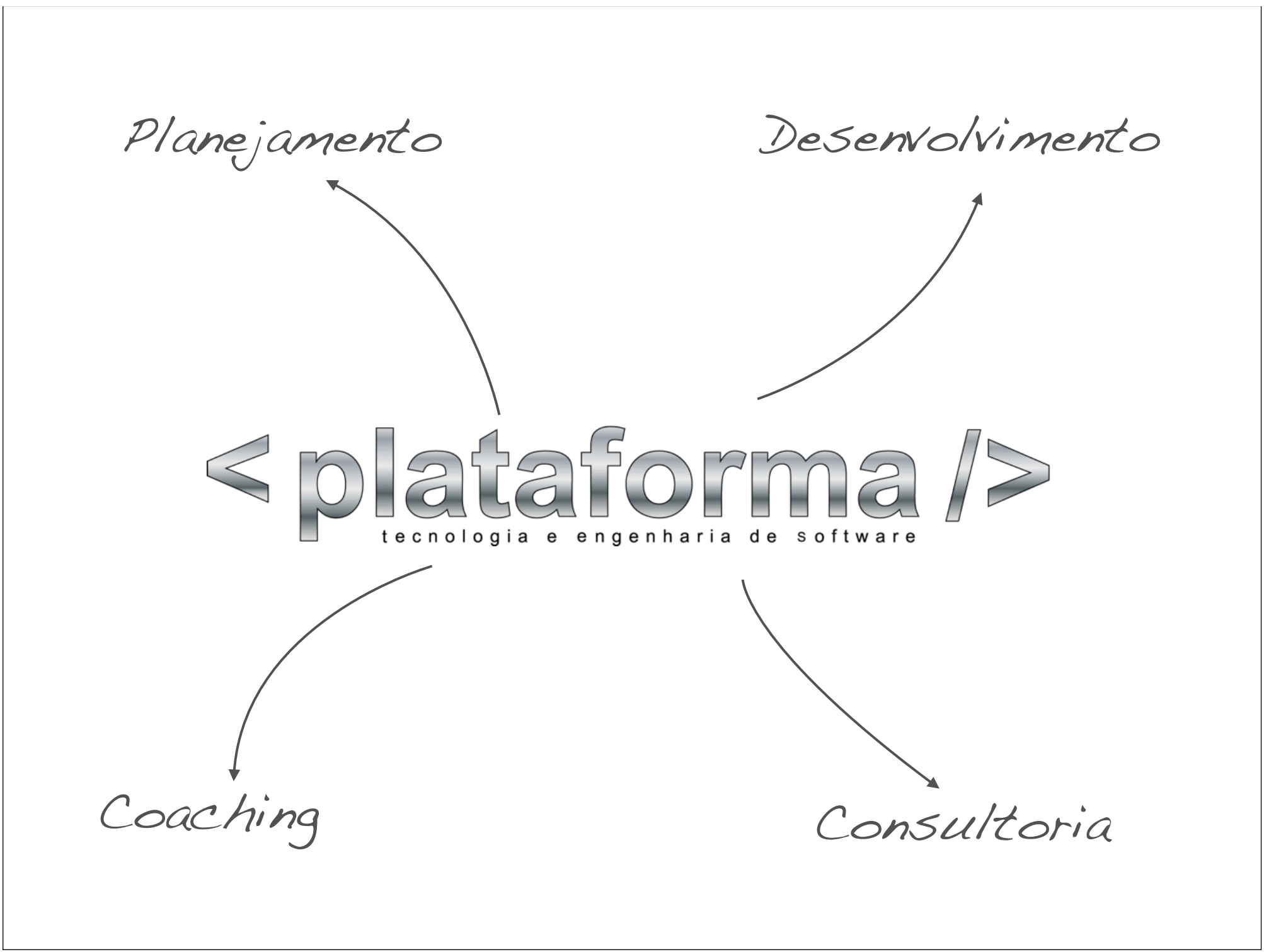

< plataforma />
tecnologia e engenharia de software

Marcelo Park

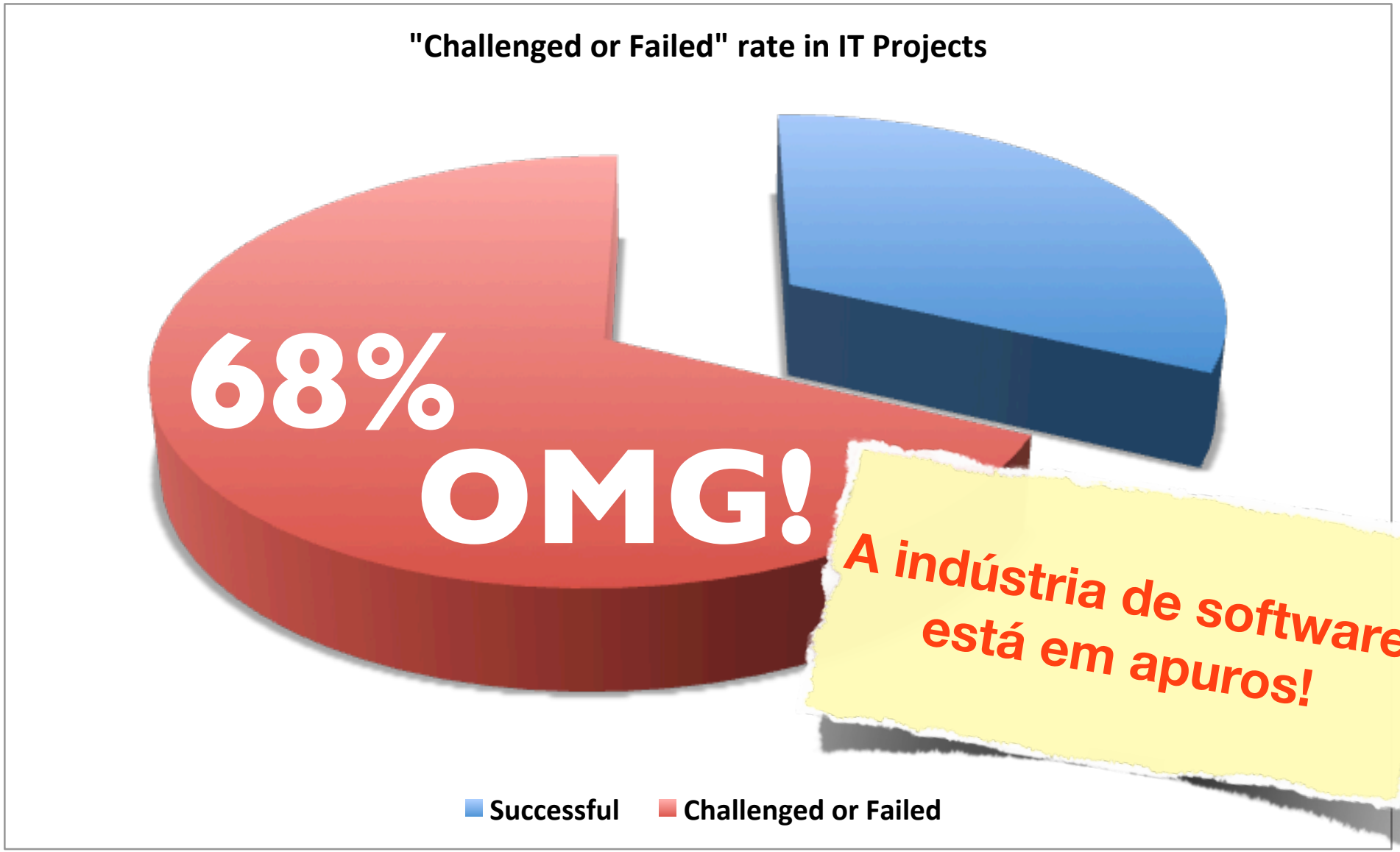



Planejamento
Desenvolvimento
< plataforma />
tecnologia e engenharia de software
Coaching
Consultoria

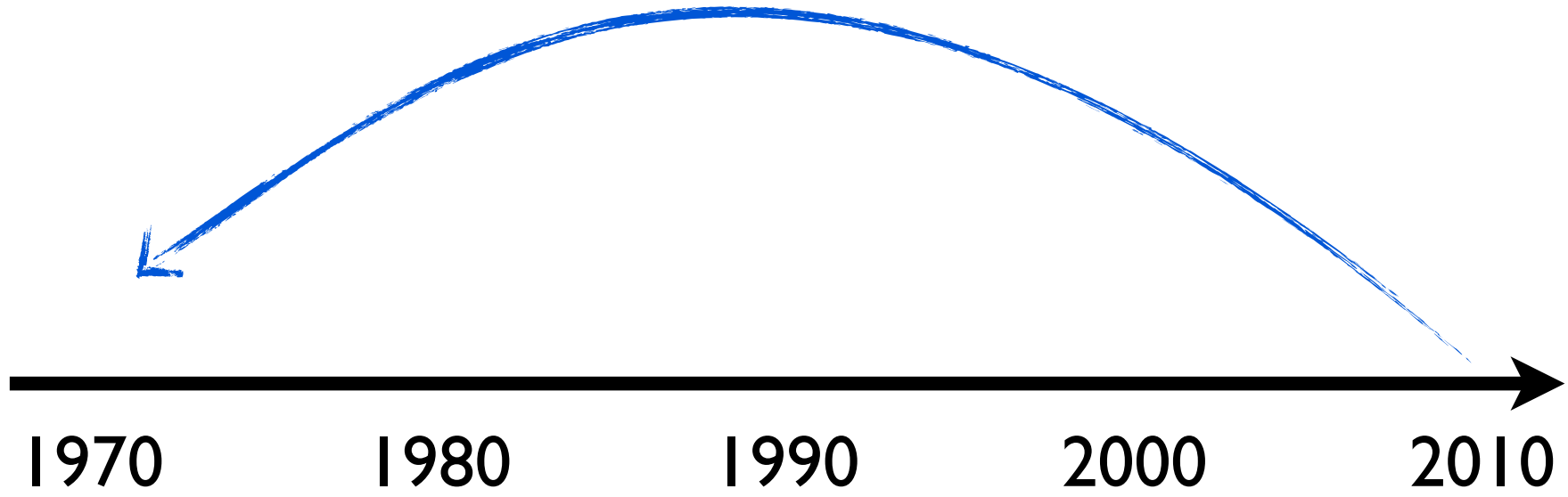
somos muito bons em


Esse livro de
Ruby on Rails
parece ser bom, hein?!!
Crafting Rails Applications

Estou aqui para alerta-los.



Fonte: Standish Group, The Chaos Report 2009


1970
1980
1990
2000
2010

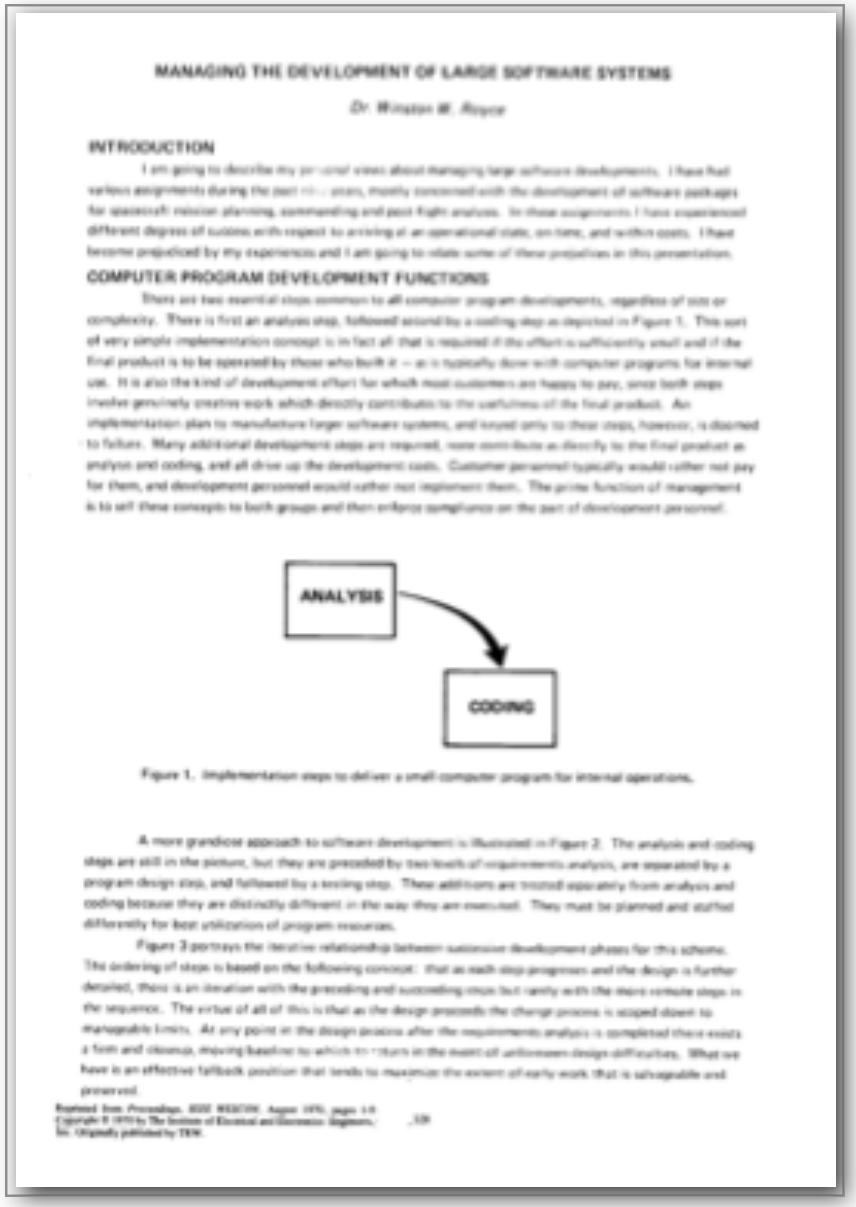
em 1970

Winston Royce publicou o paper

“Managing the development of large software systems”

Managing the development of large software systems

Dr. Winston W. Royce (11 pages)



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 spaceship 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. These 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 relationship 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 scaled down to manageable limits. At any point in the design process after the requirements analysis is completed there exists a firm and obvious, 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.

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

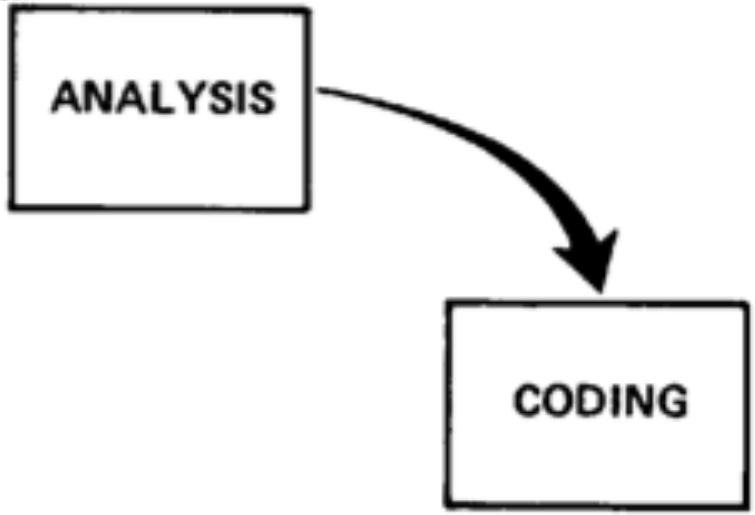


Figure 1. Implementation steps to deliver a small computer program for internal operations.

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 closeup, 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.

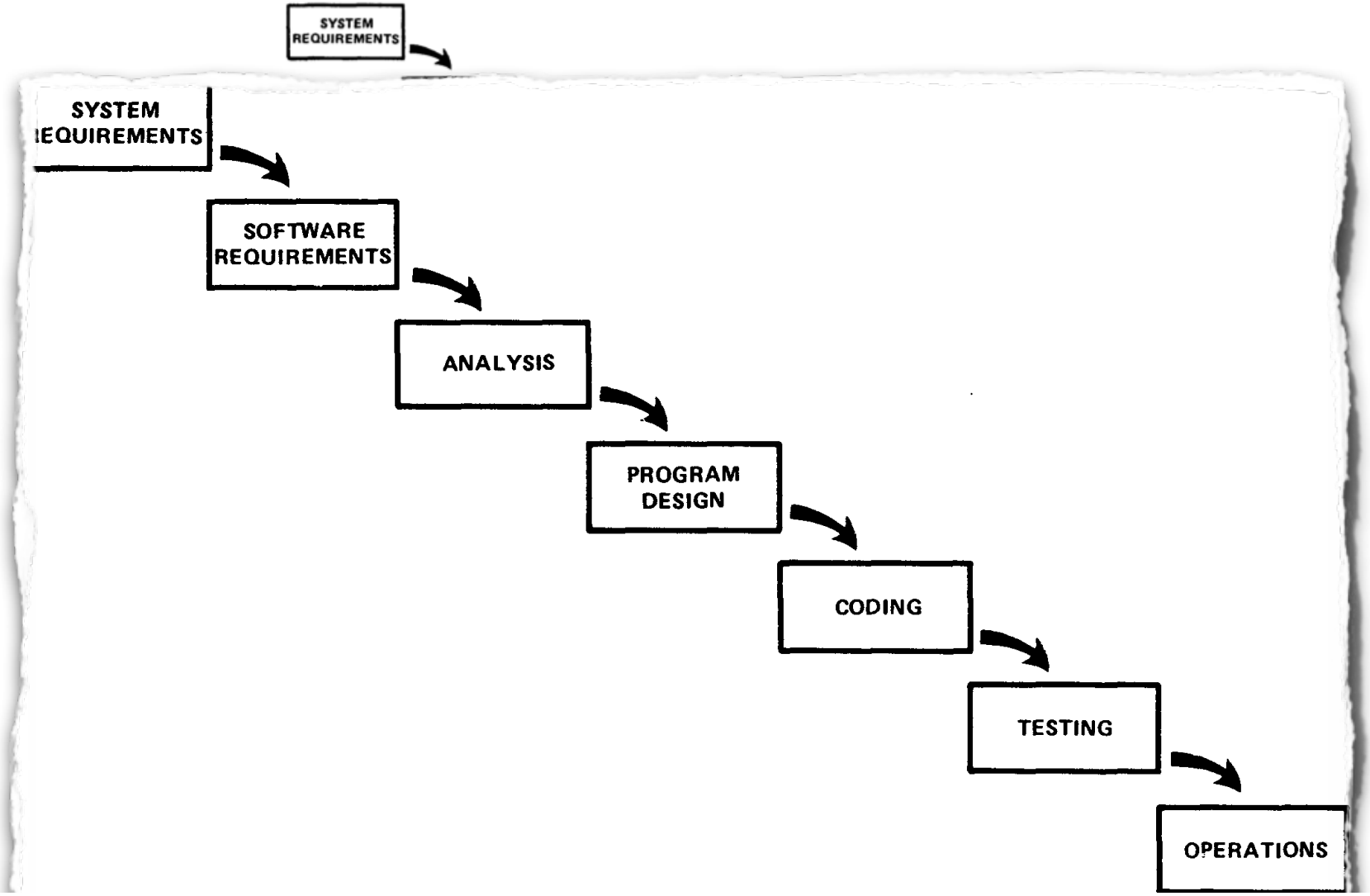


Figure 2. Implementation steps to develop a large computer program for delivery to a customer.

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 basic approach to eliminate most of the development risks.

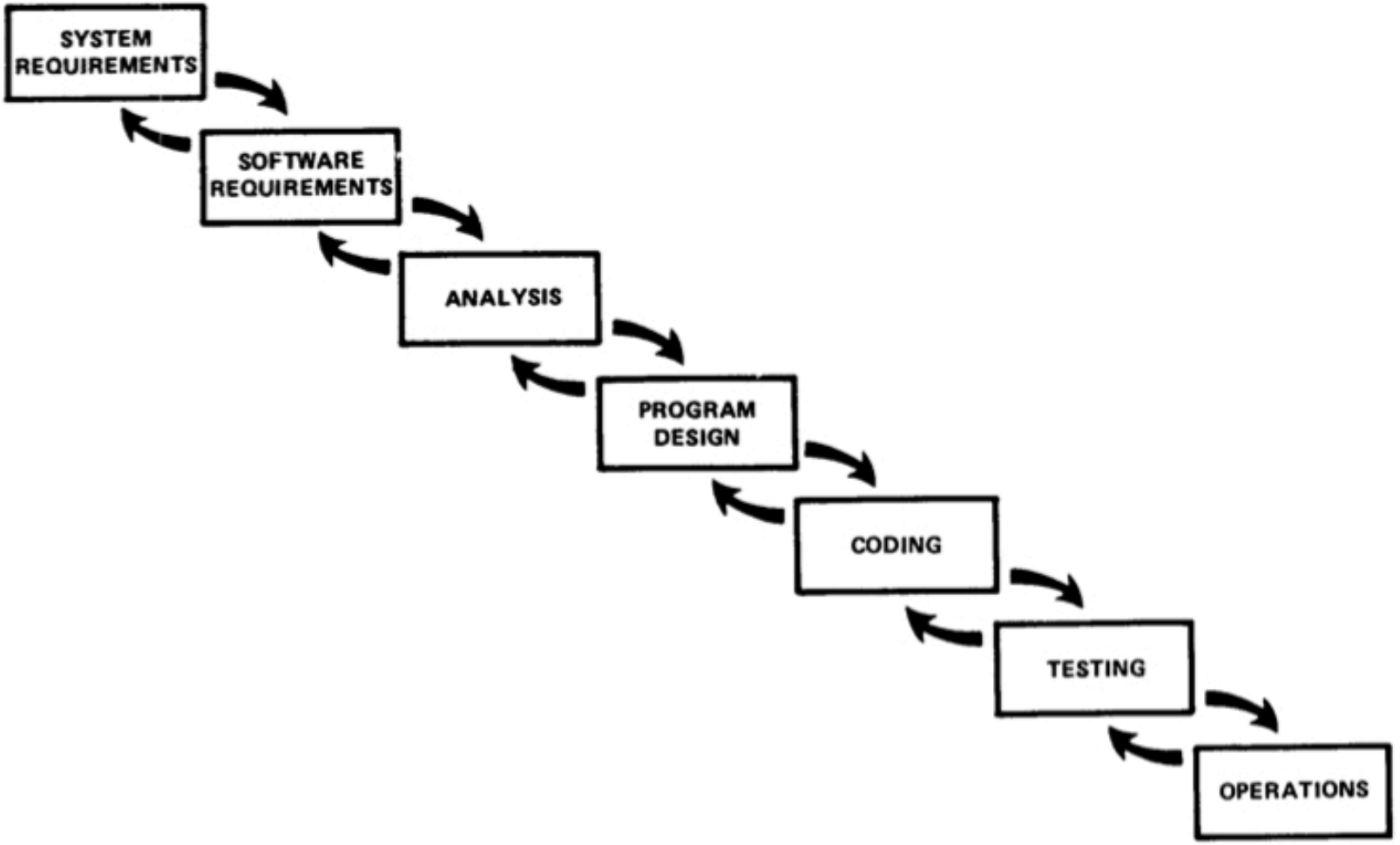


Figure 3. Hopefully, the iterative interaction between the various phases is confined to successive steps.

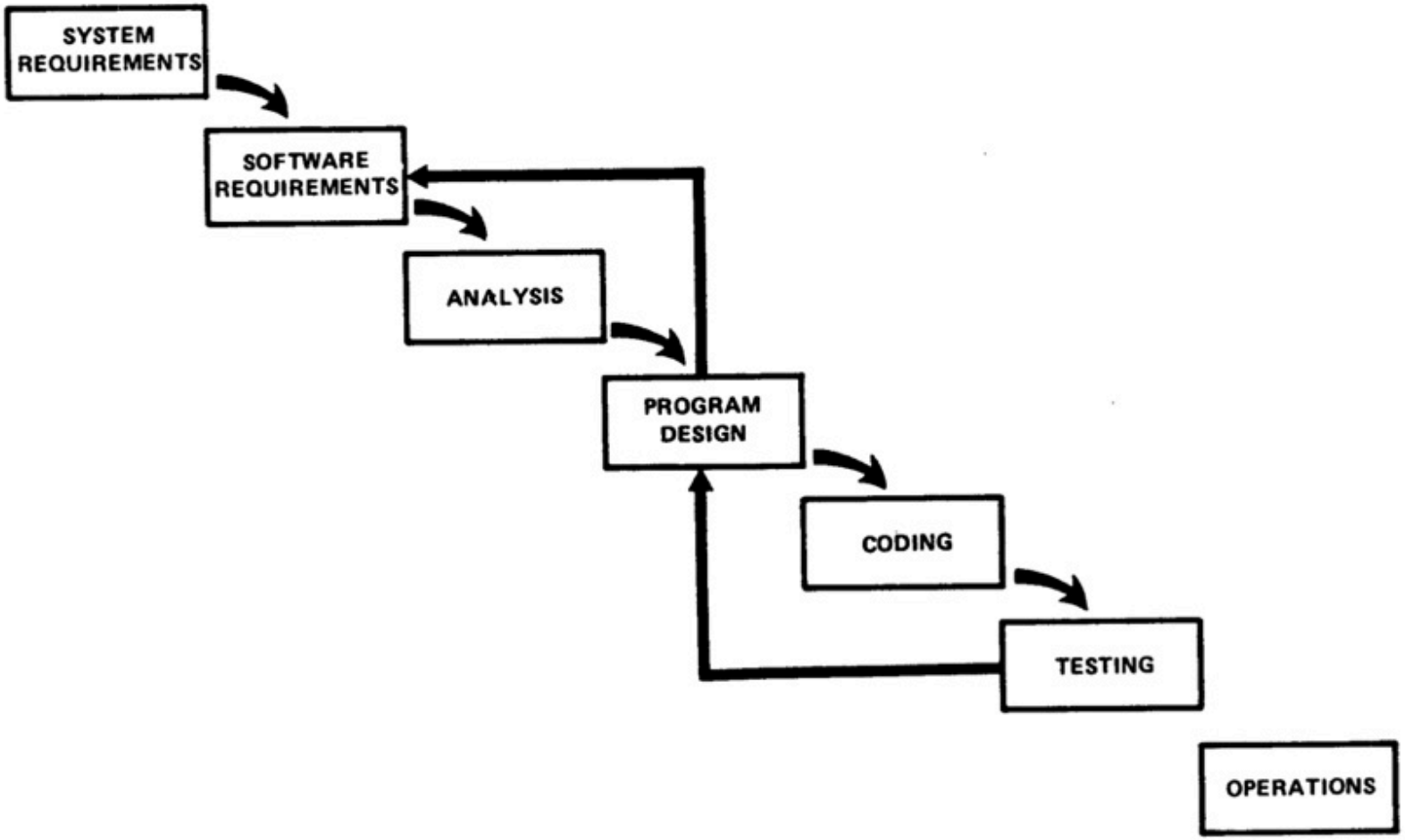
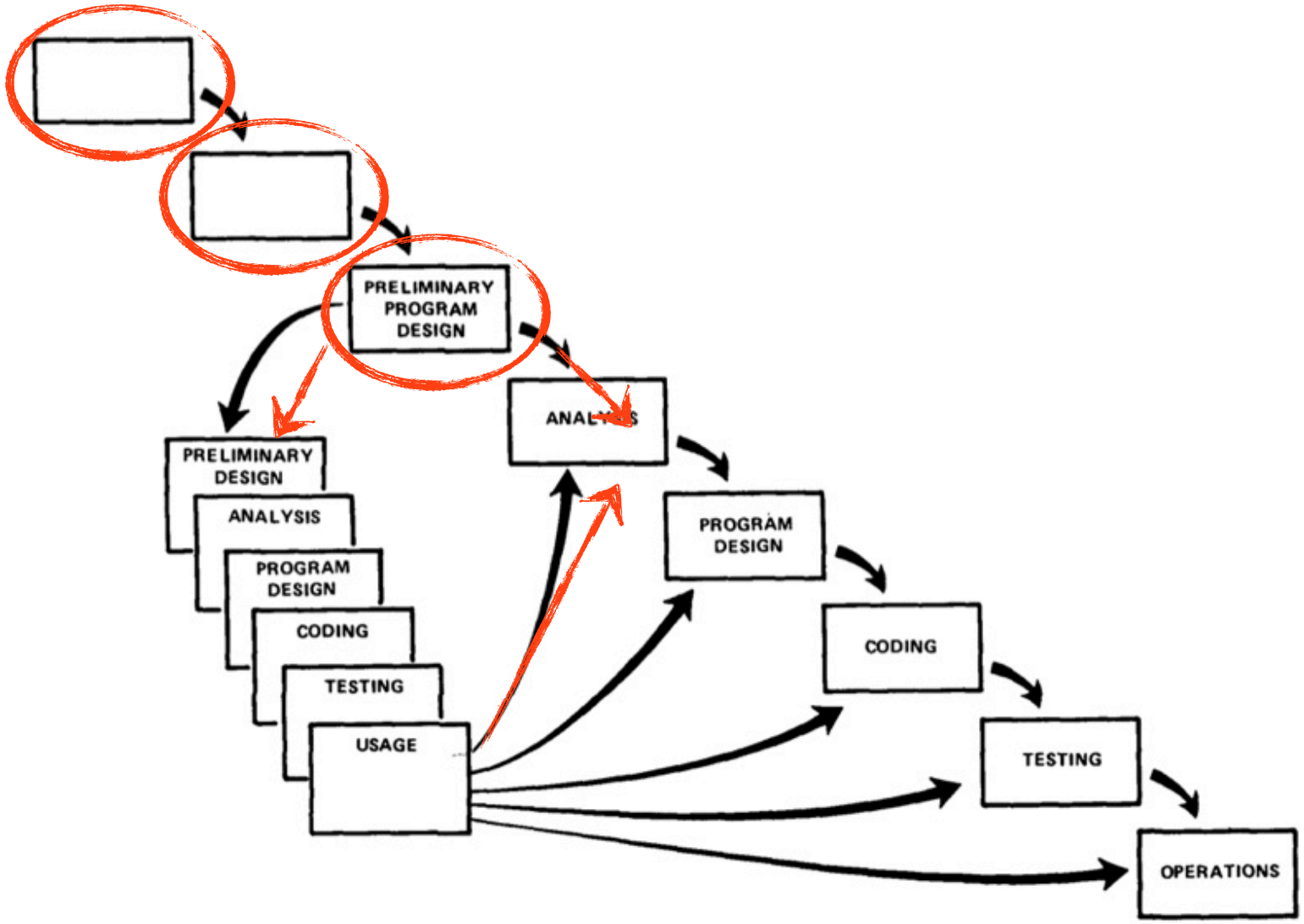
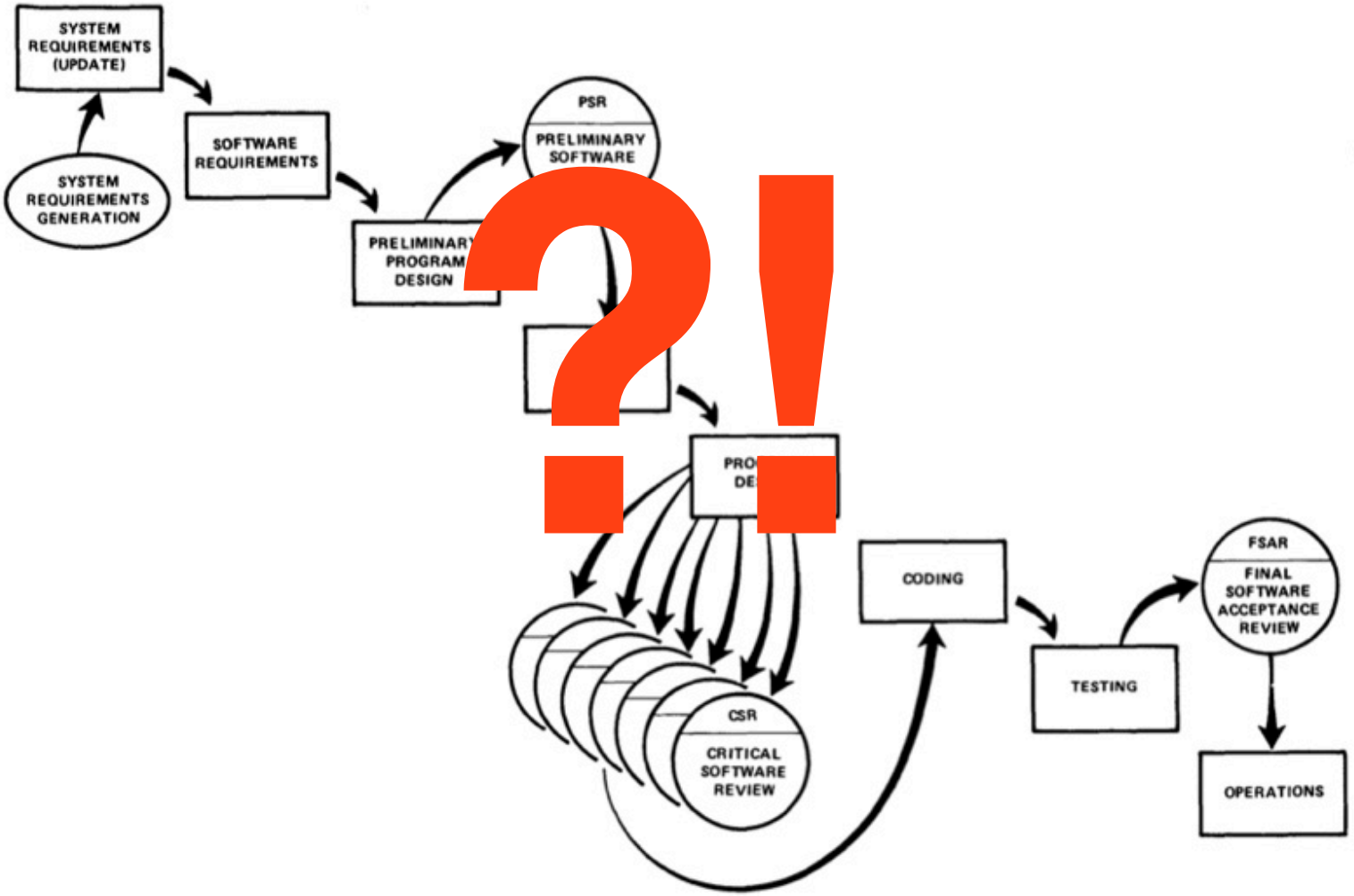


Figure 4. Unfortunately, for the process illustrated, the design iterations are never confined to the successive steps.


PRELIMINARY
PROGRAM
DESIGN
PRELIMINARY
DESIGN
ANALYSIS
PROGRAM
DESIGN
CODING
TESTING
USAGE
PROGRAM
DESIGN
CODING
TESTING
OPERATIONS


SYSTEM REQUIREMENTS (UPDATE)
SYSTEM REQUIREMENTS GENERATION
SOFTWARE REQUIREMENTS
PSR
PRELIMINARY SOFTWARE
PRELIMINARY PROGRAM DESIGN
?!
CODING
TESTING
FSAR
FINAL SOFTWARE ACCEPTANCE REVIEW
CSR
CRITICAL SOFTWARE REVIEW
OPERATIONS


TEST EVERY
LOGIC PATH
PROGRAM
CONFIGURATION
CONTROL, SPEC
MAINTENANCE
TOOLS

Onde está o resumo disso?

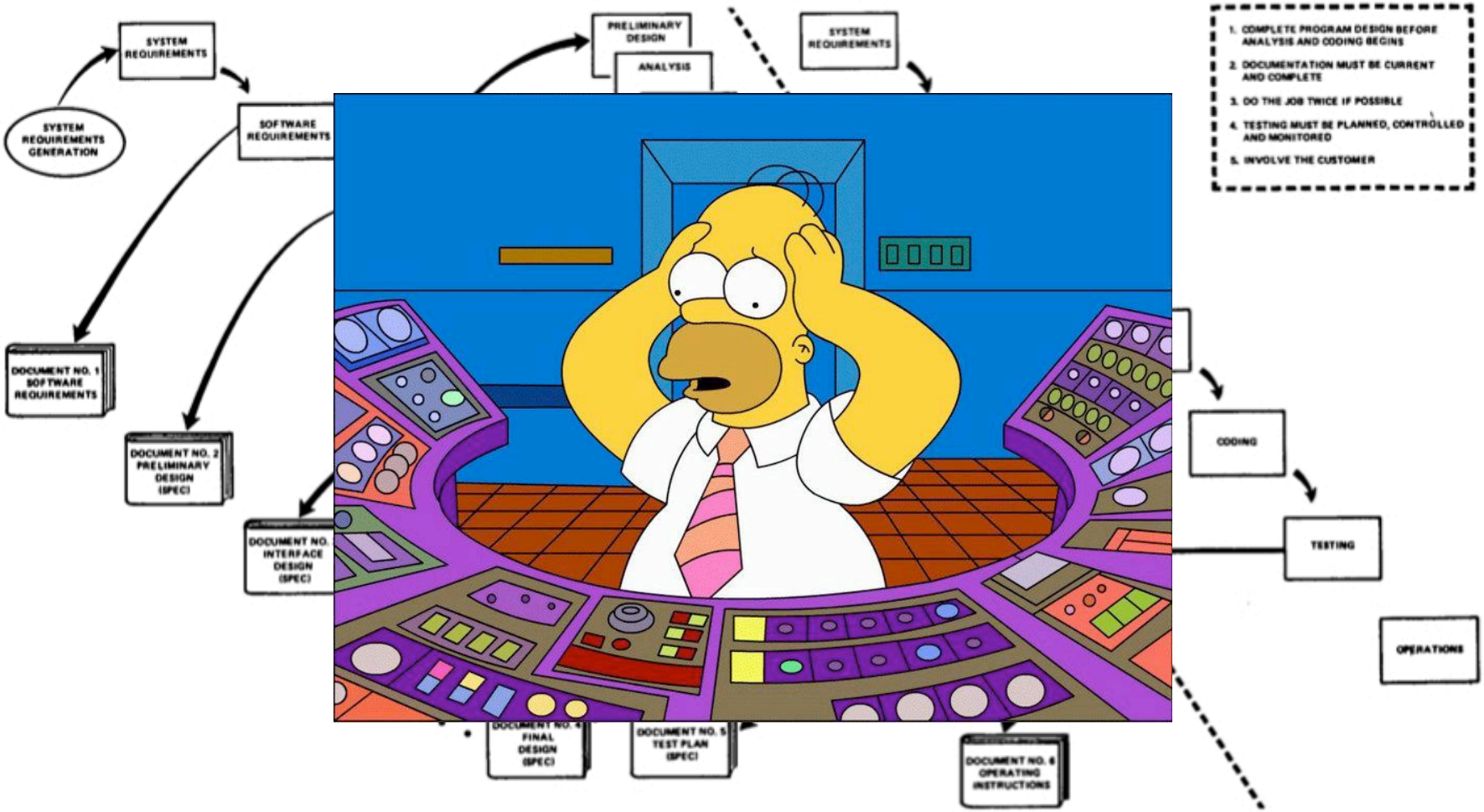


Figure 10. Summary

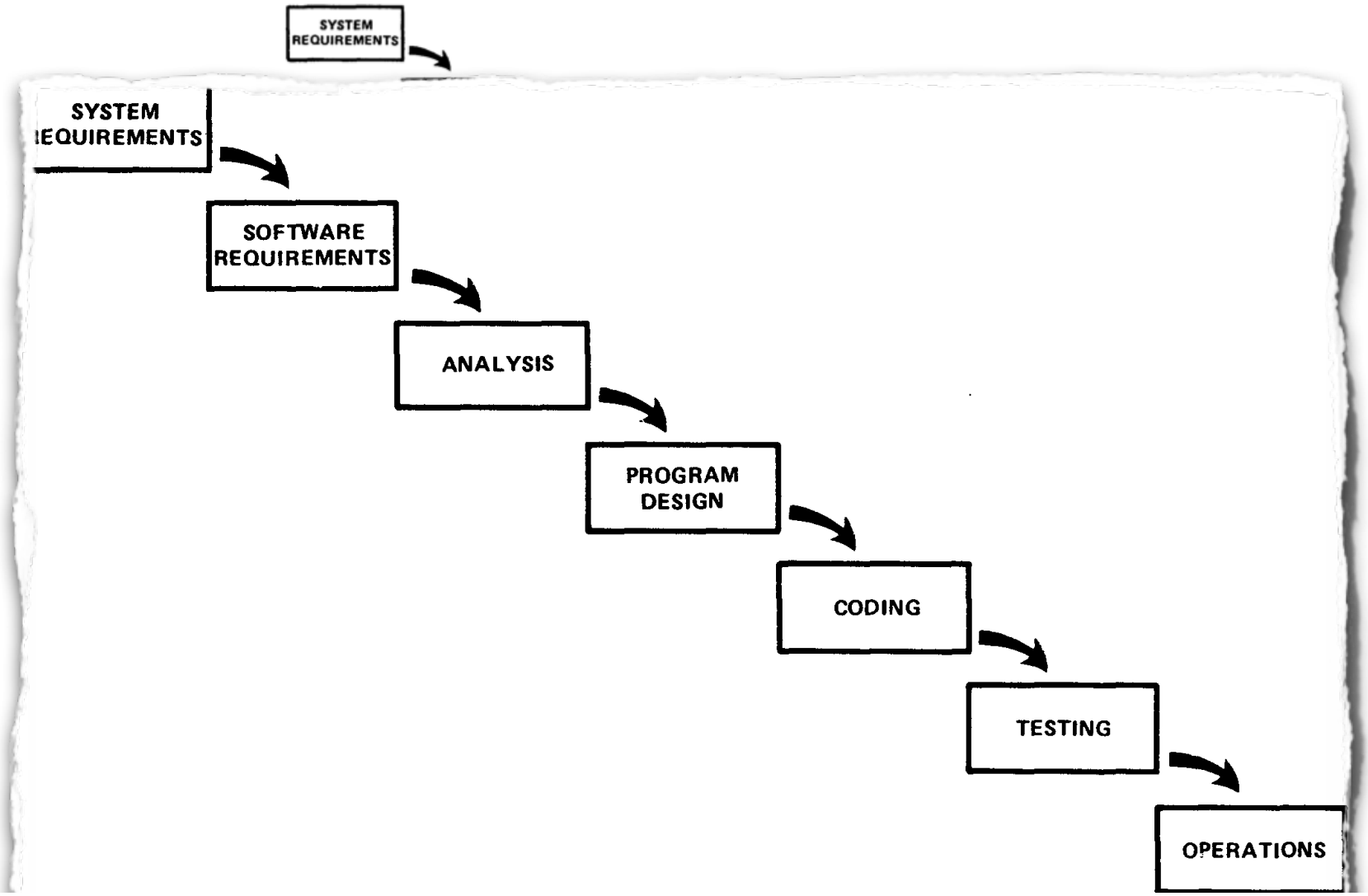
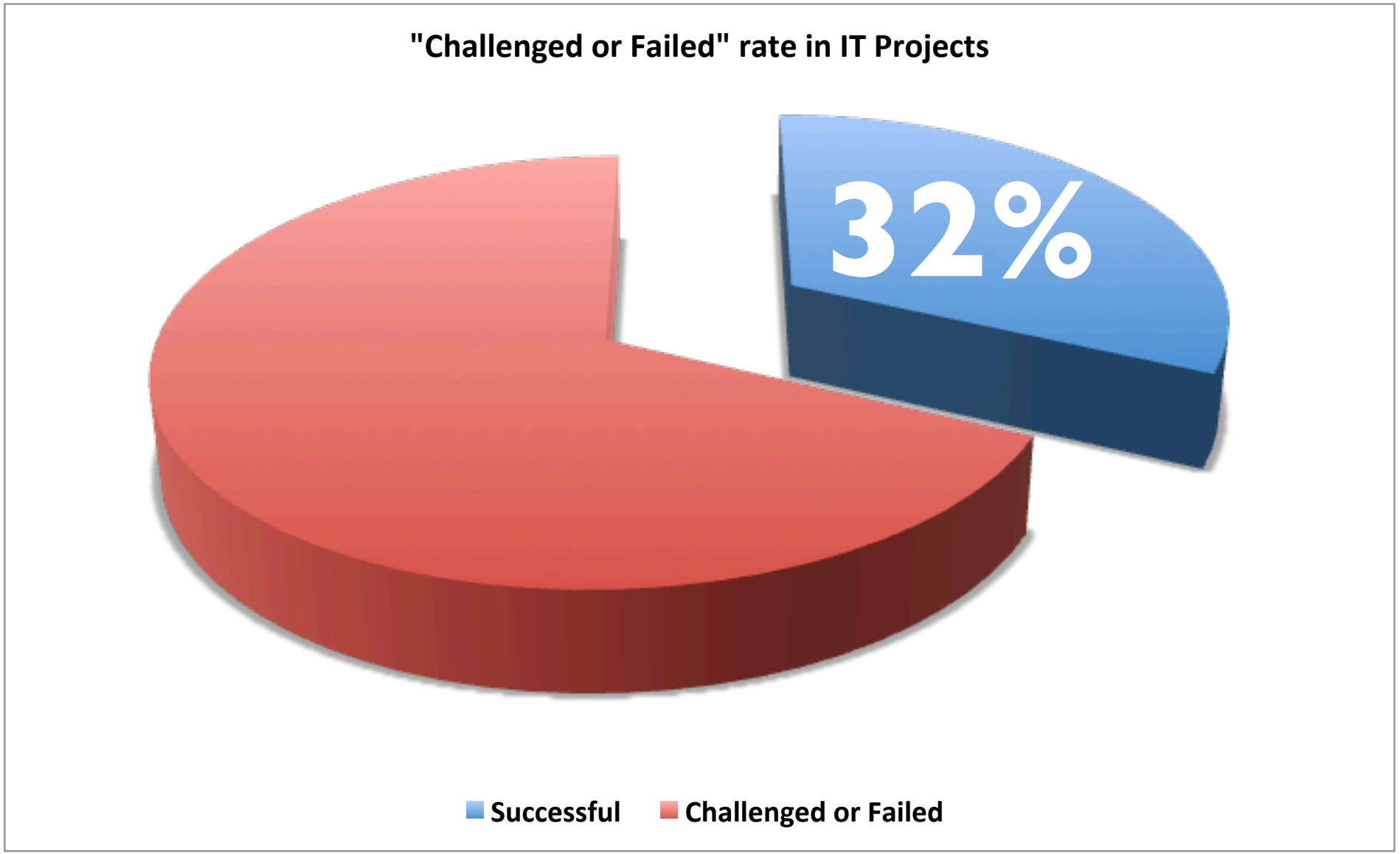


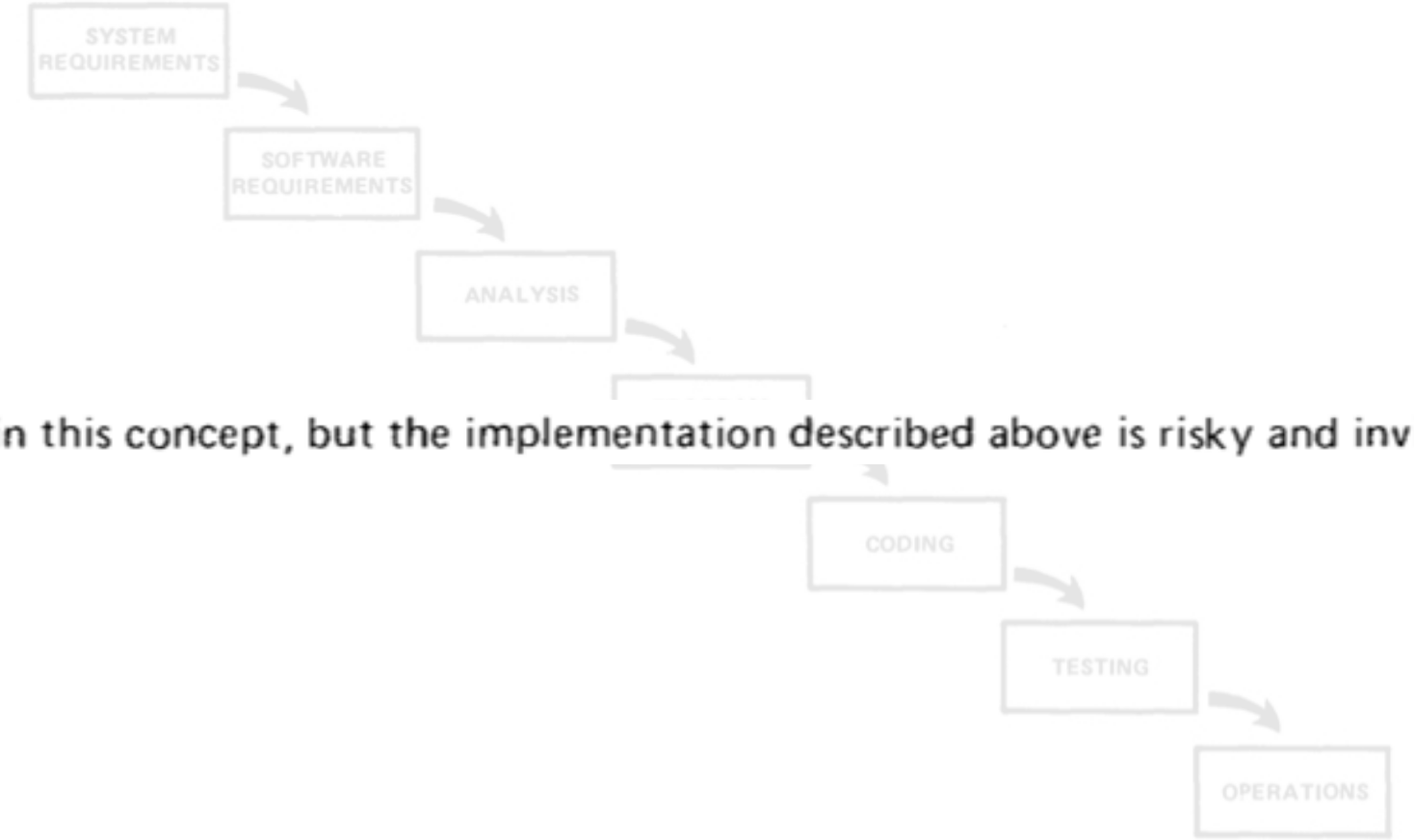
Figure 2. Implementation steps to develop a large computer program for delivery to a customer.

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 basic approach to eliminate most of the development risks.



Fonte: Standish Group, The Chaos Report 2009

Detalhe importante



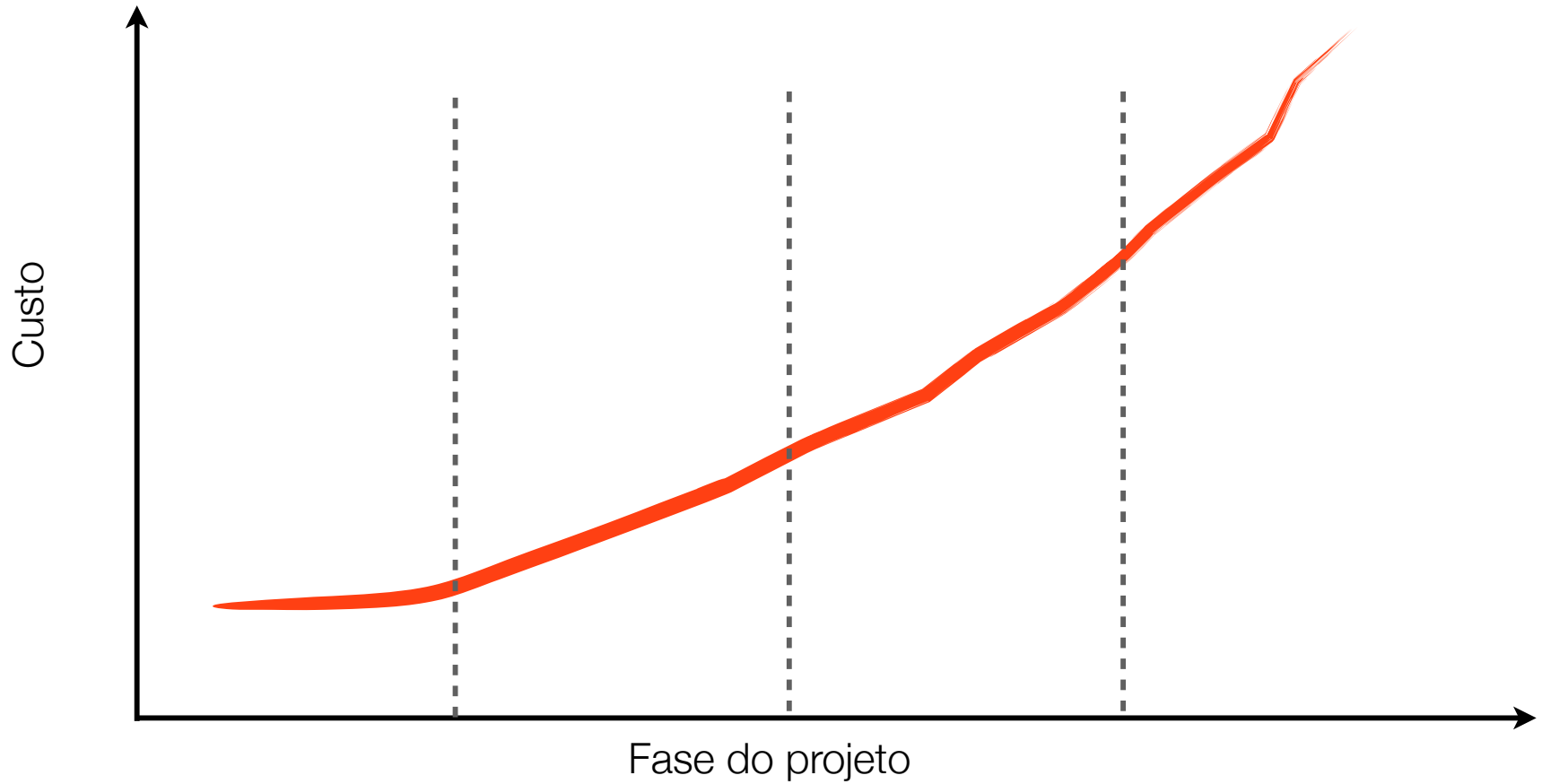
I believe in this concept, but the implementation described above is risky and invites failure.

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

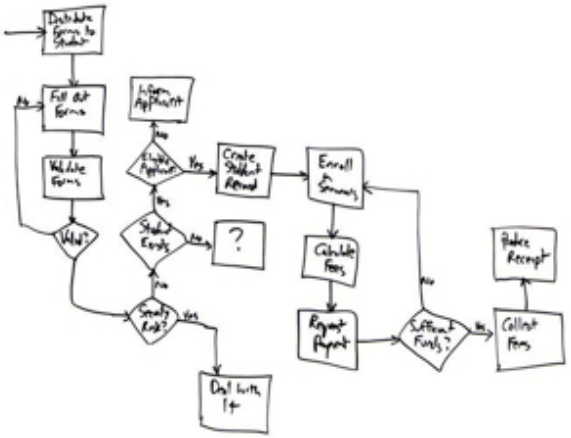
I believe in this concept, but the implementation described above is risky and invites failure. The problem is illustrated in Figure 4. The testing phase which occurs at the end of the development cycle is the first event for which timing, storage, input/output transfers, etc., are experienced as distinguished from


1970
1980
1990
2000
2010

Custo de mudança



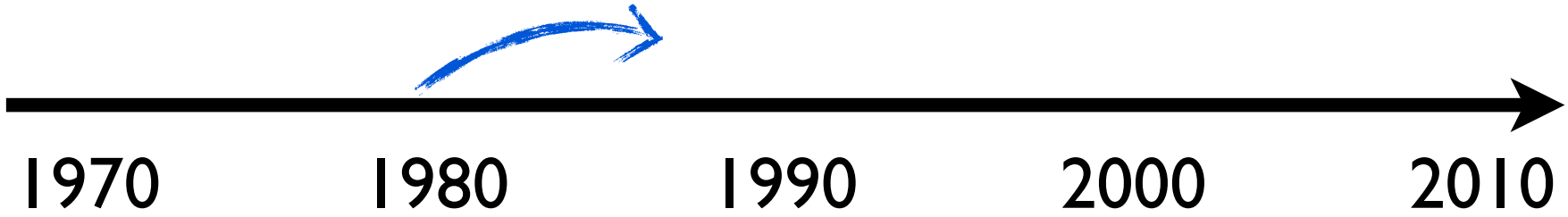
1980's

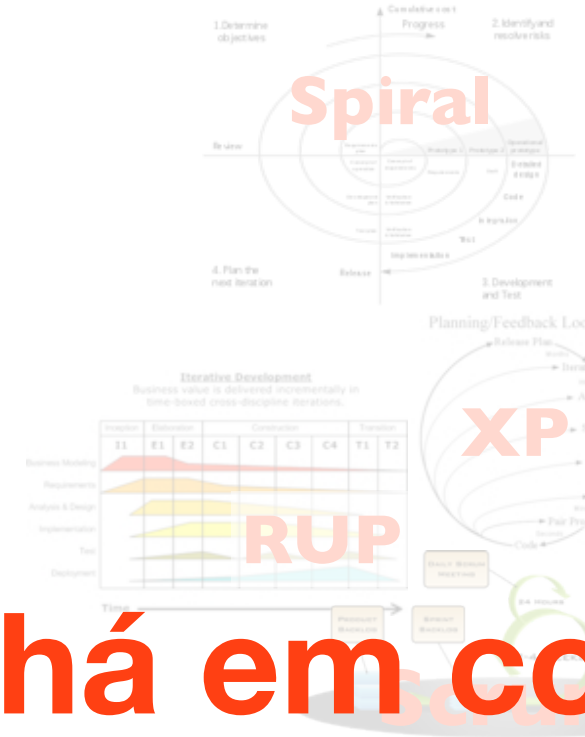


Controlar Processos

=

Reduzir Custos


1970
1980
1990
2000
2010



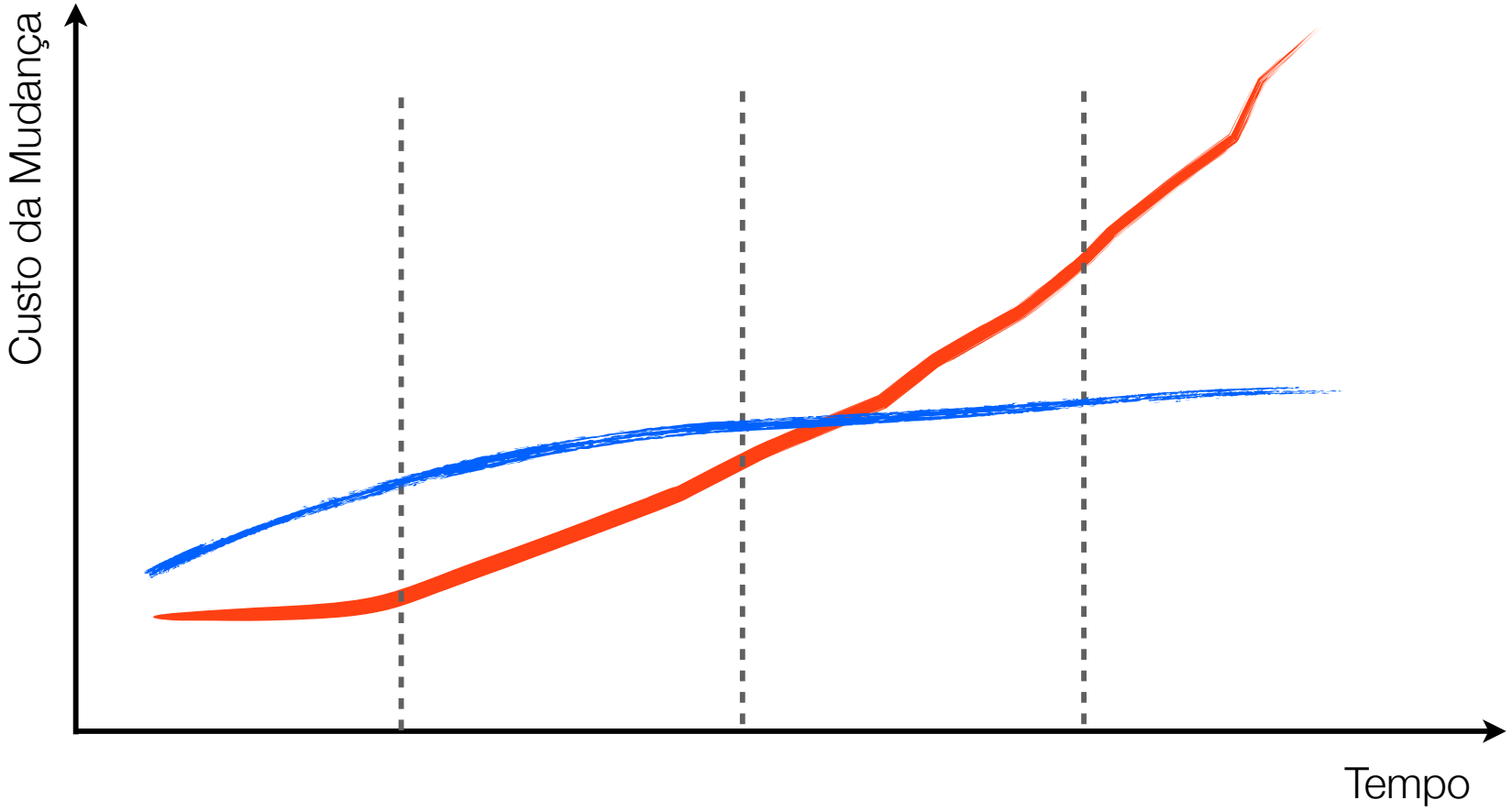
O que há em comum?

1970 1980 1990 2000 2010

Todos eles são processos **iterativos.**

***Iteração:** é o ato de repetir um processo com a intenção de alcançar um resultado desejado ou objetivo.*

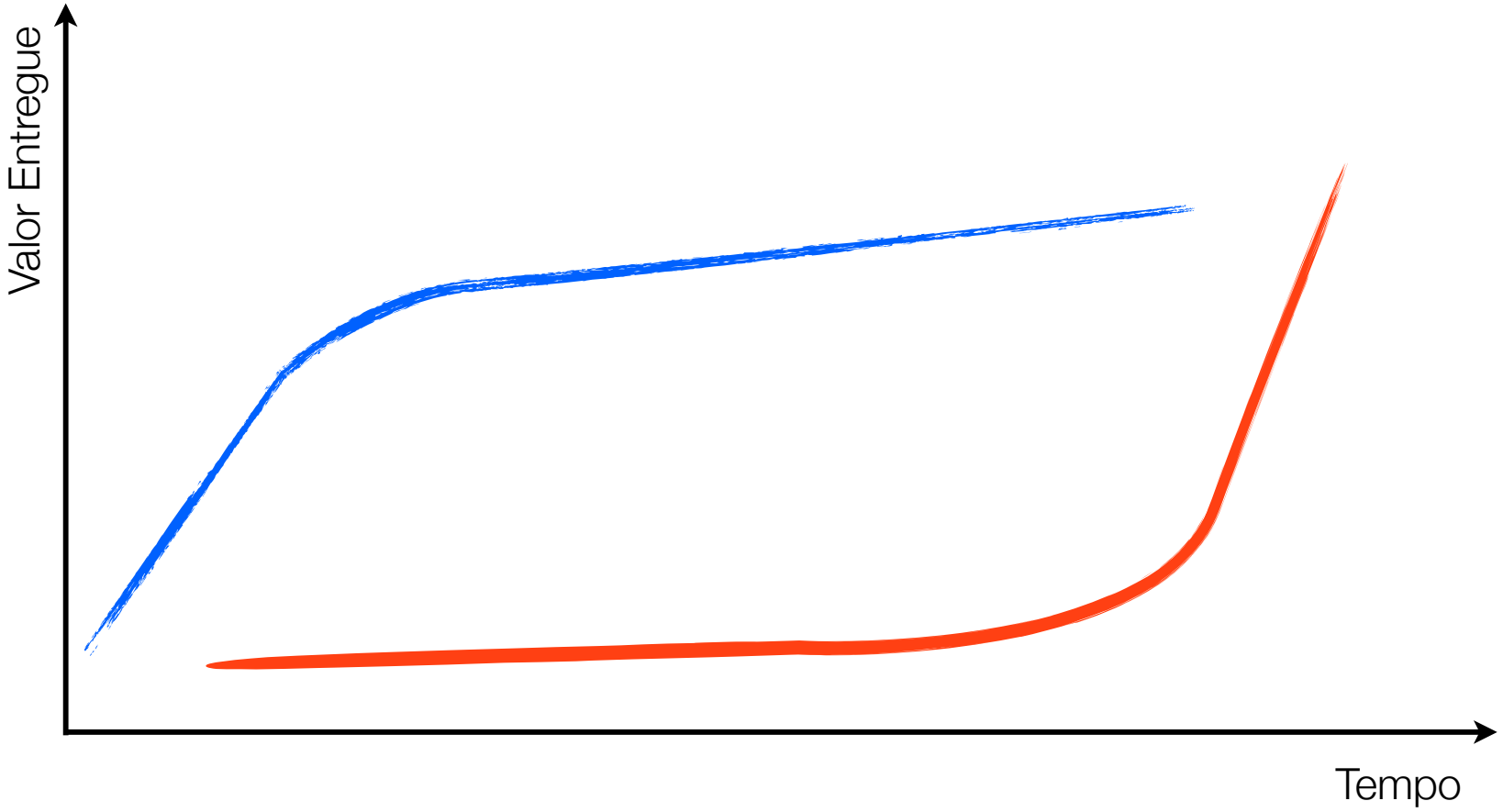
Custo de mudança



"Waterfall" (1970)

Novas abordagens 1990's

Entrega de Valor



"Waterfall" (1970)

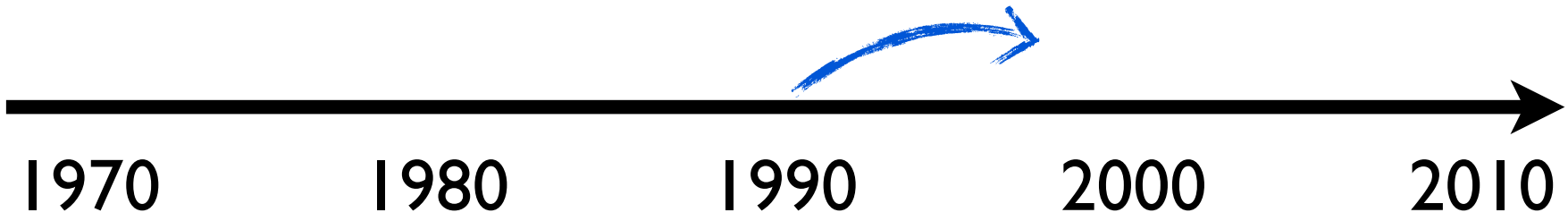
Novas abordagens 1990's

1990's

Iterações

=

Entrega de Valor


1970
1980
1990
2000
2010


1970
1980
1990
2000
2010

em 2001...

O Manifesto Ágil

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

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

That is, while there is value in the items on the right, we value the items on the left more.

Fonte: http://agilemanifesto.org

#comofaz

Como otimizar a entrega de valor?

Restrições:

Orçamento

Tempo

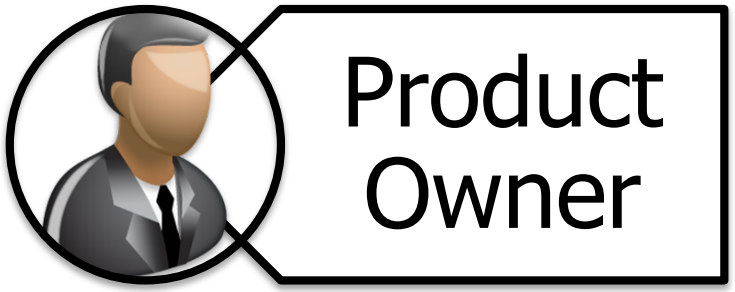
Qualidade

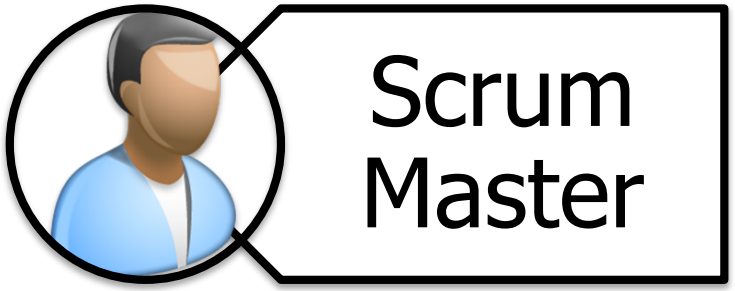
Significa que...

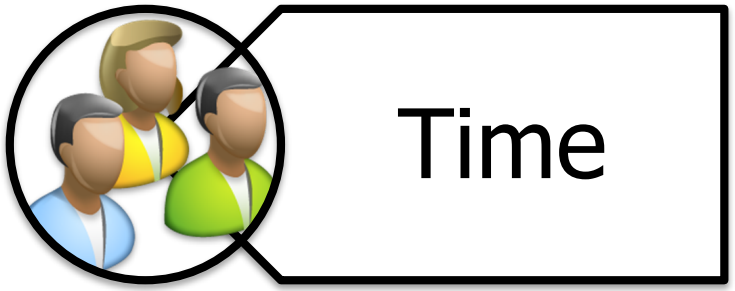
Escopo é negociável.

Scrum

Um time







Product Owner:

- Responsável pela visão do negócio
- Dono do product backlog
- Deve priorizar as estórias
- Pode ser o cliente final, um representante dele ou um analista de negócios

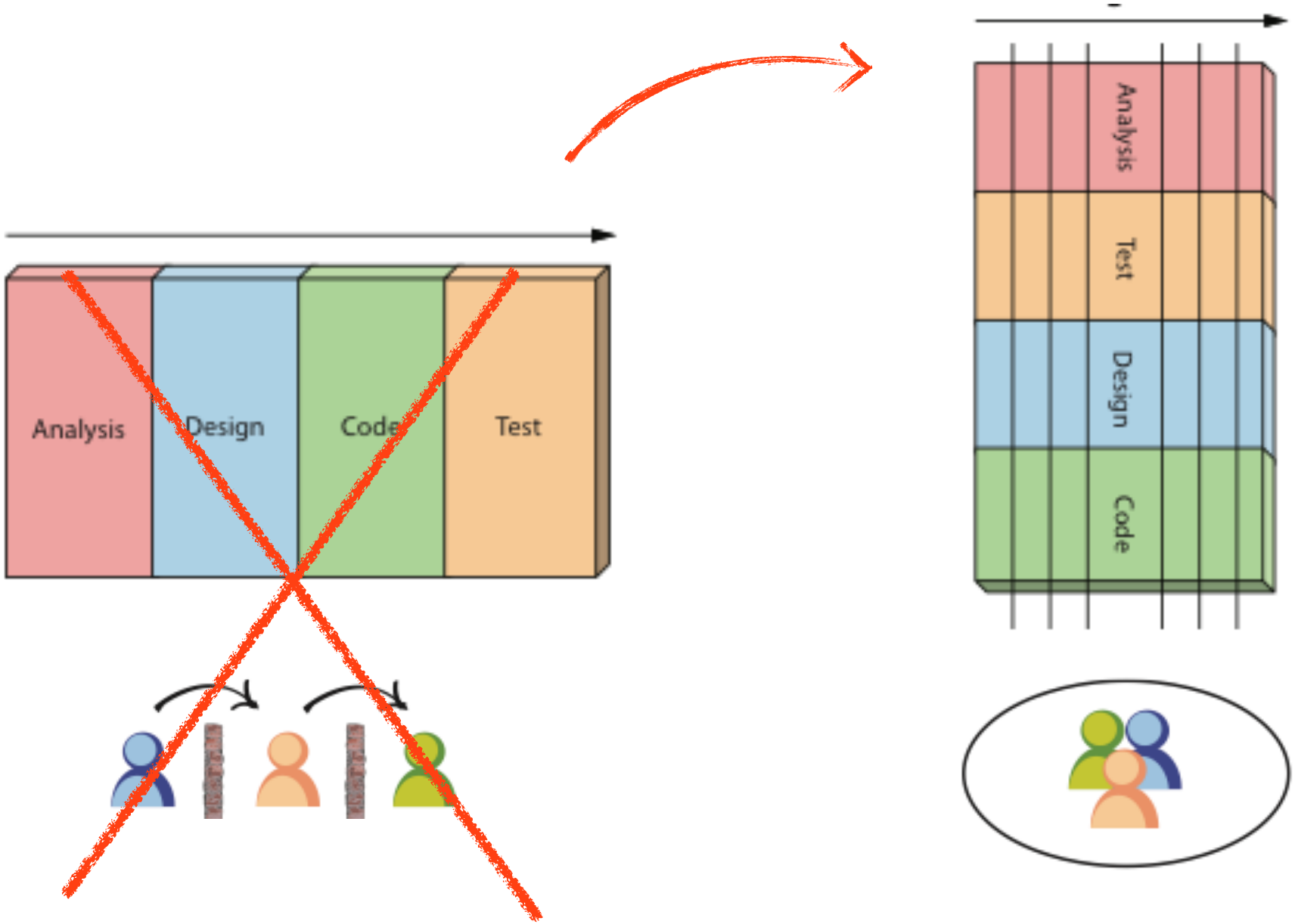
Scrum Master:

- Deve facilitar o time a ter a máxima produtividade possível
  - Deve blindar o time de eventos externos ao projeto
  - Deve resolver os impedimentos levantados pelo time

Time:

- Analisa, arquiteta, constroi, testa e implanta o software

Um time



Scrum

Release Planning | Sprint Planning | Daily (s) | Demo | Retrospectiva

Product Backlog

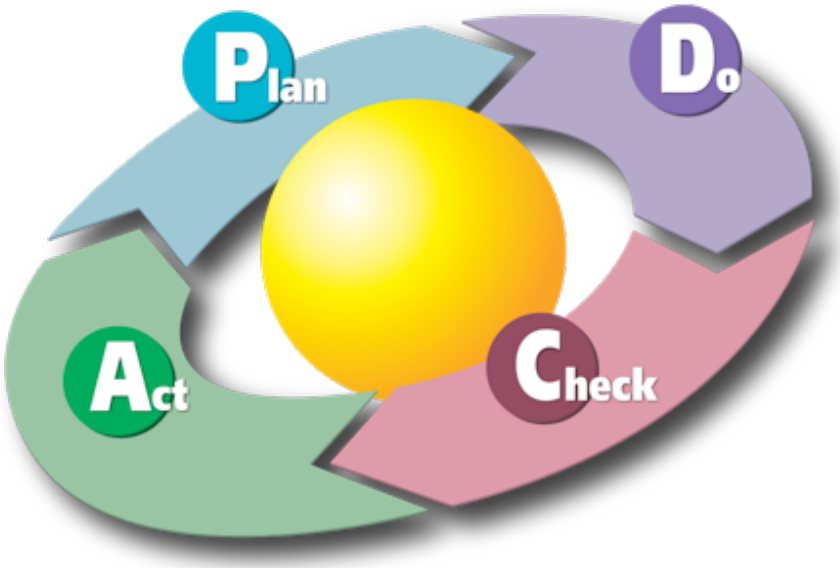
Sprint Backlog

Sincronização do time

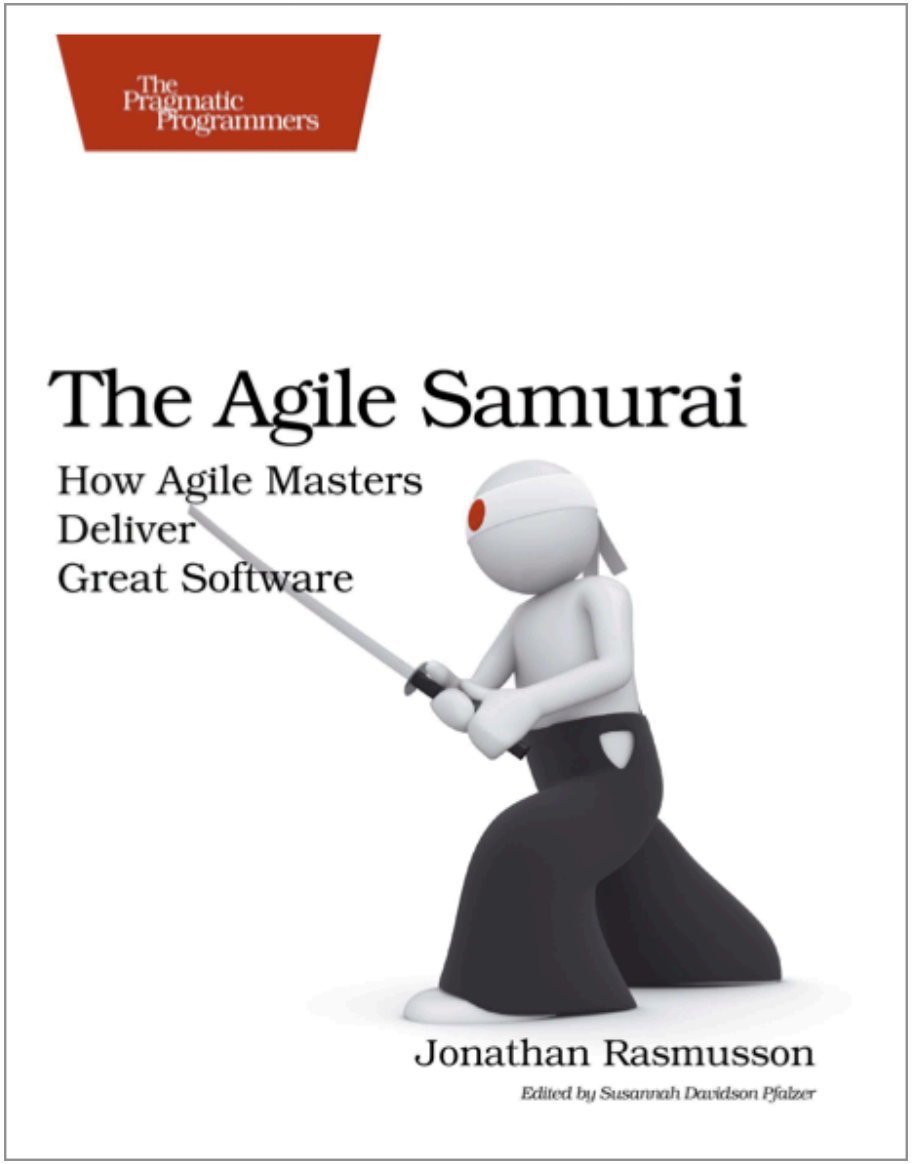
Software Funcional

Lista para Melhorias

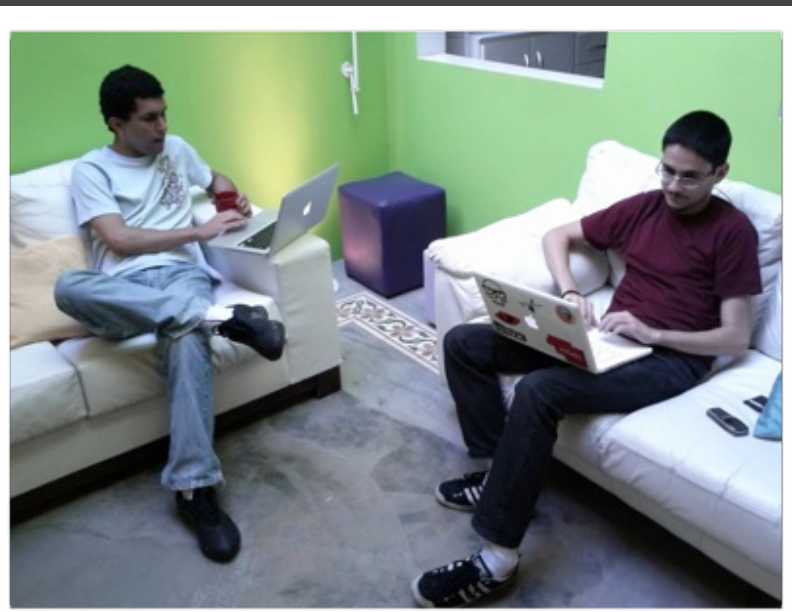
Retrospectiva



Constante busca pela Melhoria Contínua


The Pragmatic Programmers
The Agile Samurai
How Agile Masters Deliver Great Software
Jonathan Rasmusson
Edited by Susannah Davidson Pfalzer

blog.plataformatec.com.br




PULP FICTION

trabalhe@plataformatec.com.br

@plataformatec