Software Engineering TX00BW21-3001

Metropolia University of Applied Sciences 5 cr

Spring 2015

Markku Karhu and Ulla Suomela

Office: 1.145 **Tel. 09 7424 6389** **Email:** **markku.karhu@metropolia.fi**
Ulla Suomela <usuomela@gmail.com>

Class hours	40 h	Grade	Period 3	Wed 17:00 - 20.45 ETYB2327/Big Dry
Project	30 h	40%		Seminar Sat 5.3.2016 9:00 - 15:00
Seminar	24 h	20 %		
Homework	24 h			
Exam	2 h	40 %		

Software Engineering

Software - The process and its management

Project management: Software metrics, Estimation, Planning

System and software requirements analysis

Structured analysis, Object-oriented analysis

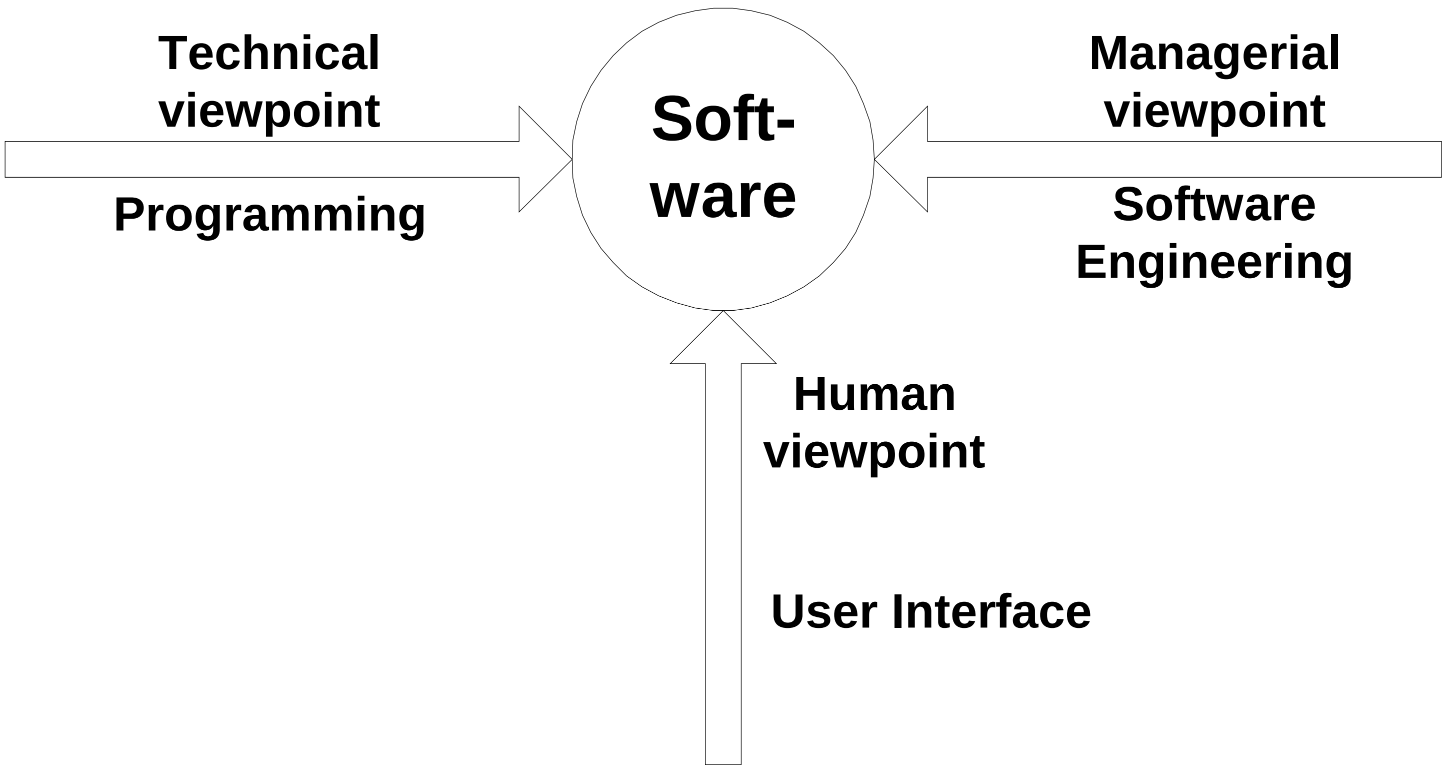
Techniques and formal methods

Textbooks

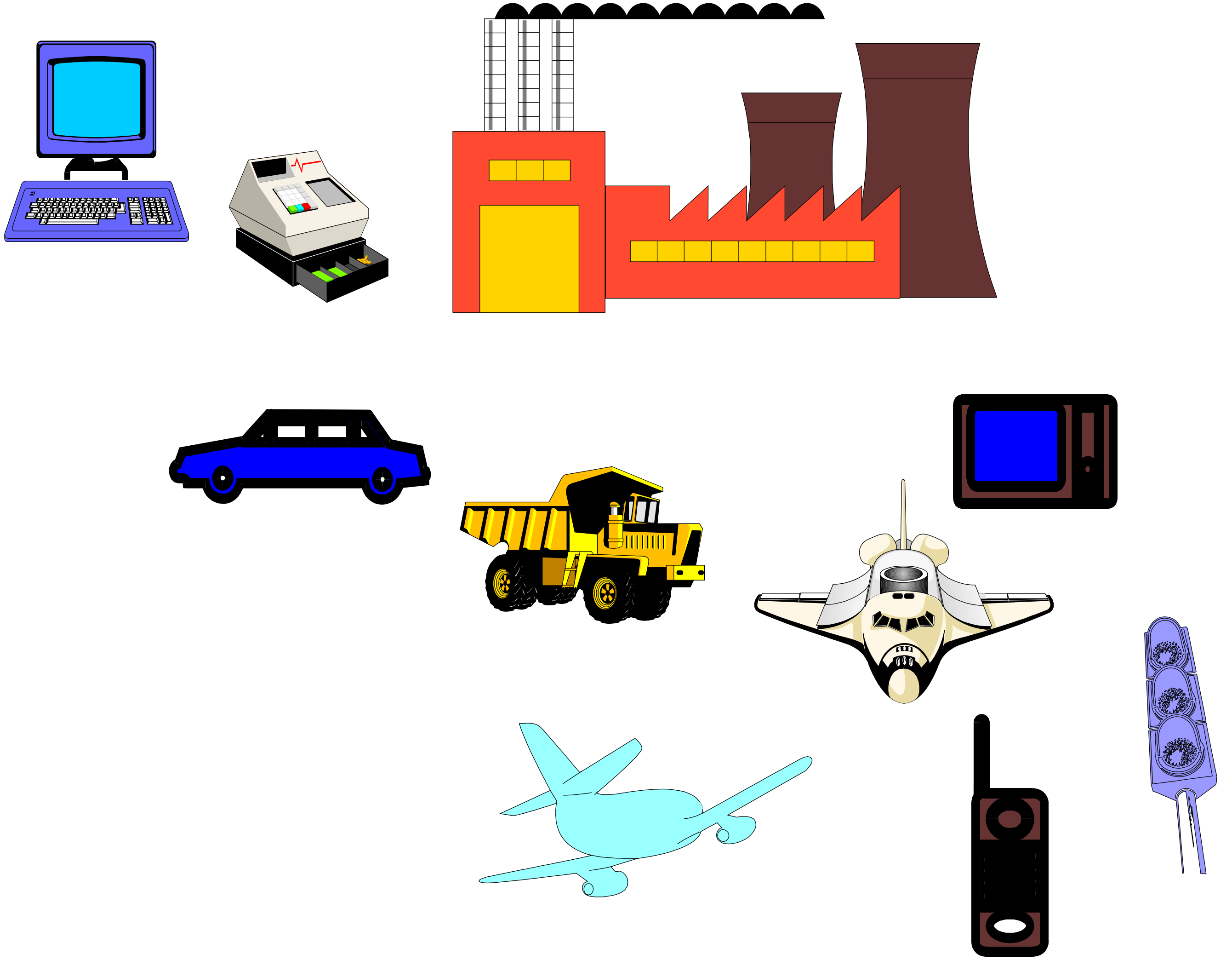
Pressman R.S.: Software Engineering, A Practitioner’s Approach, McGraw-Hill

Ian Sommerville: Software Engineering; http://iansommerville.com/software-engineering-book/

Points of View



Software Applications are everywhere



Software Applications

System software, Software tools

compilers, editors, file management utilities, operating systems, drivers, telecommunications processors

Real-time Software

monitors/analyses/controls real-world events - strict time constraints
paper mills, electricity, heat, traffic control, mobile phones

Business Software

payroll, accounts, inventory, order, contracts, budget management, recruitment

Engineering and Scientific Software

number crunching algorithms, CAD, modelling and simulation

Embedded Software

read-only memory, controls limited operations
microwave oven, fuel control, braking system

Artificial Intelligence Software (AI)

expert systems, knowledge-based systems, pattern recognition (image and voice), theorem proving, game playing, neural networks

non-numerical algorithms

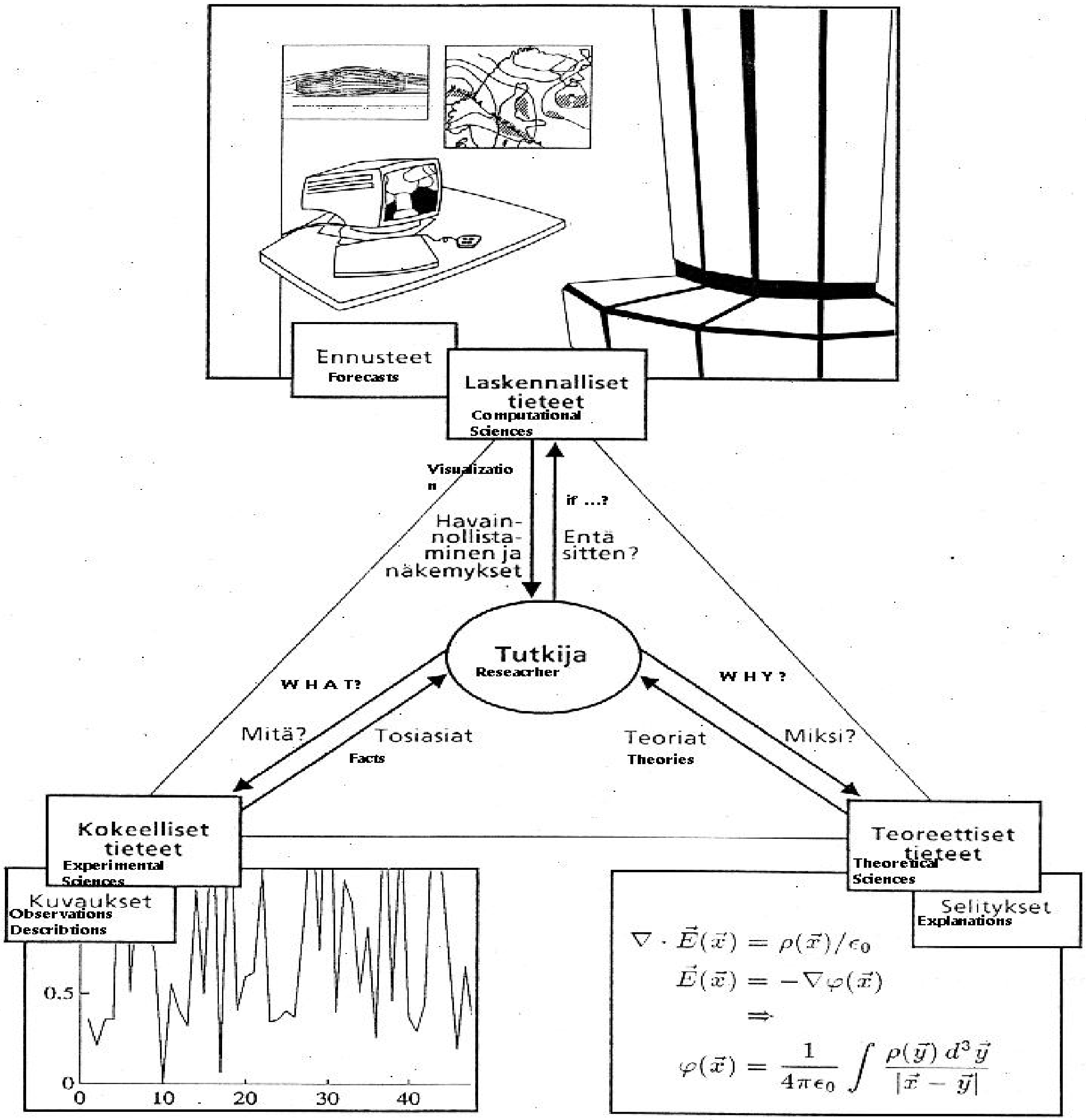
Personal Computer Software

word processing, spreadsheets, computer graphics, entertainment, database management, personal and business financial applications, external network

Web-based Applications

Complex array of content and functionality to a broad population of end-users who may be unknown

Games


Ennusteet
Forecasts
Laskennalliset tieteet
Computational Sciences
Visualization
if ...?
Havain-nollista-minen ja näkemykset
Entä sitten?
Tutkija
Reseacrher
WHAT?
Mitä?
Tosiasiat
Facts
WHY?
Teoriat
Theories
Miksi?
Kokeelliset tieteet
Experimental Sciences
Kuvaukset
Observations Describtions
Teoreettiset tieteet
Theoretical Sciences
Selitykset
Explanations
$\nabla \cdot \vec{E}(\vec{x}) = \rho(\vec{x})/\epsilon_0$
$\vec{E}(\vec{x}) = -\nabla\varphi(\vec{x})$
$\Rightarrow$
$\varphi(\vec{x}) = \frac{1}{4\pi\epsilon_0}\int \frac{\rho(\vec{y})\, d^3\vec{y}}{|\vec{x}-\vec{y}|}$
0.5
0
0
10
20
30
40

Challenges for Software Engineering

Complexity

do good planning, with few interfaces and clear and simple coding

Invisible

support project management, milestones, quality planning and assurance

Changeable

prepare for maintenance, updating and upgrading

Uniqueness

support reusability when possible, avoid unique solutions

Unscalable

properties of small applications do no apply to greater applications

Discontinuous

small error can break the whole system

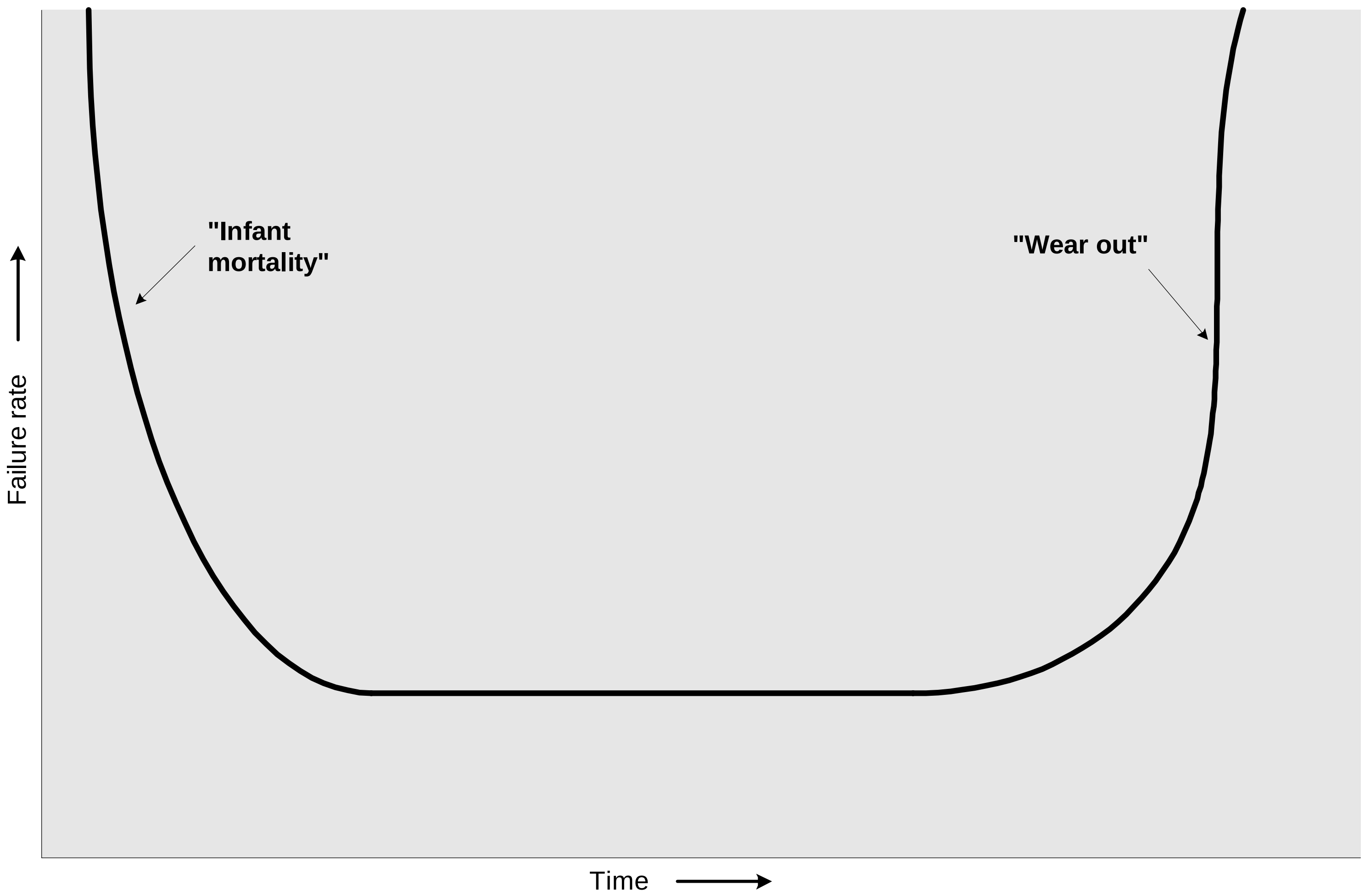


FIGURE 1.2. Failure curve for hardware.

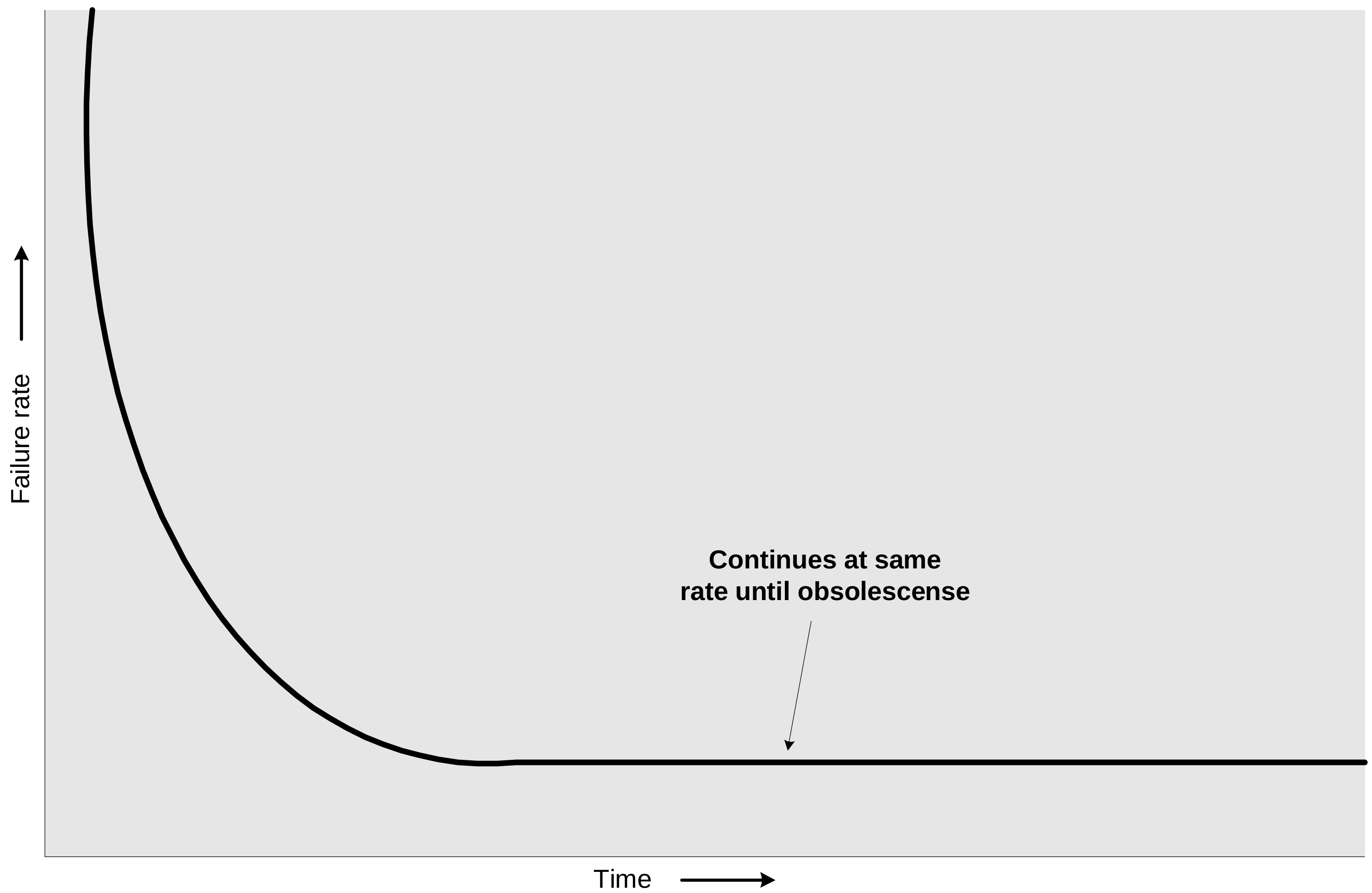


FIGURE 1.3. Failure curve for software (idealized).

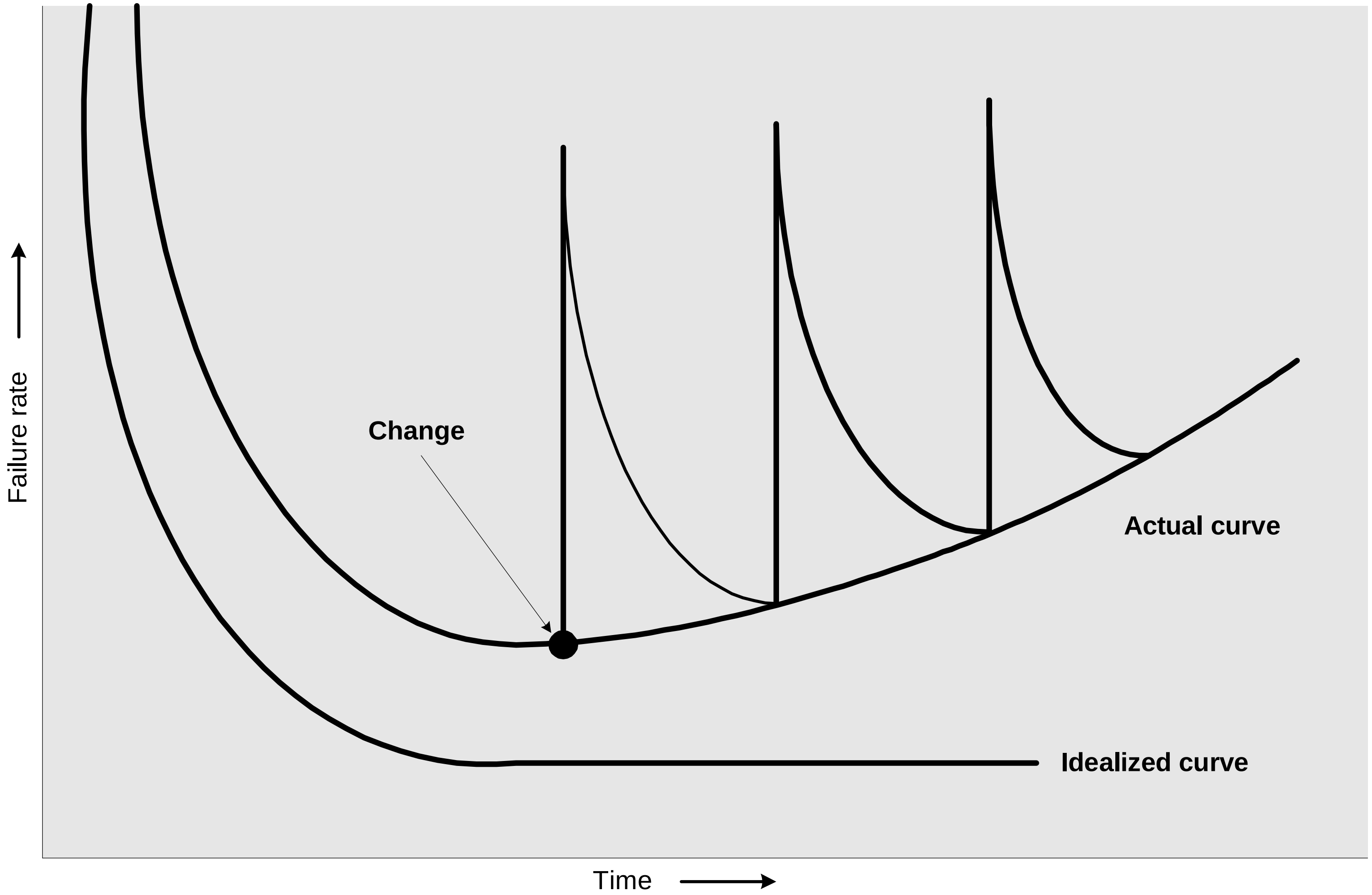


FIGURE 1.4. Actual failure curve for software.

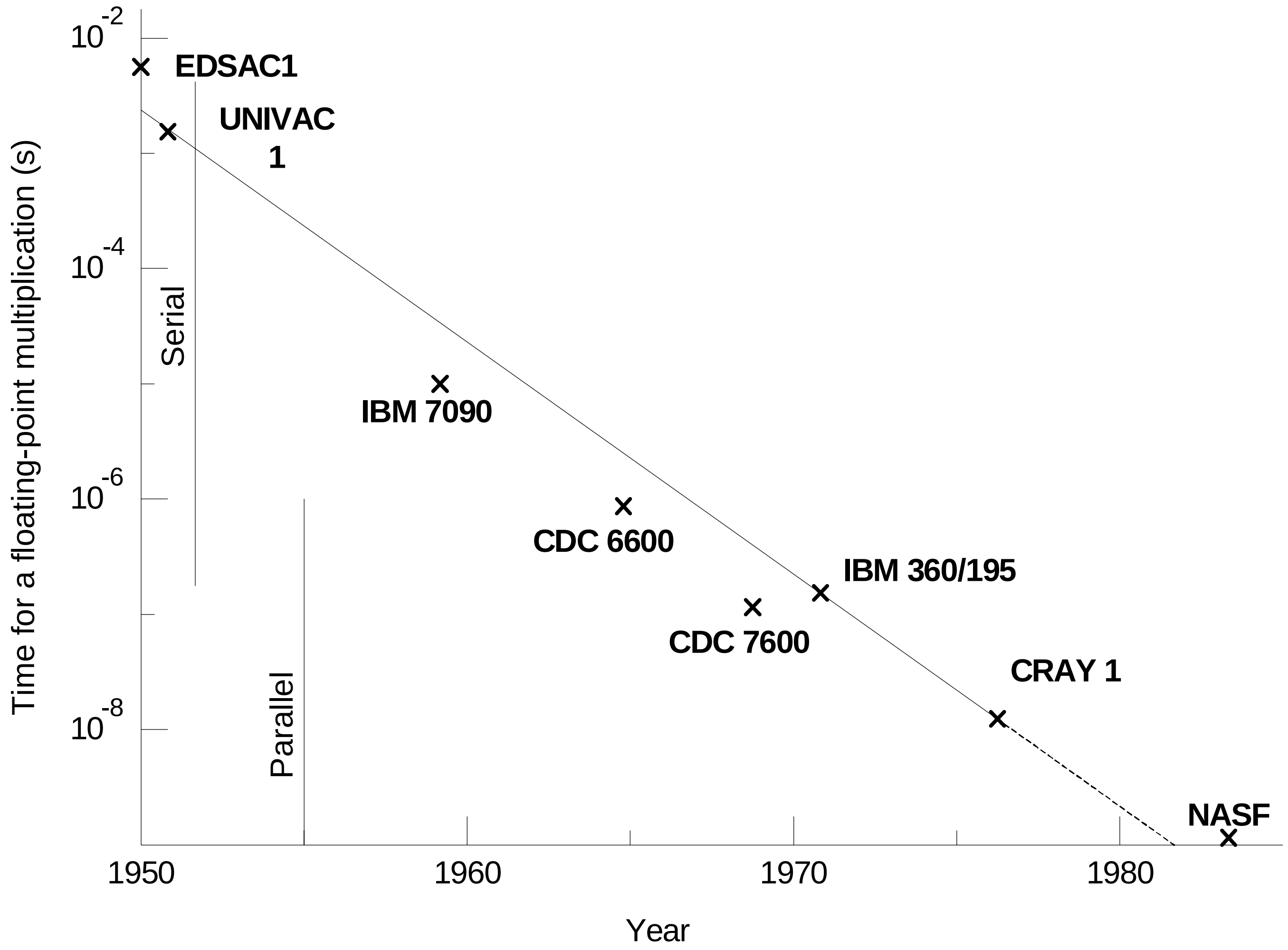
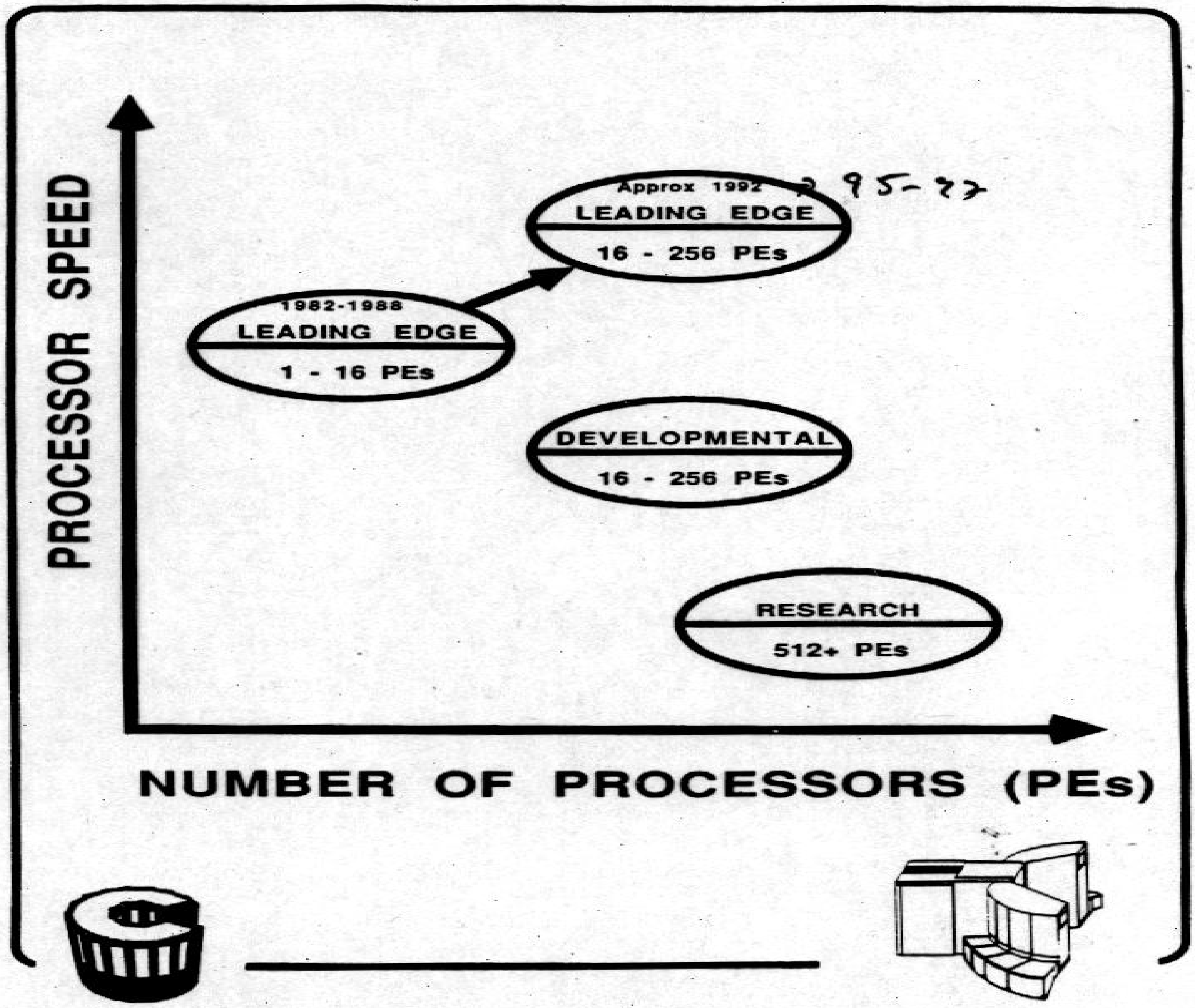


FIGURE 1.1 The history of computer arithmetic speed since 1950.
showing an increase of a factor of 10 in 5 years


PROCESSOR SPEED
Approx 1992
95-97
LEADING EDGE
16 - 256 PEs
1982-1988
LEADING EDGE
1 - 16 PEs
DEVELOPMENTAL
16 - 256 PEs
RESEARCH
512+ PEs
NUMBER OF PROCESSORS (PEs)

Moore’s law

”The observation that the logic density of silicon integrated circuits has closely followed the curve (bits per square inch) = 2^(t - 1962) where t is time in years; that is, the amount of information storable on a given amount of silicon has roughly doubled every year since the technology was invented. This relation, first uttered in 1964 by semiconductor engineer Gordon Moore (who co-founded Intel four years later) held until the late 1970s, at which point the doubling period slowed to 18 months. The doubling period remained at that value through time of writing (late 1999).
Moore's Law is apparently self-fulfilling. The implication is that somebody, somewhere is going to be able to build a better chip than you if you rest on your laurels, so you'd better start pushing hard on the problem. See also Parkinson's Law of Data and Gates's Law.

Parkinson's Law of Data

"Data expands to fill the space available for storage"; buying more memory encourages the use of more memory-intensive techniques. It has been observed since the mid-1980s that the memory usage of evolving systems tends to double roughly once every 18 months. Fortunately, memory density available for constant dollars also tends to about double once every 18 months (see Moore's Law); unfortunately, the laws of physics guarantee that the latter cannot continue indefinitely.

Gates's Law

"The speed of software halves every 18 months." This oft-cited law is an ironic comment on the tendency of software bloat to outpace the every-18-month doubling in hardware capacity per dollar predicted by Moore's Law. The reference is to Bill Gates; Microsoft is widely considered among the worst if not the worst of the perpetrators of bloat.

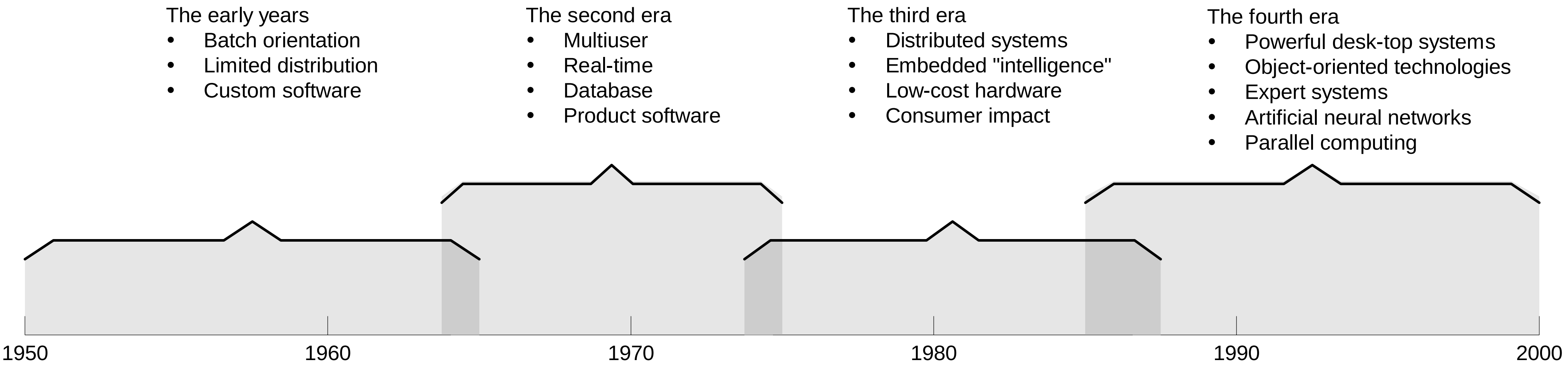
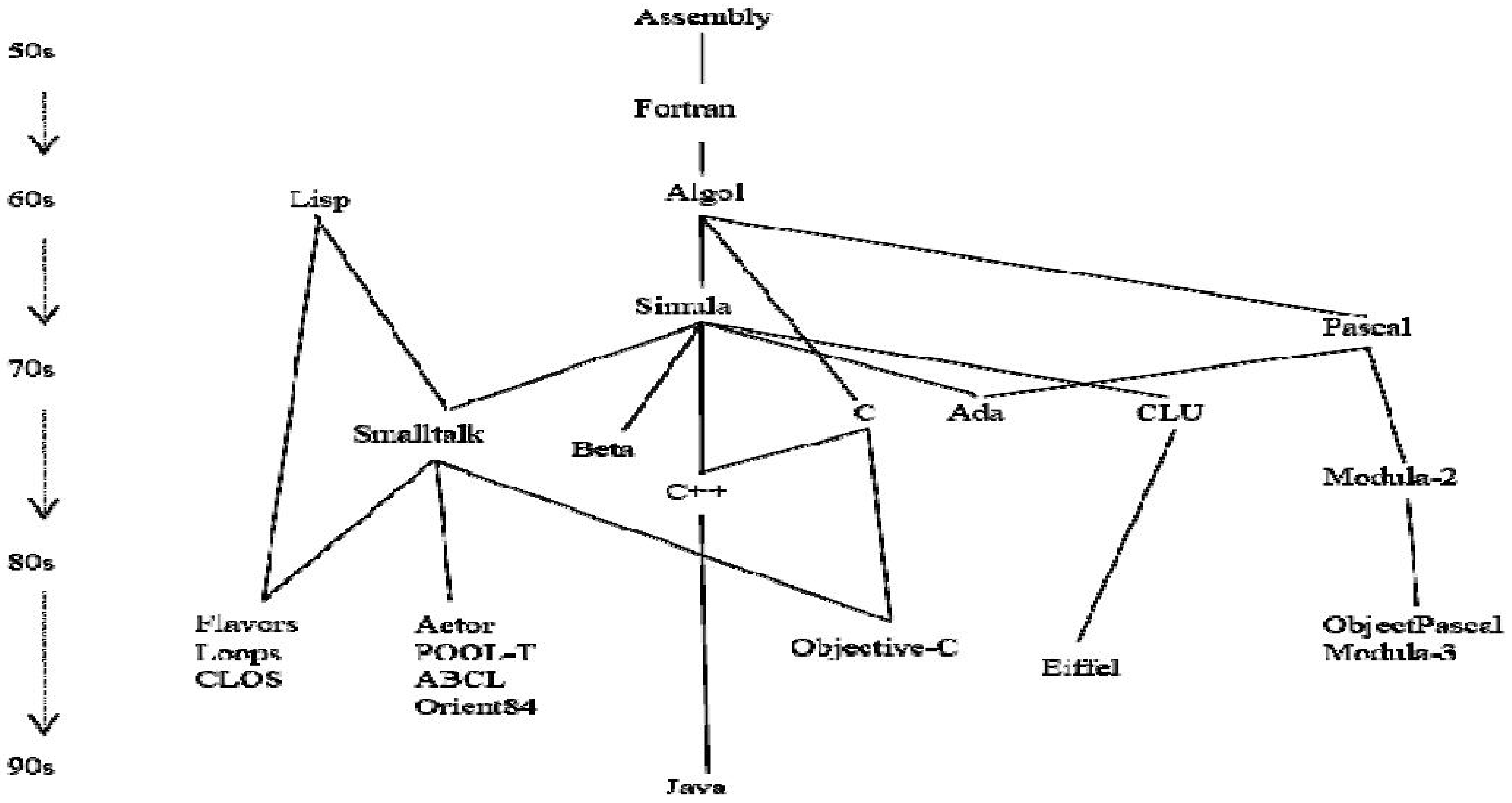
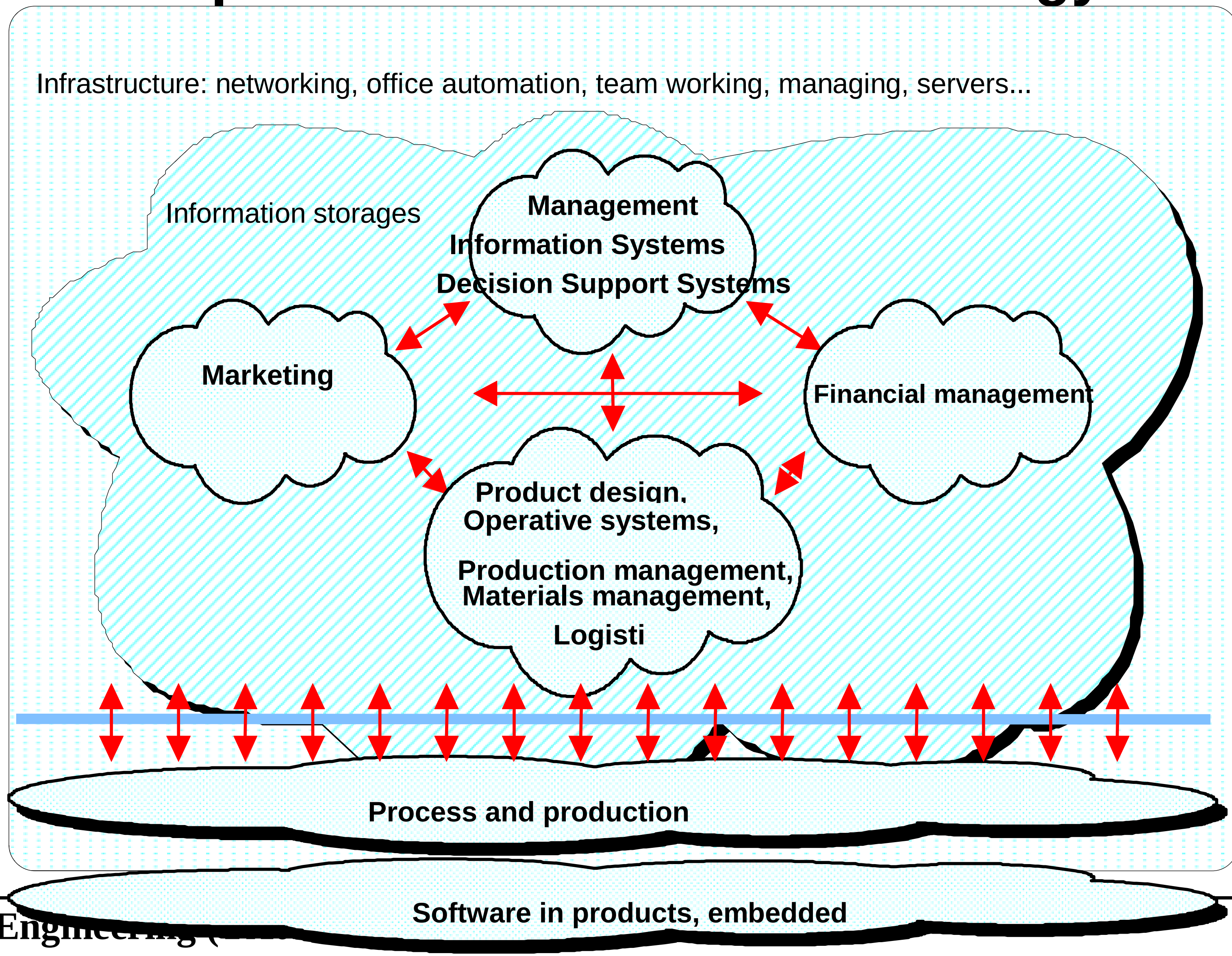


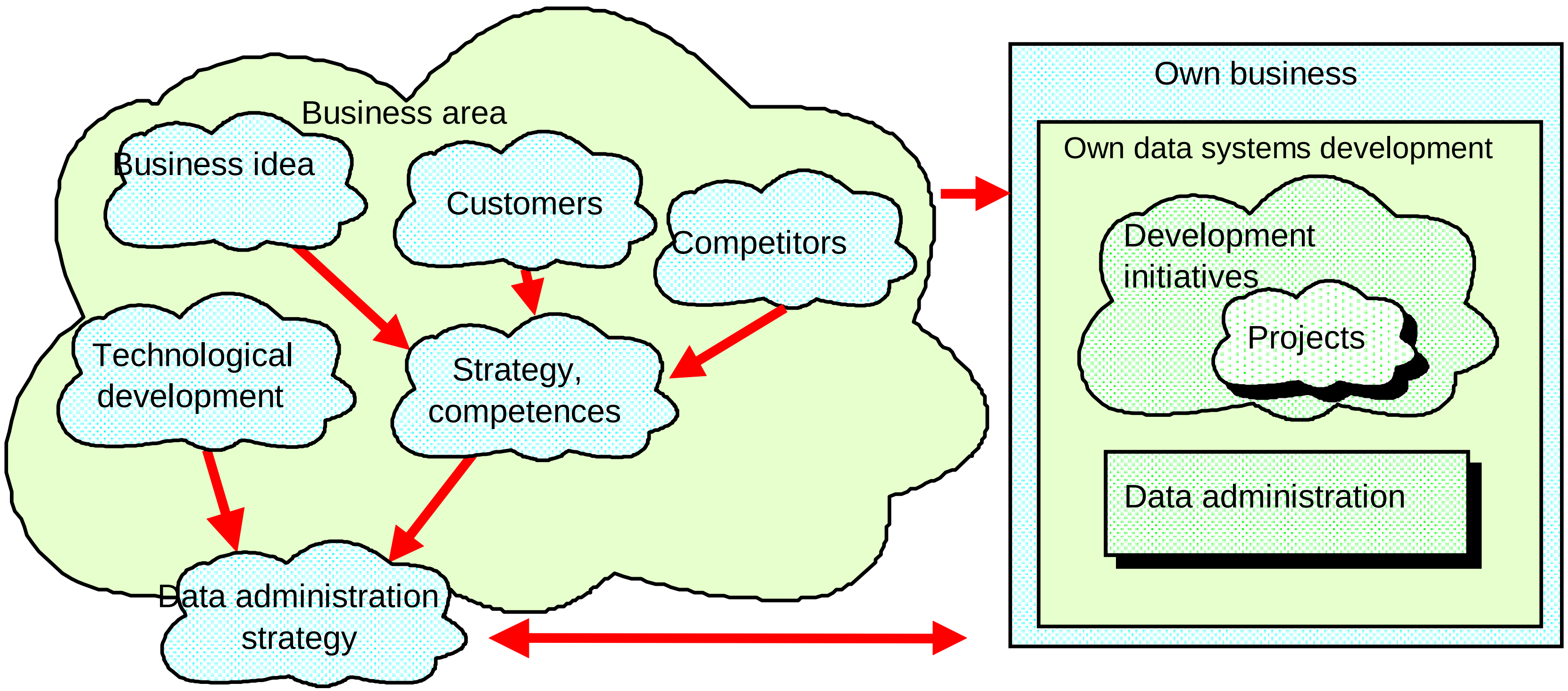
FIGURE 1.1. Evolution of software.


Assembly
Fortran
Algol
Lisp
Simula
Pascal
Smalltalk
Beta
C
Ada
CLU
C++
Modula-2
Flavors
Loops
CLOS
Actor
POOL-T
ABCL
Orient84
Objective-C
Eiffel
ObjectPascal
Modula-3
Java
50s
60s
70s
80s
90s

Corporate Information Technology



Business areas are changing



Beating the Software Crisis

- **Corporations are drowning in their own data**
- **The failure lies in software**
- **Most software are delivered late and over budget**
- **We need better software and we need it faster**
- **This is known as the software crisis**

How Software is Constructed

- **Building better software is a major challenge**
- **There have been many responses to this challenge**

Building Programs

- **Small programs can be build as a single procedure**
- **But this approach doesn’t work for larger systems**

Modular Programming

- **Larger systems require modular programming**
- **Subroutines support modular programming**
- **But modular programming requires discipline**

Structured Programming

- **Structured programming provides that discipline**
- **Functional decomposition plays a central role**
- **Structural programming is useful but limited**

Computer-Aided Software Engineering (CASE)

- **CASE automates the structured programming**
- **This helps, but it doesn’t go far enough**

Fourth-generation Languages

- **4GLs can generate programs automatically**
- **But they only work for simple, familiar problems**

Managing Information

- **Modularization has focused on procedures**
- **Data must also be modularised**

Data within Programs

- **Subroutines can share small amounts of data**
- **But sharing a collection of data leads to mysterious errors and unpredictable behaviour**
- **The solution lies in hiding information**

Data Outside of Programs

- **Some programs don’t need preserve data**
- **But most large programs have to reuse data**
- **Data can easily be preserved in files**
- **But that doesn’t work when data must be shared**

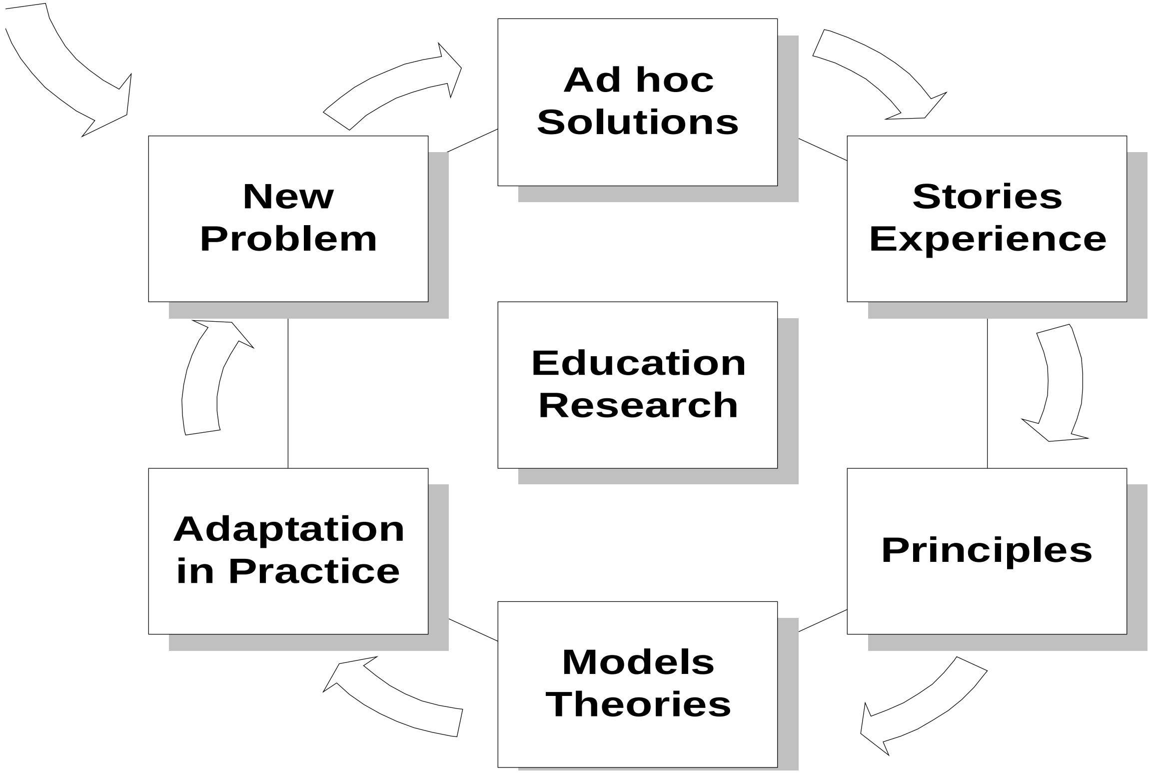
Sharing Data

- **Shared data require a database management system (DBMS)**
- **Databases contain structure as well as data**
- **The network model extended the hierarchic model**
- **Fixed data structures reduce flexibility**
- **The relational model removes most of the structure**
- **Removing structure has costs, too**

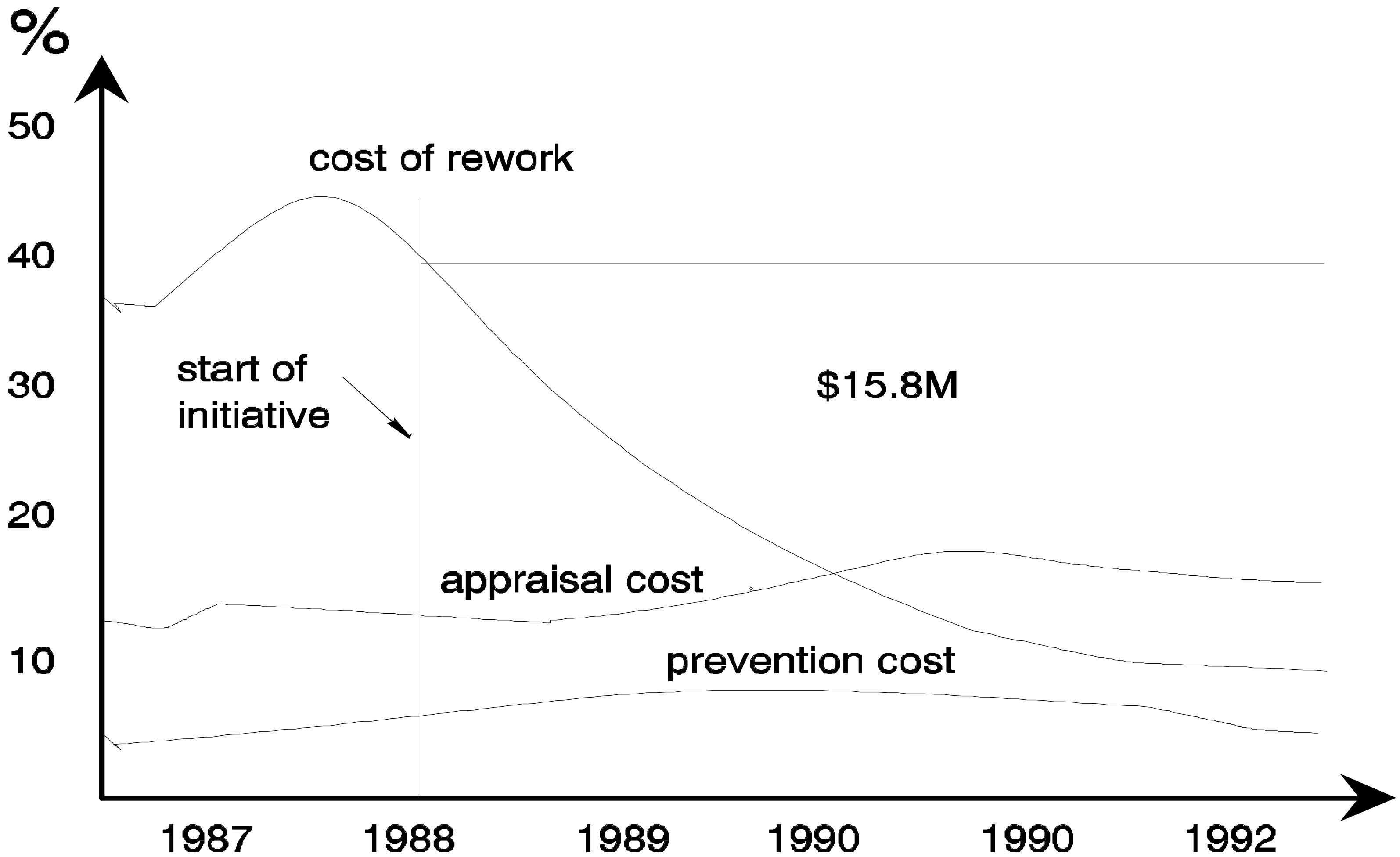
The Object-Oriented Approach

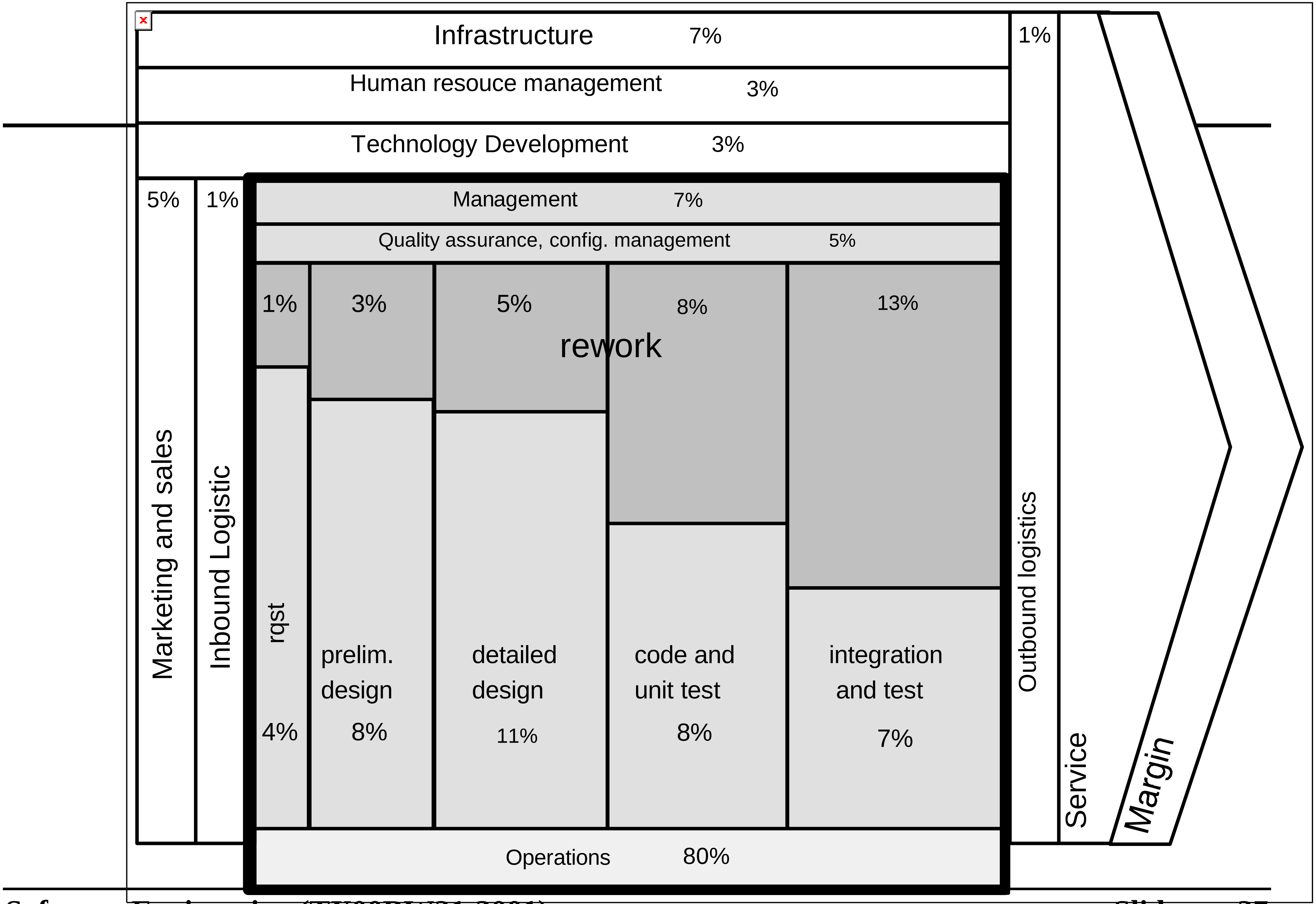
- **None of these effort has solved the software crisis**
- **We need a new approach to building systems**
- **Object-oriented technology is the new approach**

Development Cycle of a Discipline



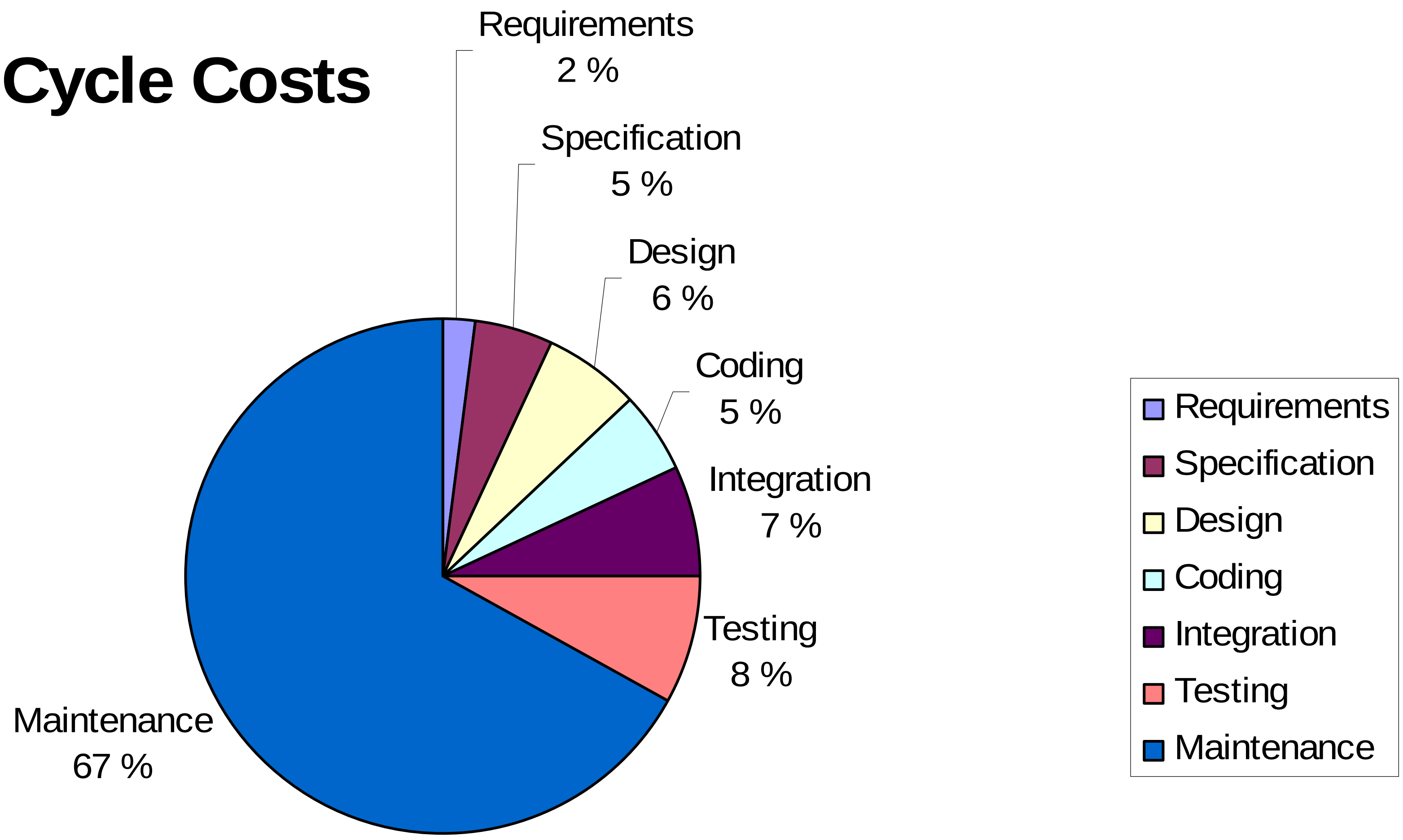
Cost of rework due to errors, failures

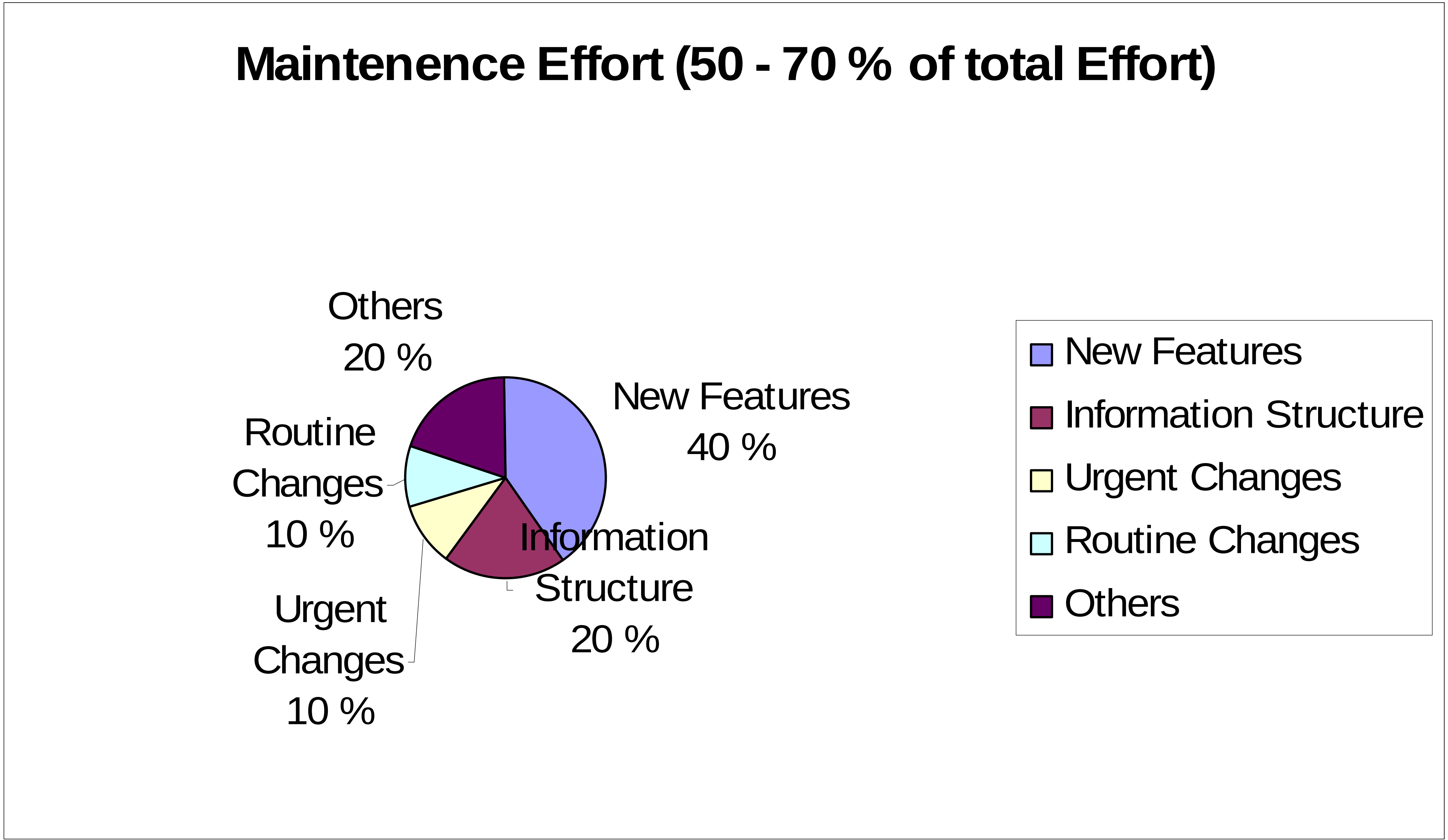




Infrastructure 7%
Human resouce management 3%
Technology Development 3%
1%
Management 7%
Quality assurance, config. management 5%
5%
1%
Marketing and sales
Inbound Logistic
1%
3%
5%
8%
13%
rework
rqst
4%
prelim. design
8%
detailed design
11%
code and unit test
8%
integration and test
7%
Operations 80%
Outbound logistics
Service
Margin

Life Cycle Costs

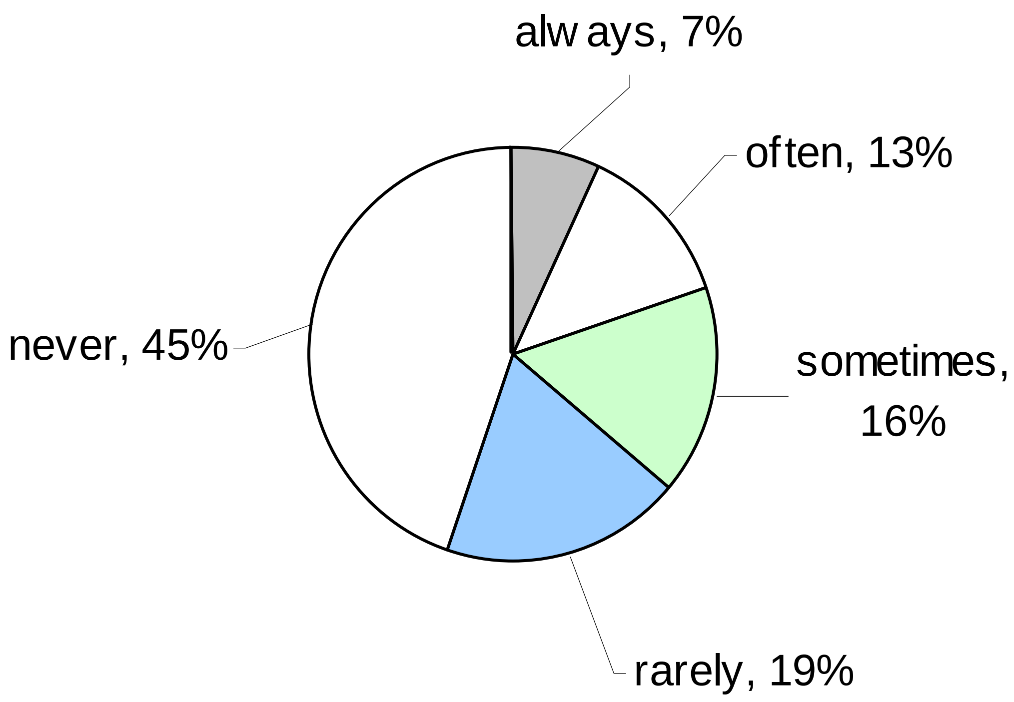



Maintenence Effort (50 - 70 % of total Effort)
Others
20 %
New Features
40 %
Routine
Changes
10 %
Information
Structure
20 %
Urgent
Changes
10 %
New Features
Information Structure
Urgent Changes
Routine Changes
Others

Facts about environment

- Actual use of requested features. ([Johnson02])

http://www.martinfowler.com/articles/xp2002.html



A General View of SW Engineering I

<u>WHAT?</u>

System analysis

- **classic life cycle: define the role of each element in a computer-based system**

Software project planning

- **scope of SW, analyse risks, allocate resources, estimate costs, define work tasks and schedule**

Requirement analysis

- **detailed definition of the information domain and function of the SW**

A General View of SW Engineering II

HOW?

Software design

- **Requirements into a set of representations (graphical, tabular, language-based) data structure, architecture, algorithmic procedure, user interface**

Coding

- **representations into an artificial language executable by the computer**

Software testing

- **uncover defects in function, in logic, in implementation**

A General View of SW Engineering III

<u>CHANGE?</u>

Correction

corrective maintenance

Adaptation

over time: adaptive maintenance (CPU, operating system, peripherals)

Enhancement

preventive maintenance: extends the original function requirements

Re-engineering

ageing software plant - reverse engineering to improve the SW’s internal working

Software Engineering Paradigms

Definition: ”The establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines”

I Classic Life Cycle

II Prototyping

III Spiral Model

IV Fourth-Generation Techniques: 4GT/4GL

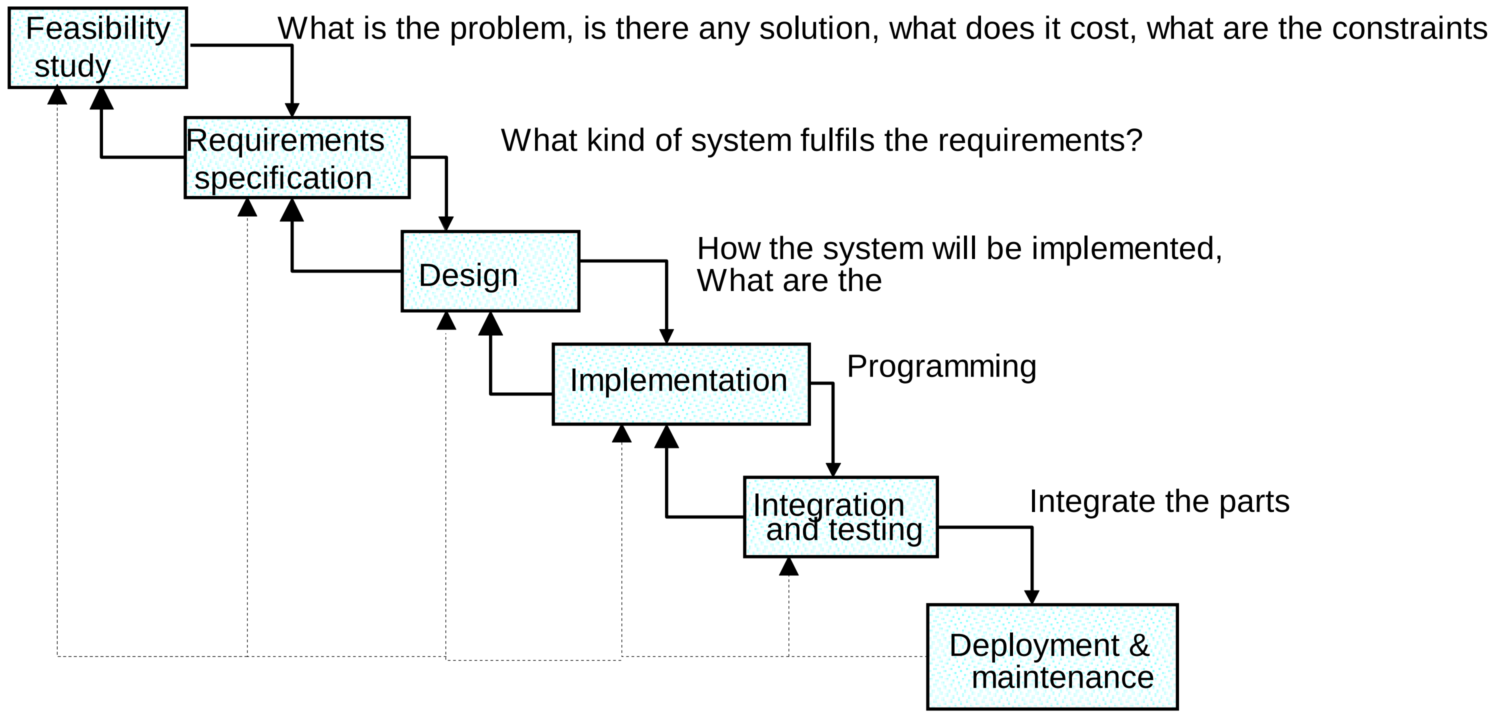
V Agile (Incremental) Methods

I Classic Life Cycle

- **System engineering and analysis: requirements for all system elements: hardware, people, databases**
- **Software requirements analysis: understanding the information domain, function, performance, interfacing; documented and reviewed with the user**
- **Design: data-, software-, hardware, and network architecture, procedural detail, interface**
- **Coding: detailed design -> mechanistic coding**
- **Testing: logical internals, functional externals; validation, verification**
- **Maintenance: errors, environmental changes, functional enhancements**

Classic Life Cycle

See the article MANAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS by *Dr. Winston W. Rovce*



II Prototyping

- **Start with general objectives, but identify detailed input, processing, or output requirements later**
- **Efficiency of an algorithm**
- **Adaptability of an operating system**
- **Form of a human-machine interaction**

A model of the software may be

- **A paper prototype or PC-based model**
- **A working prototype that implements some subset of the SW**
- **An existing program but some features are not implemented**

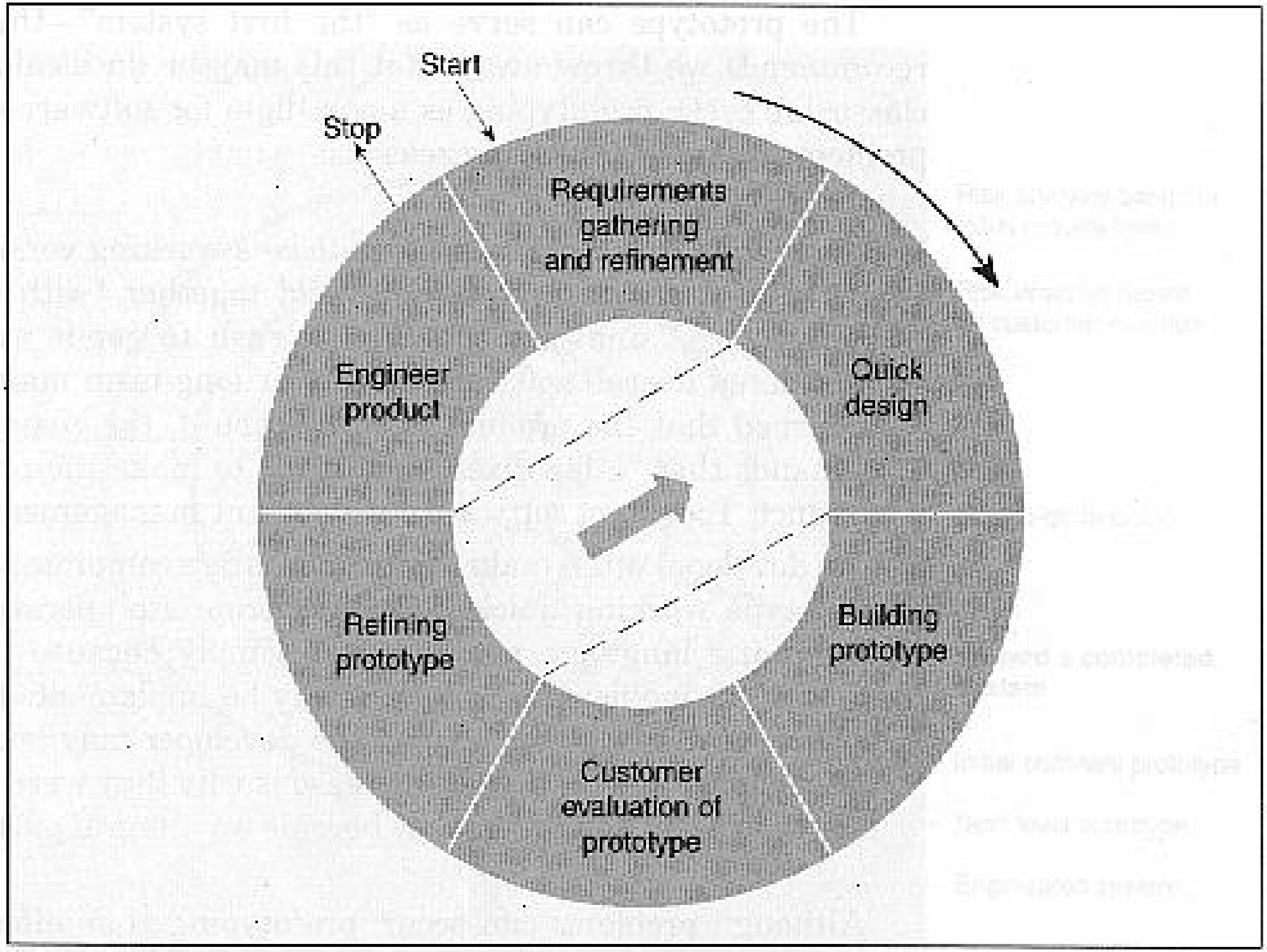


FIGURE 1.8.
Prototyping.

III Spiral Model

Encompass the best features of classic life cycle and prototyping, and risk analysis

1.Planning - determination of objectives, alternatives, and constraints
2.Risk analysis - analysis of alternatives and identification and resolution of risks
3.Engineering - development of the next-level product
4.Customer evaluation - assessment of the results of engineering

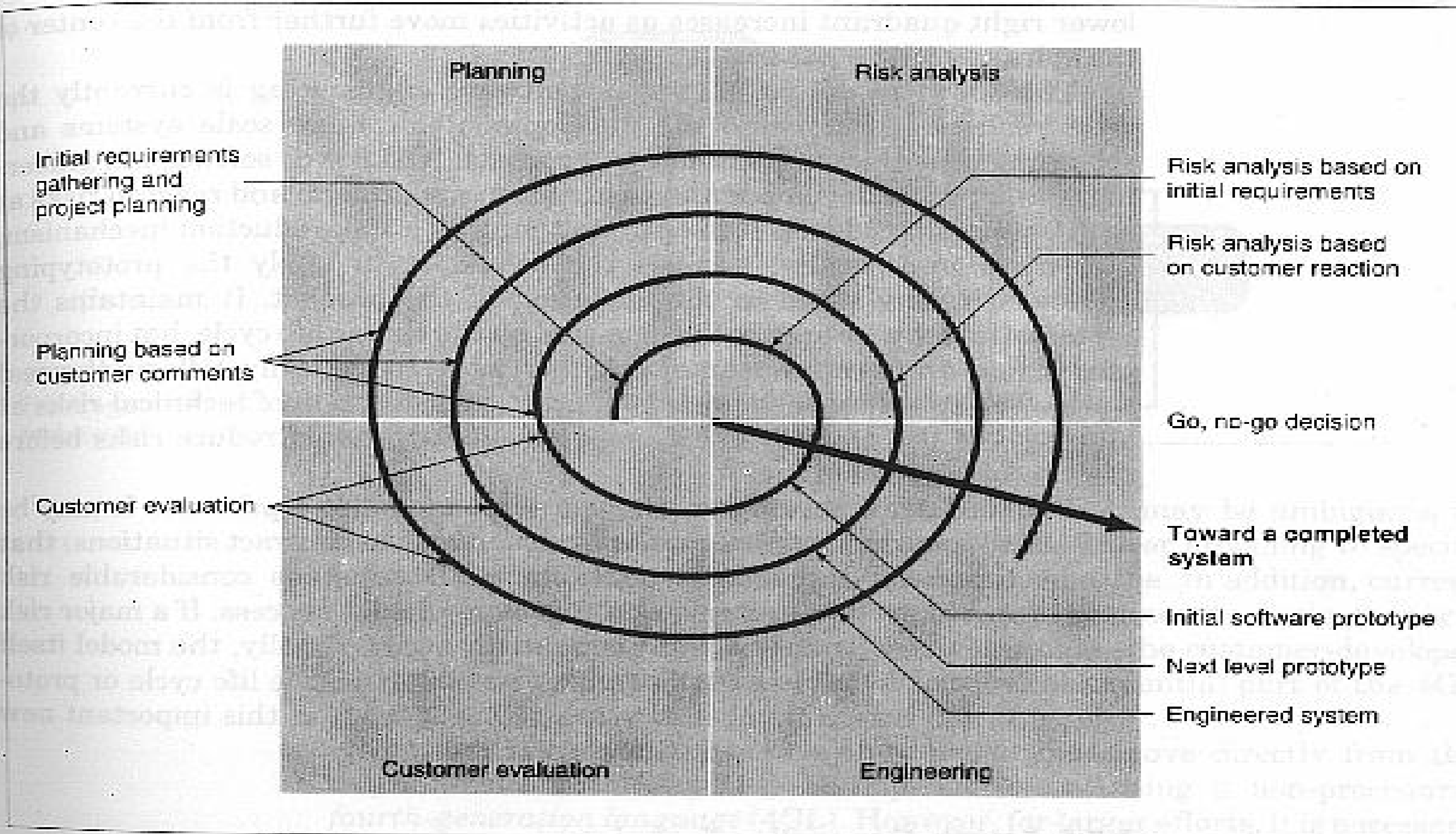
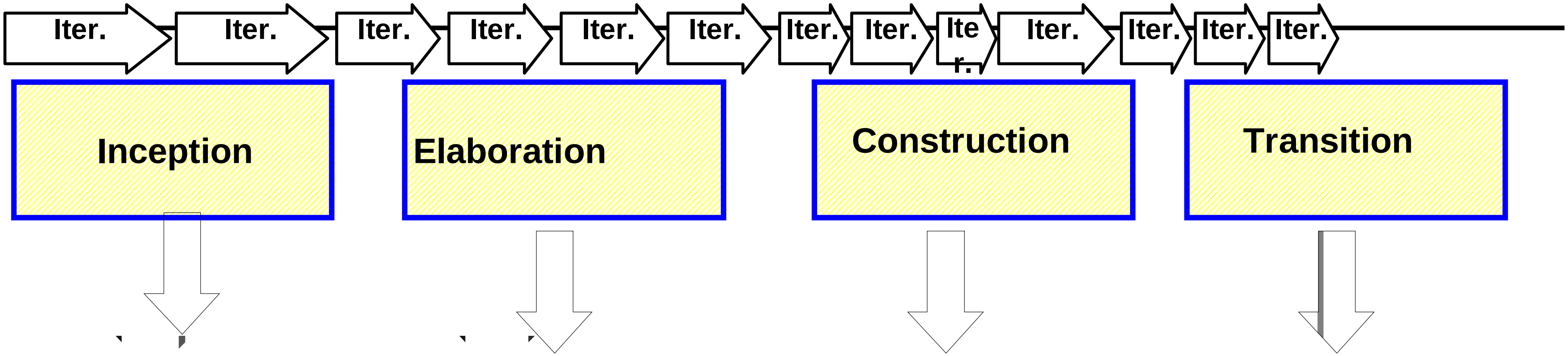


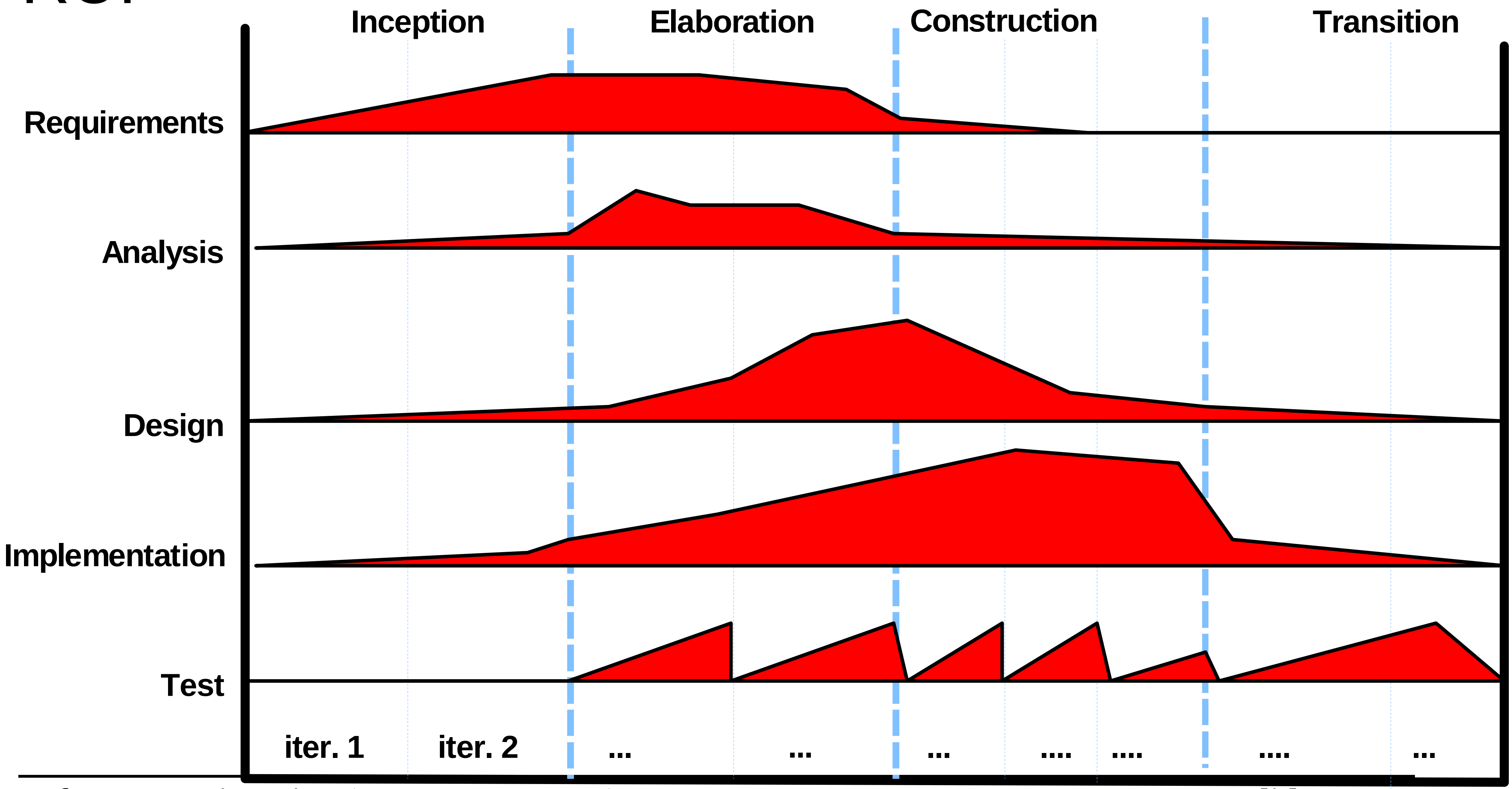
FIGURE 1.9. The spiral model.

RUP (Rational Unified Process)



A vision document	A use case model (al least 80 % complete)	The software product integrated on the adequate platforms	The transition of the software to the user community
An initial use case model (10 – 20 % complete)	Supplementary requirements	The use manuals	Beta testing
An initial project glossary	A software architecture description	A description of the current release	Parallel operation with the legacy system that is replaced
An initial risk assessment	An executable architectural prototype		Conversion of operational databases
A project plan, showing phases and iterations	A revised risk list and revised business case		Training of users and maintainers
One or several prototypes	A development plan for the overall project		Roll out the product
	An updates development case specifying the process to be used		
	A preliminary user manual		

RUP


Inception
Elaboration
Construction
Transition
Requirements
Analysis
Design
Implementation
Test
iter. 1
iter. 2
...
...
...
....
....
....
...

IV Fourth-Generation Techniques:4GT/4GL

To specify some characteristic of SW at a high level. Automatic generation of skeleton code

<u>4GT Software Development Environments:</u>

- **Nonprocedural languages for database query**
- **Report generation**
- **Data manipulation**
- **Screen interaction**
- **Definition**
- **Code generation**
- **High-level graphics capability**
- **Spreadsheet capability**

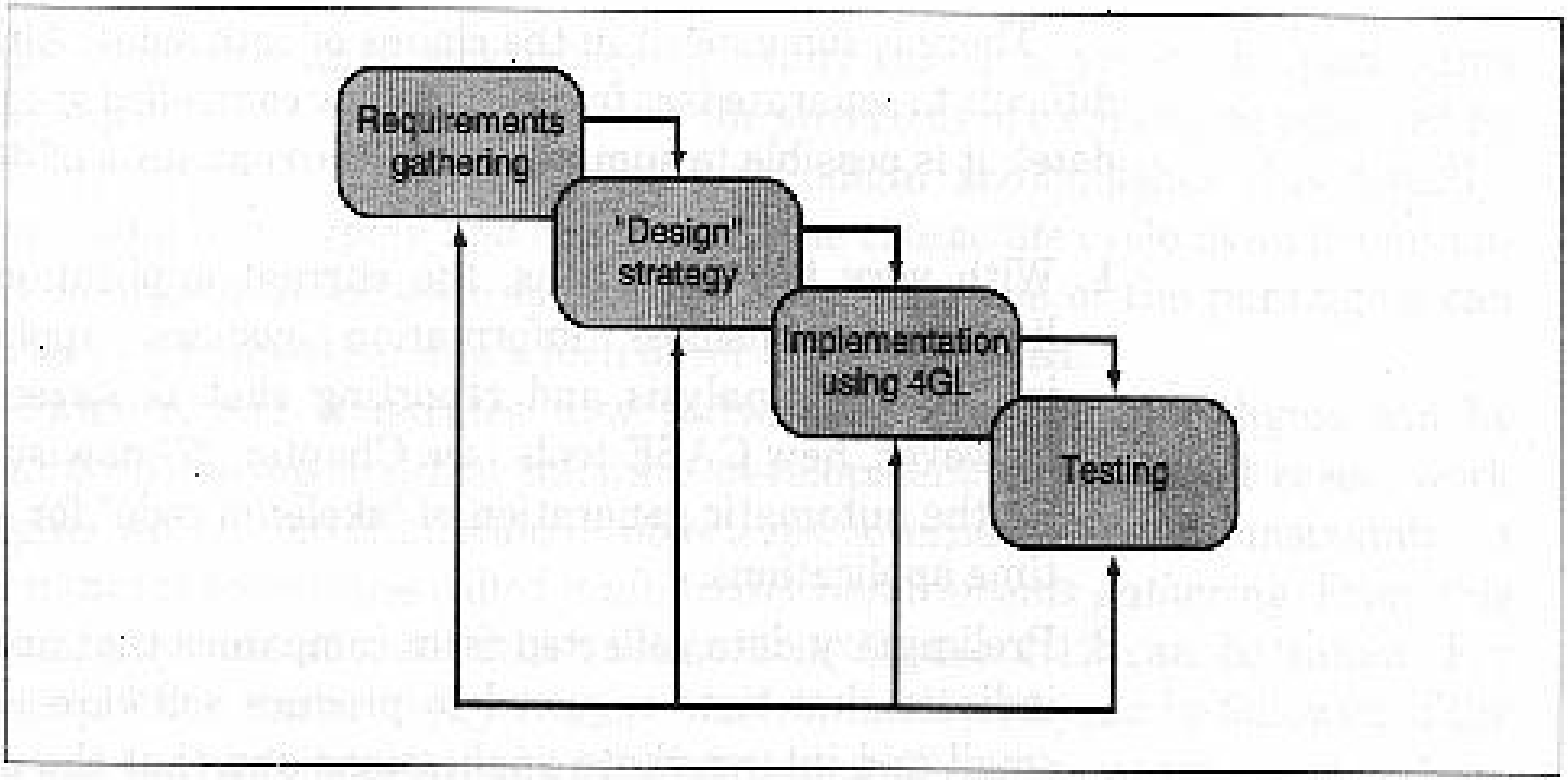
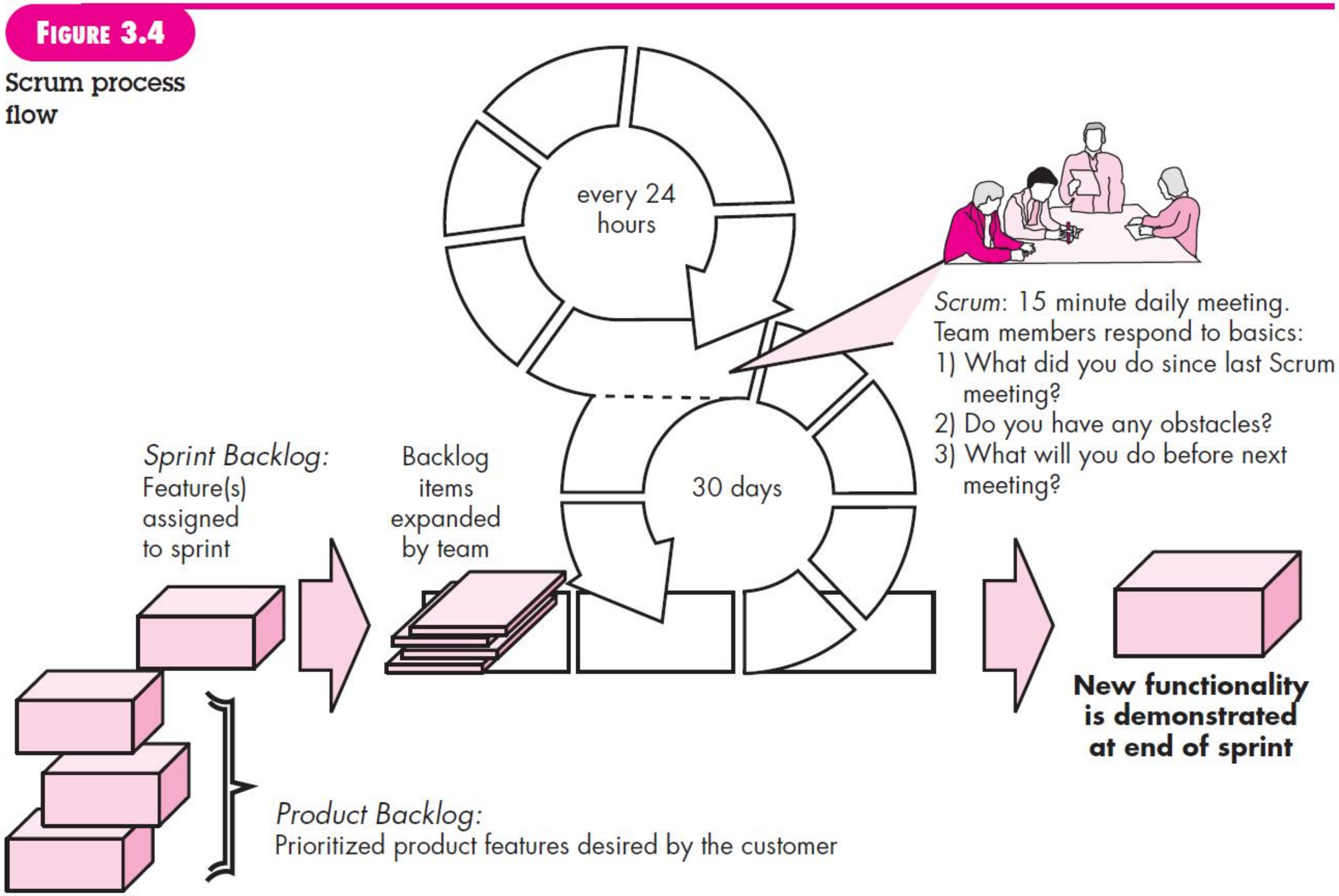
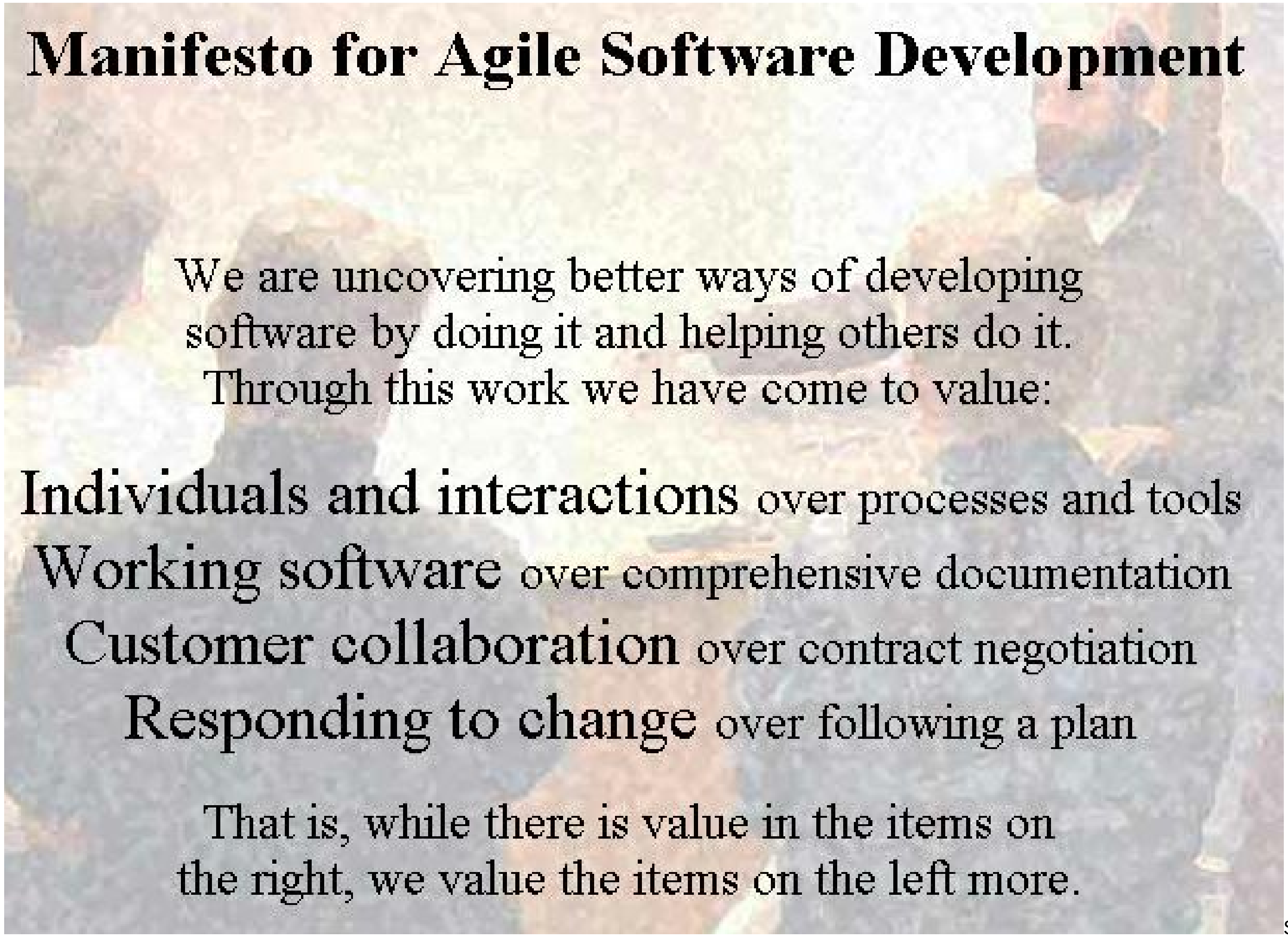


FIGURE 1.10.
Fourth-generation techniques.

V Agile (Incremental) Methods: Scrum

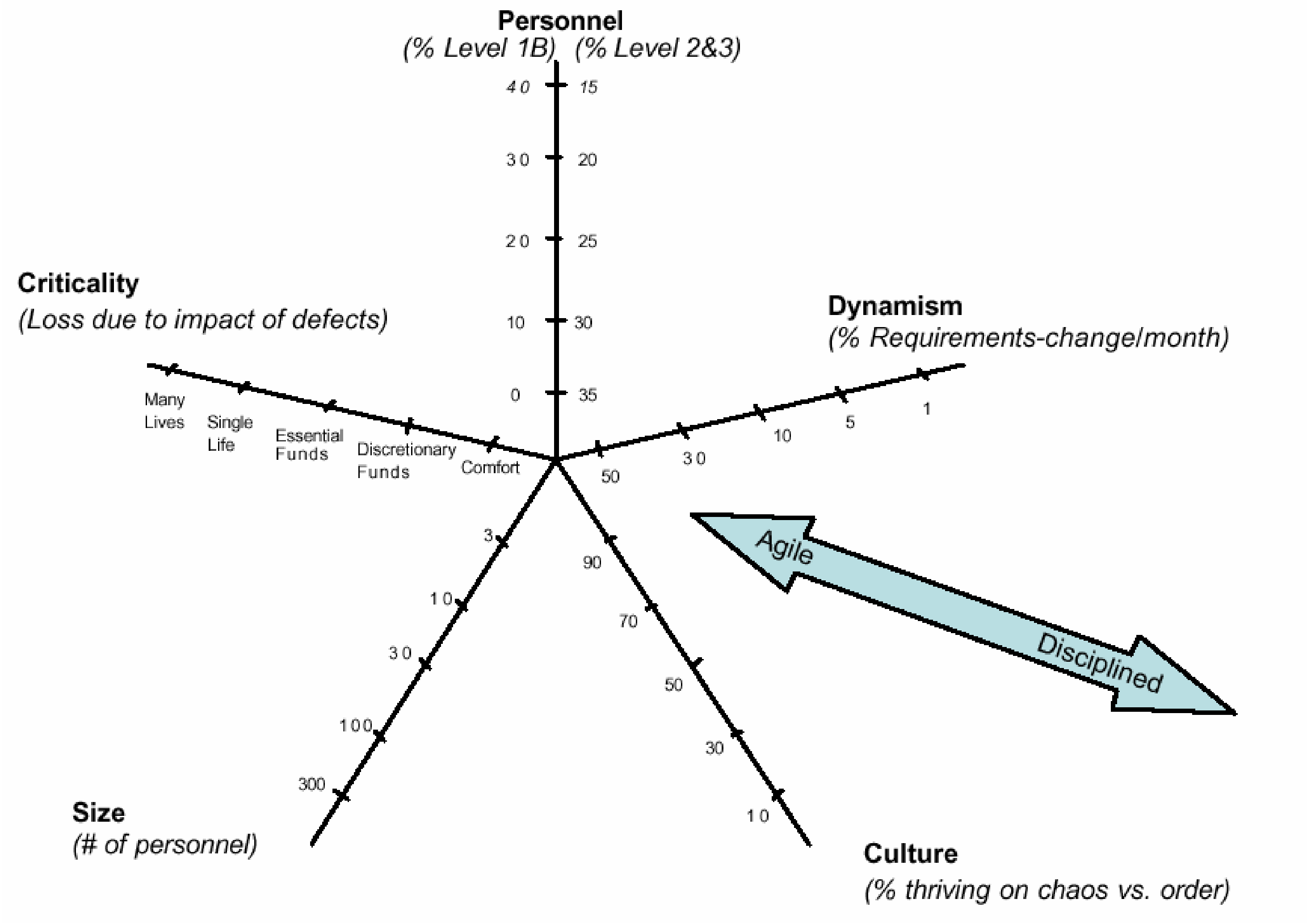


V Agile (Incremental) Methods



Source: www.agilemanifesto.org

HOW AGILE CAN YOU BE?



Source: Boehm & Turner (2003)

THE 12 AGILE PRINCIPLES (1/3)

DESCRIPTION	SUMMARY
1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software	**1. Satisfy customer through early and frequent delivery.**
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	**2. Welcome changing requirements even late in the project.**
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to a shorter timescale.	**3. Keep delivery cycles short (e.g., every couple of weeks).**
4. Business people and developers must work together daily throughout the project	**4. Business people and developers work together daily throughout the project.**

THE 12 AGILE PRINCIPLES (2/3)

5. Build project around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

5. Build projects around motivated individuals.

6. The most efficient and effective method of conveying information to and within development team is face-to-face conversation.

6. Place emphasis on face-to-face Communication.

7. Working software is the primary measure for progress.

7. Working software is the primary measure of progress.

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

8. Promote sustainable development pace.

THE 12 AGILE PRINCIPLES (3/3)

9. Continuous attention to technical excellence and good design enhances agility.

9. Continuous attention to technical excellence and good design.

10. Simplicity – the art of maximizing the amount of work not done – is essential.

10. Simplicity is Essential.

11. The best architectures, requirements, and designs emerge from self-organizing teams.

11. The best results emerge from self-organizing teams.

12. At regular intervals, the team reflect on how to become more effective, then tunes and adjusts its behavior accordingly.

12. Team reflects regularly where and how to improve.

AGILE SW DEVELOPMENT IN ONE SLIDE

Volatile development environment

Emphasis on individuals interactions and working software

Light and sufficient

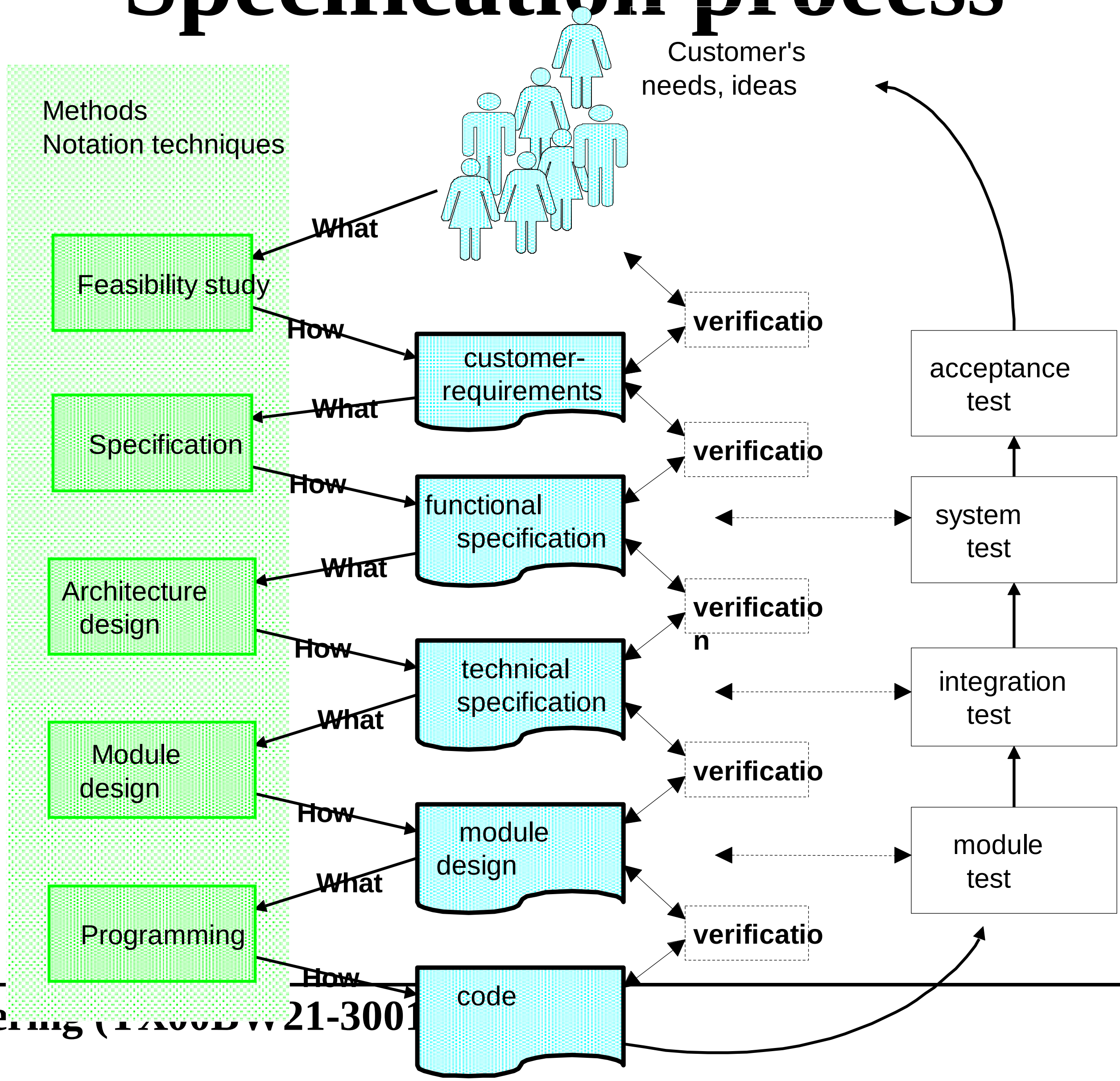
Methods are incremental, cooperative, straightforward and adaptive

Small teams, co-located environment, non-safety critical Software, object-oriented platforms

Encountered problems

Classic Life Cycle	Prototyping	Spiral Model	4GT
• Sequential flow is rarely followed in real project sand iteration is needed • Difficult to state all requirements. • How uncertainty? • The customer must have patience because working version is not available until late	• The customer does not recognise that the SW is hold together with chewing gum and baling wire • Inappropriate OS, inefficient algorithms, less-than-ideal choices	• Not a panacea • Not easily controllable • Demands considerable risk assessment and expertise • New method and has not been used widely	• What is actually required? • The user is unable or unwilling to specify in the manner a 4DT forces • Not much easier than classic languages • The produced code inefficient

Specification process



V Model or W Model