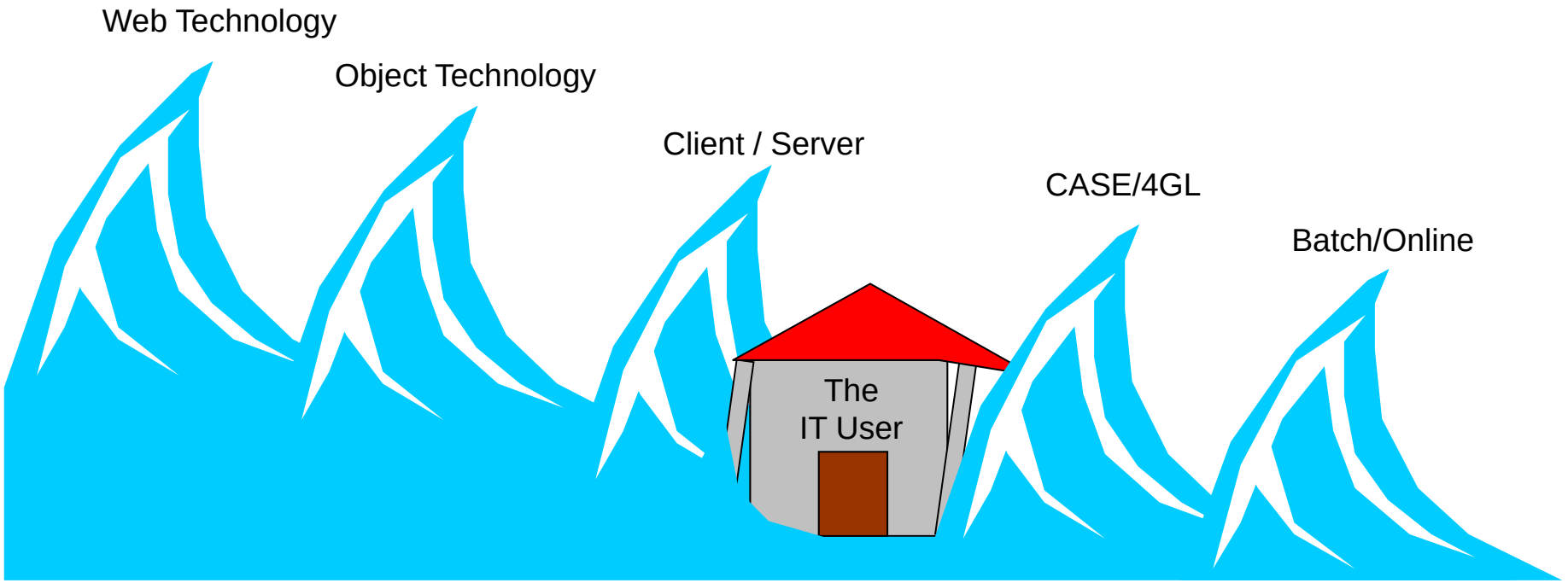
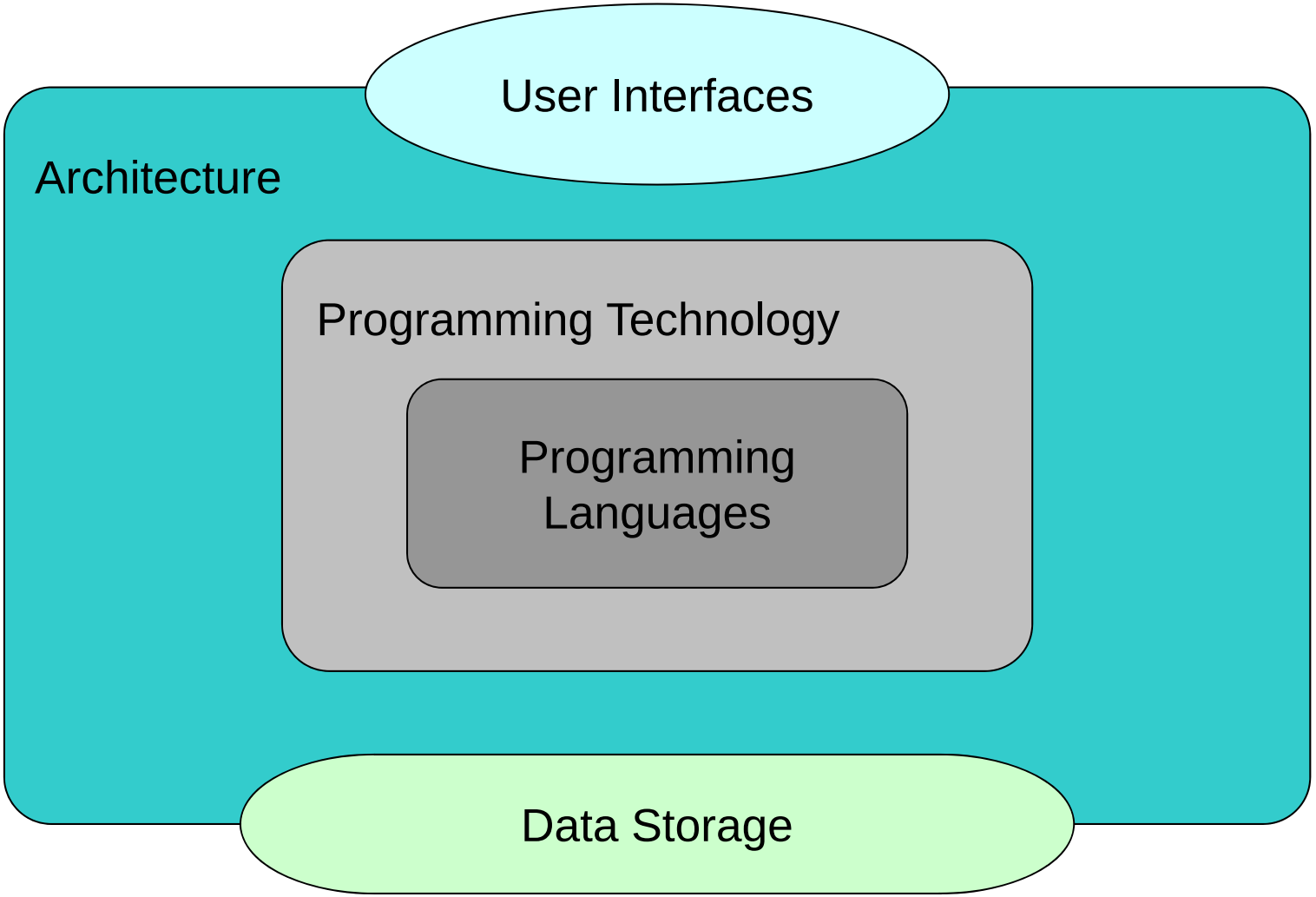
Software Product Evolution

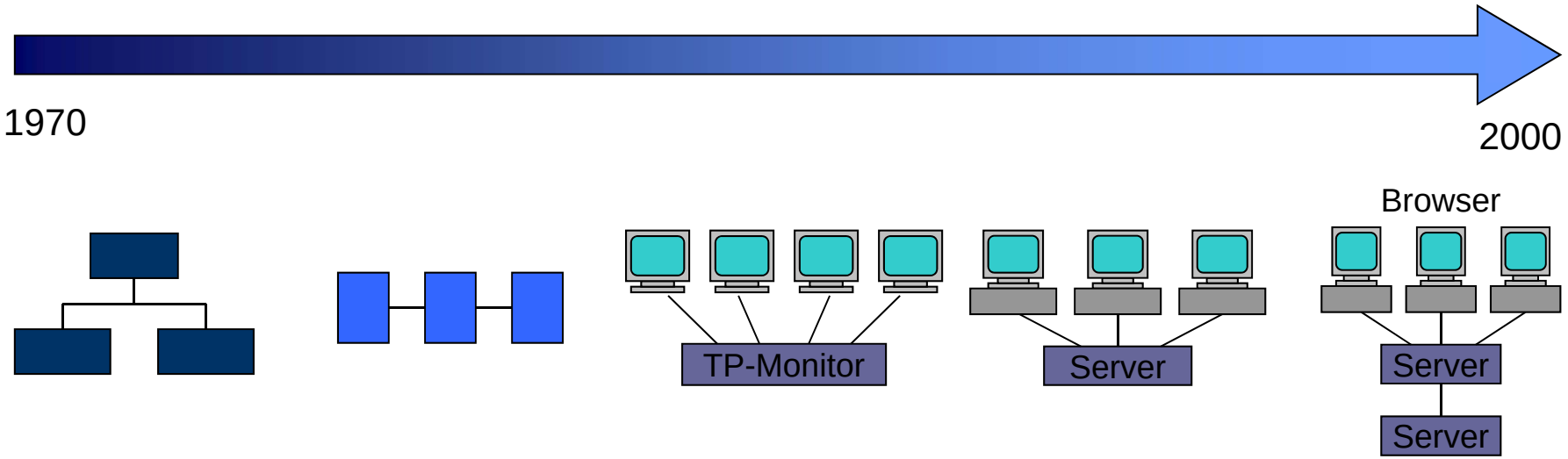
The Waves of Change



Components of an Application Environment



Evolution of System Architecture



Hierarchical Standalone Architecture	Batch Sequential Architecture	Online Transaction Architecture	Client / Server Architecture	Web2 Architecture
One Step	Some Steps	One Layer Thin Client	Two Layers Fat Client	Some Layers Thin Client

Evolution of Database Technology

1970

2000

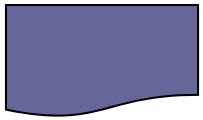
VSAM ISAM	IMS UDS	CODASYL IDMS	DB2 Oracle	DB2 Oracle
VSAM Files	Hierarchical Databases	Networked Databases	Relational Databases	Object Relational Databases
•Fixed physical Records •Variable logical Records •Differing Record Formats •Primary and secondary Search Items •Nested Access Logic	•Fixed Record Size •Variable Segments •Differing Segment Structures •Pointer Chaining •Physical and logical Accesses •Hierarchical Access Logic	•Fixed Records •Variable Records •Overlaying Record Structures •Pointer Chaining •Primary und secondary Seatch Items •Multiple Access Paths •Navigation through the Database	•Tabels of Tuples •Variable Length of Lines •Fixed Number of Attributes •Primary & Foreign Keys •Complex Access Logik •Independence of the Data from the Program	•Objects •Tabels •Differing Formats •Indexing and Pointer Chaining •Multiplex Access Logic •Adaptablility •XML Translation

Evolution of User Interfaces

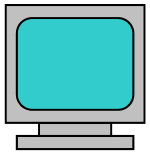
1970

2000

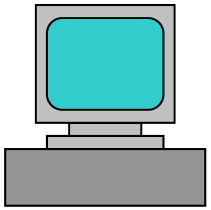
Punched Cards
Punched Tapes
Batch Systems



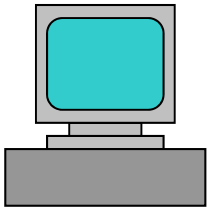
Record Reader
Batch Systems



Terminal
Monitor
Online Systems
TP-Monitors
IMS DC
CICS
UTM
TIP

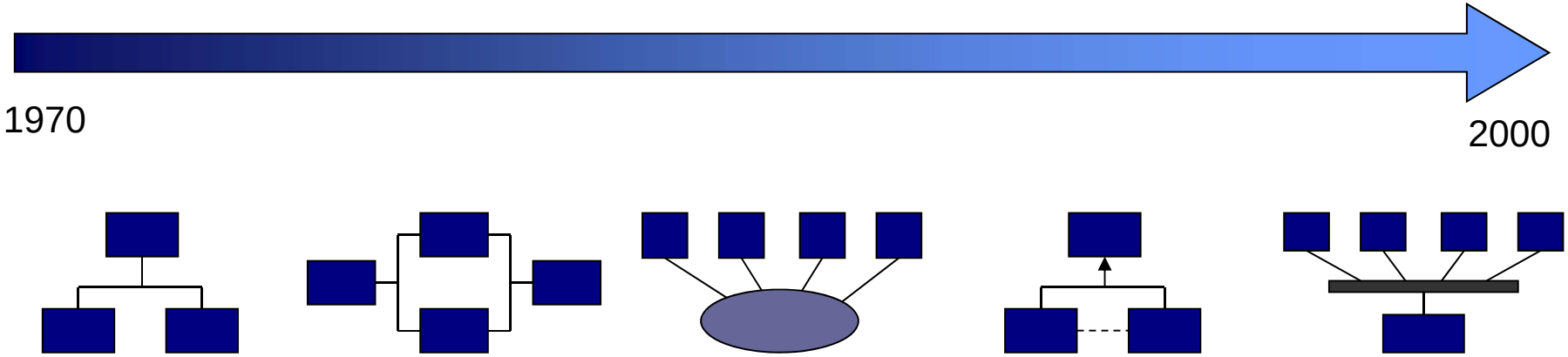


GUI
Fat Client
Client/Server
Systems



Web Browser
Thin Client
Web based
Systems

Evolution of Programming Techniques



Normalized Programming

- File affected Programming
- Input, Processing, Output

Structured Programming

- Control Flow affected Programming
- Sequence, Choice, Repetition

Data oriented Programming

- Database affected Programming
- Create, Update, Retrieve, Delete

Object oriented Programming

- Encapsulating of Data and Functions
- Inheritance
- Polymorphism
- Dynamical Binding

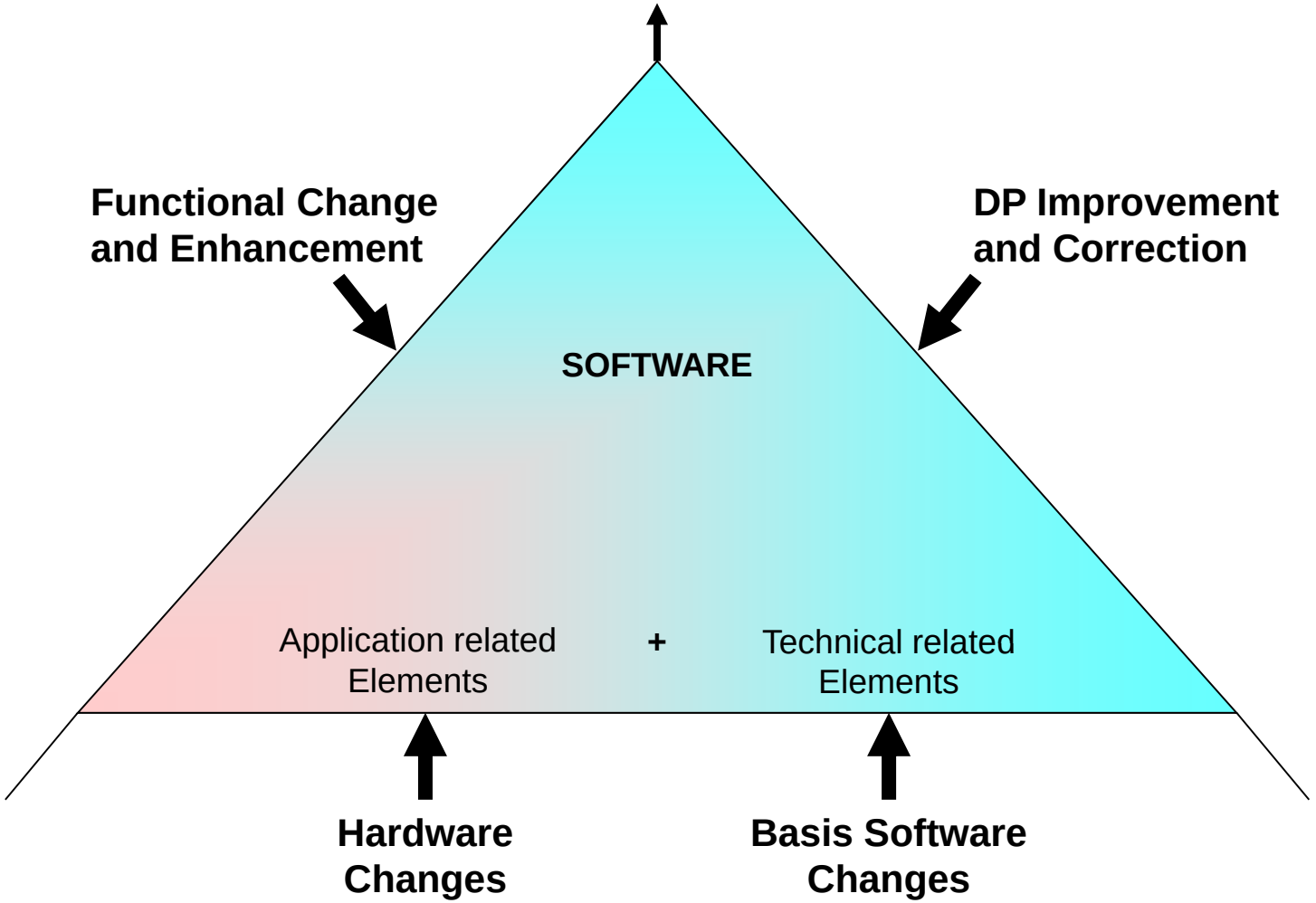
Component Programming

- Encapsulating of Functions
- Hiding of States
- Interfaces as Wrappers
- Division of Data and Functions

Evolution of Programming Languages

Assembler	Procedural Languages	4GL Languages	OO Languages	Mark-Up Languages
IBM Siemens Nixdorf	Fortran COBOL PL/I Delta	ADS-Online APS Ideal Natural	C++ OO-Cobol Java C#	HTML XML JSP PHP

Principle of Continuous Change



Software is embedded in an instable world.
Thus, it is subject to permanent change.

(according to Les Belady)

Environmental Influences on IT Systems

1) Technological Changes

- centralized Databases,
- distributed Processing,
- Automatisation of Offices,
- Micro Processors

2) Judicial Changes

- Tax Laws,
- social Laws,
- Business Laws

3) Economical Changes

- Growth,
- Consolidation / Conserving Actions
- Changes of Markets

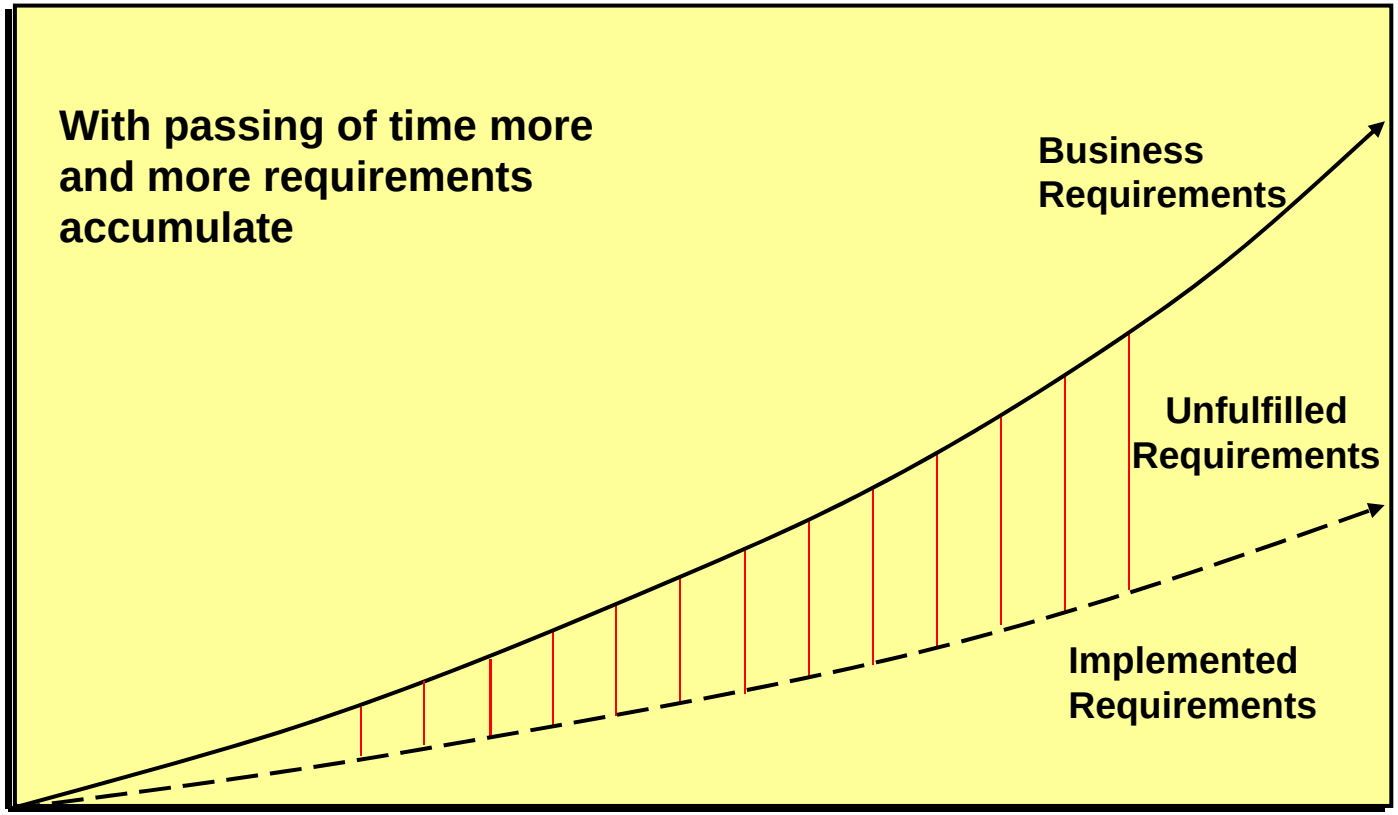
4) Social Changes

- Attitude to Technology,
- Self conception of the employees,
- Authority Structures

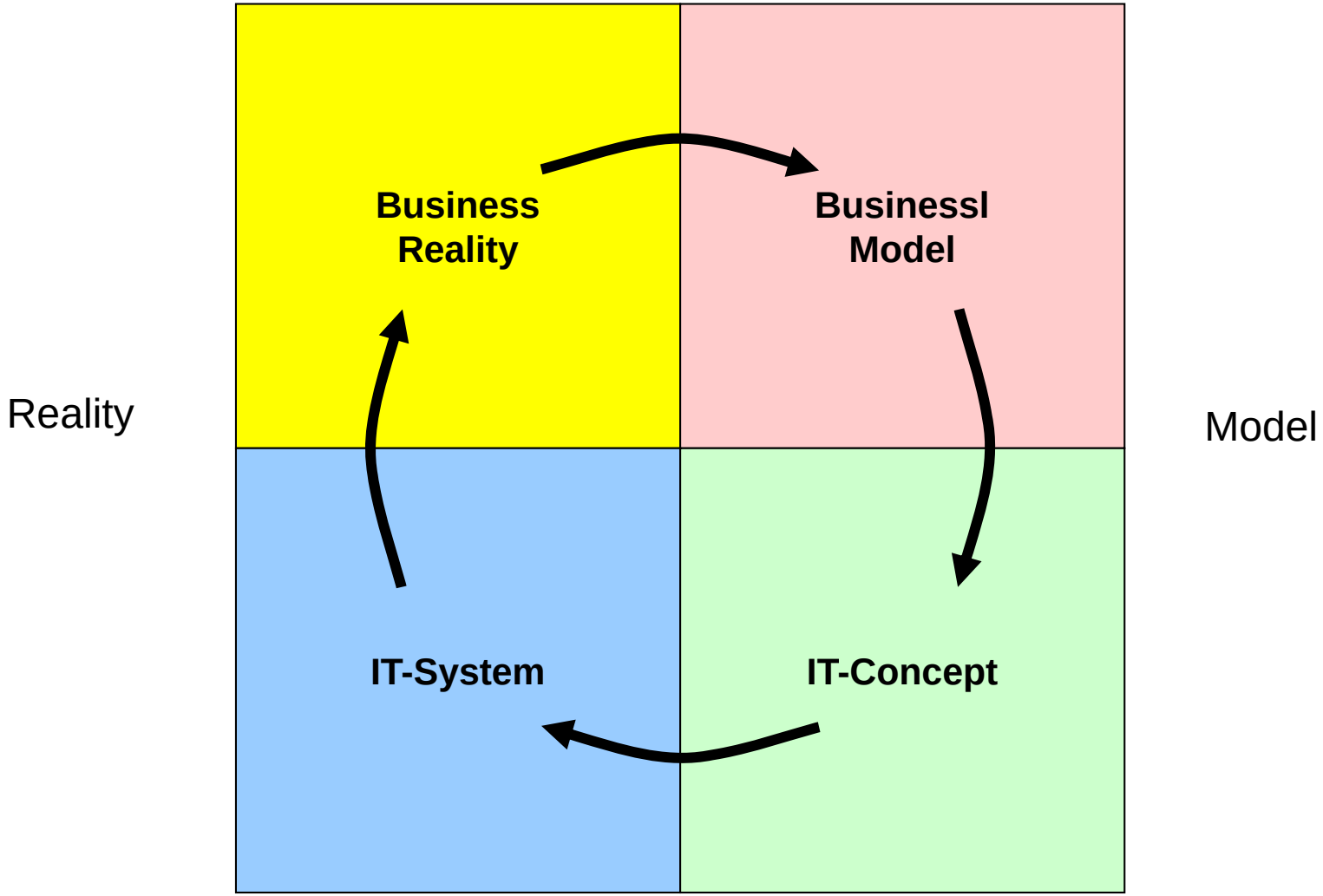
5) Organisatorial Changes

- Centralisation,
- Organisational Structures (central/decentral),
- Management Style (authoritarian/democratical),
- Division of work/Job specification

The Requirements Jam

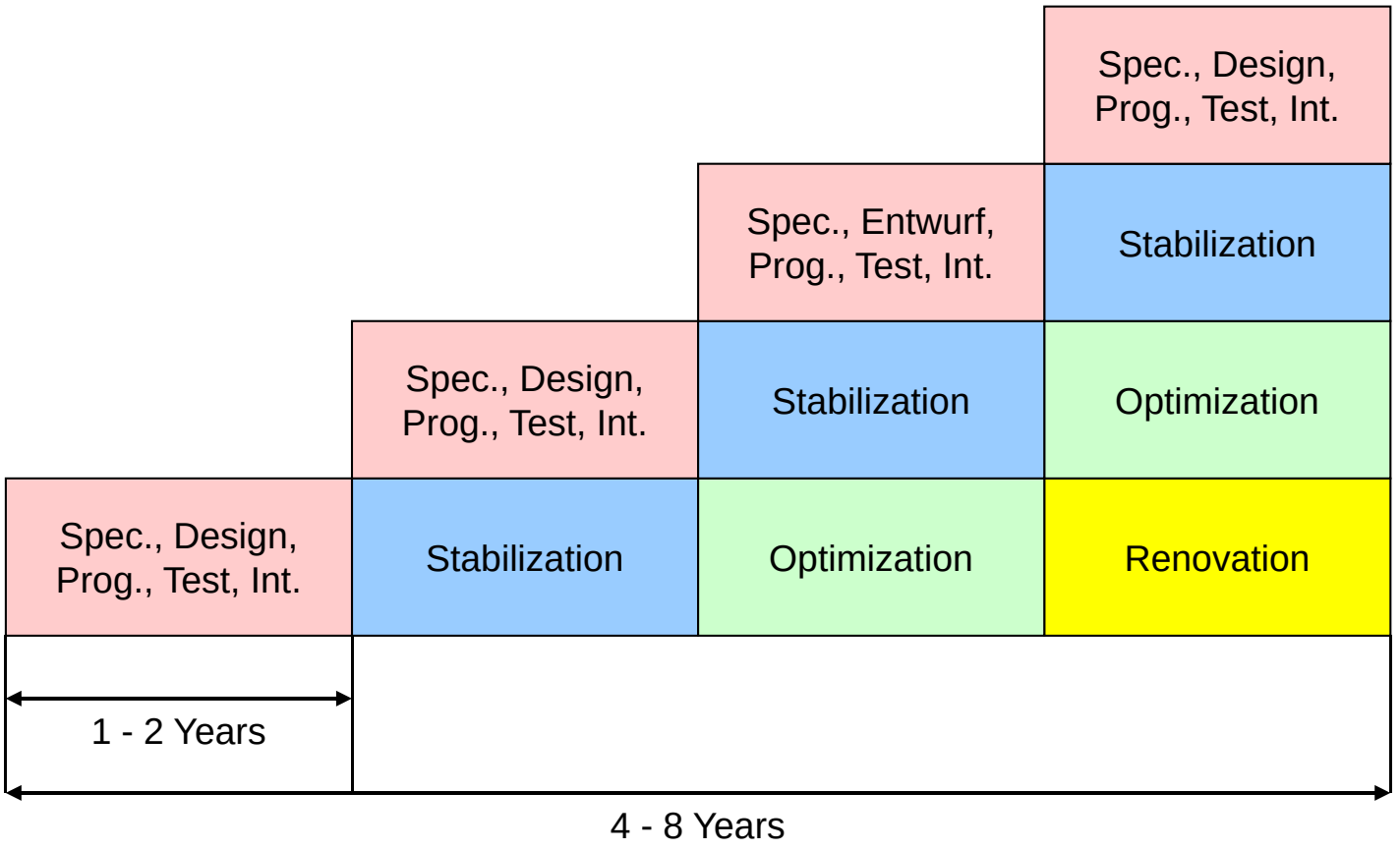


Evolution as a Control Feedback Loop

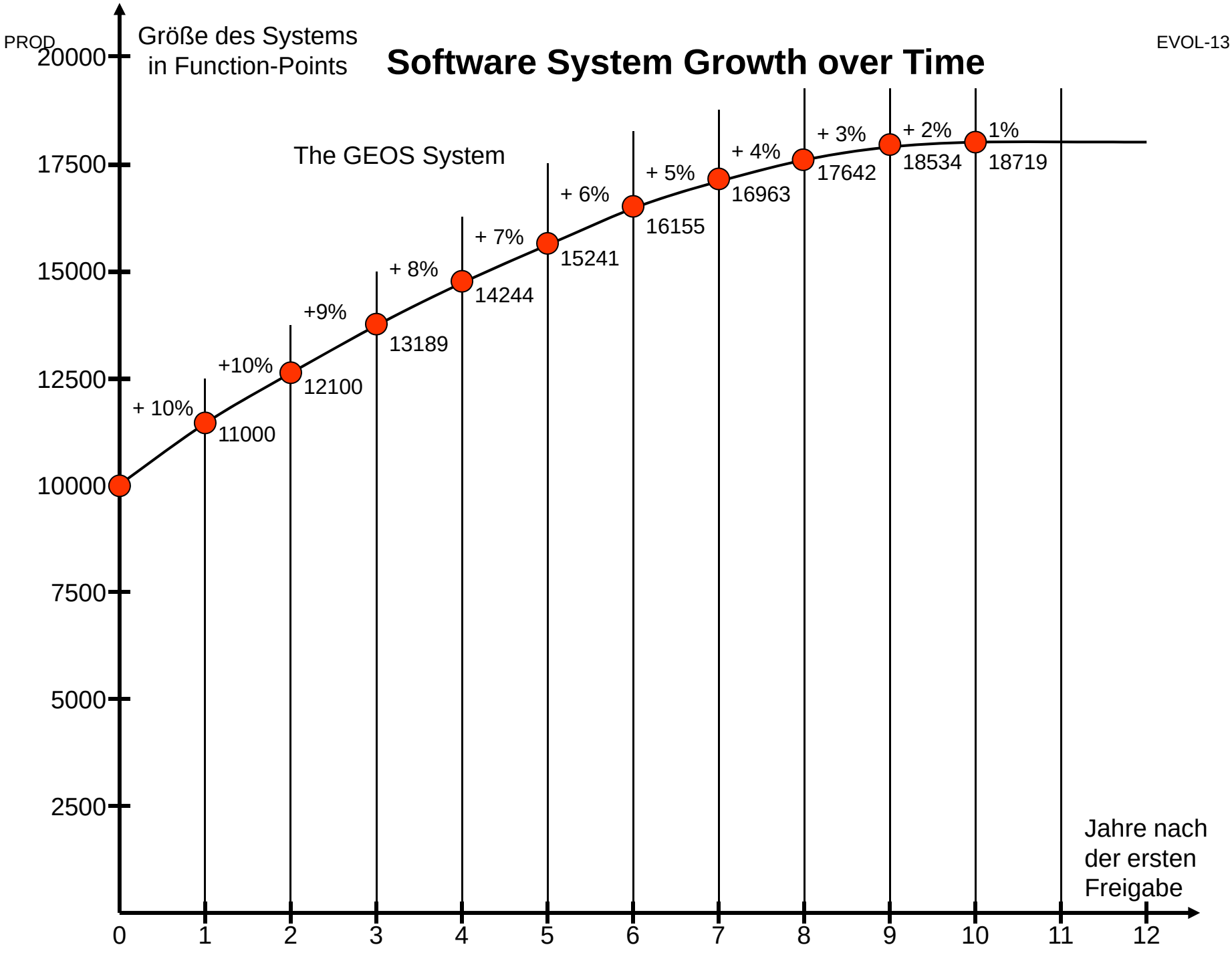


Evolutionary Software Development

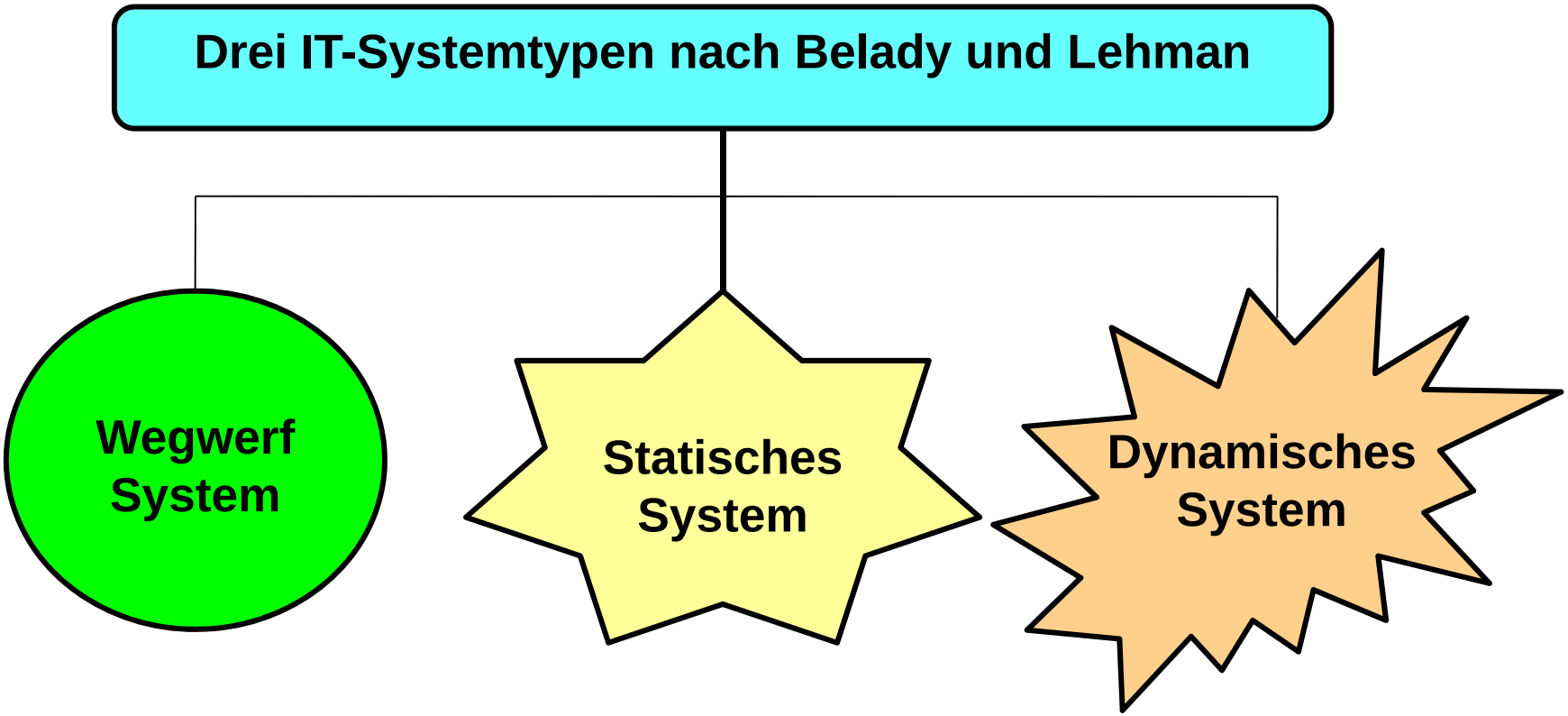
(according to Tom Gilb, 1985)



Software System Growth over Time



Software Evolution by Lehmann und Belady



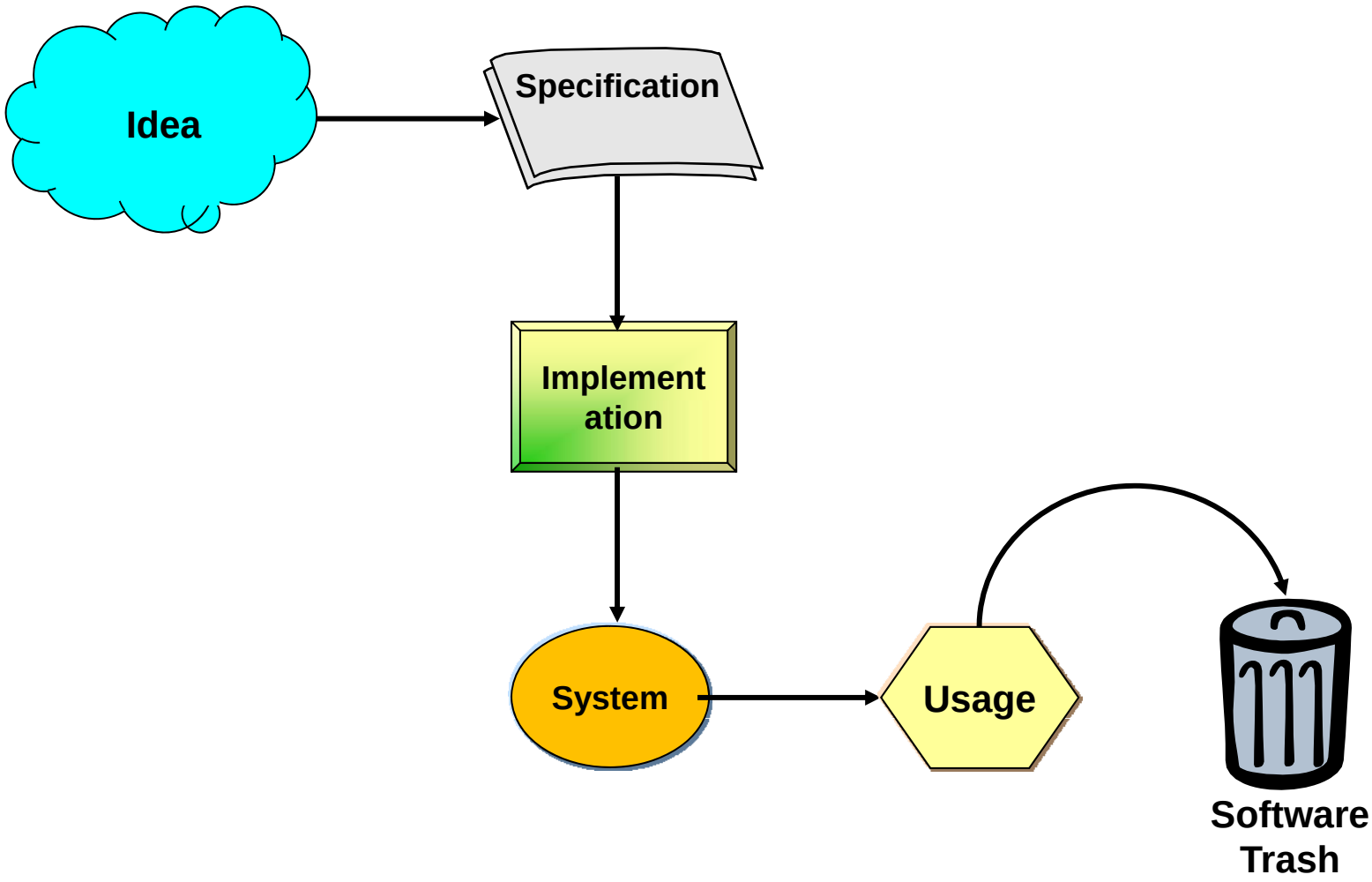
S-Systemtyp
wird spezifiziert, implementiert und weggeworfen

P-Systemtyp
wird spezifiziert, implementiert und fortgeschrieben

E-Systemtyp
wird konzipiert, und fortdauernd adaptiert

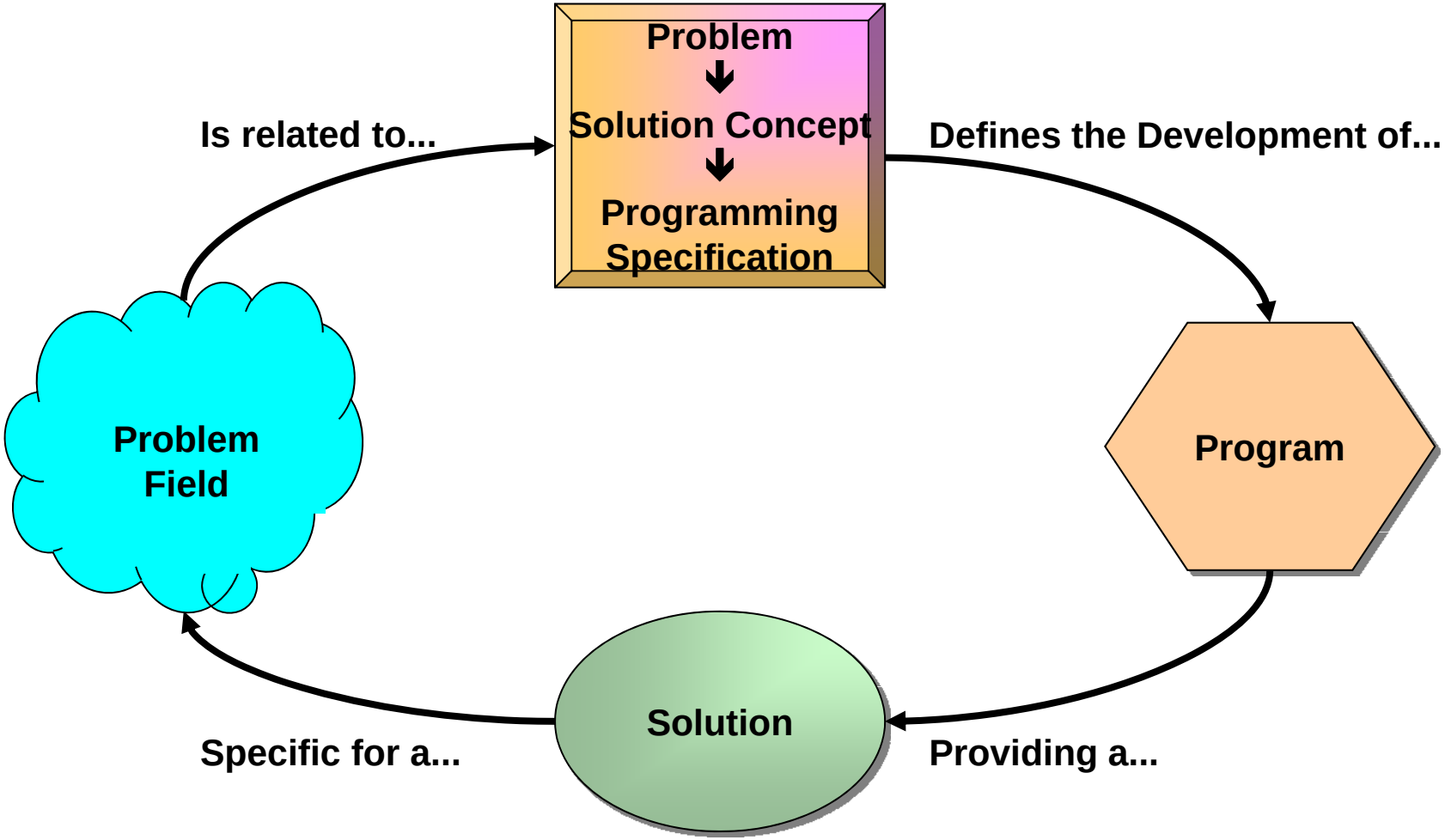
Academic Press
London, 1985

Throwaway Application Systems



Such systems have a limited life time - less than theee years

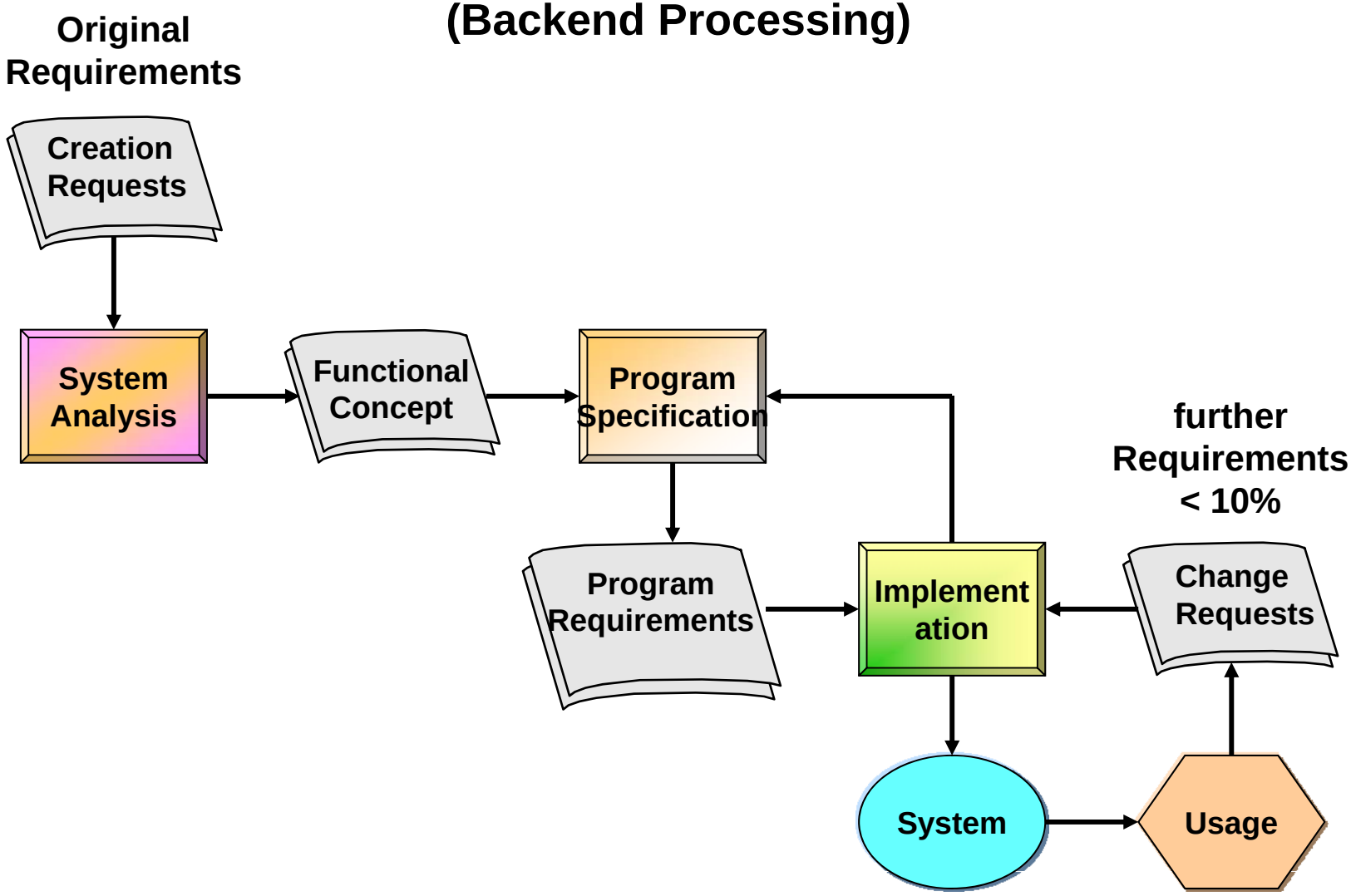
S-Systems = Temporary Solutions



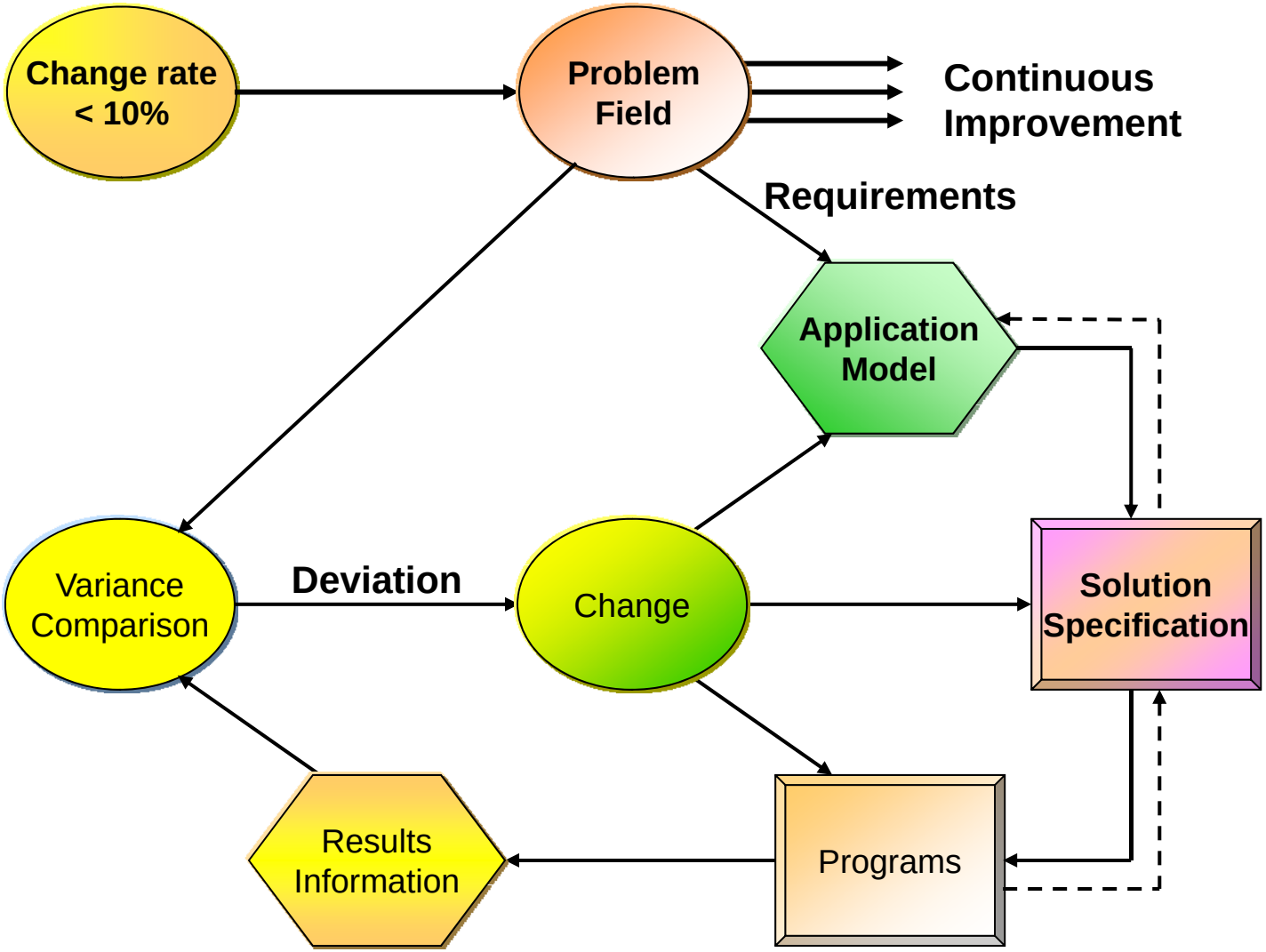
The program is customized to a given problem.

Standard Information Technology

(Backend Processing)

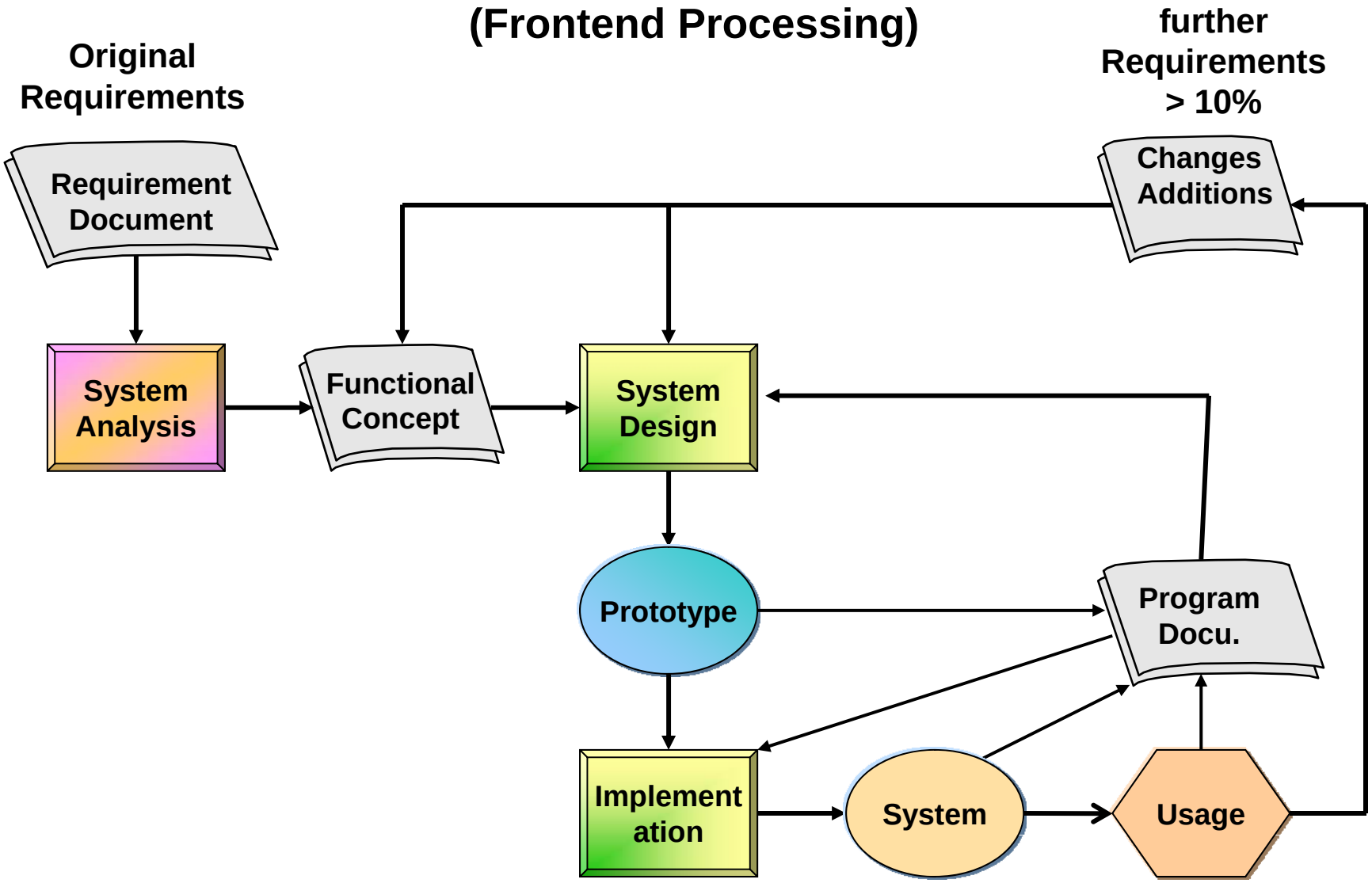


P Systems = Static Application Systems

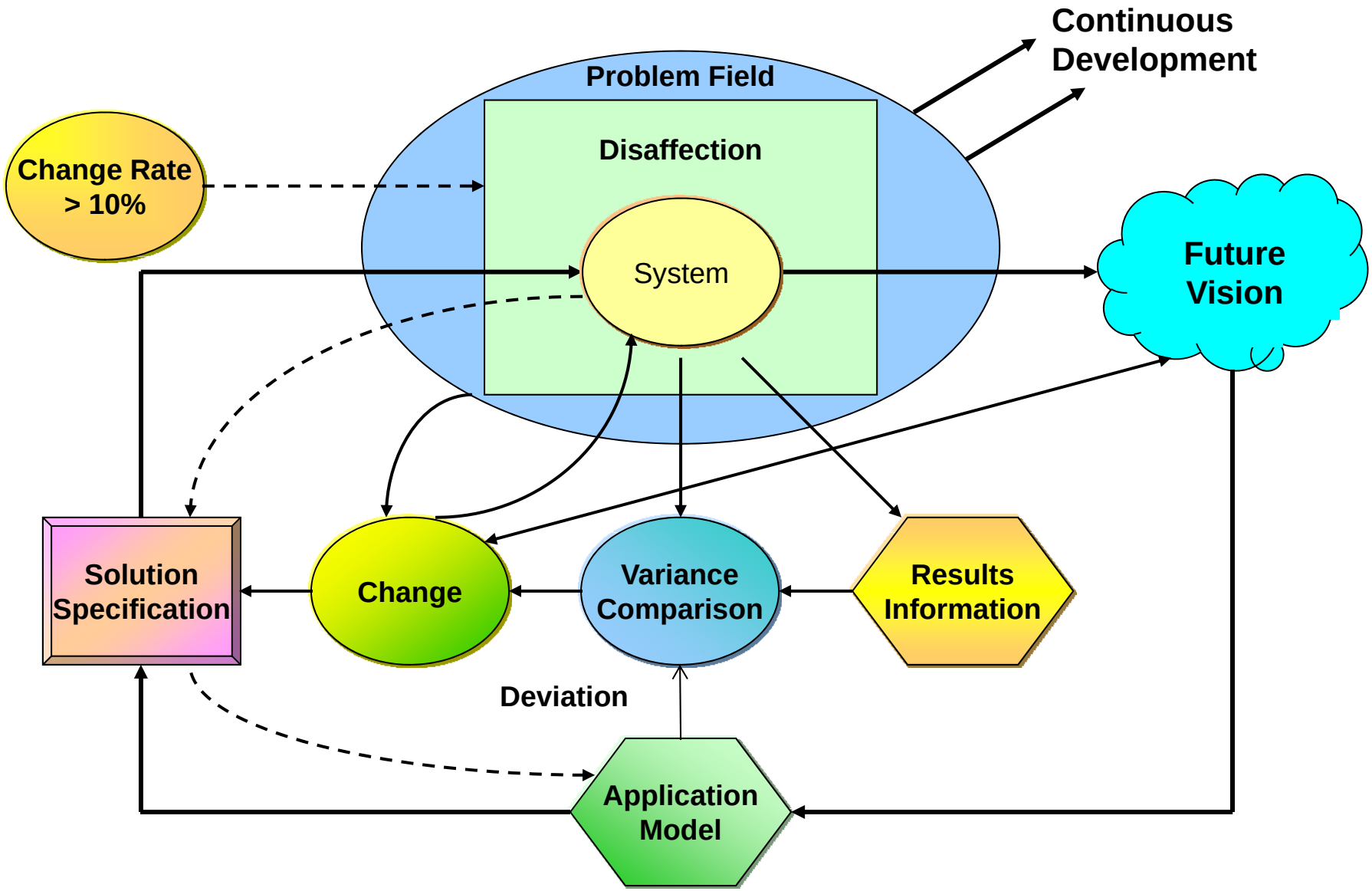


Flexible Information Technology

(Frontend Processing)



E-Systems = Dynamic Application Systems



The Laws of Software Evolution

I. Law of continuous change

Useful software mirrors processes in the real world. If these processes change, also the software has to change in order to be consist with them. Software that does not change or changes slower than the real world gets more and more useless.

II. Law of increasing complexity

In the beginning, the functionality of a system correlates with its form. Each change or enhancement of the functionality enlarges the gap between form and functionality. With increasing functionality and unchanged form the complexity of the system increases. Each evolution step gets more difficult.

III. Law of decreasing quality

Because each evolution step widens the gap between form and functionality, the quality of the system decreases. More and more conflicts between new and old requirements lead to bad compromises. This in turn leads to system erosion and decreasing quality.

IV. Law of diminishing productivity

More and more effort has to be put in the adaptation of the original form. This effort is missing for the evolution of the functionality. As a consequence the productivity decreases and the costs for evolution rise.

V. Law of restricted growth

The increasing complexity together with the decreasing quality slows down the evolution and stops it at a point in time. The evolution steps become smaller and smaller while the release intervals become larger and larger.

Consequences of the Laws of Evolution

(according to Belady & Lehmann)

1.) **The software manager must know the limits of changeability and extensibility.**

2.) **The software manager has to develop a long term plan to guarantee a continuous and controllable enhancement.**

3.) **The software specification has to be adapted just like the programs.**

4.) **The software documentation may not be disregarded.**

5.) **The software manager has to maintain a consistent and integrated development environment to ensure the continuity of his work.**

6.) **The software always has to be constantly checked regarding its quality to avoid erosion.**

7.) **The management has to take care of keeping the same developing methods during the whole life cycle. One may not change tires on a running car.**

Reasons for Software Mortality

1) Childhood mortality

- The software is unsuited from the beginning.

2) Unable to adapt

- Hardware changes (software depends on certain hardware)
- Software changes (software on other software)
- Requirement changes (software is design for a certain level)

3) Defectiveness

- Original errors
- Second level errors

4) Inadequacy

- The software no longer solves the problem.

5) Obsolete

- The software does not use the latest technical features.

6) Dependencies on humans

- The software can only be maintained by certain key persons, who threaten to leave the organization.

Possibilities for Life Elongation

1.) More modularization
2.) Better interfaces
3.) More independent components
4.) More strict standardization
5.) Limited data access (information hiding)
6.) Division in core and periphery components
7.) Greater use of parameters
8.) Built-in portability (virtual machines)
9.) Fewer hard-coded data (constants and literals)
10.) More and better documentation
11.) Repeatable test procedures
12.) More independence from personnel

➔ Constant Renovation / Refactoring

Continual Quality Assurance

Conclusions

- **A software product runs through five phases:**
  - **Conception (prototype development)**
  - **Initial development (1. release)**
  - **Evolution (maintenance & evolution)**
  - **Conservation (maintenance)**
  - **Redemption (displacement by another product)**
- **The software life cycle has to be planed, organized and controlled by the product management.**
- **The software product has to be maintained and evolved in constant intervals.**
- **The software product has to be fixed between the releases if necessary. (emergency fixes).**
- **The software product must be developed in small and manageable steps (evolutionary development)**
- **Software maintenance & evolution have to take place within a proper organizational infrastructure..**