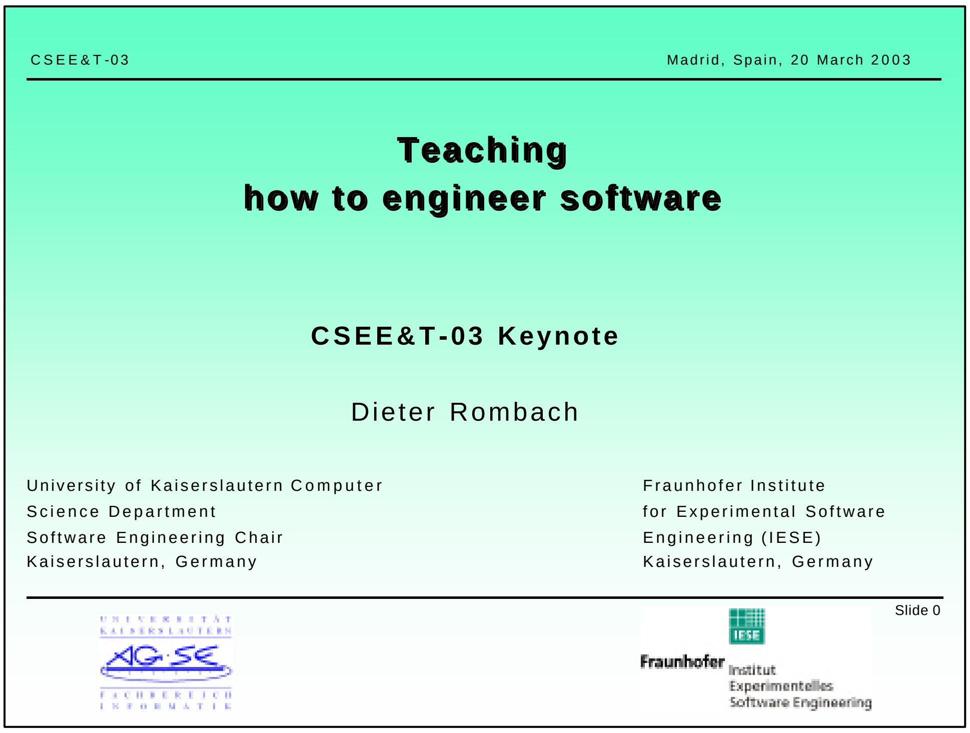


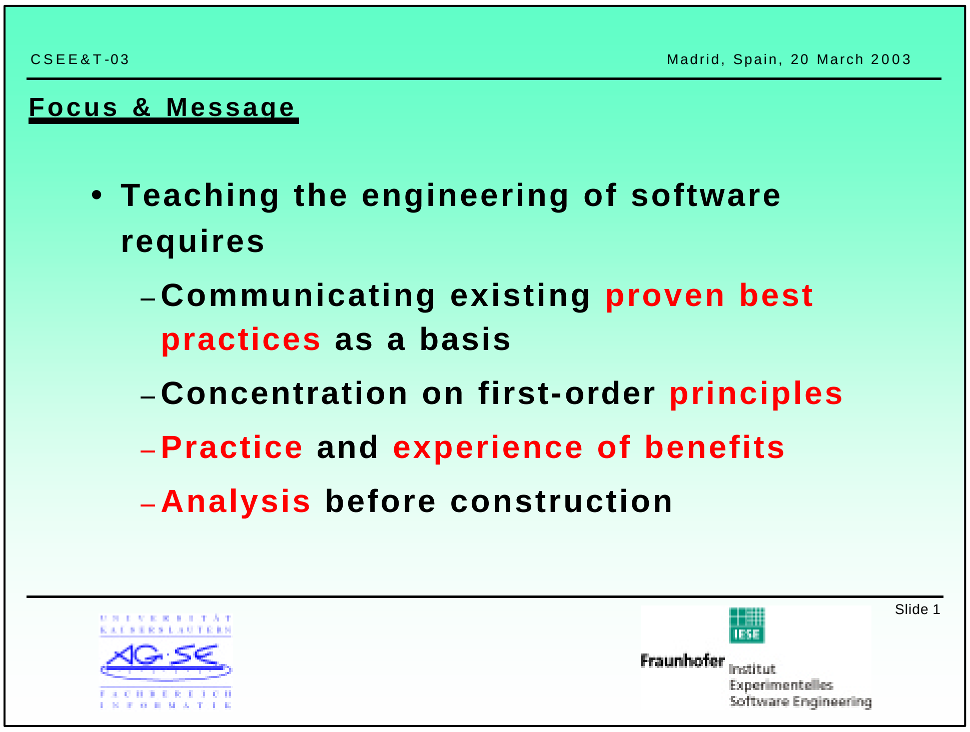
Teaching how to engineer software

CSEE&T-03 Keynote

Dieter Rombach

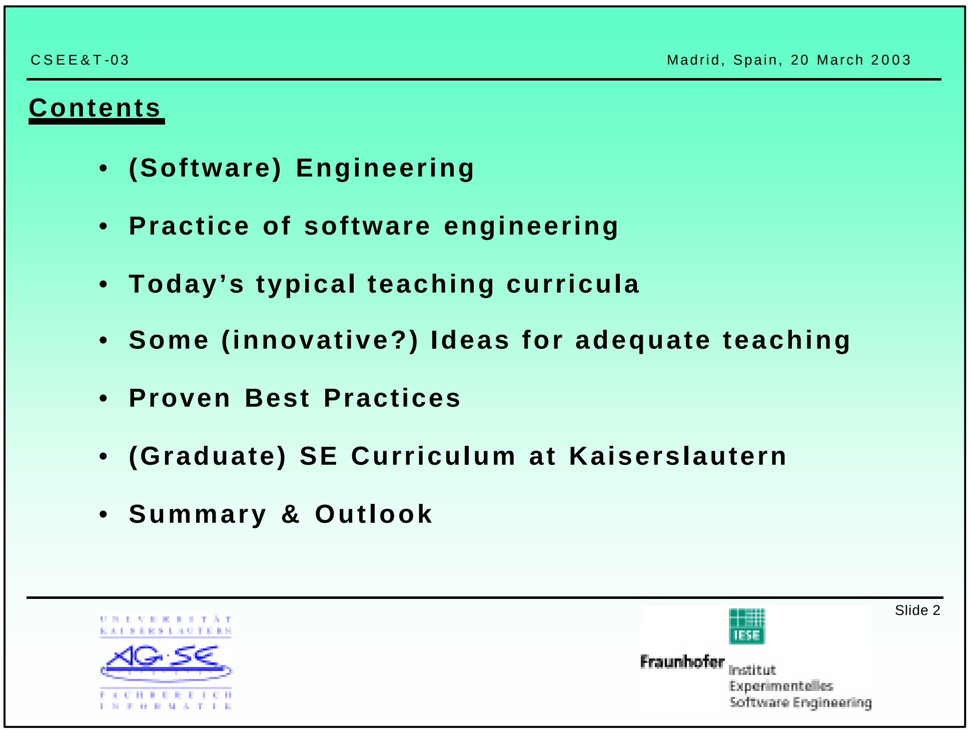
University of Kaiserslautern Computer Science Department
Software Engineering Chair
Kaiserslautern, Germany

Fraunhofer Institute for Experimental Software Engineering (IESE)
Kaiserslautern, Germany



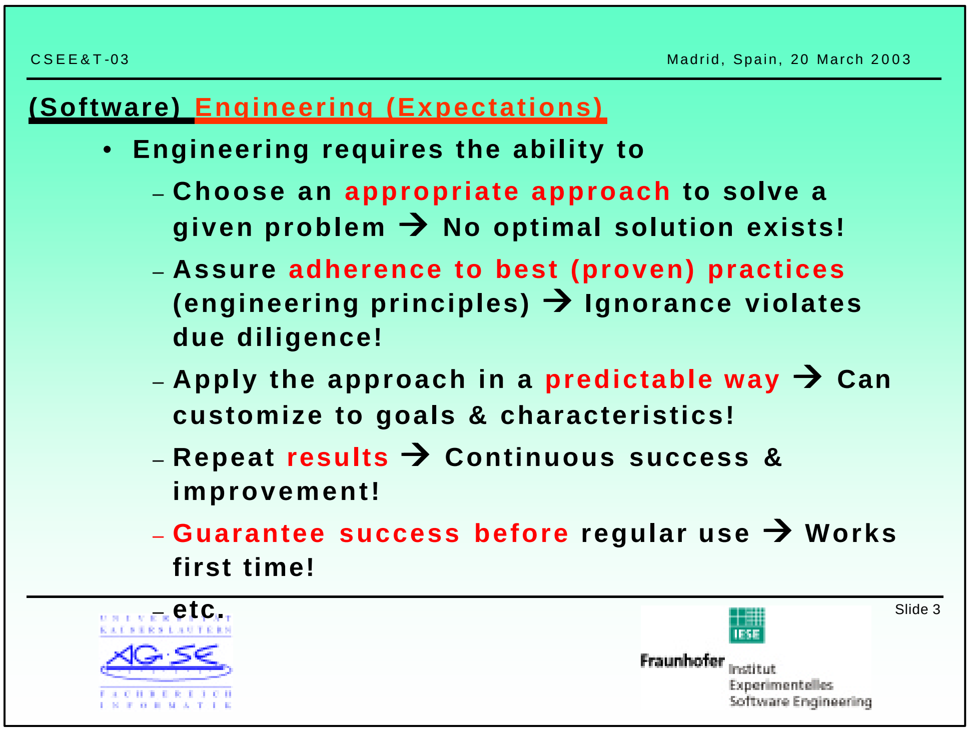
Focus & Message

- Teaching the engineering of software requires
  - Communicating existing proven best practices as a basis
  - Concentration on first-order principles
  - Practice and experience of benefits
  - Analysis before construction



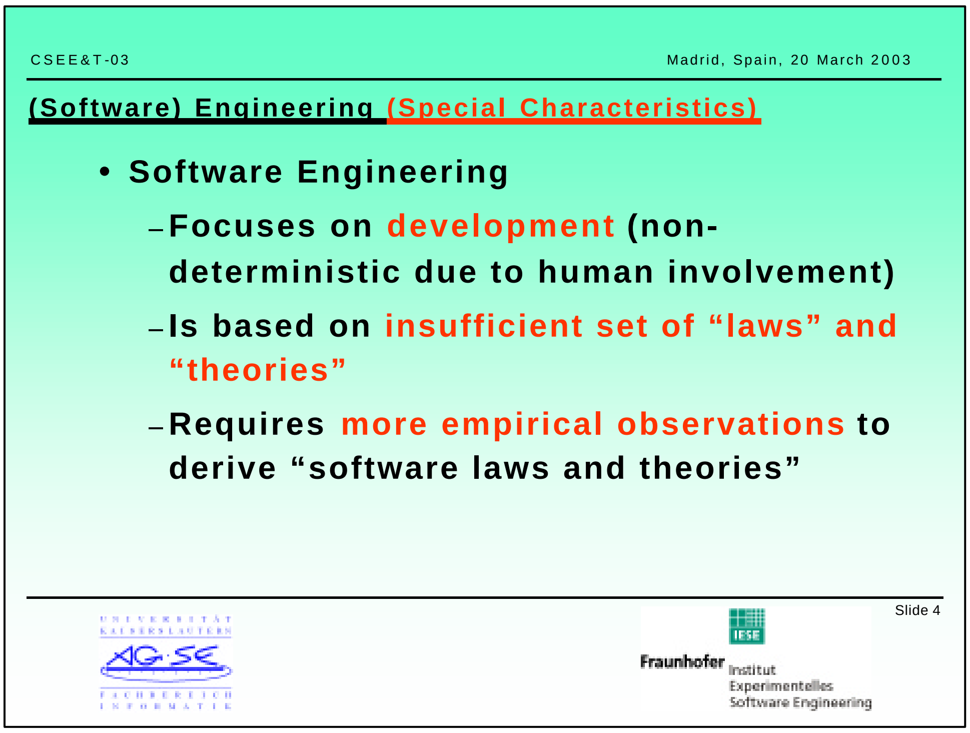
Contents

- (Software) Engineering
- Practice of software engineering
- Today's typical teaching curricula
- Some (innovative?) Ideas for adequate teaching
- Proven Best Practices
- (Graduate) SE Curriculum at Kaiserslautern
- Summary & Outlook



(Software) Engineering (Expectations)

- Engineering requires the ability to
  - Choose an appropriate approach to solve a given problem → No optimal solution exists!
  - Assure adherence to best (proven) practices (engineering principles) → Ignorance violates due diligence!
  - Apply the approach in a predictable way → Can customize to goals & characteristics!
  - Repeat results → Continuous success & improvement!
  - Guarantee success before regular use → Works first time!
  - etc.



CSEE&T-03 Madrid, Spain, 20 March 2003

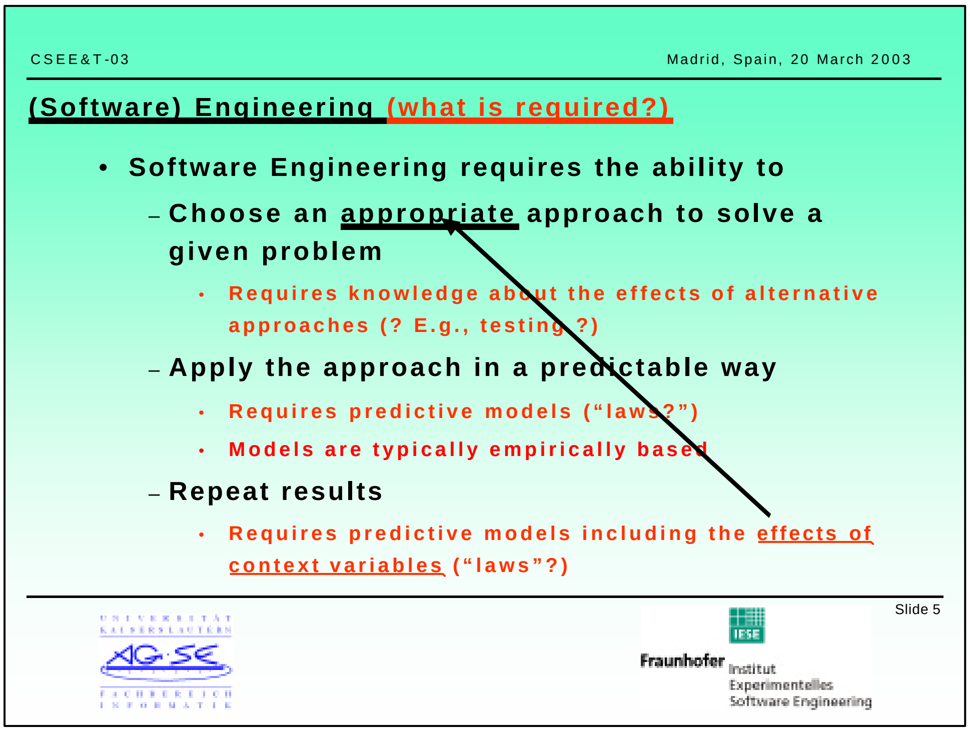
(Software) Engineering (Special Characteristics)

- Software Engineering
  - Focuses on development (non-deterministic due to human involvement)
  - Is based on insufficient set of “laws” and “theories”
  - Requires more empirical observations to derive “software laws and theories”

Slide 4

UNIVERSITÄT KAISERSLAUTERN AG SE FACHBEREICH INFORMATIK

IESE Fraunhofer Institut Experimentelles Software Engineering



CSEE&T-03 Madrid, Spain, 20 March 2003

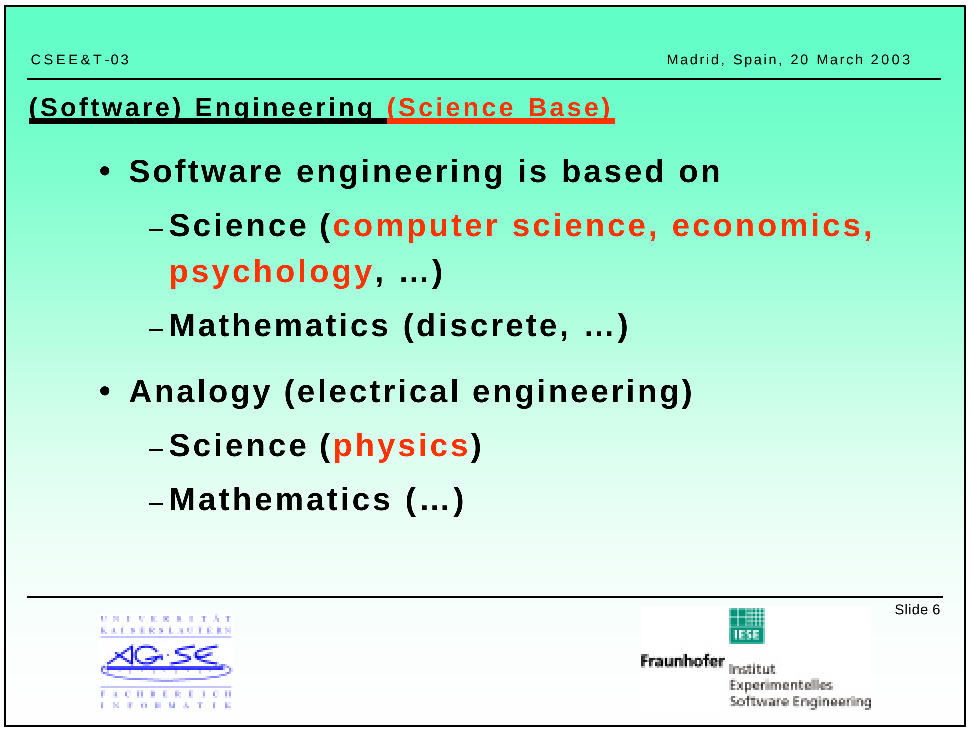
(Software) Engineering (what is required?)

- Software Engineering requires the ability to
  - Choose an <u>appropriate</u> approach to solve a given problem
    - Requires knowledge about the effects of alternative approaches (? E.g., testing ?)
  - Apply the approach in a predictable way
    - Requires predictive models (“laws?”)
    - Models are typically empirically based
  - Repeat results
    - Requires predictive models including the <u>effects of context variables</u> (“laws”?)

Slide 5

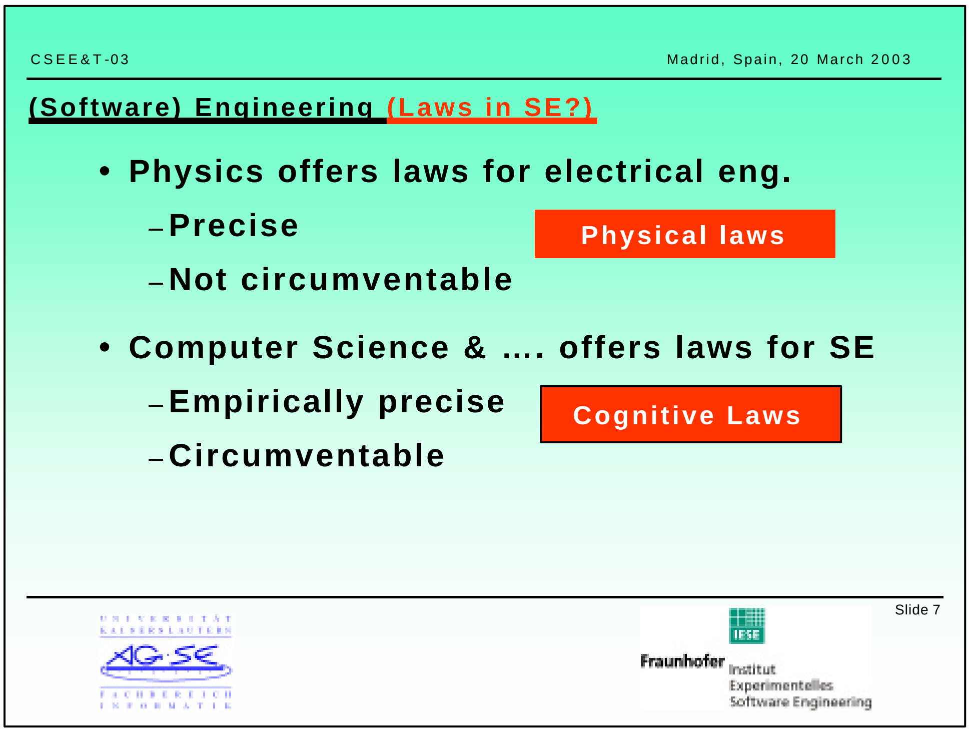
UNIVERSITÄT KAISERSLAUTERN AG SE FACHBEREICH INFORMATIK

IESE Fraunhofer Institut Experimentelles Software Engineering



(Software) Engineering (Science Base)

- Software engineering is based on
  - Science (computer science, economics, psychology, …)
  - Mathematics (discrete, …)
- Analogy (electrical engineering)
  - Science (physics)
  - Mathematics (…)



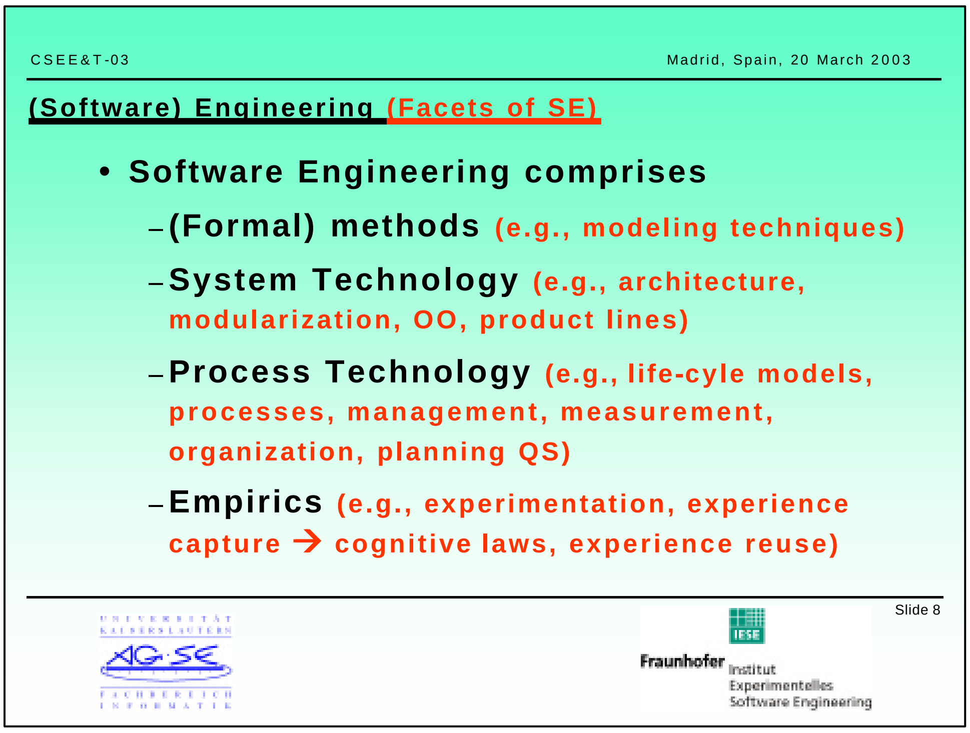
(Software) Engineering (Laws in SE?)

- Physics offers laws for electrical eng.
  - Precise
  - Not circumventable

Physical laws

- Computer Science & …. offers laws for SE
  - Empirically precise
  - Circumventable

Cognitive Laws



CSEE&T-03 Madrid, Spain, 20 March 2003

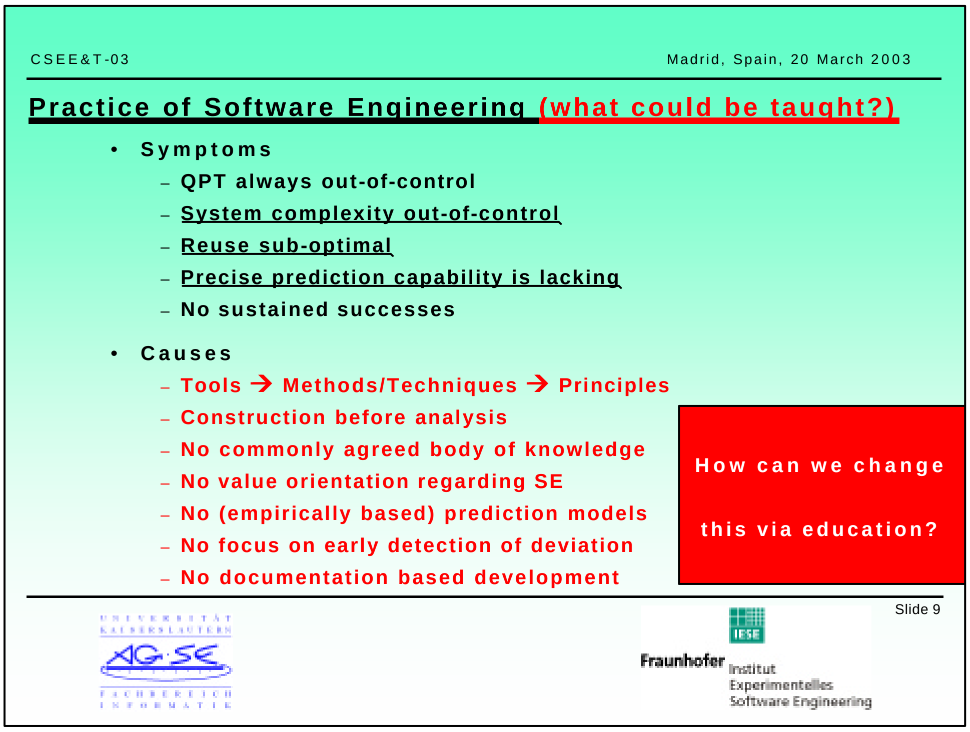
(Software) Engineering (Facets of SE)

- Software Engineering comprises
  - (Formal) methods (e.g., modeling techniques)
  - System Technology (e.g., architecture, modularization, OO, product lines)
  - Process Technology (e.g., life-cyle models, processes, management, measurement, organization, planning QS)
  - Empirics (e.g., experimentation, experience capture → cognitive laws, experience reuse)

Slide 8

UNIVERSITÄT KAISERSLAUTERN AG-SE FACHBEREICH INFORMATIK

IESE Fraunhofer Institut Experimentelles Software Engineering



CSEE&T-03 Madrid, Spain, 20 March 2003

Practice of Software Engineering (what could be taught?)

- Symptoms
  - QPT always out-of-control
  - System complexity out-of-control
  - Reuse sub-optimal
  - Precise prediction capability is lacking
  - No sustained successes
- Causes
  - Tools → Methods/Techniques → Principles
  - Construction before analysis
  - No commonly agreed body of knowledge
  - No value orientation regarding SE
  - No (empirically based) prediction models
  - No focus on early detection of deviation
  - No documentation based development

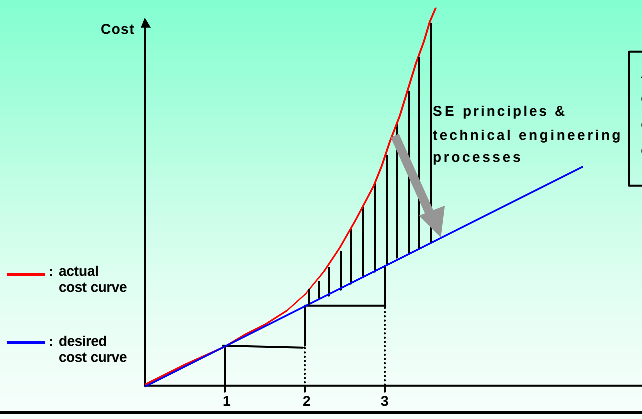
How can we change this via education?

Slide 9

UNIVERSITÄT KAISERSLAUTERN AG-SE FACHBEREICH INFORMATIK

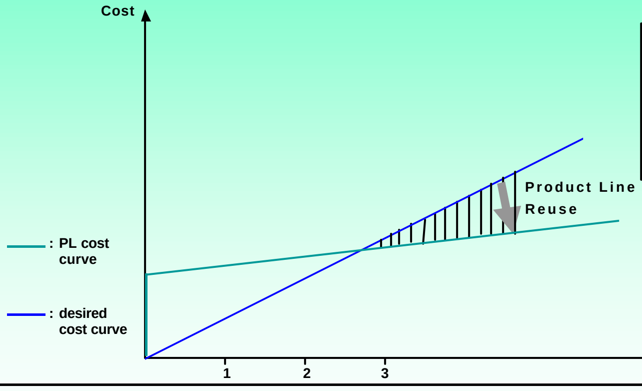
IESE Fraunhofer Institut Experimentelles Software Engineering

System Complexity out-of-control



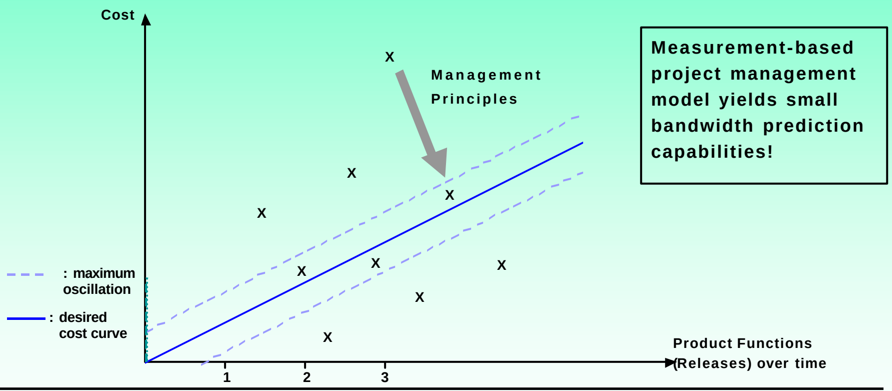
Avoiding deterioration of product complexity over time yields linear cost curve!

Reuse is suboptimal



Exploiting reuse potential with proper scoping (tool support) yields constant cost curve!

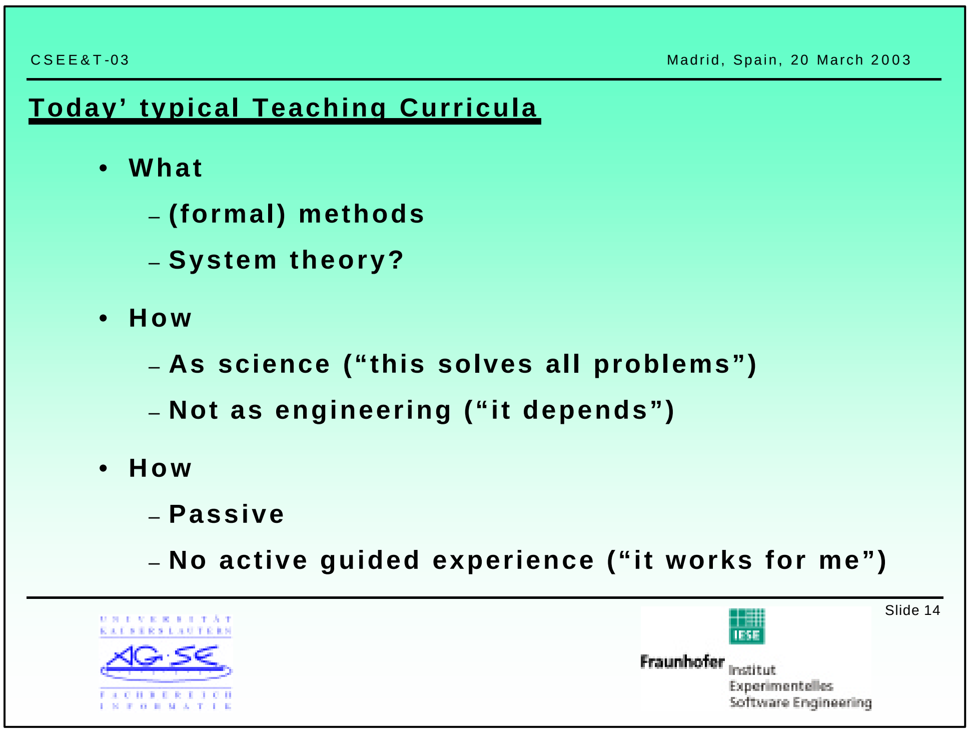
Precise Prediction Capability is lacking



Measurement-based project management model yields small bandwidth prediction capabilities!

Practice of Software Engineering (Challenge)

- Establish proper academic education & industrial training
- The more students graduate with sound software engineering background, the better practice will get


CSEE&T-03 Madrid, Spain, 20 March 2003

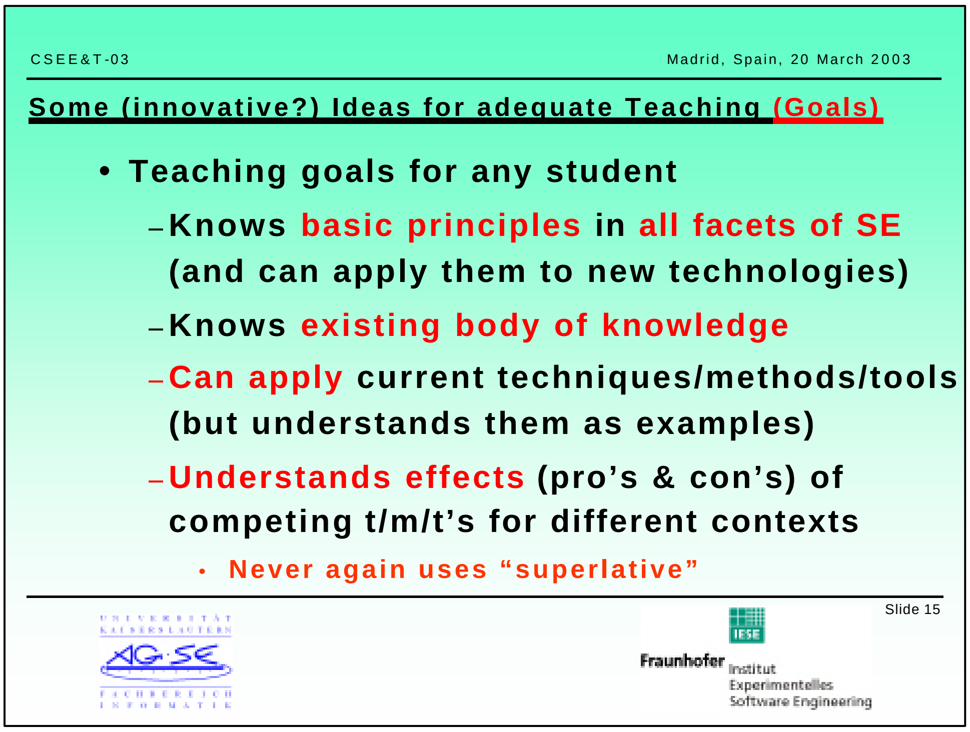
Today' typical Teaching Curricula

- What
  - (formal) methods
  - System theory?
- How
  - As science ("this solves all problems")
  - Not as engineering ("it depends")
- How
  - Passive
  - No active guided experience ("it works for me")

UNIVERSITÄT KAISERSLAUTERN AG SE FACHBEREICH INFORMATIK

IESE Fraunhofer Institut Experimentelles Software Engineering

Slide 14


CSEE&T-03 Madrid, Spain, 20 March 2003

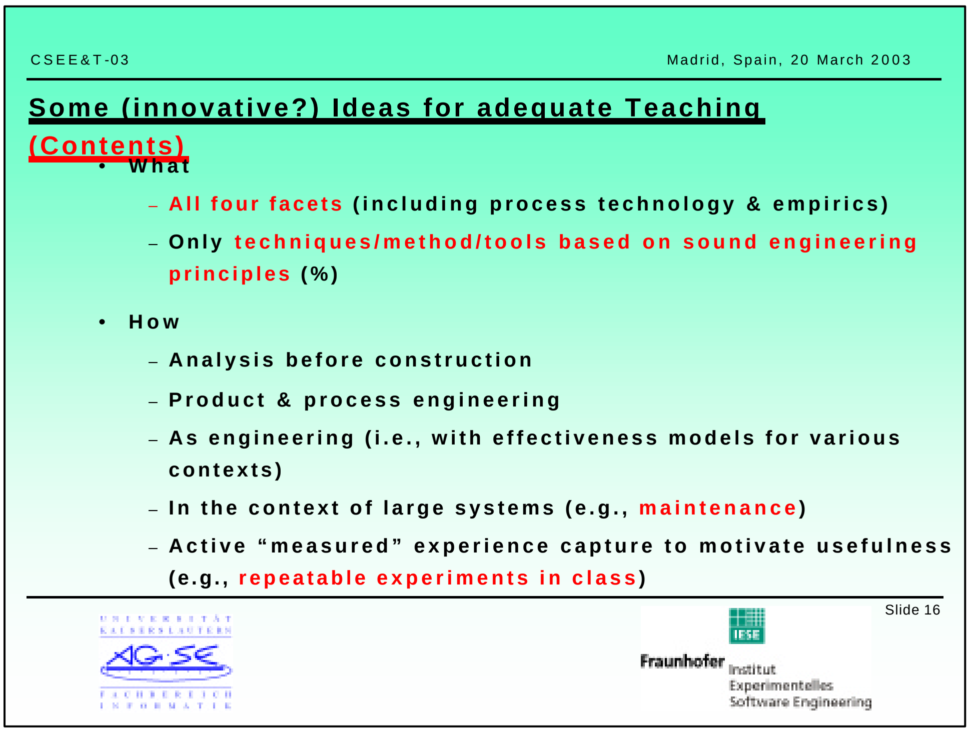
Some (innovative?) Ideas for adequate Teaching (Goals)

- Teaching goals for any student
  - Knows basic principles in all facets of SE (and can apply them to new technologies)
  - Knows existing body of knowledge
  - Can apply current techniques/methods/tools (but understands them as examples)
  - Understands effects (pro's & con's) of competing t/m/t's for different contexts
    - Never again uses "superlative"

UNIVERSITÄT KAISERSLAUTERN AG SE FACHBEREICH INFORMATIK

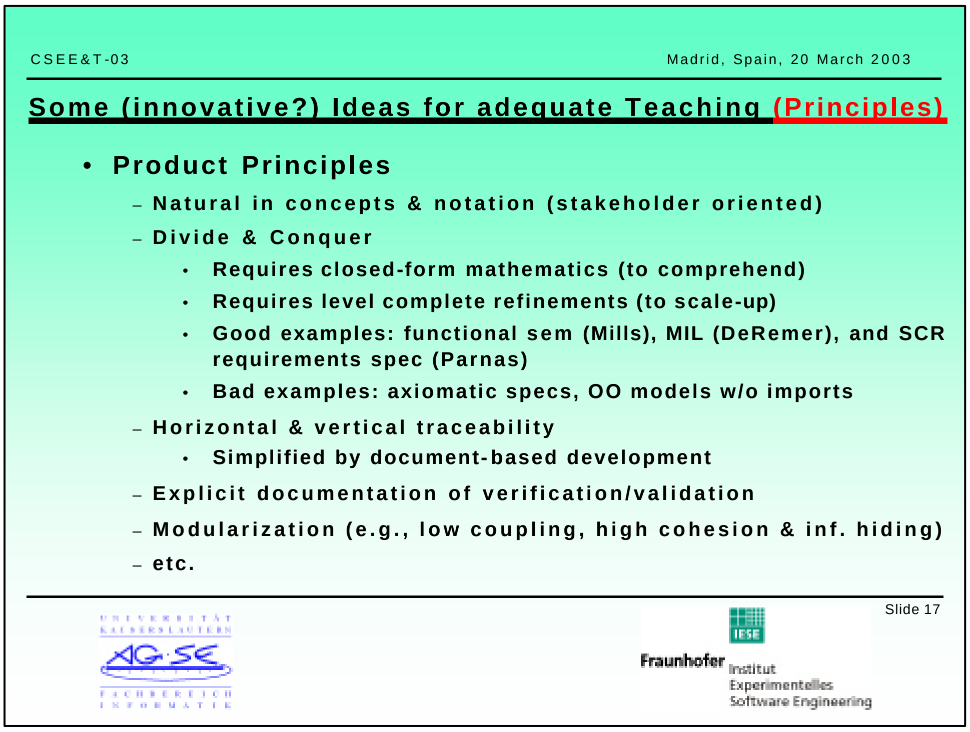
IESE Fraunhofer Institut Experimentelles Software Engineering

Slide 15



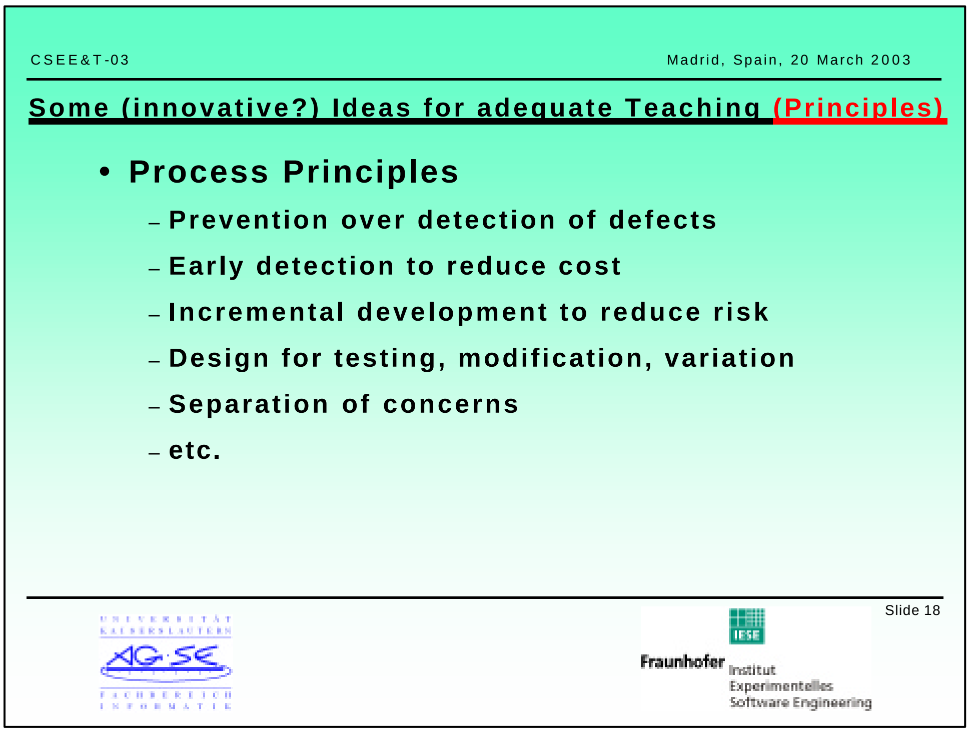
Some (innovative?) Ideas for adequate Teaching (Contents)

- What
  - All four facets (including process technology & empirics)
  - Only techniques/method/tools based on sound engineering principles (%)
- How
  - Analysis before construction
  - Product & process engineering
  - As engineering (i.e., with effectiveness models for various contexts)
  - In the context of large systems (e.g., maintenance)
  - Active "measured" experience capture to motivate usefulness (e.g., repeatable experiments in class)



Some (innovative?) Ideas for adequate Teaching (Principles)

- Product Principles
  - Natural in concepts & notation (stakeholder oriented)
  - Divide & Conquer
    - Requires closed-form mathematics (to comprehend)
    - Requires level complete refinements (to scale-up)
    - Good examples: functional sem (Mills), MIL (DeRemer), and SCR requirements spec (Parnas)
    - Bad examples: axiomatic specs, OO models w/o imports
  - Horizontal & vertical traceability
    - Simplified by document-based development
  - Explicit documentation of verification/validation
  - Modularization (e.g., low coupling, high cohesion & inf. hiding)
  - etc.



CSEE&T-03 Madrid, Spain, 20 March 2003

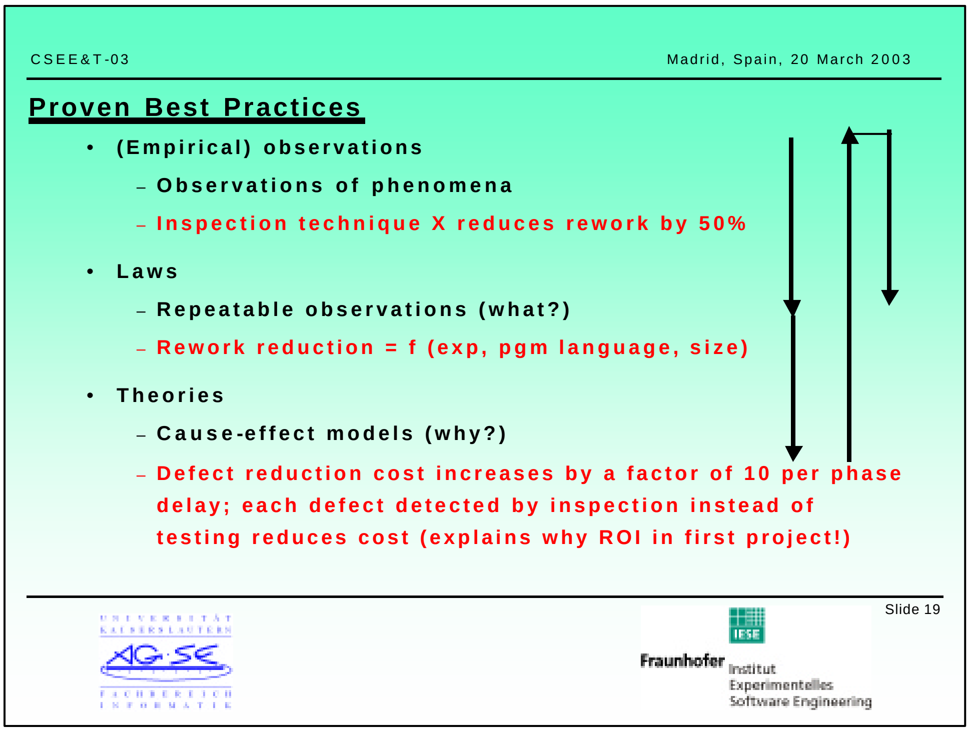
Some (innovative?) Ideas for adequate Teaching (Principles)

- Process Principles
  - Prevention over detection of defects
  - Early detection to reduce cost
  - Incremental development to reduce risk
  - Design for testing, modification, variation
  - Separation of concerns
  - etc.

Slide 18

UNIVERSITÄT KAISERSLAUTERN AG SE FACHBEREICH INFORMATIK

Fraunhofer Institut Experimentelles Software Engineering IESE



CSEE&T-03 Madrid, Spain, 20 March 2003

Proven Best Practices

- (Empirical) observations
  - Observations of phenomena
  - Inspection technique X reduces rework by 50%
- Laws
  - Repeatable observations (what?)
  - Rework reduction = f (exp, pgm language, size)
- Theories
  - Cause-effect models (why?)
  - Defect reduction cost increases by a factor of 10 per phase delay; each defect detected by inspection instead of testing reduces cost (explains why ROI in first project!)

Slide 19

UNIVERSITÄT KAISERSLAUTERN AG SE FACHBEREICH INFORMATIK

Fraunhofer Institut Experimentelles Software Engineering IESE

Proven Best Practices

- Lots of observations, laws & theories exis
- Professionalism requires
  - Application of existing knowledge
  - Justification of voiding (by accepting responsibility)
- In case of failures
  - Adherence to best practice (i.e., existing knowledge)
  - Otherwise violation of due diligence
  - Accountability
  - Example: Company does not establish vertical traceability & modification results in operational failure ➔ accountable!

Complete change

of current practice!

The Fraunhofer IESE Series on Software Engineering

'giving the reader a proven basis for engineering complex software systems'

rombach@iese.fhg.de

A Handbook of Software and Systems Engineering

Empirical Observations, Laws and Theories

Albert Endres
Dieter Rombach

Fraunhofer Institut Experimentelles Software Engineering
PEARSON Addison Wesley

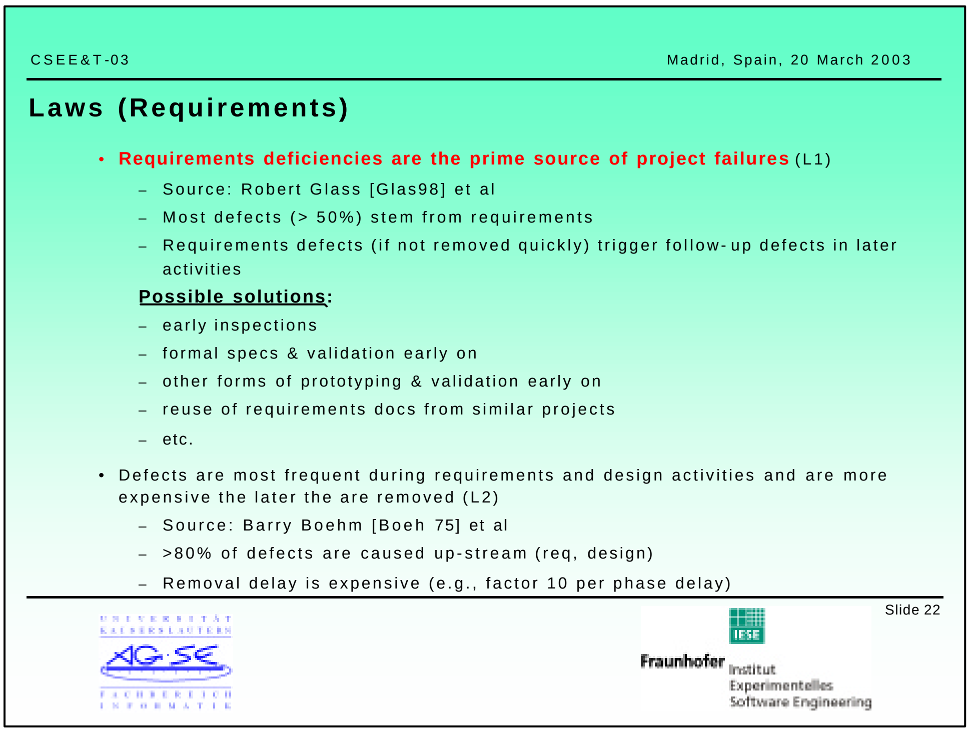
Fraunhofer IESE Series

Handbook capturing existing body of knowledge

Students can learn about existing body of knowledge

Practitioners can avoid negligance of due dilligance

Additions are welcome for next edition of book



CSEE&T-03 Madrid, Spain, 20 March 2003

Laws (Requirements)

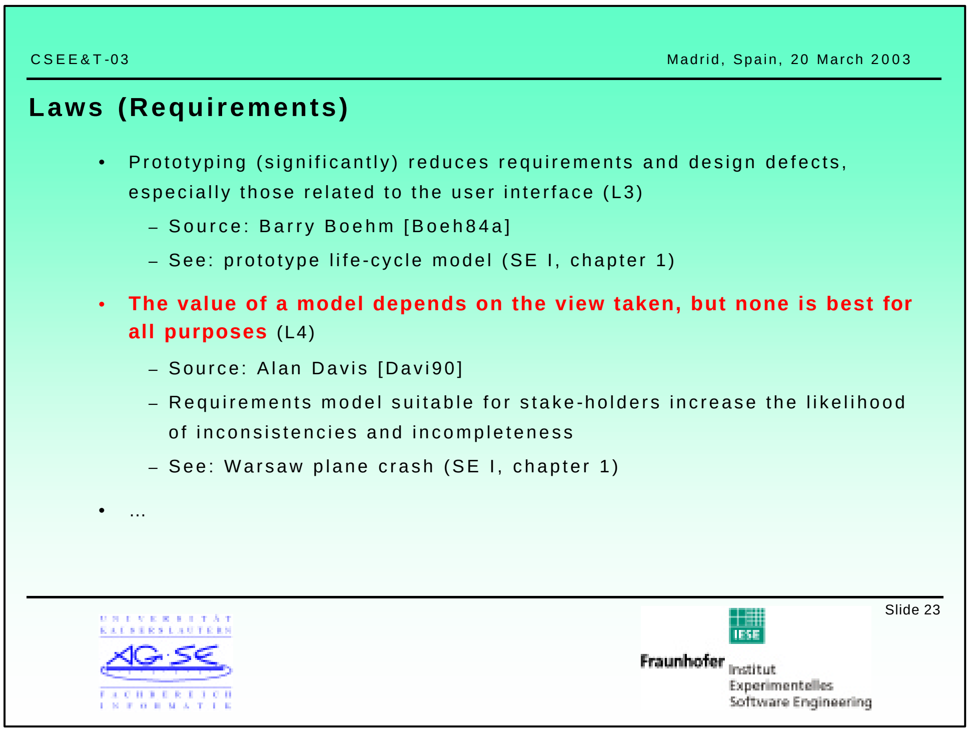
- **Requirements deficiencies are the prime source of project failures** (L1)
  - Source: Robert Glass [Glas98] et al
  - Most defects (> 50%) stem from requirements
  - Requirements defects (if not removed quickly) trigger follow-up defects in later activities

  Possible solutions:
  - early inspections
  - formal specs & validation early on
  - other forms of prototyping & validation early on
  - reuse of requirements docs from similar projects
  - etc.
- Defects are most frequent during requirements and design activities and are more expensive the later the are removed (L2)
  - Source: Barry Boehm [Boeh 75] et al
  - >80% of defects are caused up-stream (req, design)
  - Removal delay is expensive (e.g., factor 10 per phase delay)

UNIVERSITÄT KAISERSLAUTERN AG SE FACHBEREICH INFORMATIK

IESE Fraunhofer Institut Experimentelles Software Engineering

Slide 22



CSEE&T-03 Madrid, Spain, 20 March 2003

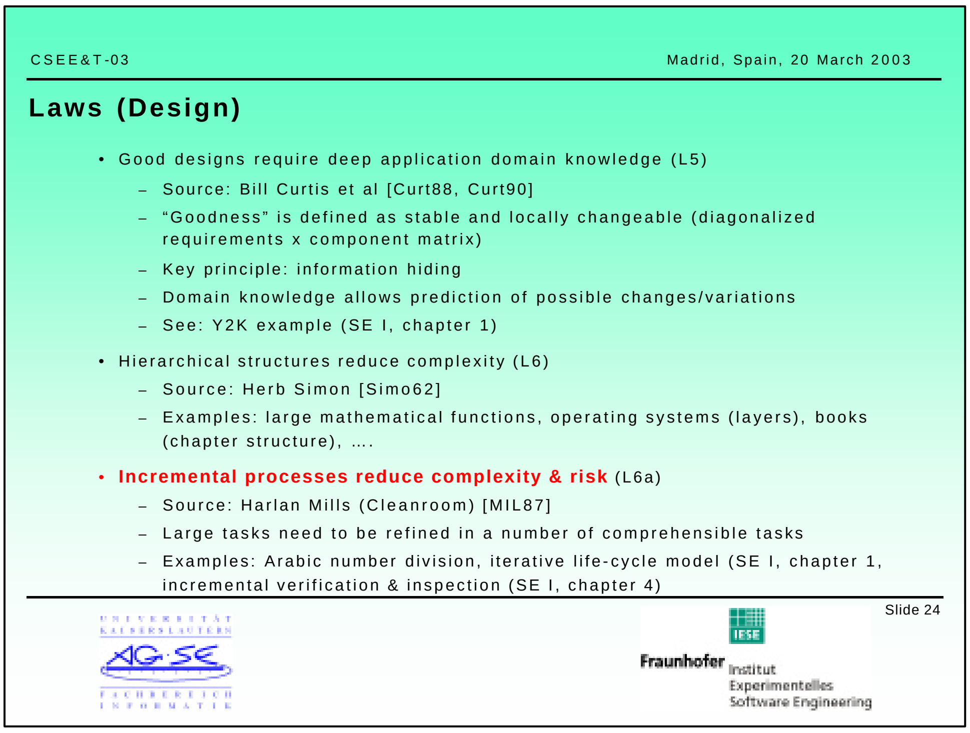
Laws (Requirements)

- Prototyping (significantly) reduces requirements and design defects, especially those related to the user interface (L3)
  - Source: Barry Boehm [Boeh84a]
  - See: prototype life-cycle model (SE I, chapter 1)
- **The value of a model depends on the view taken, but none is best for all purposes** (L4)
  - Source: Alan Davis [Davi90]
  - Requirements model suitable for stake-holders increase the likelihood of inconsistencies and incompleteness
  - See: Warsaw plane crash (SE I, chapter 1)
- ...

UNIVERSITÄT KAISERSLAUTERN AG SE FACHBEREICH INFORMATIK

IESE Fraunhofer Institut Experimentelles Software Engineering

Slide 23

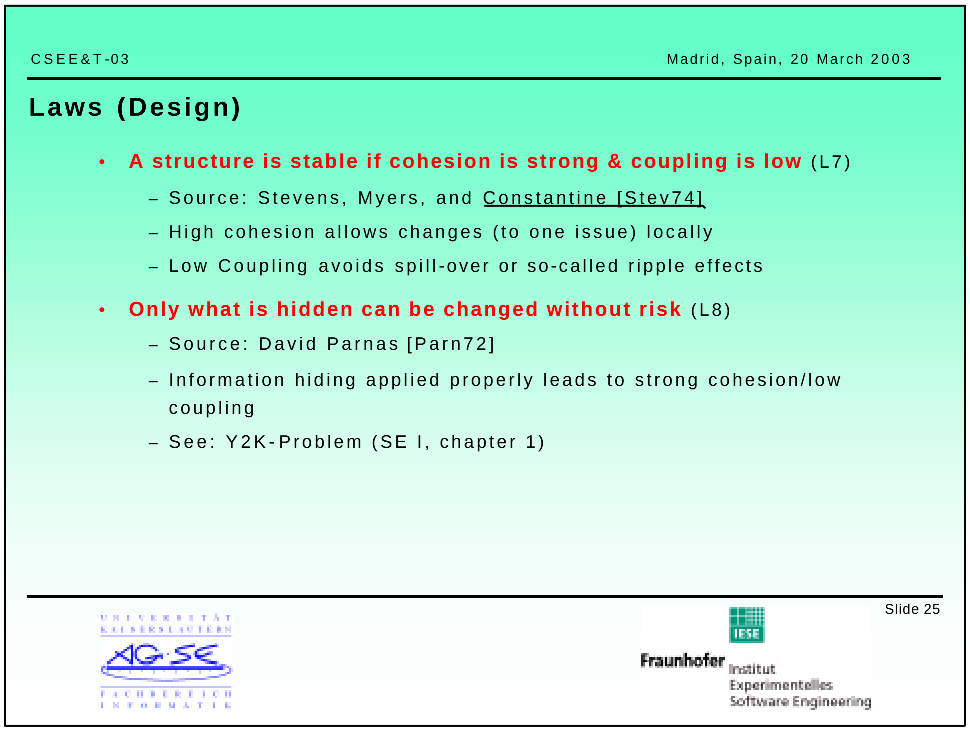


Laws (Design)

- Good designs require deep application domain knowledge (L5)
  - Source: Bill Curtis et al [Curt88, Curt90]
  - "Goodness" is defined as stable and locally changeable (diagonalized requirements x component matrix)
  - Key principle: information hiding
  - Domain knowledge allows prediction of possible changes/variations
  - See: Y2K example (SE I, chapter 1)
- Hierarchical structures reduce complexity (L6)
  - Source: Herb Simon [Simo62]
  - Examples: large mathematical functions, operating systems (layers), books (chapter structure), ….
- **Incremental processes reduce complexity & risk** (L6a)
  - Source: Harlan Mills (Cleanroom) [MIL87]
  - Large tasks need to be refined in a number of comprehensible tasks
  - Examples: Arabic number division, iterative life-cycle model (SE I, chapter 1, incremental verification & inspection (SE I, chapter 4)

UNIVERSITÄT KAISERSLAUTERN AG SE FACHBEREICH INFORMATIK

Fraunhofer IESE Institut Experimentelles Software Engineering

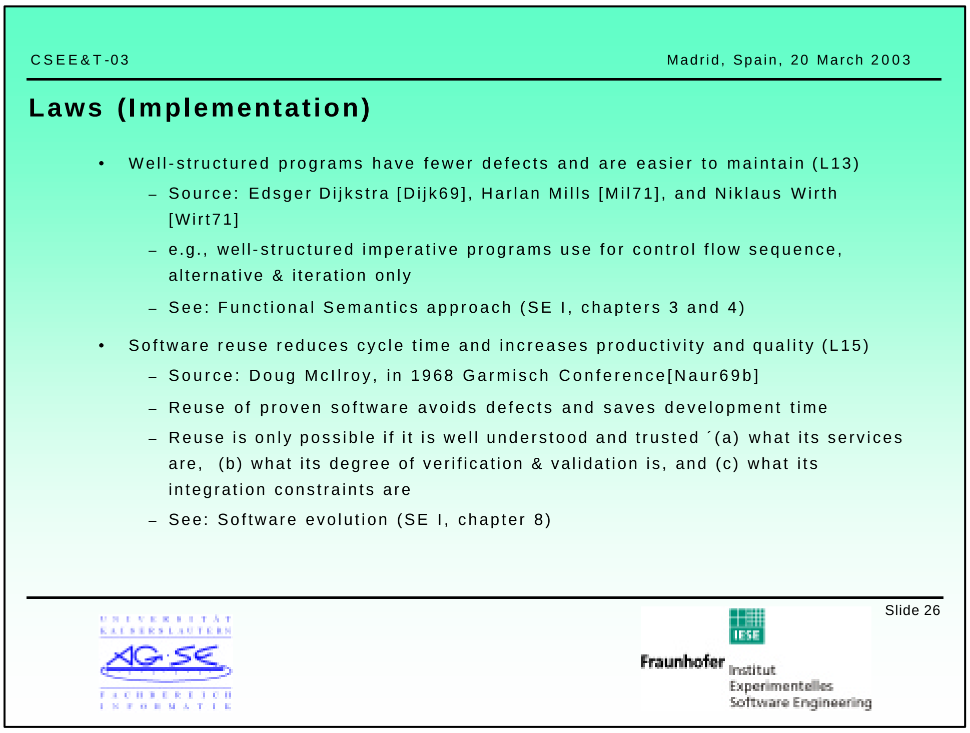


Laws (Design)

- **A structure is stable if cohesion is strong & coupling is low** (L7)
  - Source: Stevens, Myers, and Constantine [Stev74]
  - High cohesion allows changes (to one issue) locally
  - Low Coupling avoids spill-over or so-called ripple effects
- **Only what is hidden can be changed without risk** (L8)
  - Source: David Parnas [Parn72]
  - Information hiding applied properly leads to strong cohesion/low coupling
  - See: Y2K-Problem (SE I, chapter 1)

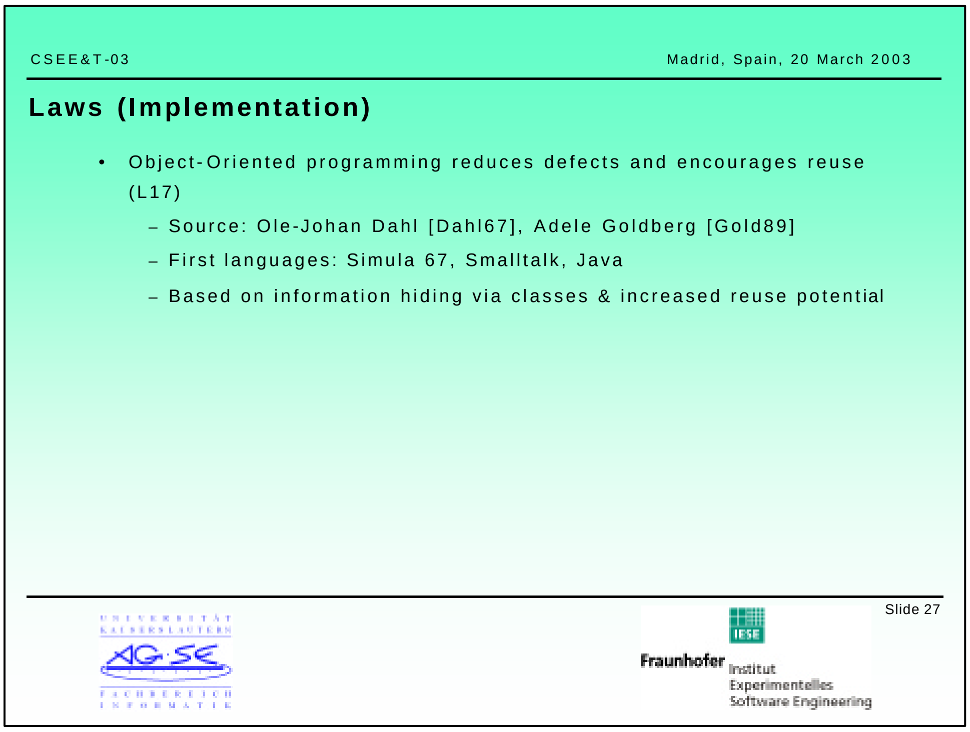
UNIVERSITÄT KAISERSLAUTERN AG SE FACHBEREICH INFORMATIK

Fraunhofer IESE Institut Experimentelles Software Engineering



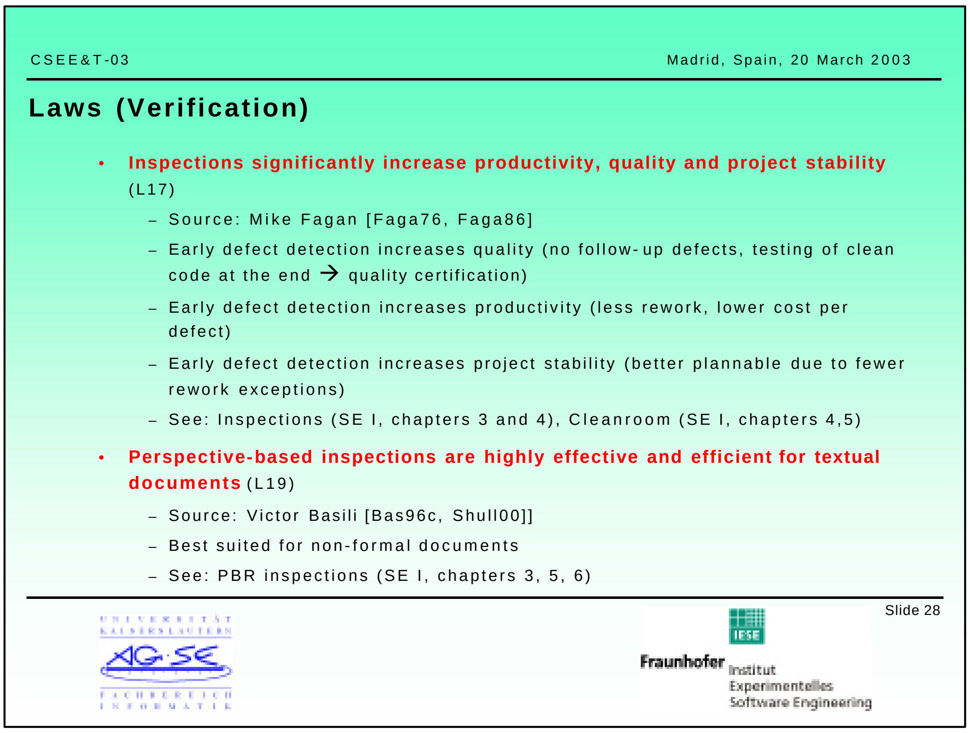
Laws (Implementation)

- Well-structured programs have fewer defects and are easier to maintain (L13)
  - Source: Edsger Dijkstra [Dijk69], Harlan Mills [Mil71], and Niklaus Wirth [Wirt71]
  - e.g., well-structured imperative programs use for control flow sequence, alternative & iteration only
  - See: Functional Semantics approach (SE I, chapters 3 and 4)
- Software reuse reduces cycle time and increases productivity and quality (L15)
  - Source: Doug McIlroy, in 1968 Garmisch Conference[Naur69b]
  - Reuse of proven software avoids defects and saves development time
  - Reuse is only possible if it is well understood and trusted ´(a) what its services are, (b) what its degree of verification & validation is, and (c) what its integration constraints are
  - See: Software evolution (SE I, chapter 8)



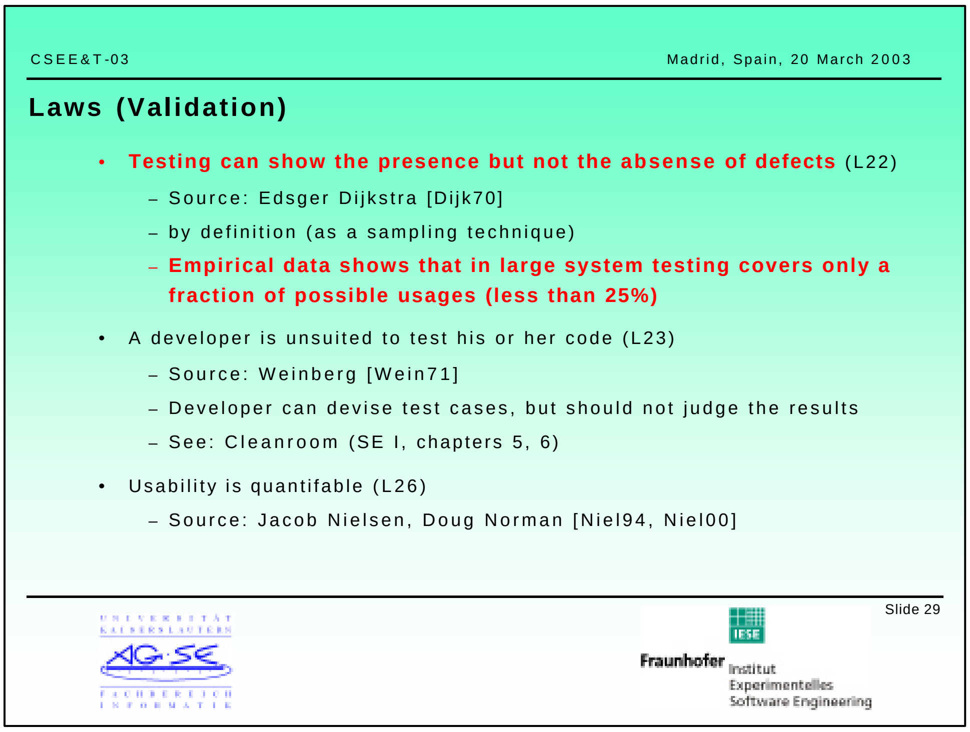
Laws (Implementation)

- Object-Oriented programming reduces defects and encourages reuse (L17)
  - Source: Ole-Johan Dahl [Dahl67], Adele Goldberg [Gold89]
  - First languages: Simula 67, Smalltalk, Java
  - Based on information hiding via classes & increased reuse potential



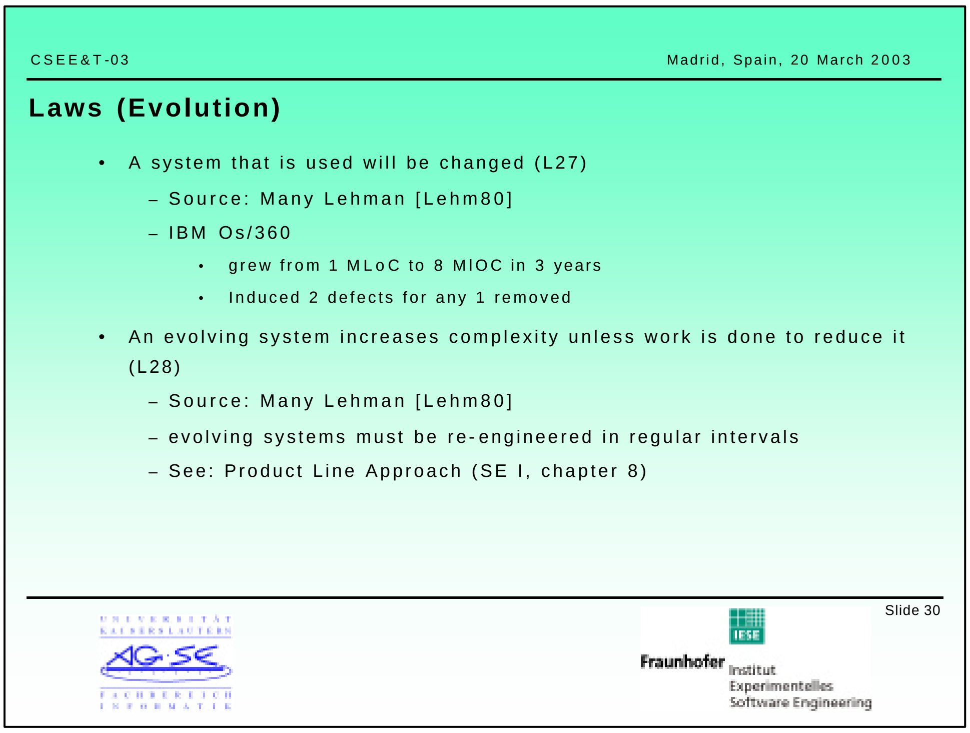
Laws (Verification)

- **Inspections significantly increase productivity, quality and project stability** (L17)
  - Source: Mike Fagan [Faga76, Faga86]
  - Early defect detection increases quality (no follow-up defects, testing of clean code at the end → quality certification)
  - Early defect detection increases productivity (less rework, lower cost per defect)
  - Early defect detection increases project stability (better plannable due to fewer rework exceptions)
  - See: Inspections (SE I, chapters 3 and 4), Cleanroom (SE I, chapters 4,5)
- **Perspective-based inspections are highly effective and efficient for textual documents** (L19)
  - Source: Victor Basili [Bas96c, Shull00]]
  - Best suited for non-formal documents
  - See: PBR inspections (SE I, chapters 3, 5, 6)



Laws (Validation)

- **Testing can show the presence but not the absense of defects** (L22)
  - Source: Edsger Dijkstra [Dijk70]
  - by definition (as a sampling technique)
  - **Empirical data shows that in large system testing covers only a fraction of possible usages (less than 25%)**
- A developer is unsuited to test his or her code (L23)
  - Source: Weinberg [Wein71]
  - Developer can devise test cases, but should not judge the results
  - See: Cleanroom (SE I, chapters 5, 6)
- Usability is quantifable (L26)
  - Source: Jacob Nielsen, Doug Norman [Niel94, Niel00]



CSEE&T-03 Madrid, Spain, 20 March 2003

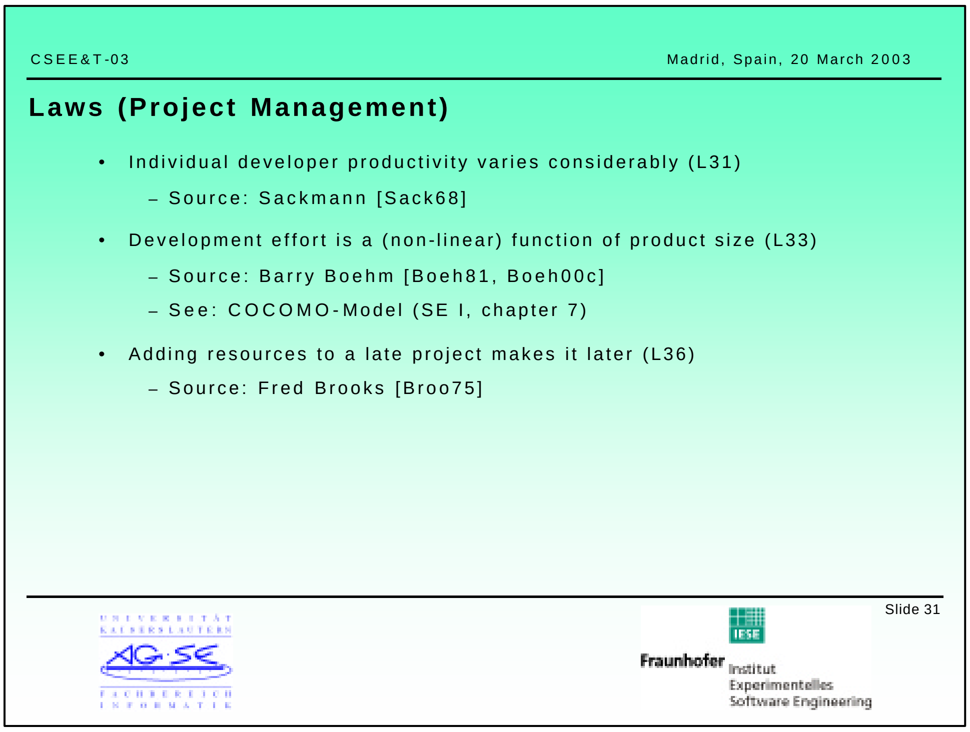
Laws (Evolution)

- A system that is used will be changed (L27)
  - Source: Many Lehman [Lehm80]
  - IBM Os/360
    - grew from 1 MLoC to 8 MIOC in 3 years
    - Induced 2 defects for any 1 removed
- An evolving system increases complexity unless work is done to reduce it (L28)
  - Source: Many Lehman [Lehm80]
  - evolving systems must be re-engineered in regular intervals
  - See: Product Line Approach (SE I, chapter 8)

UNIVERSITÄT KAISERSLAUTERN AG SE FACHBEREICH INFORMATIK

IESE Fraunhofer Institut Experimentelles Software Engineering

Slide 30



CSEE&T-03 Madrid, Spain, 20 March 2003

Laws (Project Management)

- Individual developer productivity varies considerably (L31)
  - Source: Sackmann [Sack68]
- Development effort is a (non-linear) function of product size (L33)
  - Source: Barry Boehm [Boeh81, Boeh00c]
  - See: COCOMO-Model (SE I, chapter 7)
- Adding resources to a late project makes it later (L36)
  - Source: Fred Brooks [Broo75]

UNIVERSITÄT KAISERSLAUTERN AG SE FACHBEREICH INFORMATIK

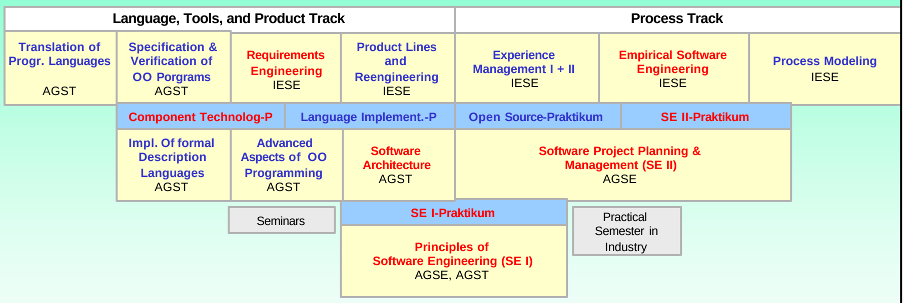
IESE Fraunhofer Institut Experimentelles Software Engineering

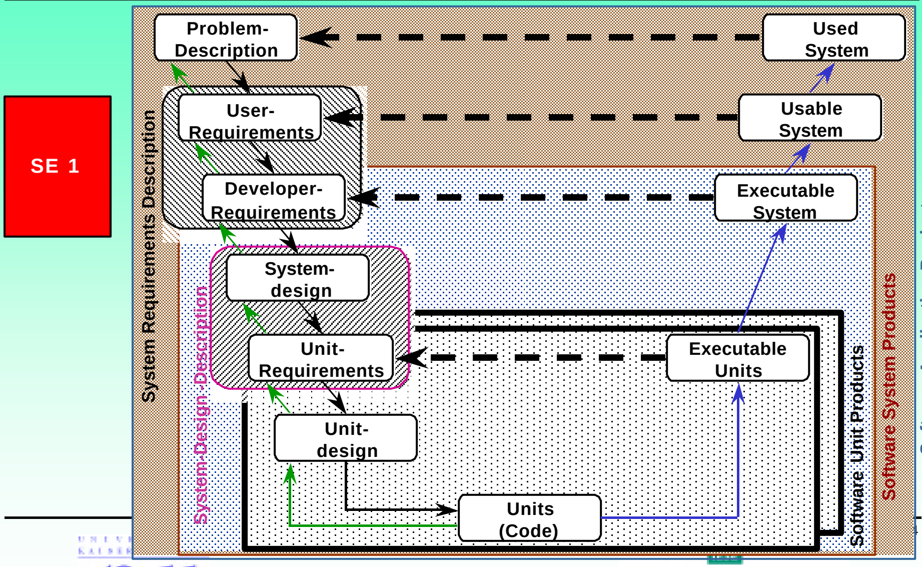
Slide 31

(Graduate) SE Curriculum at Kaiserslautern

- Comprehensive set of classes (taught by university & industry folks)
- Experimental studies included to motivate key lessons learned
  - Unit inspection more efficient than testing
  - Traceable design documentation reduces effort & risk of change
  - Informal (req) documents can be inspected efficiently (> 90%)
- Practical courses (1 semester each)
  - Use large systems to be changed
  - Team work (based on roles)
  - Real customer
  - Goals: running product & process improvements

SE Curriculum
Software Engineering / Software Technology
(University of Kaiserslautern)

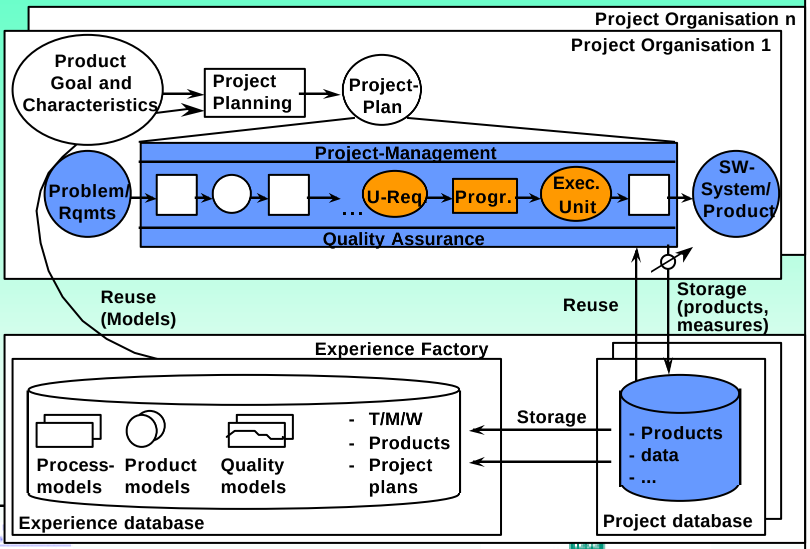

Language, Tools, and Product Track
Process Track
Translation of Progr. Languages AGST
Specification & Verification of OO Porgrams AGST
Requirements Engineering IESE
Product Lines and Reengineering IESE
Experience Management I + II IESE
Empirical Software Engineering IESE
Process Modeling IESE
Component Technolog-P
Language Implement.-P
Open Source-Praktikum
SE II-Praktikum
Impl. Of formal Description Languages AGST
Advanced Aspects of OO Programming AGST
Software Architecture AGST
Software Project Planning & Management (SE II) AGSE
Seminars
SE I-Praktikum
Principles of Software Engineering (SE I) AGSE, AGST
Practical Semester in Industry

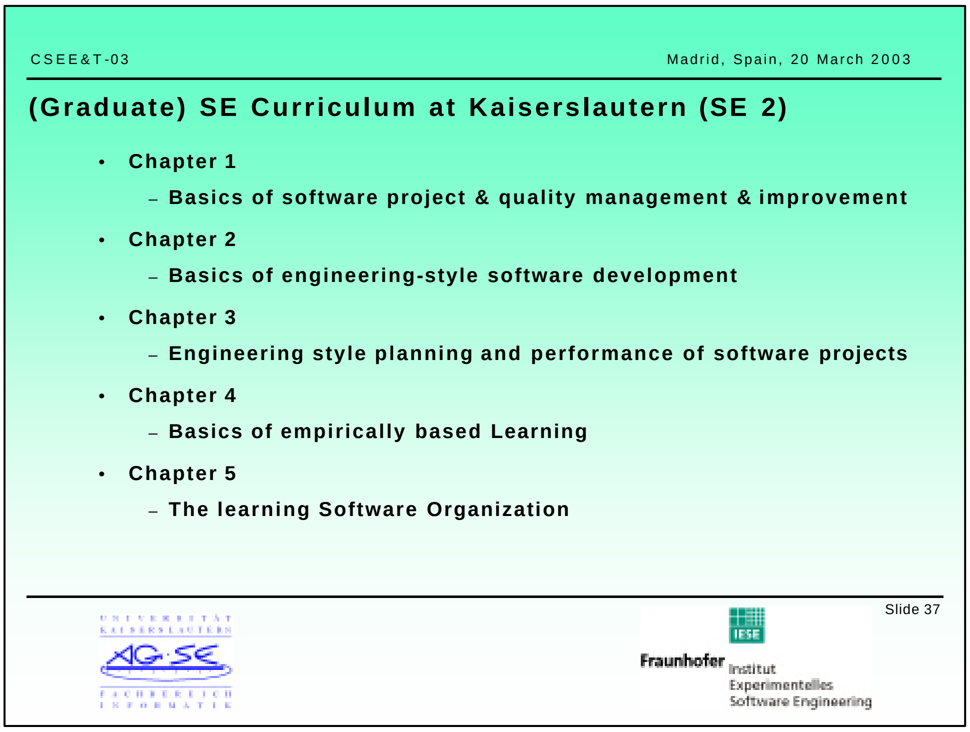


(Graduate) SE Curriculum at Kaiserslautern (SE 1)

- Chapter 1: Introduction & Motivation
- Chapter 2: Summary of existing Knowledge (see book)
- Chapter 3: Basics of Software Engineering (e.g., principles, modeling & architecture, quantification)
- Chapter 4: Software Unit Engineering
- Chapter 5: Software System Engineering
- Chapter 6: Software Application Engineering
- Chapter 7: Basics of Software Project & Quality Management
- Chapter 8: Software Evolution

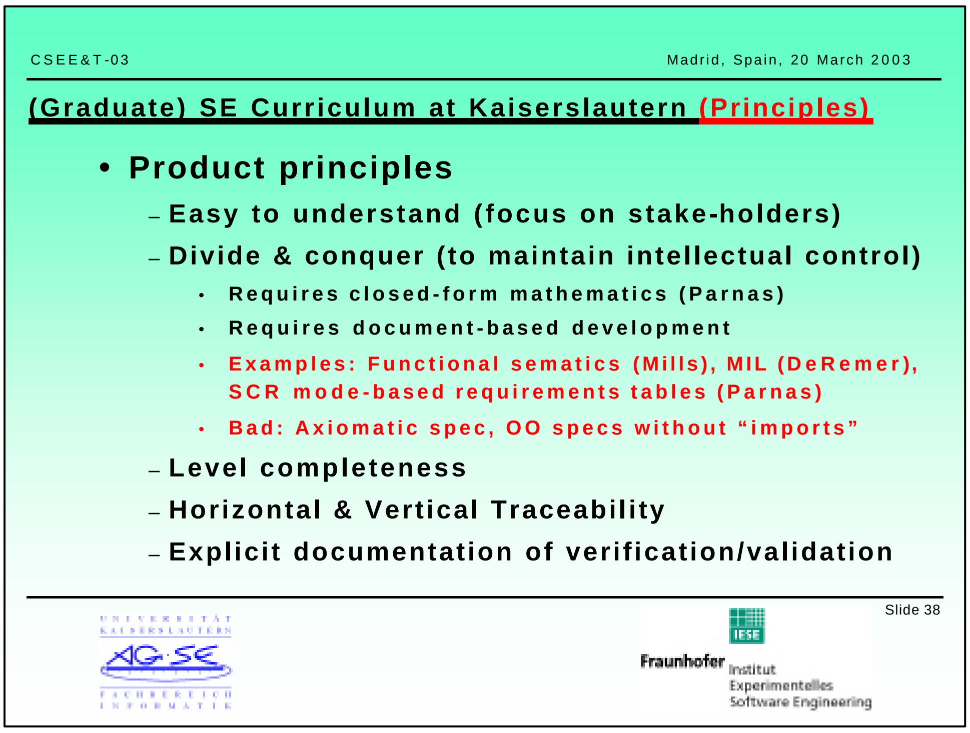
SE 2





(Graduate) SE Curriculum at Kaiserslautern (SE 2)

- Chapter 1
  - Basics of software project & quality management & improvement
- Chapter 2
  - Basics of engineering-style software development
- Chapter 3
  - Engineering style planning and performance of software projects
- Chapter 4
  - Basics of empirically based Learning
- Chapter 5
  - The learning Software Organization


CSEE&T-03 Madrid, Spain, 20 March 2003

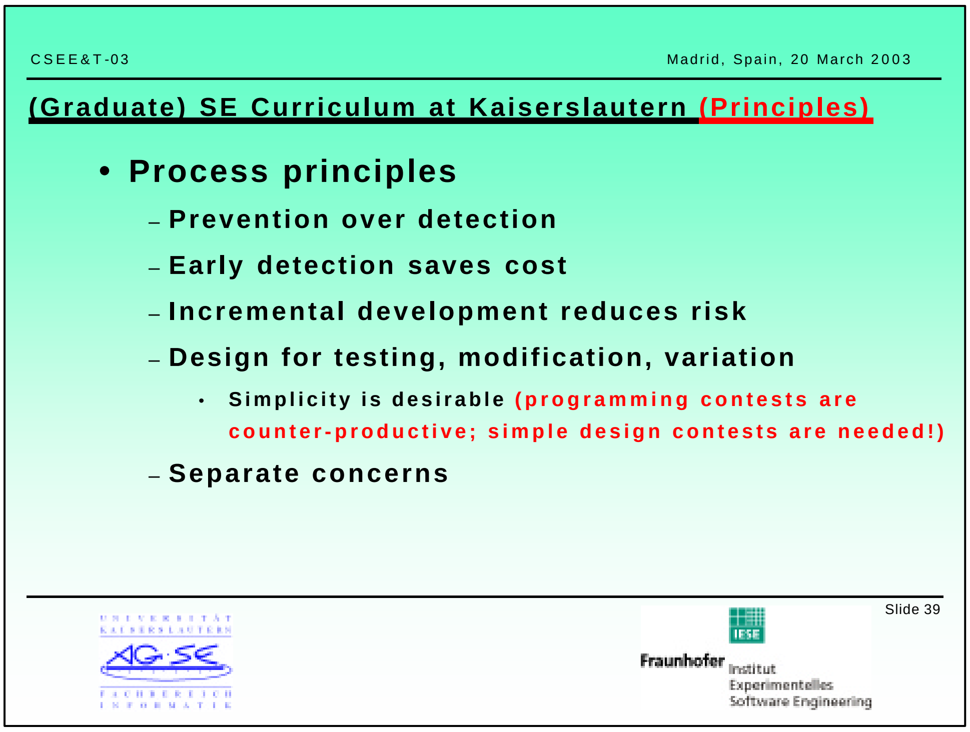
(Graduate) SE Curriculum at Kaiserslautern (Principles)

- Product principles
  - Easy to understand (focus on stake-holders)
  - Divide & conquer (to maintain intellectual control)
    - Requires closed-form mathematics (Parnas)
    - Requires document-based development
    - Examples: Functional sematics (Mills), MIL (DeRemer), SCR mode-based requirements tables (Parnas)
    - Bad: Axiomatic spec, OO specs without “imports”
  - Level completeness
  - Horizontal & Vertical Traceability
  - Explicit documentation of verification/validation

UNIVERSITÄT KAISERSLAUTERN AG SE FACHBEREICH INFORMATIK

Fraunhofer Institut Experimentelles Software Engineering

Slide 38


CSEE&T-03 Madrid, Spain, 20 March 2003

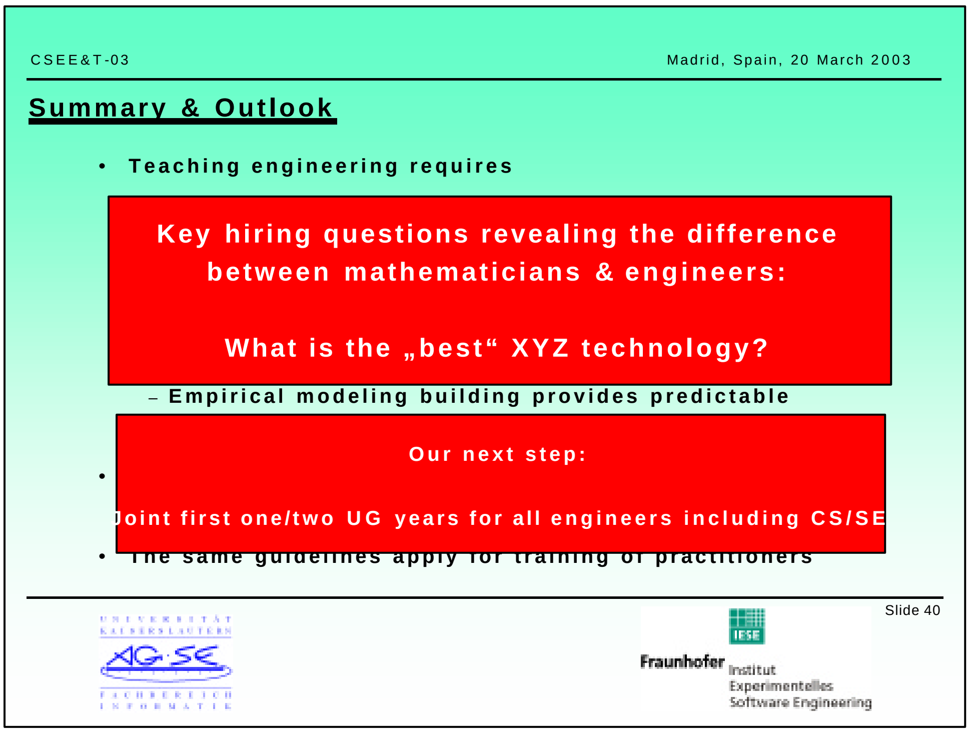
(Graduate) SE Curriculum at Kaiserslautern (Principles)

- Process principles
  - Prevention over detection
  - Early detection saves cost
  - Incremental development reduces risk
  - Design for testing, modification, variation
    - Simplicity is desirable (programming contests are counter-productive; simple design contests are needed!)
  - Separate concerns

UNIVERSITÄT KAISERSLAUTERN AG SE FACHBEREICH INFORMATIK

Fraunhofer Institut Experimentelles Software Engineering

Slide 39



CSEE&T-03 Madrid, Spain, 20 March 2003

Summary & Outlook

- Teaching engineering requires

Key hiring questions revealing the difference between mathematicians & engineers:

What is the „best“ XYZ technology?

– Empirical modeling building provides predictable

-

Our next step:

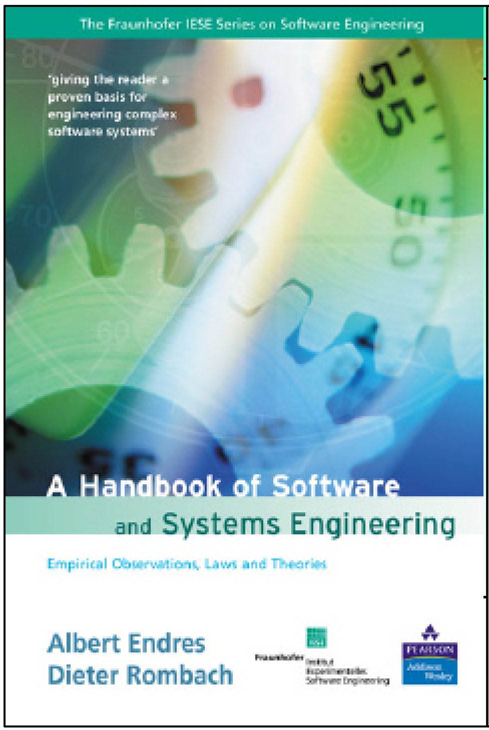
Joint first one/two UG years for all engineers including CS/SE

- The same guidelines apply for training of practitioners

UNIVERSITÄT KAISERSLAUTERN AG SE FACHBEREICH INFORMATIK

Fraunhofer Institut Experimentelles Software Engineering IESE

Slide 40


The Fraunhofer IESE Series on Software Engineering
'giving the reader a proven basis for engineering complex software systems'
A Handbook of Software and Systems Engineering
Empirical Observations, Laws and Theories
Albert Endres
Dieter Rombach
Fraunhofer Institut Experimentelles Software Engineering
PEARSON Addison Wesley

The book is available as of today!