



University of Zurich



Software Product Management

Product Launch and Lifecycle

Dec 07, 2010

Samuel Fricker
(fricker@ifi.uzh.ch)



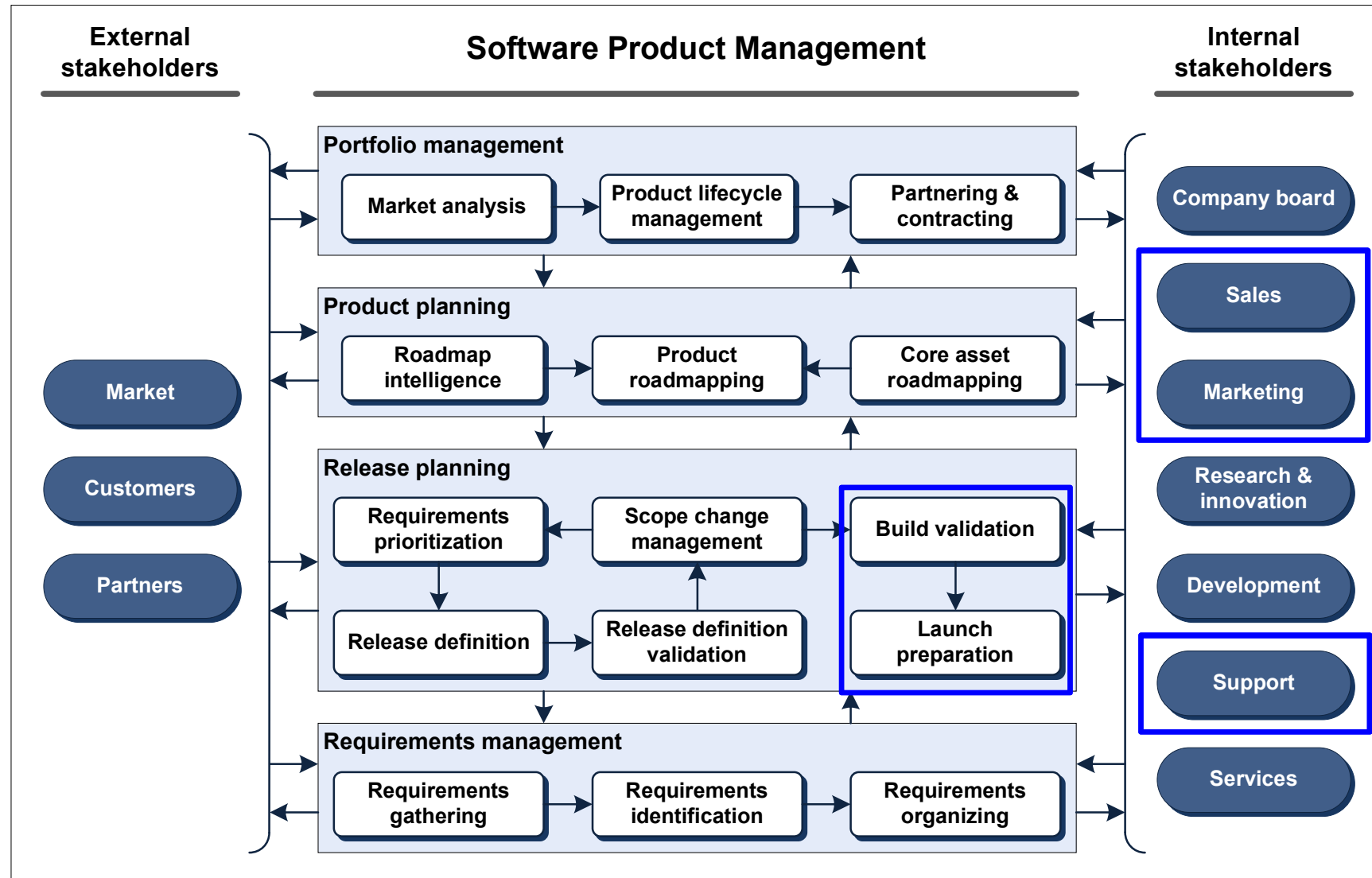
Focus



	Market Analysis	Product Analysis	Product Strategy	Product Planning	Development	Marketing	Sales and Distribution	Support and Services
Often on corporate level			Portfolio Management	Resource Allocation	Resource Allocation	Marketing Strategy and Plan	Sales Strategy and Plan	
Product (family) level	Market Research Market Sizing Market Problems Technology Assessment Competitive Analysis	Product Performance Customer Satisfaction Win/loss Analysis Opportunities	Positioning Delivery Model Pricing Model Pricing Business Case Make or buy Ecosystem Legal terms Protection of Intellectual Property	Roadmap Release Plan Requirements Management Functional Specification	Project Plan Technical Specification Project Requirements Management Implementation Quality Assurance Technology Innovation	Launch Plan Customer Analysis Partner Management Operational Marketing Material	Channel Preparation Operational Sales Operational Distribution Material	Customer Support Technical Support Marketing Support Sales Support Services Preparation Operational Services Provision



Focus





Contents

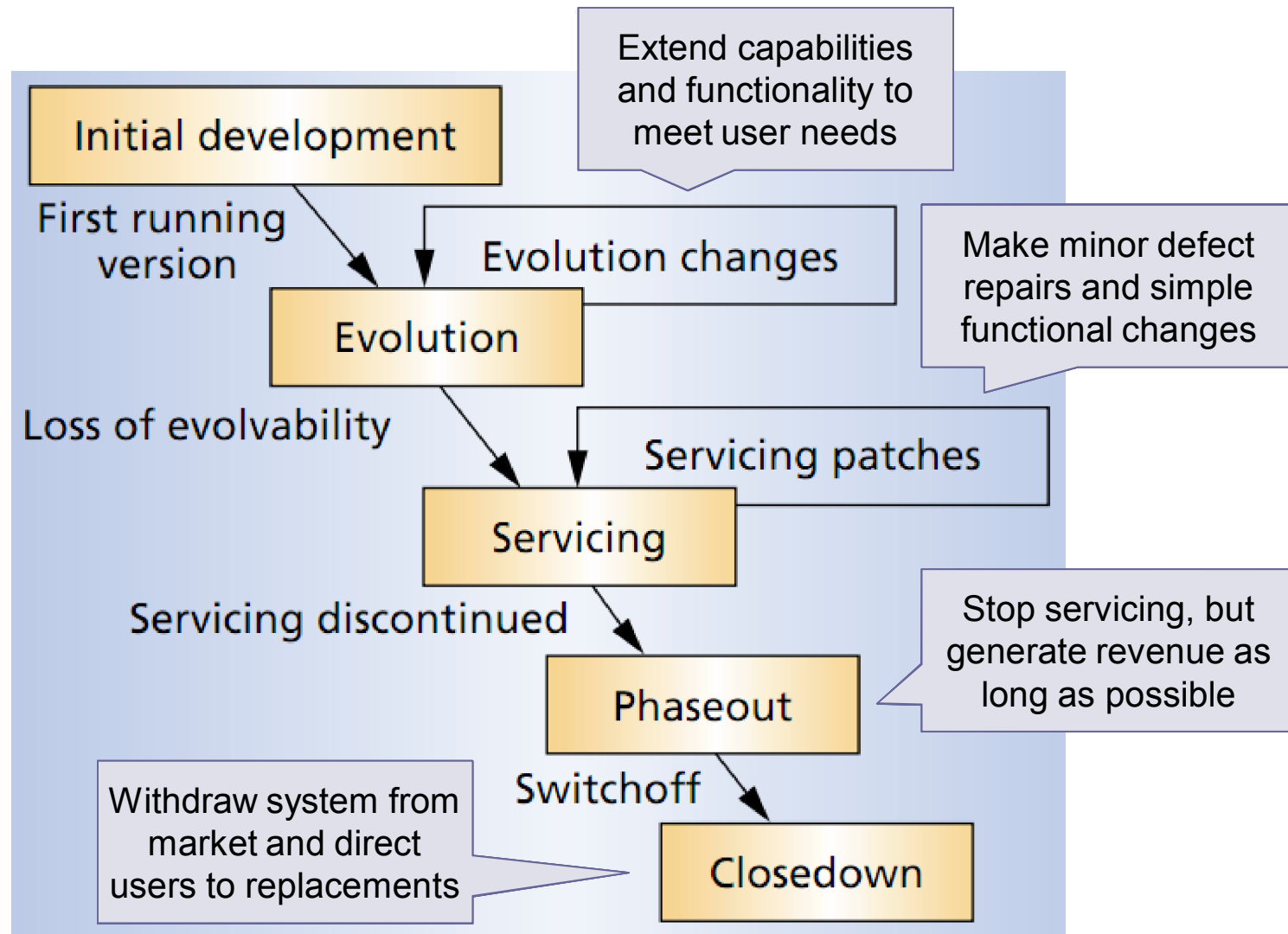
Learning Objectives

- Understand software lifecycle
- Understand critical practices to support software launch and lifecycle
 - Acceptance Testing
 - Delivery
 - Product Support Organization

Agenda

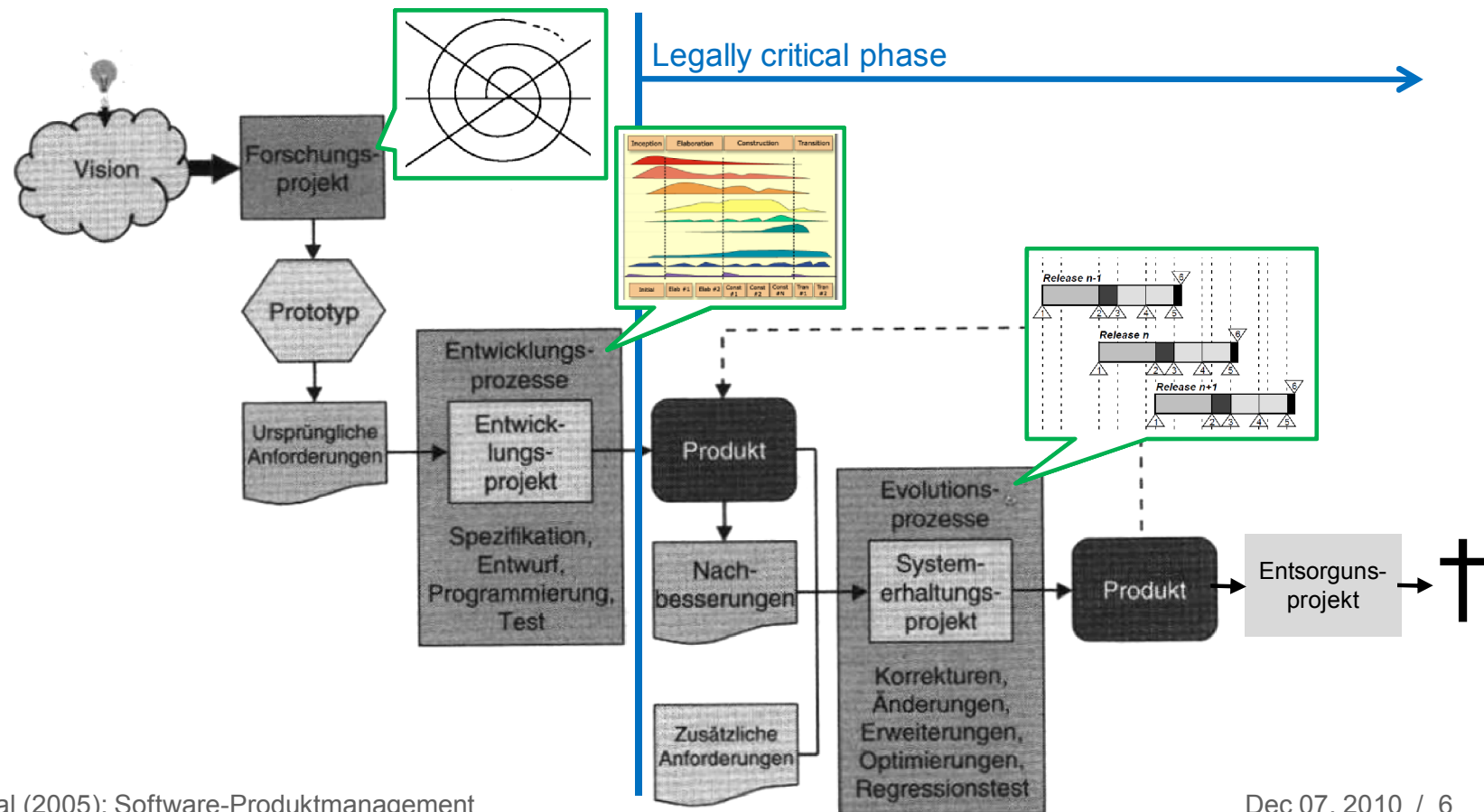
- Software Lifecycle
- Software Launch
 - Acceptance Testing
 - Delivery
- Software Lifecycle
 - Evolution
 - Servicing
 - Phase-Out and Close-Down

Software Stages



Technical View: SPM = Lifecycle Management

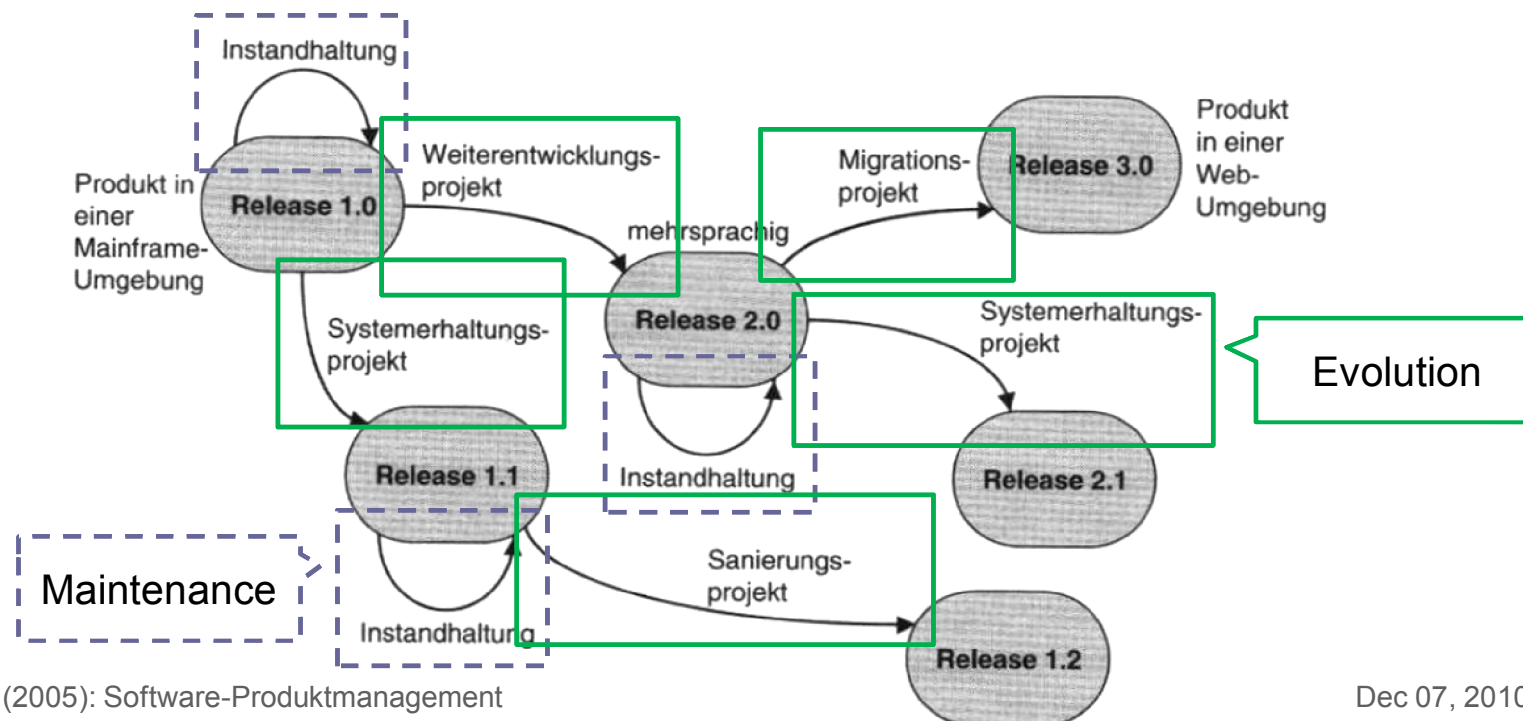
- Software Engineering in the Large
 - Prototype, Product, Project, Process



Technical View: SPM = Software Evolution



- Project of Projects
 - Starts with birth, ends with death of product
 - Define development goals to evolve product
 - Control goal achievement



Laws of Software Evolution (Software used by Humans)



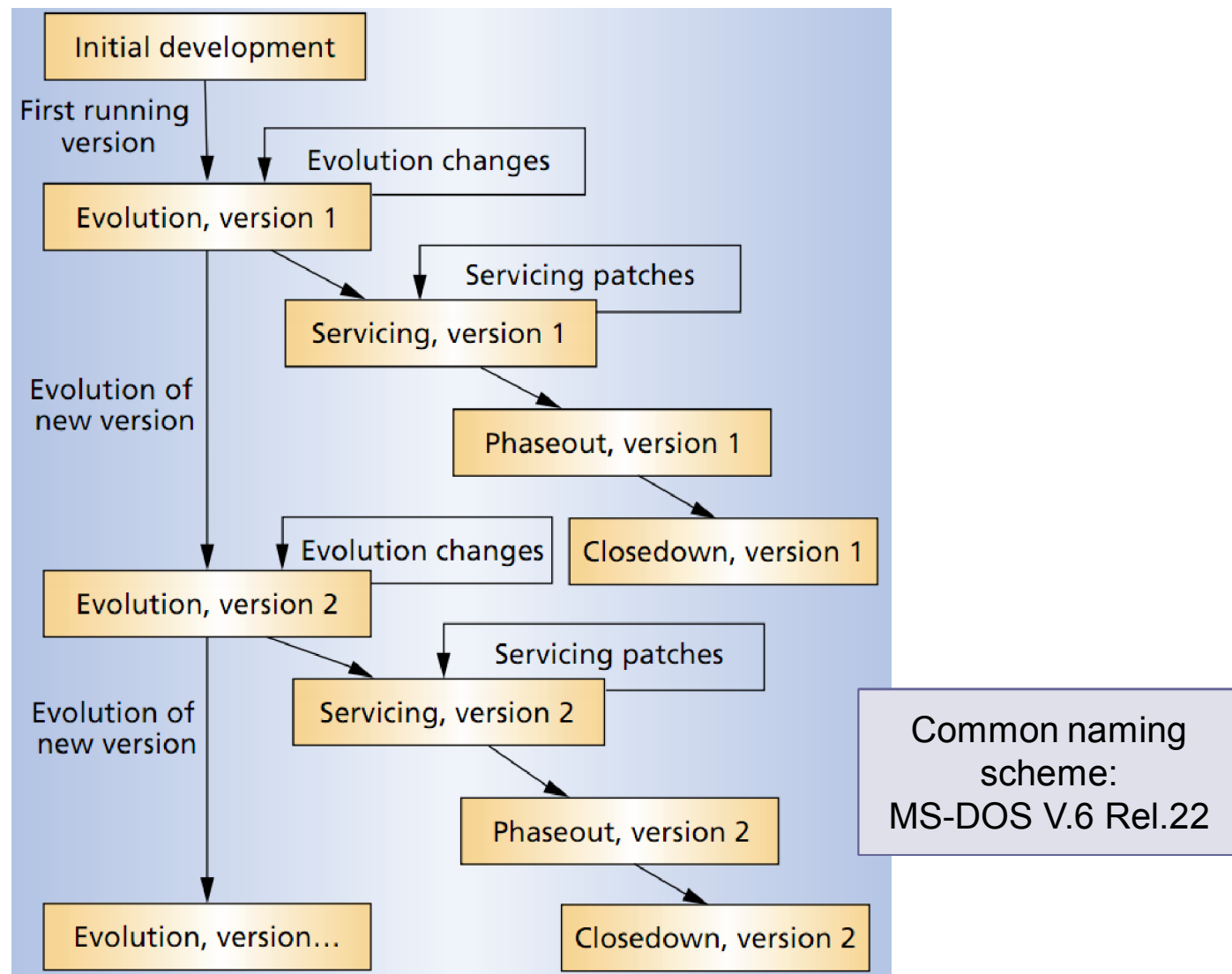
University of Zurich



Law	Explanation
Continuing Change	A software must be continually adapted else it becomes progressively less satisfactory in use.
Increasing Complexity	As a software is evolved its complexity increases unless work is done to maintain or reduce it.
Self Regulation	Global software evolution processes are self-regulating based on feedback to achieve stability.
Conservation of Organizational Stability	Average activity rate in software process tends to remain constant over system lifetime or segments of that lifetime.
Conservation of Familiarity	The average incremental growth (growth rate trend) of software tends to decline.
Continuing Growth	The functional capability of software must be continually increased to maintain user satisfaction over the system lifetime.
Declining Quality	Unless rigorously adapted to take into account changes in the operational environment, the quality of a software will appear to decline as it is evolved
Feedback System	Software evolution processes are multi-level, multi-loop, multi-agent feedback systems



Versioned Staged Model





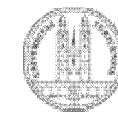
Contents

Learning Objectives

- Understand software lifecycle
- Understand critical practices to support software launch and lifecycle
 - Acceptance Testing
 - Delivery
 - Product Support Organization

Agenda

- Software Lifecycle
- Software Launch
 - Acceptance Testing
 - Delivery
- Software Lifecycle
 - Evolution
 - Servicing
 - Phase-Out and Close-Down



Acceptance Testing

- Acceptance test is a (formal) test conducted to determine whether or not a system satisfied its acceptance criteria and it enables the customer to determine whether or not to accept the system.
- The objective is to provide confidence that the delivered system meets the business requirements of the customer (and users).

Term	Introduced by
- <i>“functional tests”</i>	Beck, Extreme Programming Explained, 1/e [6]
- <i>“customer tests”</i>	Jeffries [17], Beck, Extreme Programming Explained, 2/e
- <i>“customer-inspired tests”</i>	Beck, Extreme Programming Explained, 1/e [6]
- <i>“story-tests”</i> and <i>“story-test-driven development”</i>	Kerievsky [20]
- <i>“specification by example”</i>	Fowler [14]
- <i>“coaching tests”</i>	Marick [22]
- <i>“examples”, “business-facing example”, and “example-driven-development”</i>	Marick [21]
- <i>“conditions of satisfaction”</i>	Cohn [10]
- <i>“scenario tests”</i>	Kaner [18]
- <i>“soap opera tests”</i>	Buwalda [8]
- <i>“formal qualification tests”</i>	e.g. DOD [11]
- <i>“system tests”</i>	e.g. IEEE [15]



Decision Tables

- As formality increases, tests and requirements become indistinguishable. At the limit, they are equivalent.
- One first determines the structure of the table, making sure that the headers cover all possible cases, then turns one's attention to completing the individual entries in the table.

Purchase tickets and check acknowledgment			Requirement (Scenario)
Variables	Play	Quantity	Acknowledgment?
Values	Phantom	2	You have purchased 2 tickets to Phantom of the Opera.
	Wizard	1	You have purchased 1 ticket to The Wizard of Oz.
	Cats	1	The play 'Cats' is not currently playing in a theater.
	Independent		Dependent



Specification Episodes

Entire Workflow,
including user actions
and system responses,
in customer-legible form
and formal-enough for
test automation.

Procedural
Table

Declarative
Table

Establishes a well-
documented language
shared between
development and
stakeholders

User: *bob* searches for play: *Phantom* where: *Toronto* when: *July 2007*

Search Results	
play	status
The Phantom of the Opera	available
Phantom Band	available
No Face Phantom	sold out

reject user: *bob* chooses play: *Phantomas*

system error: *No such play. Try again.*

user: *bob* chooses play: *The Phantom of the Opera*

Play Availability		
date	time	venue
July 15, 2007	2:00 pm	Pantages theater
July 15, 2007	8:00 pm	Pantages theater
July 16, 2007	8:00 pm	Pantages theater

user: *bob* chooses showing at: *8:00 pm* on: *July 15, 2007*

Seating Section Availability			
play	date	time	seating
The Phantom of the Opera	July 15, 2007	8:00 pm	main floor
The Phantom of the Opera	July 15, 2007	8:00 pm	first balcony
The Phantom of the Opera	July 15, 2007	8:00 pm	second balcony

user: *bob* chooses seating: *first balcony*

Discount Availability	
type	price
full price	\$95.00
student	\$80.00
senior	\$80.00
child	\$32.00

user: *bob* chooses discount: *full price* quantity: *2*

...

Sad Path

Happy Path

Not covered: boundary
value analysis, path
analysis, and state
transition analysis.



Evaluation

user: *bob* chooses showing at: *8:00 pm* on: *July 15, 2007*

Seating Section Availability			
play	date	time	seating
The Phantom of the Opera	July 15, 2007	8:00 pm	m
The Phantom of the Opera	July 15, 2007	8:00 pm	fi
The Phantom of the Opera	July 15, 2007	8:00 pm	se

user: *bob* chooses seating: *first balcony*

Discount Availability	
type	price
full price	\$95.00
student	\$80.00
senior	\$80.00
child	\$32.00

Discount Availability

type	price
full price	\$95.00
student	\$80.00
senior	\$80.00
child	\$32.00 <i>expected</i>
	\$31.99 <i>actual</i>

Correct Cases

Erroneous Case

user: *bob* chooses discount: *full price* quantity: *2*



Challenges with FIT Testing



- Concurrency

time is now	2006/10/03 10:24:00 am
user:	<i>bob</i> chooses showing at: 8:00 pm on: July 15, 2007

Extension

Seating Section Availability				
play	date	time	seating	quantity available
The Phantom of the Opera	July 15, 2007	8:00 pm	main floor	10
The Phantom of the Opera	July 15, 2007	8:00 pm	first balcony	5
The Phantom of the Opera	July 15, 2007	8:00 pm	second balcony	2

time is now	2006/10/03 10:24:10 am
user:	greg chooses showing at: 8:00 pm on: July 15, 2007

Seating Section Availability				
play	date	time	seating	quantity available
The Phantom of the Opera	July 15, 2007	8:00 pm	main floor	10
The Phantom of the Opera	July 15, 2007	8:00 pm	first balcony	5
The Phantom of the Opera	July 15, 2007	8:00 pm	second balcony	2

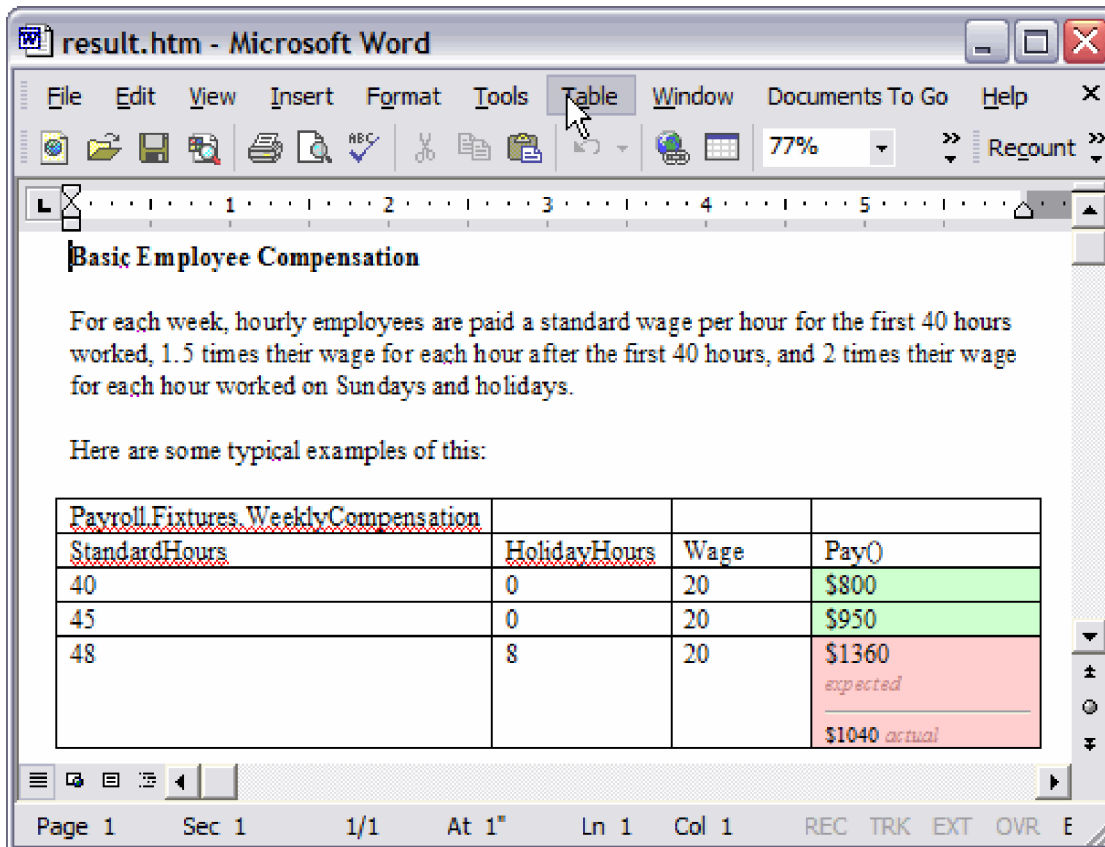
- Non-functional requirements, e.g. Performance

ensure	transaction:	choose showing	executes:	1,000	times in less than:	5	seconds
--------	--------------	----------------	-----------	-------	---------------------	---	---------

Extension

FIT: Framework for Integrated Test

- Translation of the customer view to the development view (of automated tests)



Basic Employee Compensation

For each week, hourly employees are paid a standard wage per hour for the first 40 hours worked, 1.5 times their wage for each hour after the first 40 hours, and 2 times their wage for each hour worked on Sundays and holidays.

Here are some typical examples of this:

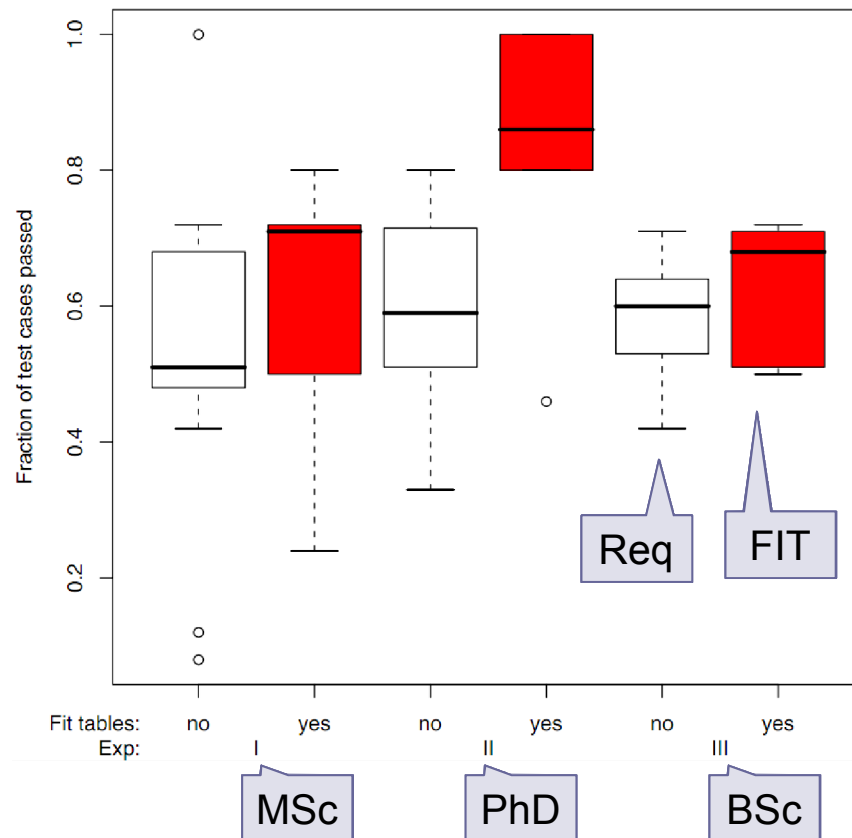
StandardHours	HolidayHours	Wage	Pay()
40	0	20	\$800
45	0	20	\$950
48	8	20	\$1360 <i>expected</i>
			\$1040 <i>actual</i>

```
public class WeeklyCompensation : ColumnFixture
{
    public int StandardHours;
    public int HolidayHours;
    public Currency Wage;

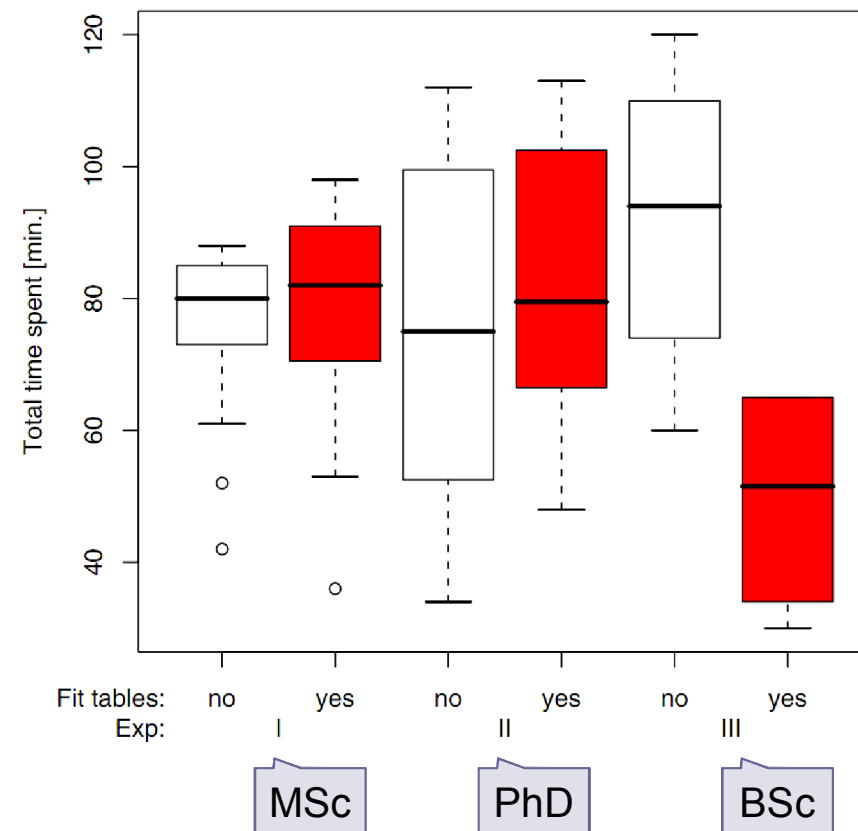
    public Currency Pay()
    {
        WeeklyTimesheet timesheet = new WeeklyTimesheet();
        return timesheet.CalculatePay(Wage);
    }
}
```

Evaluation of FIT Approach

Does the presence of Fit tables help programmers to improve the correctness of maintained code after maintenance/evolution tasks?



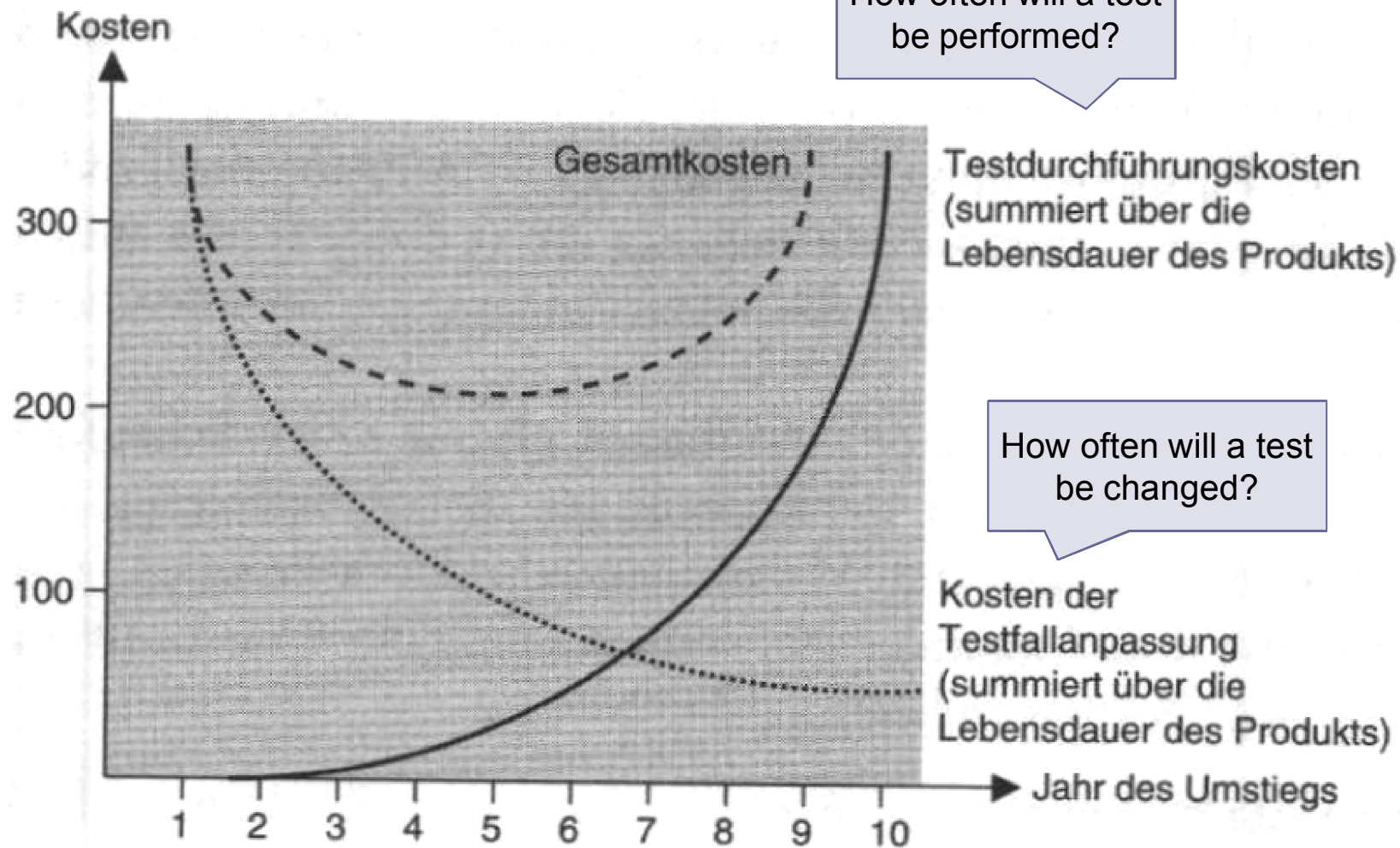
Does the presence of Fit tables affect the time in the execution of maintenance interventions?



Cost of Manual vs. Automated Testing



University of Zurich





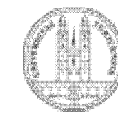
Contents

Learning Objectives

- Understand software lifecycle
- Understand critical practices to support software launch and lifecycle
 - Acceptance Testing
 - Delivery
 - Product Support Organization

Agenda

- Software Lifecycle
- Software Launch
 - Acceptance Testing
 - Delivery
- Software Lifecycle
 - Evolution
 - Servicing
 - Phase-Out and Close-Down



Decision to Deliver: Cost and Value of an Update

- Motivation for Product Management

$$V_{\text{value}}(\text{Update}) > V_{\text{cost}}(\text{Update})$$

- $V_{\text{value}}(\text{newUpdatePackage}) =$
newCustomers x priceNewRelease +
oldCustomers x priceUpdate + costReduction(support)
- $V_{\text{cost}}(\text{newUpdatePackage}) =$
cost(development) + cost(updateCurrentCustomers) +
cost(increasedSupport) + cost(marketing) +
cost(deliveryToCustomers) + cost(packageCreation)

- Motivation for Customers:

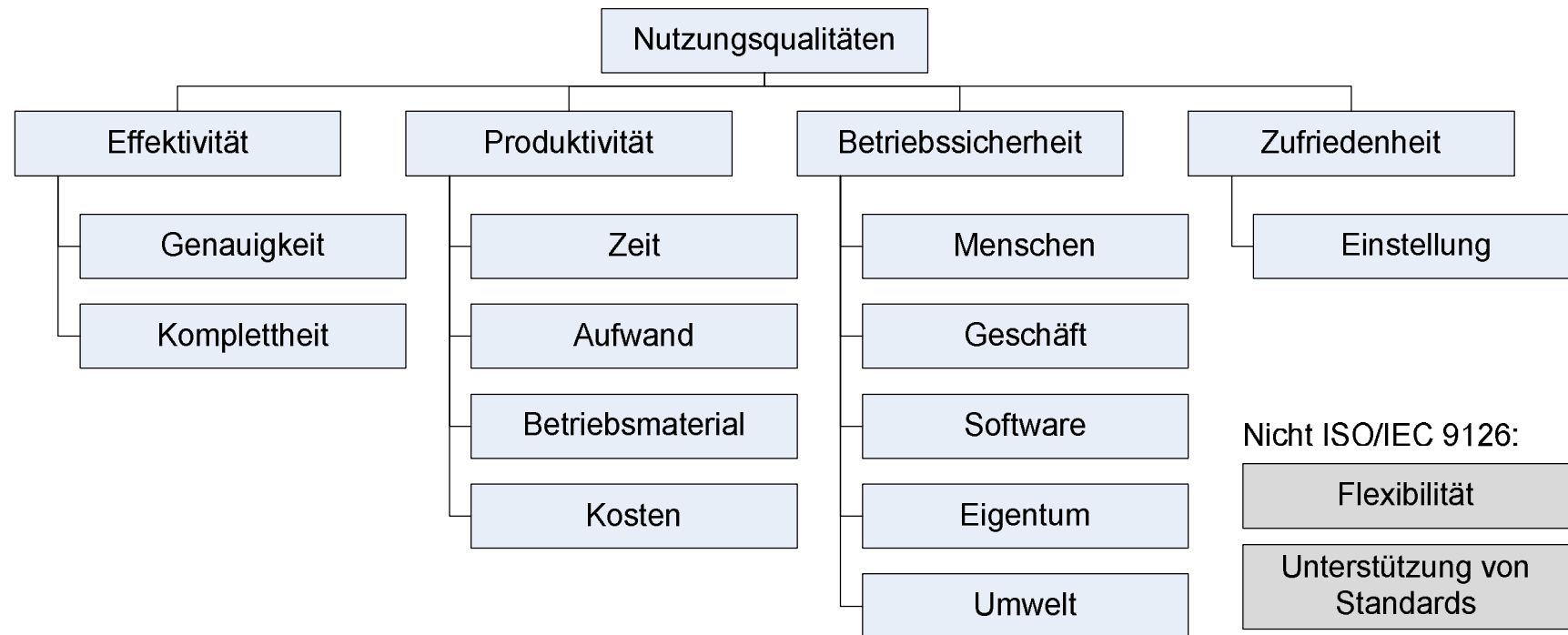
$$C_{\text{value}}(\text{Update}) > C_{\text{cost}}(\text{Update})$$

- $C_{\text{value}}(\text{Update}) =$
value(newFeatures) + value(removalOfWorkarounds)
- $C_{\text{cost}}(\text{Update}) =$
cost(Downtime) + cost(Training) + cost(updateEffort) + cost(lostFunctionality)
+ cost(paymentToVendor)

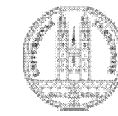
Typical Benefits of Software Products in a Business Context



University of Zurich



Typical Needs of People: Maslow Pyramid



University of Zurich



Selbstverwirklichung

Individualität, Talententfaltung, Perfektion, Erleuchtung.

Soziale Anerkennung

Höhere Wertschätzung durch Status, Respekt, Anerkennung (Auszeichnungen, Lob), Wohlstand, Geld, Einfluss, private und berufliche Erfolge, mentale und körperliche Stärke.

Soziale Beziehungen

Familie, Freundeskreis, Partnerschaft, Liebe, Intimität, Kommunikation.

Sicherheit

Recht und Ordnung, Schutz vor Gefahren, fester Arbeitsplatz, Absicherung.

Körperliche Bedürfnisse

Freiheit, Atmung, Schlaf, Nahrung, Wärme, Gesundheit, Wohnraum, Sexualität.

Ten Misconceptions about Product Software Releasing



- Customers want to stay up-to-date
 - If a newer release package does not provide the customer with new functions, why should she update ($C_{\text{cost}} > C_{\text{value}}$)?
- Customers must stay up-to-date
 - The customer does not necessarily feel limited in the use of an old product ($C_{\text{cost}} > C_{\text{value}}$).
- Release $n+1$ is better for a customer than release n
 - GUI changes may lead to training cost ($C_{\text{cost}} > C_{\text{value}}$).
- Fixes can be postponed to the next major release
 - Major releases risk to be delayed, delaying the fixes ($V_{\text{cost}} > V_{\text{value}}$).
- Workarounds must be avoided at all costs
 - No need for change as long as the workarounds are known and common practice ($C_{\text{cost}} > C_{\text{value}}$).

Ten Misconceptions about Product Software Releasing



University of Zurich



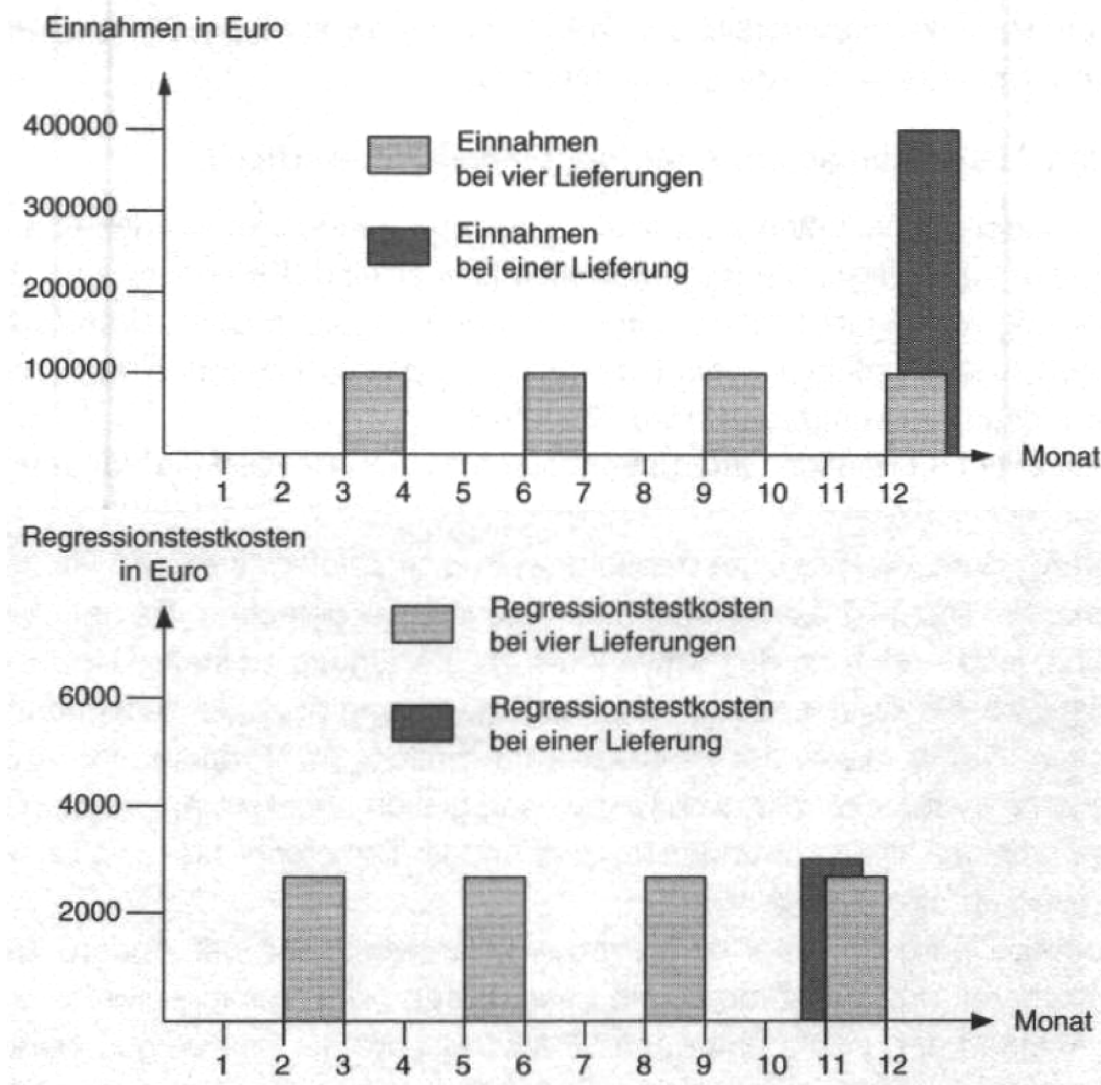
- Customers always want new features
 - GUI changes may lead to decreased efficiency and increased error rates of users ($C_{\text{cost}} > C_{\text{value}}$).
- Releasing too often is bad
 - Too frequent releases may increase the number of bug reports and decrease product experience ($V_{\text{cost}} > V_{\text{value}}$ and $C_{\text{cost}} > C_{\text{value}}$).
- A quiet customer is a happy customer
 - Helpdesk use may indicate customer satisfaction. A silence customer may have gone with a competitor (not understood $C_{\text{value}} > C_{\text{cost}}$).
- Customers read release notes
 - Customers are interested in tailored communication (less C_{cost}).
- Having many different releases out in the field is bad
 - Customers with old versions may be charged service contracts and should not be forced to update ($V_{\text{cost}} > V_{\text{value}}$).

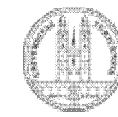


Recommendations

- Number of releases: bug fix > minor > major
- Major update
 - Cost of development >> any other cost
 - May generate or increase business
- Bug fix
 - Creation of the release package $\sim$ cost of development
 - May (only) reduce support calls
- Cost reduction
 - Many customers: introduce automated release package creation and software delivery through a network to reduce V_{cost} .
- Attractiveness increase
 - Ensure that a release increases C_{value} .
 - Introduce automated update process and inform about new features with indexable release notes to reduce C_{cost} .

Frequent vs. Few Deliveries





Contents

Learning Objectives

- Understand software lifecycle
- Understand critical practices to support software launch and lifecycle
 - Acceptance Testing
 - Delivery
 - Product Support Organization

Agenda

- Software Lifecycle
- Software Launch
 - Acceptance Testing
 - Delivery
- Software Lifecycle
 - Evolution
 - Servicing
 - Phase-Out and Close-Down

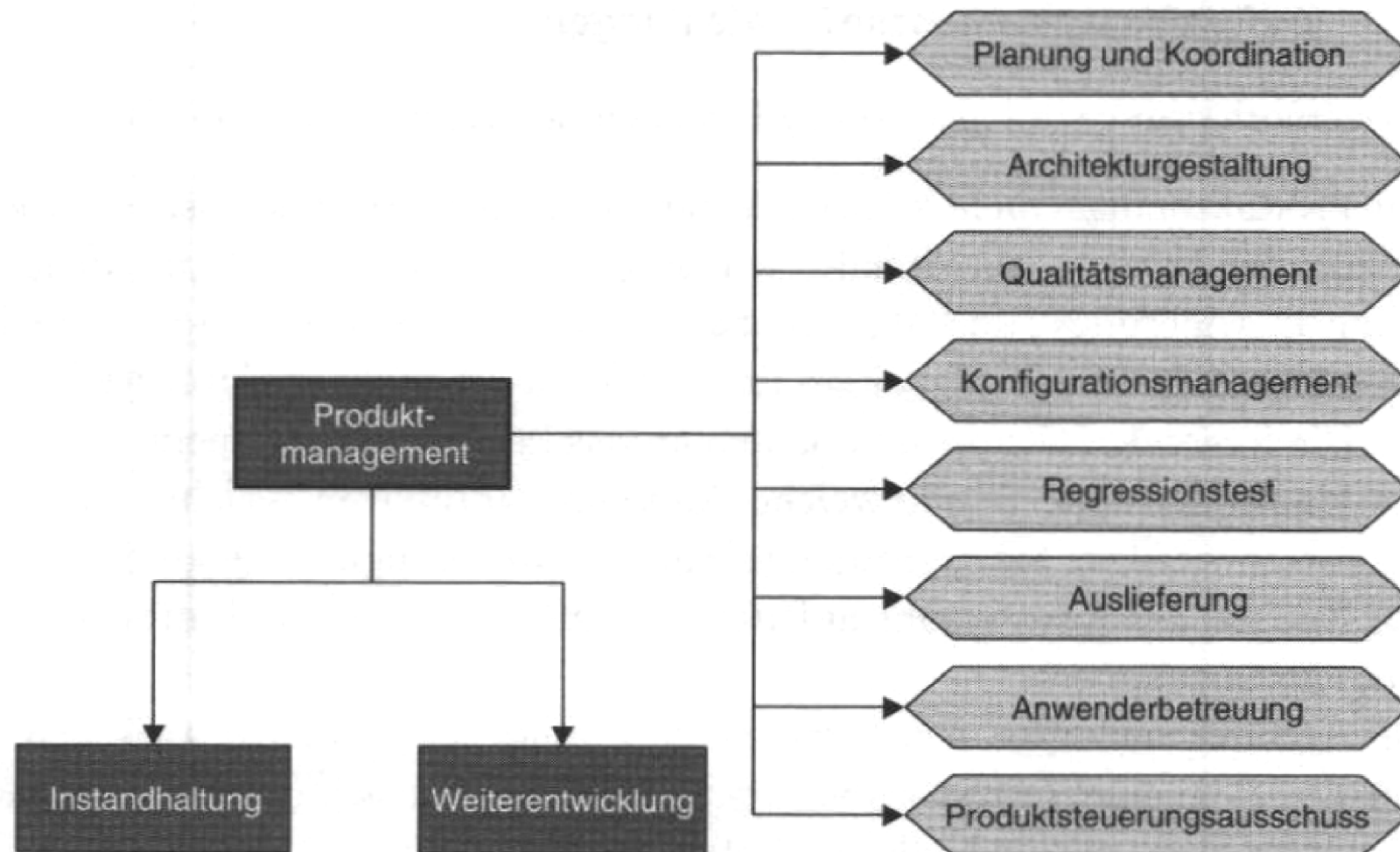


Initial Development

- Software team expertise
 - System architecture
 - Software product that satisfies initial requirements
 - First user base
-
- Often the software is released during evolution after the software has gone through several **internal iterations** to address glaring deficiencies and ensure a stable fault rate.
 - The **release date** is based on sometimes-conflicting technical and business considerations, such as time to market, time to delivery, software stability, and fault rates.



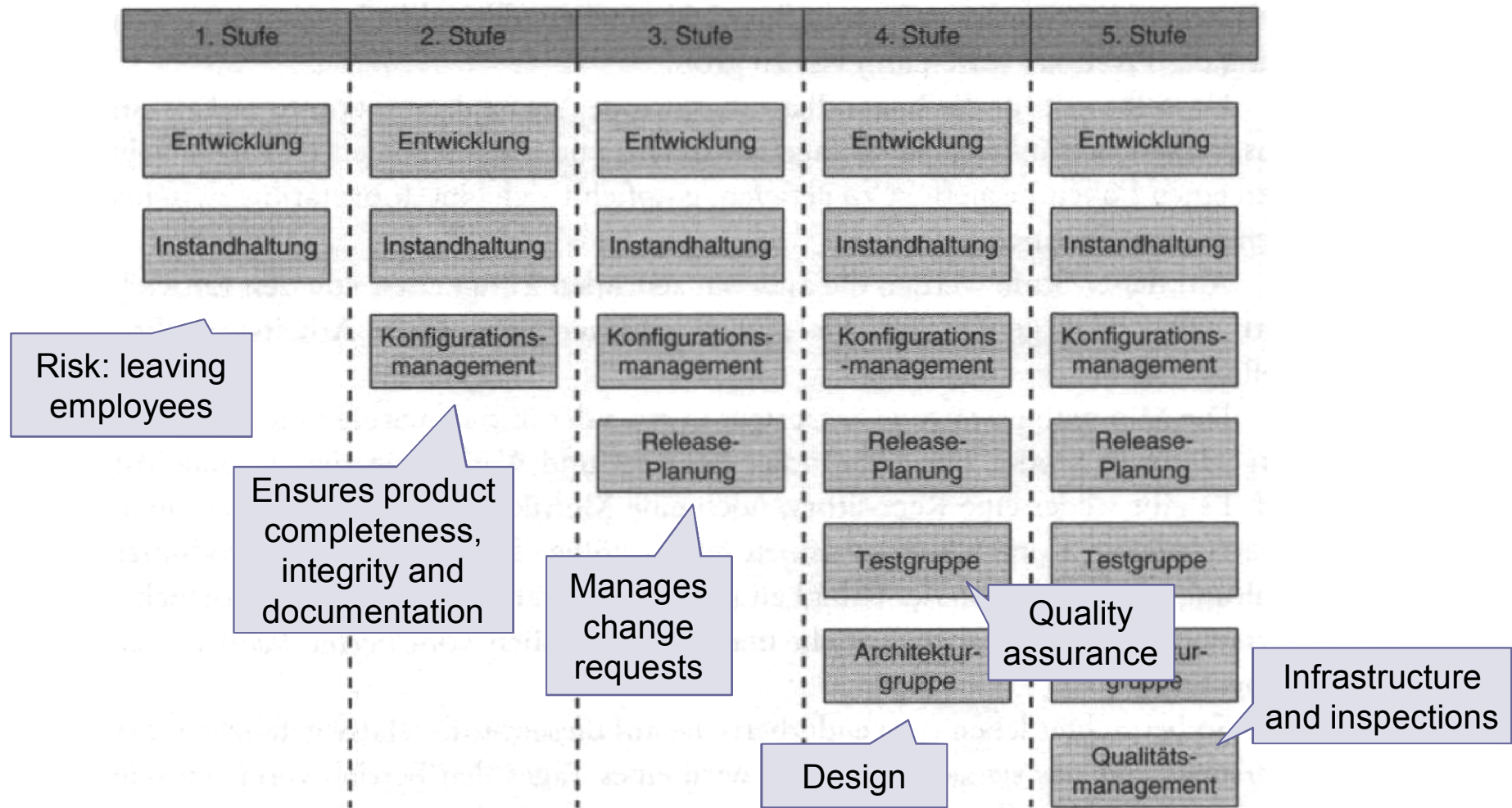
Product Support Organization



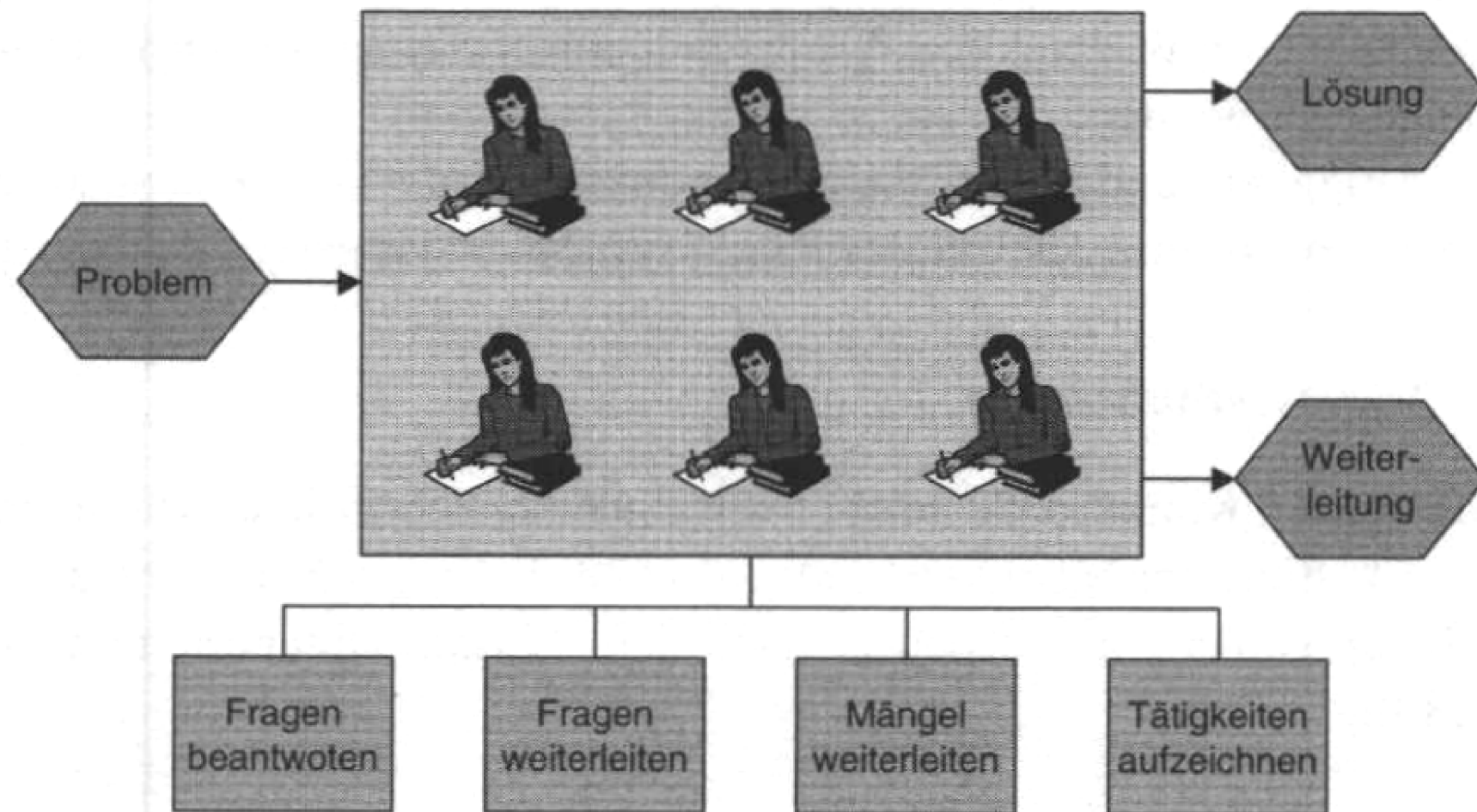
Growth of a Product Support Organization



University of Zurich



Helpdesk Process



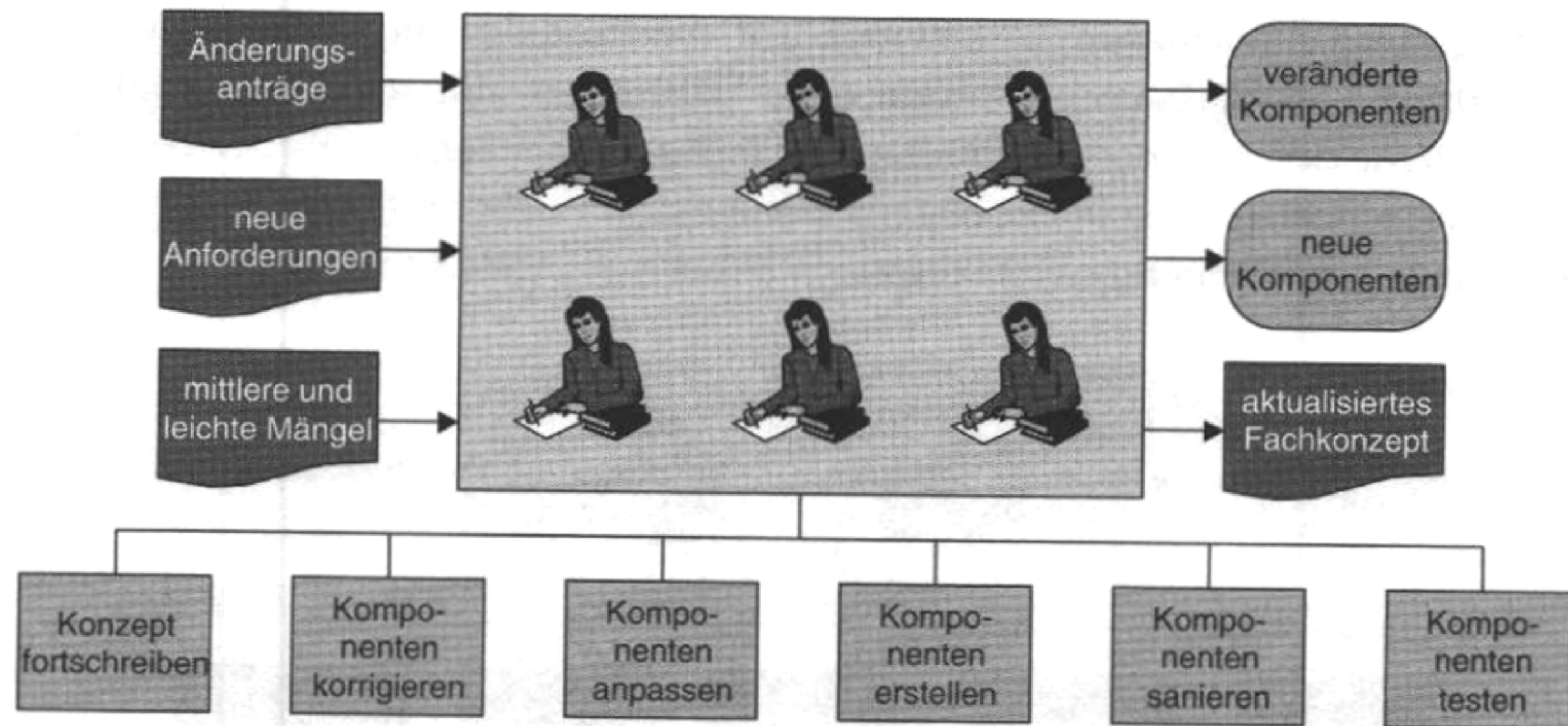


Evolution

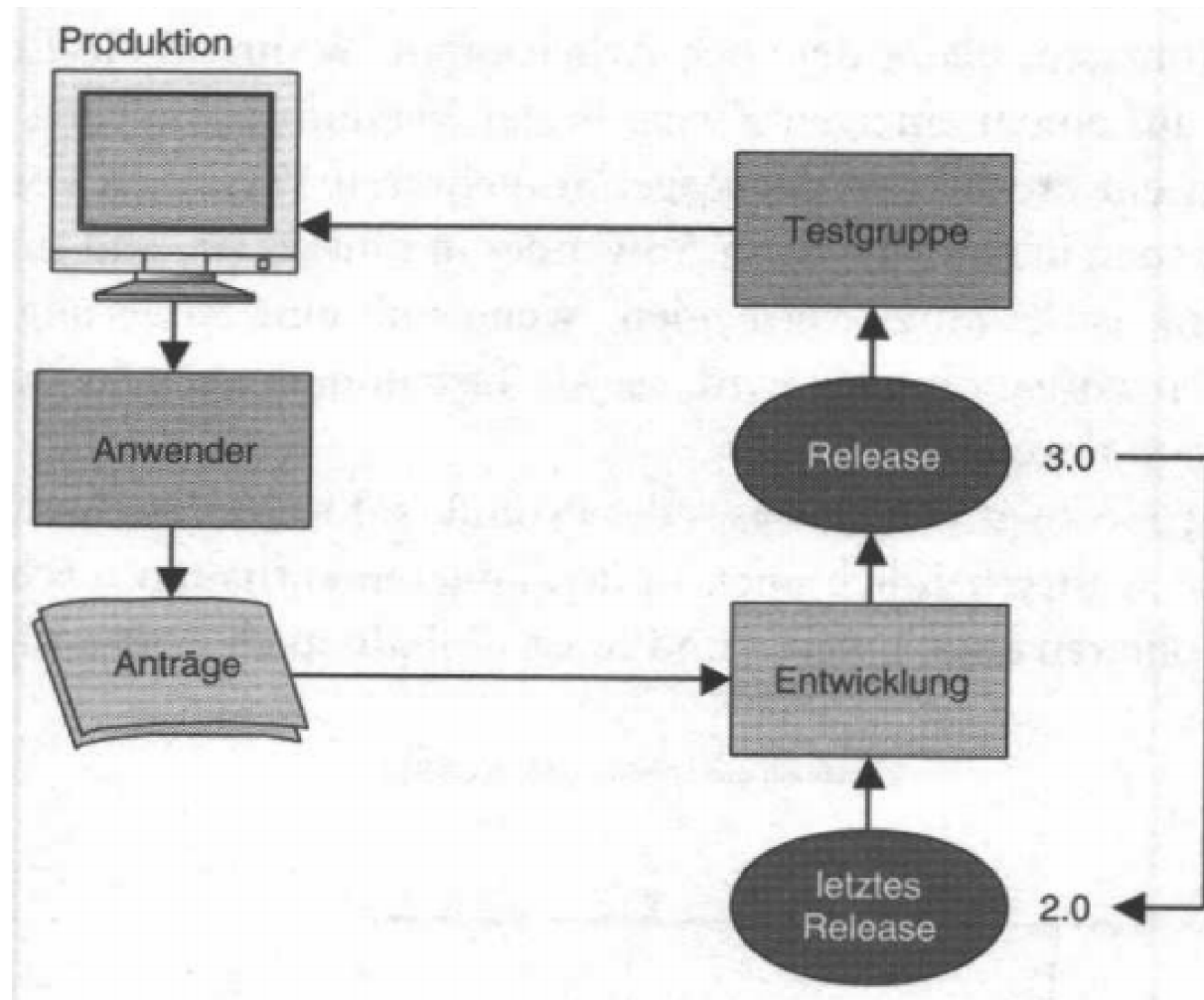
- 30%+ feature change: iterative changes, modifications, and deletions of functionality
- Learning process of the supplier based on feedback from user base
- Increased customer demands
- Competitive pressures
- Legislative actions
- Changes in business practice or operating environment



Evolution Process



Release Management Process

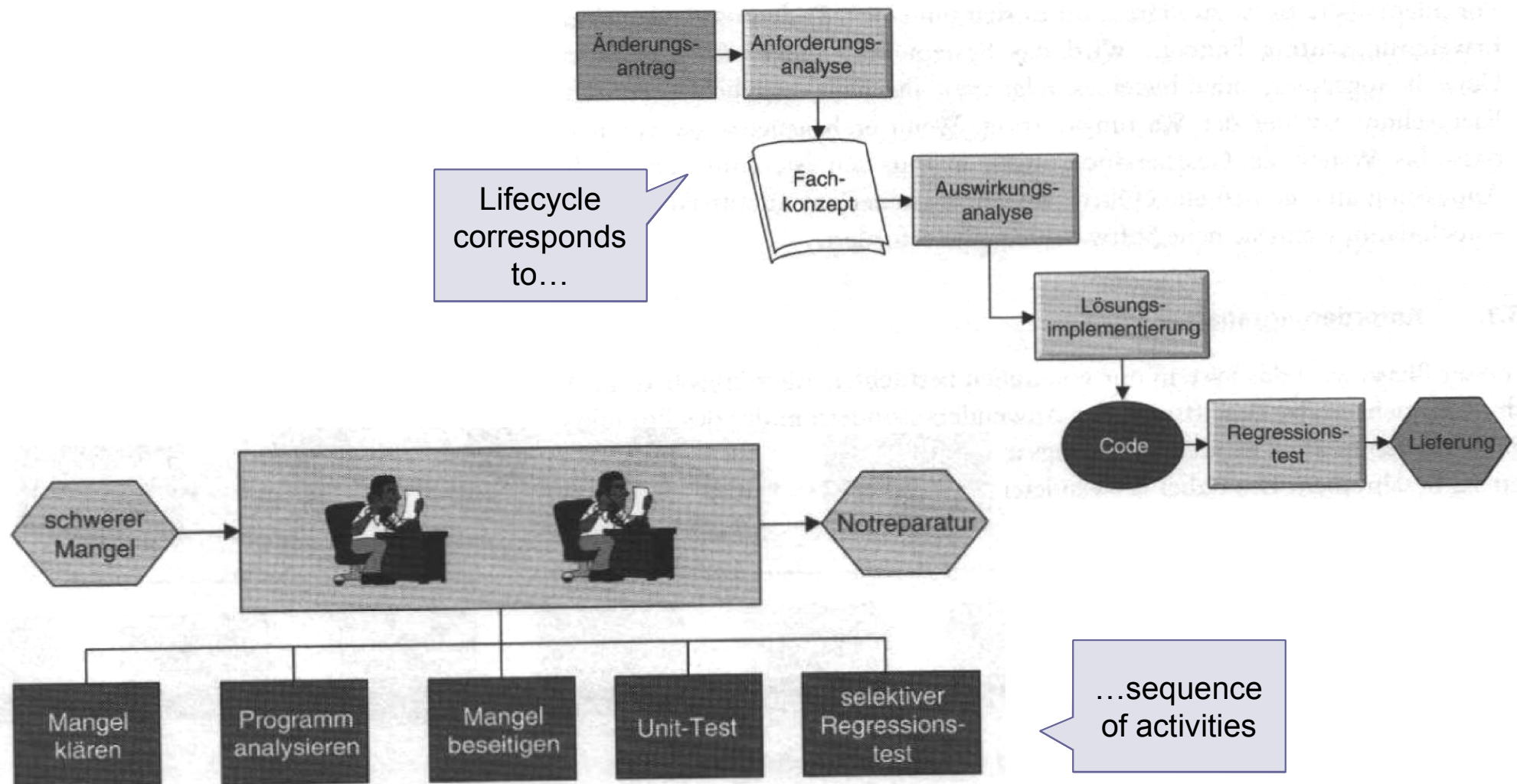




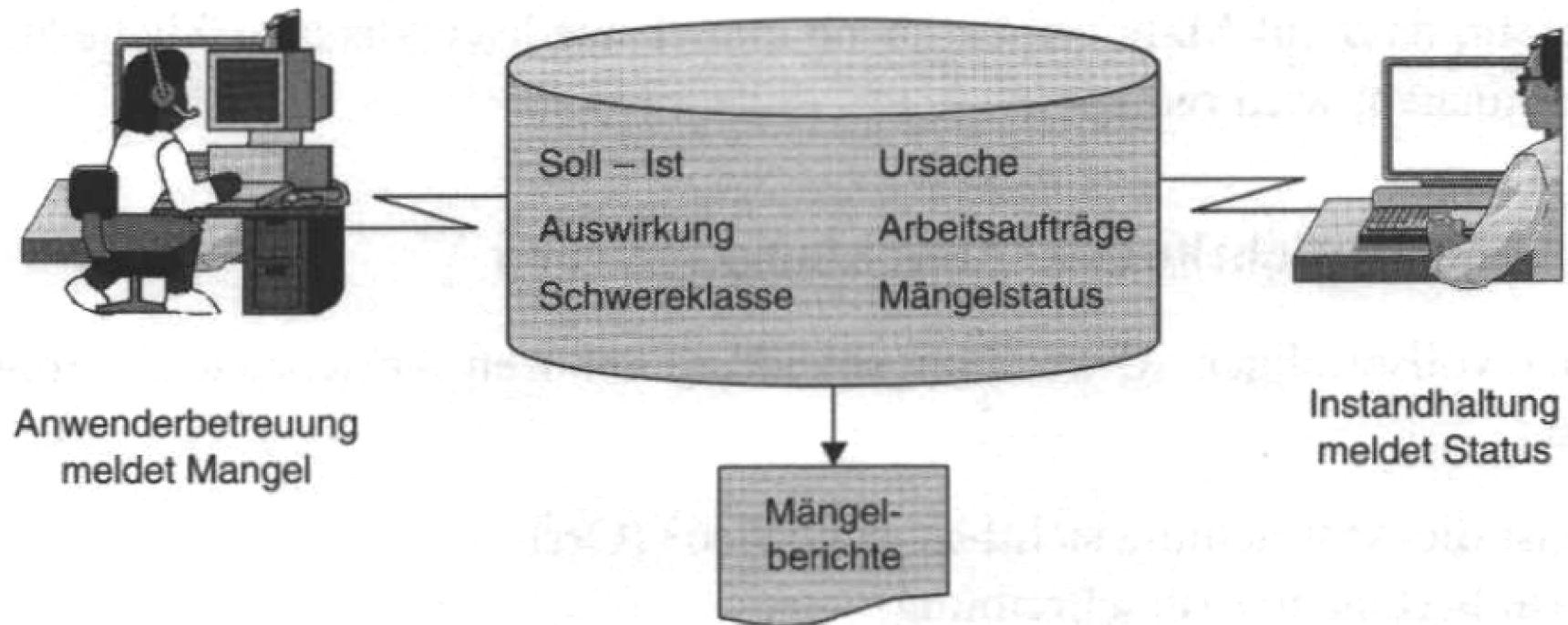
Servicing

- Software enters servicing or saturation stage and is considered aging, decayed, or legacy.
 - Decreasing fitness of architecture to system context
 - Decreasing skills of development team with regard to software solution
- Changes are both difficult and expensive, so developers minimize them or do them as wrappers, which are simply modifications to inputs and outputs, leaving the old software untouched.
 - Each change further degrades the architecture, pushing it deeper into servicing

Servicing Process

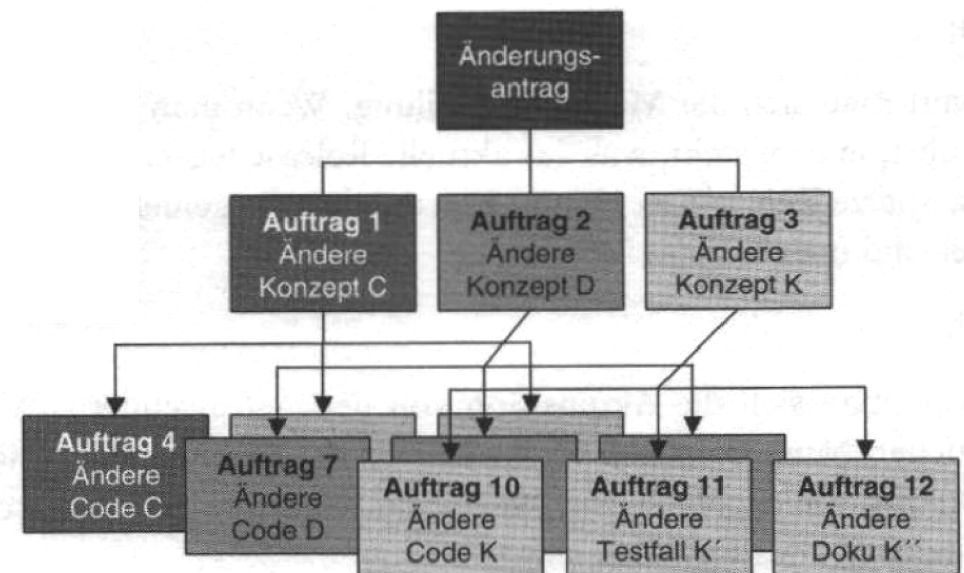
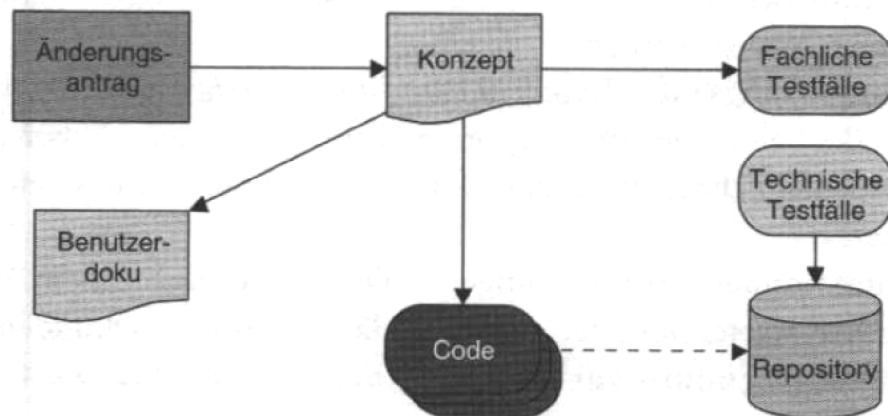


Issue Database

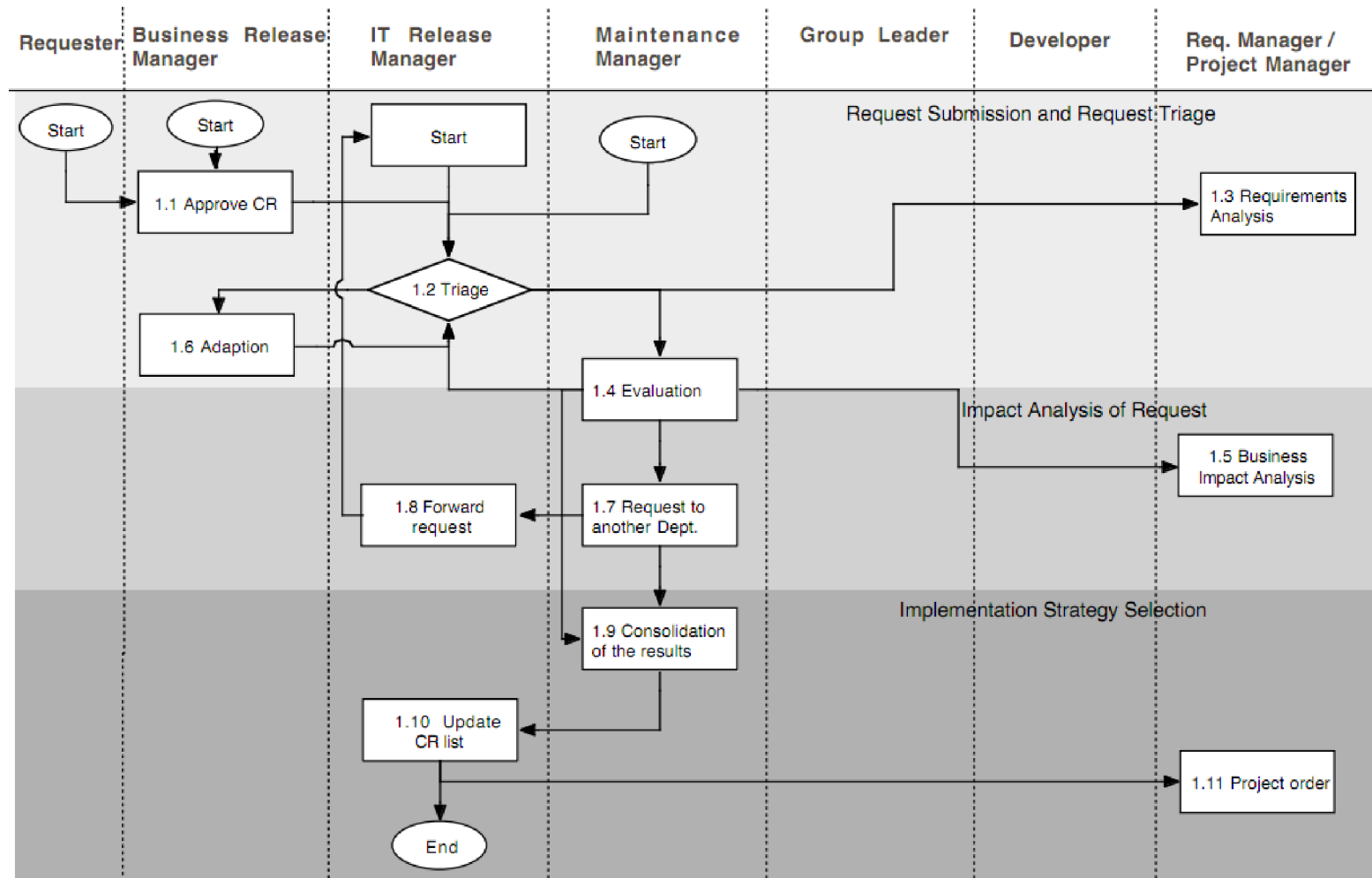




Issue Implementation

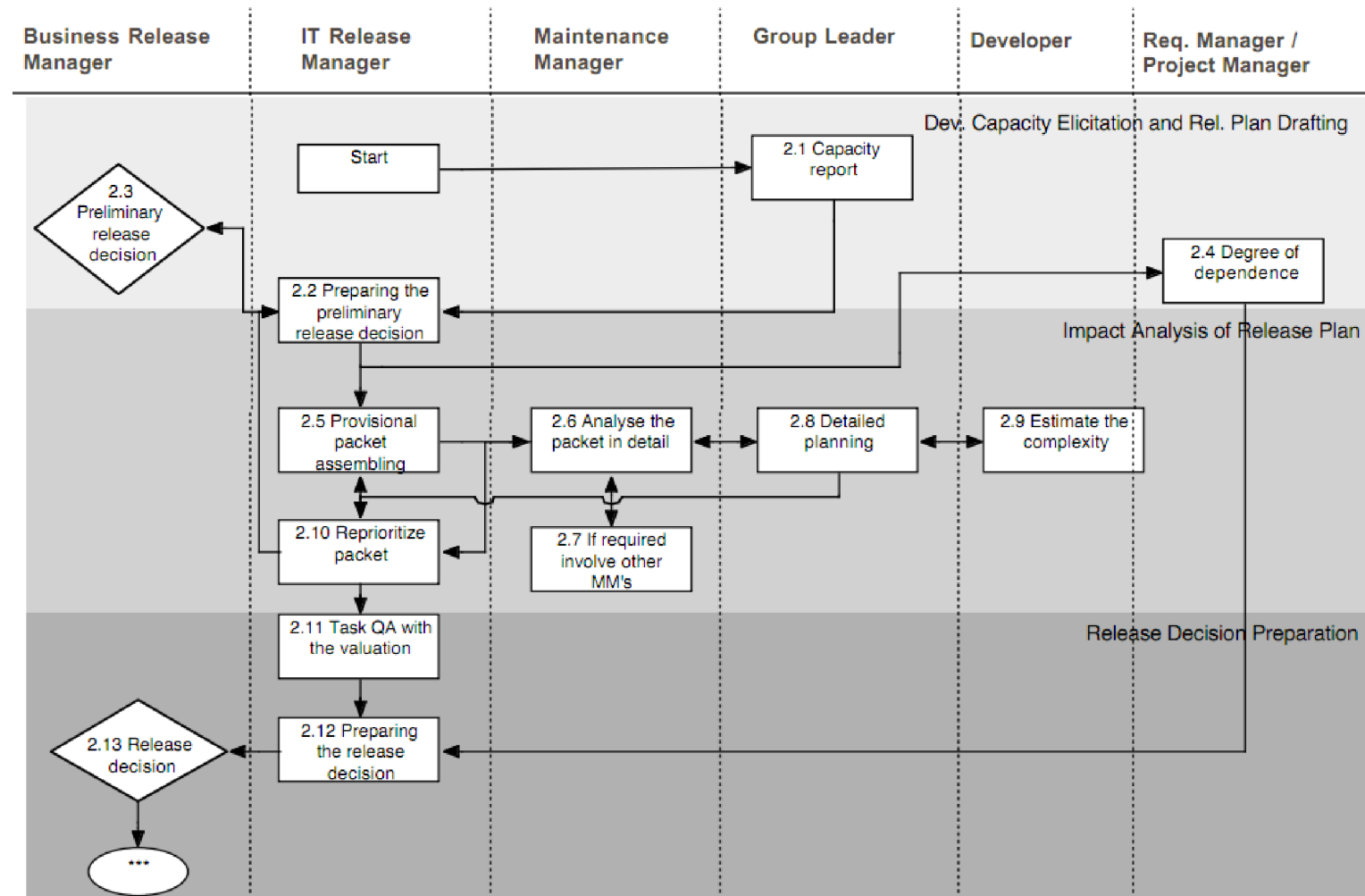


Real-World Example

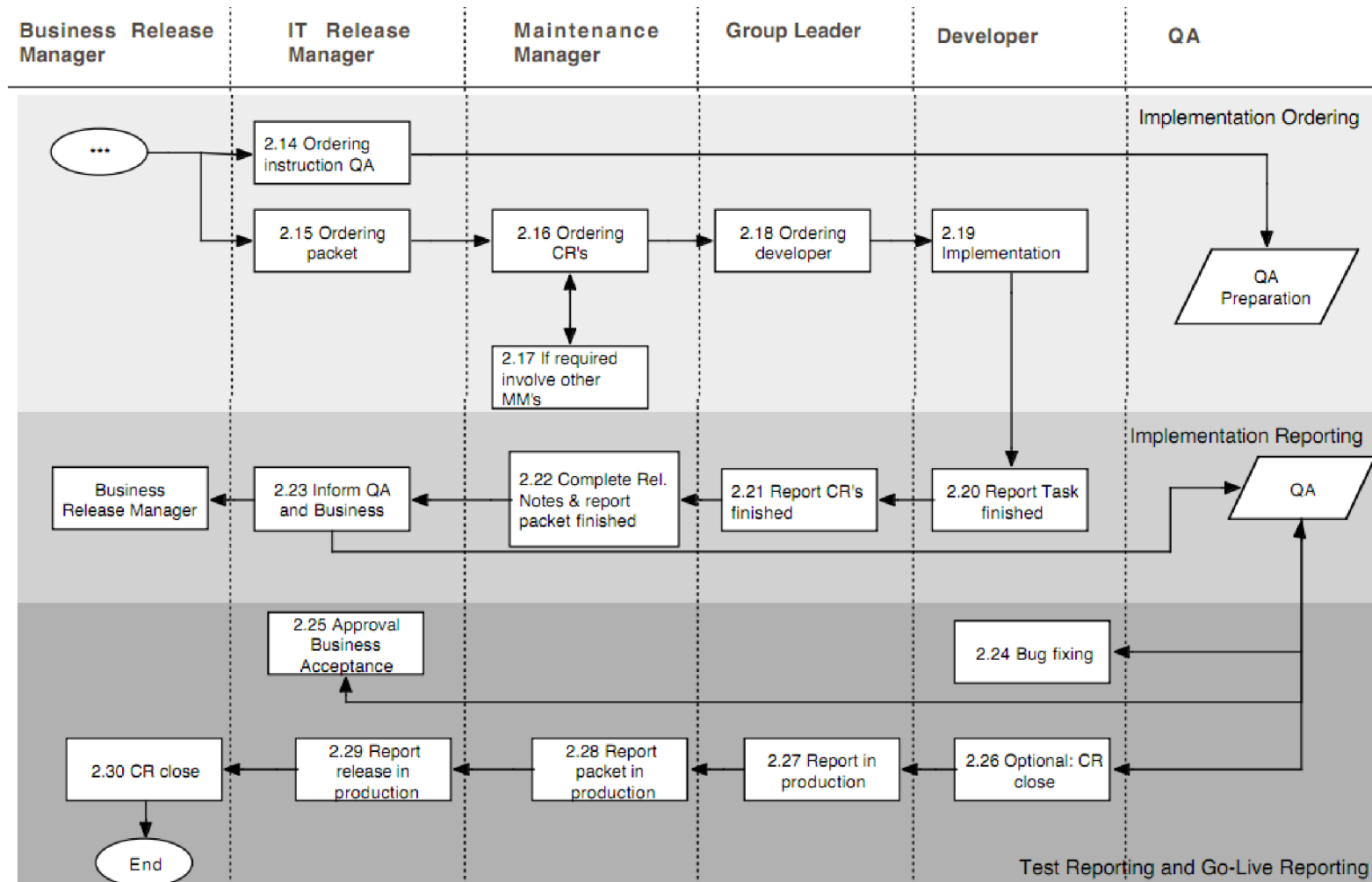


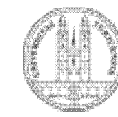


Real-World Example



Real-World Example





Issue Analysis

- Goal: recognize how product quality evolves

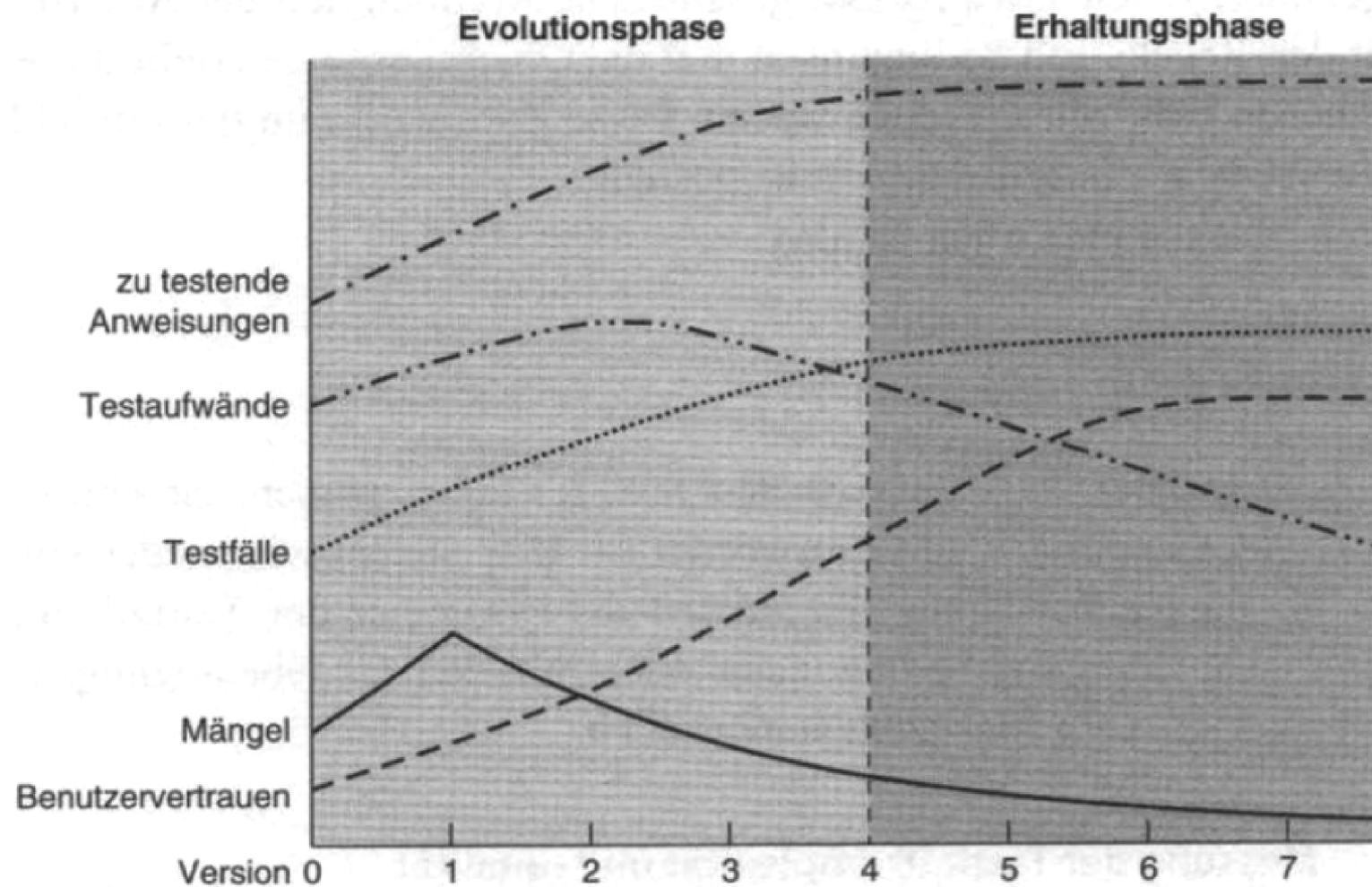
TEST SUMMARY REPORT (by Test Cycle)					
DATE: 01.06.2004					FIRM: TENSOR
SYSTEM: TENSOR		SYSTEM DEFECTS			PAGE: 01
Release	Faults	Severe	Medium	Minor	Undefined
1.2.0	0001	0000	0000	0001	0000
1.2.2	0003	0000	0002	0001	0000
1.2.4	0021	0000	0007	0014	0000
1.2.6	0011	0000	0002	0009	0000
1.2.8	0006	0000	0002	0004	0000
1.2.10	0002	0001	0001	0000	0000

Differences due to changing software quality and differences in user population and usage duration

Insights should lead to focused improvements and process development

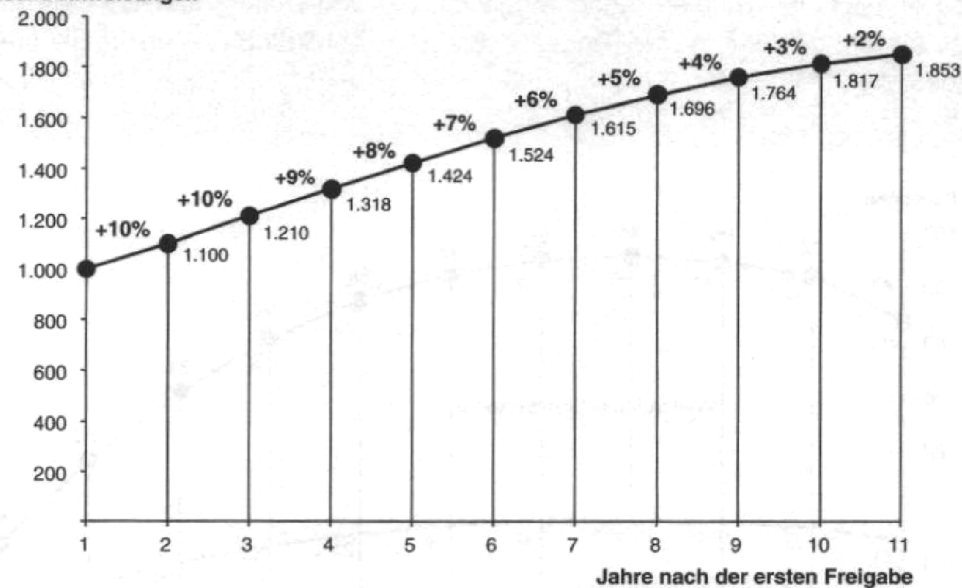
TEST SUMMARY REPORT (by Cause)					
DATE: 01.06.2004					FIRM: TENSOR
SYSTEM: TENSOR		DEFECT RATES (per 1000)			PAGE: 04
Component	Analysis	Design	Code	Config	Documentation
KFE	0.030	0.017	0.300	0.000	0.000
ZEU	0.000	0.000	0.470	0.000	0.001
KKD	0.003	0.208	0.809	0.010	0.007
WER	0.000	0.002	1.704	0.000	0.003
AQA	0.040	0.037	1.001	0.001	0.006
...	...	...	...	...	...

Evolution of Key Metrics

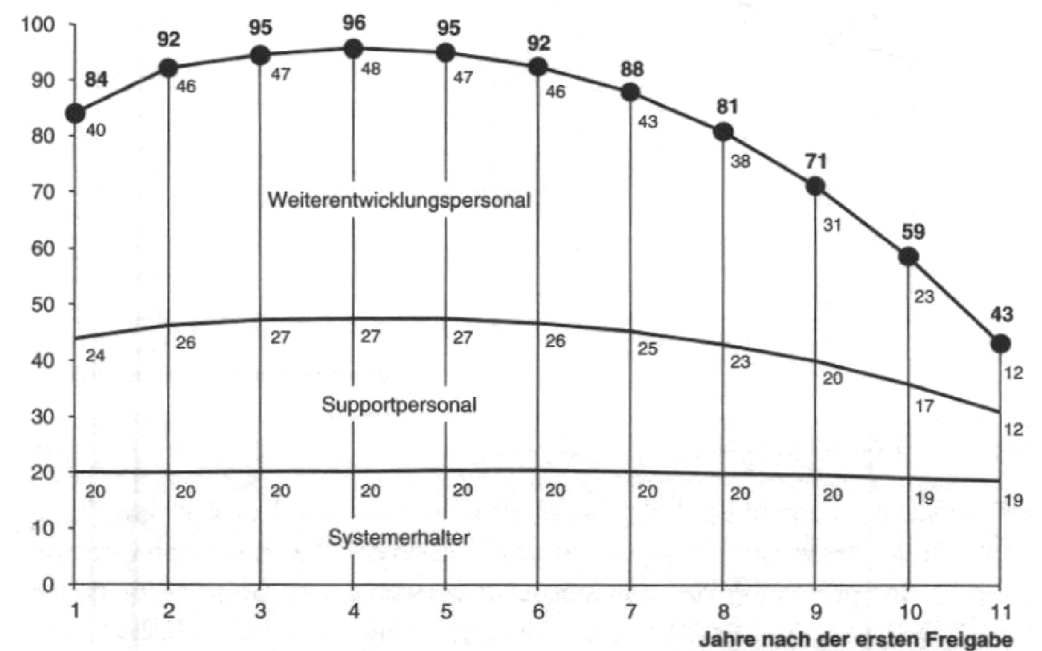


Model Calculations

Größe des Systems in
tausend Anweisungen



Personen





Phaseout



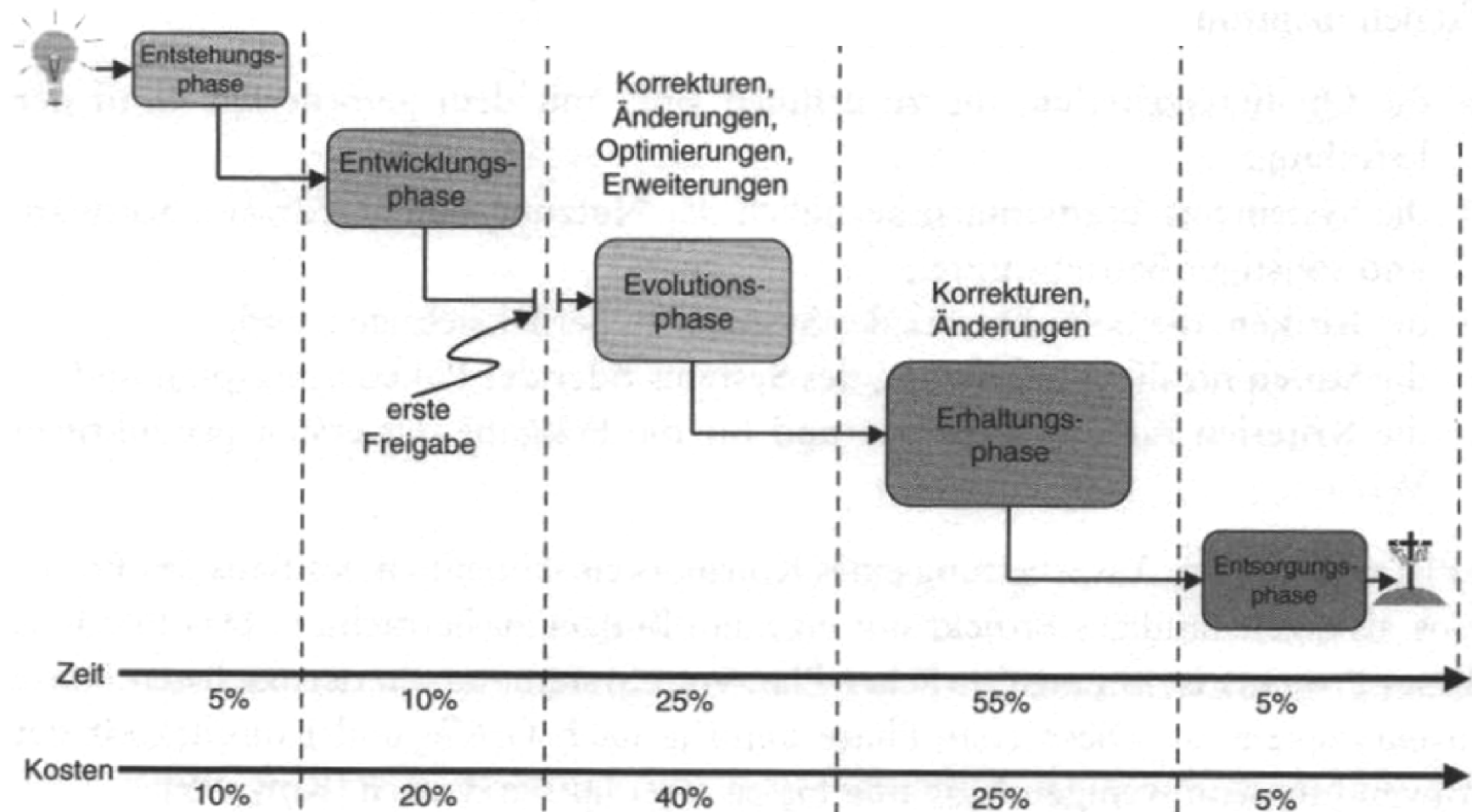
- No more servicing
- Supplier tries to generate revenue, or other benefits, from the unchanged software as long as possible
- Software, with no changes made, becomes increasingly outdated
- Users must work around software deficiencies
- Growing backlog of change requests hinders return to servicing stage



Closedown

- Supplier shuts down the software
- Supplier directs users to a replacement system
- Residual responsibilities
 - Source code retention
 - Legal liability
- Migrate data to a new system
- Examples of shut-downs:
<http://37signals.com/svn/posts/2682-the-creators-of-no-longer-with-us-products-explain-what-went-wrong>

Relative Time and Cost





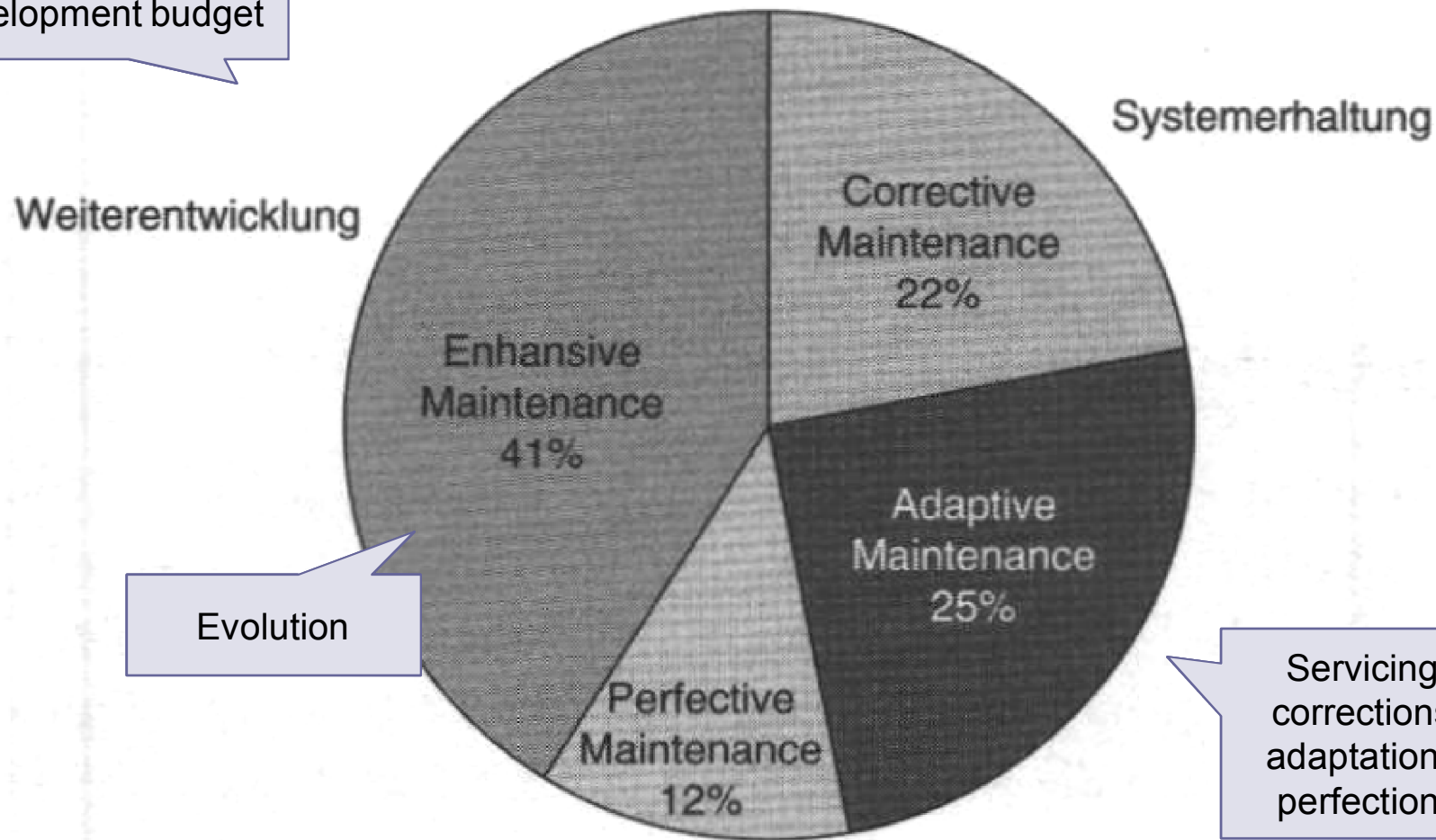
Changes over Software Phases

	Expertise	Architecture	Software Decay	Economics
Initial	Staff must understand domain, solutions to domain problems, program properties	Initial architecture		Heavy investment with no return. Balance evolvability with time-to-market.
Evolution		Partial architecture rebuilds to facilitate future changes	Architecture degradation requires expertise, which continuously leaves. Less expertise leads to more degradation.	Sales, market, and revenue good. Invest in next version.
Servicing	Staff must understand software interfaces	Wrappers damages the architecture and makes it hard to understand		Sales fall off and key team members turn to other projects. Work is predictable and well understood.
Phaseout	Staff must understand how to execute the software	No change	Phase out wrapped component for component	Extend revenue generation as long as possible.
Close-down				



Relative Development Budget (487 Organizations)

Ca. 50% of total
development budget





Contents

Learning Objectives

- Understand software lifecycle
- Understand critical practices to support software launch and lifecycle
 - Acceptance Testing
 - Delivery
 - Product Support Organization

Agenda

- Software Lifecycle
- Software Launch
 - Acceptance Testing
 - Delivery
- Software Lifecycle
 - Evolution
 - Servicing
 - Phase-Out and Close-Down