Industry-Relevant Software Engineering Research and Education

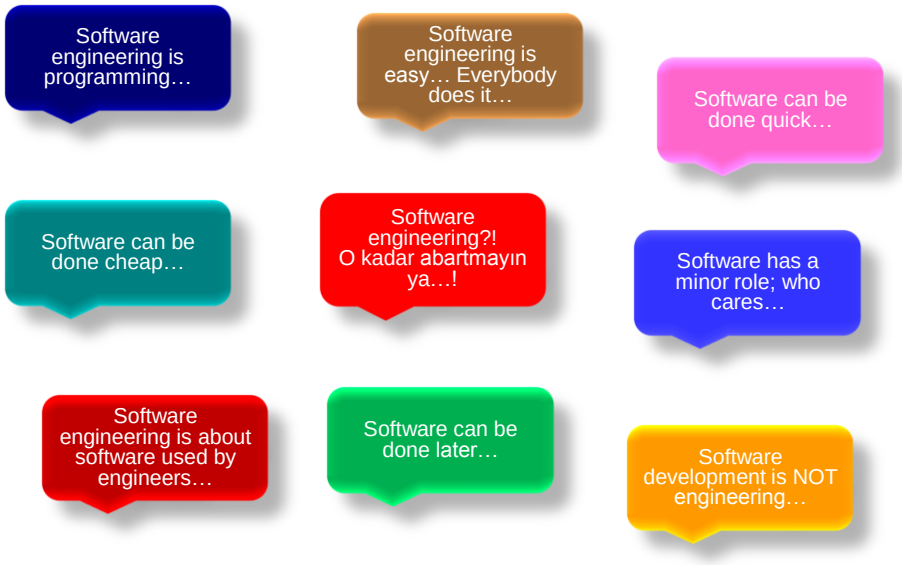
III. Ulusal Yazılım Mühendisliği Sempozyumu ve Sergisi
Bilkent University, Ankara, Turkey
28 September 2007

Bedir Tekinerdoğan

University of Twente
Faculty of Electrical Engineering, Mathematics and Computer Science
Dept. of Computer Science - Software Engineering Group
P.O. Box 217 7500 AE Enschede, The Netherlands
http://www.cs.utwente.nl/~bedir
B.Tekinerdogan@ewi.utwente.nl

Software Engineers Wanted…

Some Misconceptions…



Origin and Basis…

NATO Conference

- The term “Software Engineering” was first introduced in a Computer Science conference in Garmisch, Germany, 7th to 11th October 1968 presented by NATO Science Committee
- Software crisis in Software Practice:
  - Projects over budget, behind schedule and with low quality

Nato Conference – Importance of Software

One of the major motivations for the organizing of the conference was an awareness of the rapidly increasing importance of computer software systems in many activities of society. Thus, although much of the conference was concerned with detailed technical questions, many of the discussions were of a more general nature, and should be of interest to a wide spectrum of readers. It is for the benefit of this wider audience that representative discussions of various points relating to the impact of software engineering on society have been abstracted from later sections of this Report, and collected in this introductory section.

First, three quotations which indicate the rate of growth of software:

Helms: In Europe alone there are about 10,000 installed computers — this number is increasing at a rate of anywhere from 25 per cent to 50 per cent per year. The quality of software provided for these computers will soon affect more than a quarter of a million analysts and programmers.

David: No less a person than T.J. Watson said that OS/360 cost IBM over $50 million dollars a year during its preparation, and at least 5000 man-years' investment. TSS/360 is said to be in the 1000 man-year category. It has been said, too, that development costs for software equal the development costs for hardware in establishing a new machine line.

d'Agapeyeff: In 1958 a European general purpose computer manufacturer often had less than 50 software programmers, now they probably number 1,000-2,000 people; what will be needed in 1978?

Yet this growth rate was viewed with more alarm than pride.

Excerpt from in Software Engineering Conference Proceedings in 1968, page 15

Nato Conference – Immaturity of SE

There was general agreement that 'software engineering' is in a very rudimentary stage of development as compared with the established branches of engineering.

McIlroy: We undoubtedly produce software by backward techniques. We undoubtedly get the short end of the stick in confrontations with hardware people because they are the industrialists and we are the crofters. Software production today appears in the scale of industrialization somewhere below the more backward construction industries.

Kolence: Programming management will continue to deserve its current poor reputation for cost and schedule effectiveness until such time as a more complete understanding of the program design process is achieved.

Fraser: One of the problems that is central to the software production process is to identify the nature of progress and to find some way of measuring it. Only one thing seems to be clear just now. It is that program construction is not always a simple progression in which each act of assembly represents a distinct forward step and that the final product can be described simply as the sum of many sub-assemblies.

Graham: Today we tend to go on for years, with tremendous investments to find that the system, which was not well understood to start with, does not work as anticipated. We build systems like the Wright brothers built airplanes — build the whole thing, push it off the cliff, let it crash, and start over again.

Excerpt from in Software Engineering Conference Proceedings in 1968, page 17

Software Crisis…

There was a considerable amount of debate on what some members chose to call the 'software crisis' or the 'software gap'. As will be seen from the quotations below, the conference members had widely differing views on the seriousness, or otherwise, of the situation, and on the extent of the problem areas.

David and Fraser: (from their *Position paper*)

»There is a widening gap between ambitions and achievements in software engineering. This gap appears in several dimensions: between promises to users and performance achieved by software, between what seems to be ultimately possible and what is achievable now and between estimates of software costs and expenditures. The gap is arising at a time when the consequences of software failure in all its aspects are becoming increasingly serious. Particularly alarming is the seemingly unavoidable fallibility of large software, since a malfunction in an advanced hardware-software system can be a matter of life and death, not only for individuals, but also for vehicles carrying hundreds of people and ultimately for nations as well.«

Hastings: I am very disturbed that an aura of gloom has fallen over this assembly. I work in an environment of many large installations using OS/360. These are complex systems, being used for many very sophisticated applications. People are doing what they need to do, at a much lower cost than ever before; and they seem to be reasonably satisfied. Perhaps their systems do not meet everybody's need, they don't meet the time sharing people's demands for example, but I don't think software engineering should be confused with time sharing system engineering. Areas like traffic control, hospital patient monitoring, etc., are very explosive, but are very distinct from general purpose computing.

Gillette: We are in many ways in an analogous position to the aircraft industry, which also has problems producing systems on schedule and to specification. We perhaps have more examples of bad large systems than good, but we are a young industry and are learning how to do better.

Randell: There are of course many good systems, but are any of these good enough to have human life tied on-line to them, in the sense that if they fail for more than a few seconds, there is a fair chance of one or more people being killed?

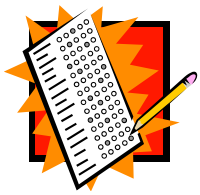
Excerpt from in Software Engineering Conference Proceedings in 1968, page 120

Software *Engineering?*

Debate on Software Engineering (SE)

Which one is the right answer…? ☺

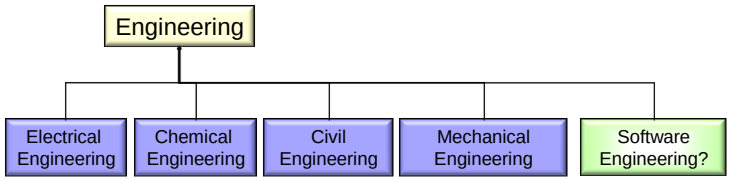
a) SE is not an engineering discipline…
b) SE can not be an engineering discipline due to the different nature of software…
c) SE is rapidly approaching towards becoming a mature engineering discipline…
d) Sure, SE is an engineering discipline…
e) none of the above…



Engineering - Definition

- Creating cost-effective solutions
  - Engineering is about solving problems with economical use of all resources
- to practical problems
  - Engineering deals with practical problems whose solutions matter to people outside the engineering domain: the customers
- by applying scientific knowledge
  - Engineering solves a problem in a particular way, by applying science, mathematics and design analysis
- building things
  - Engineering emphasizes the solutions, which are tangible artifacts
- in the service of mankind
  - Engineering not only serves the immediate customer, but also develops technology and expertise that will support the society

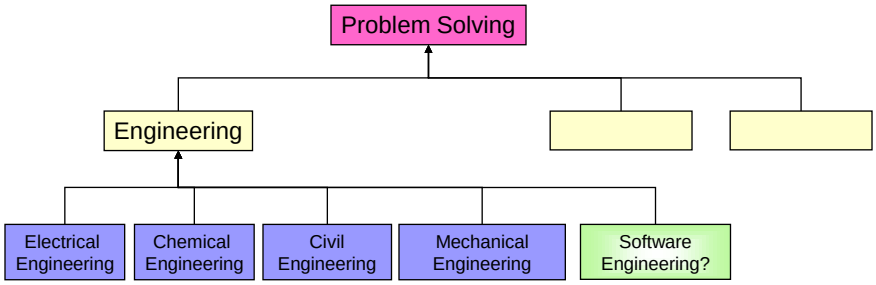
(M. Shaw and D. Garlan. Software Architecture: Perspectives on an Emerging Discipline, Prentice Hall, 1996)



Software Engineering is Problem Solving

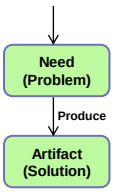
- Software Engineering is Engineering?
- Engineering is Problem Solving

- How has engineering evolved?
- What are the lessons that we can derive from history of Traditional Engineering wrt Problem Solving?



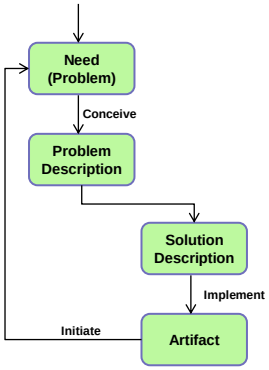
Primitive Problem Solving - Engineering

- Production in the early societies was basically done by hand and therefore they are also called craft-based societies
- craftsmen do not and often cannot, externalize their works in descriptive representations
- no prior activity of describing the solution like drawing or modeling before the production of the artifact
- almost no application of scientific knowledge of science. The available knowledge related with the craft process was stored in the artifact itself
- production of the artifacts is basically controlled by tradition



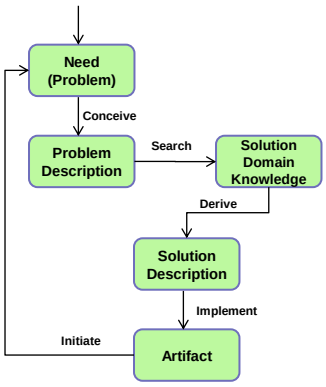
Design-Driven Problem Solving - Engineering

- Over time, the size and the complexity of the artifacts exceeded the cognitive capacity of a single craftsman
- many craftsmen were involved in the production and communication about the production process
- This initiated the necessity for drafting or designing, whereby the artifact is represented through a drawing before the actual production.
- Through drafting, engineers could communicate about the production of the artifact, evaluate the artifact before production and use the drafting or design as a guide for production.



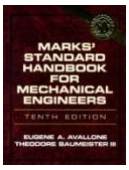
Domain-Driven Problem Solving - Engineering

- Classical engineers were restricted in their accomplishments when scientific knowledge was lacking.
- Over time, scientific knowledge gradually evolved while forming the basis for the introduction of new engineering disciplines.
- New advancements in physics and mathematics formed the basics of several engineering disciplines.
- The vastly increased use of scientific principles to the solution of practical problems and the past experimental experiences increasingly resulted also in the production of new types of artifacts.



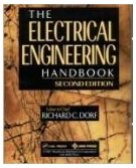
Current Domain Knowledge in Engineering

- each mature engineering is based on a rich scientific knowledge that has been developed over several centuries
  - Chemical engineering → Chemistry
  - Mechanical and Electrical Engineering → Physics, etc
- compiled in several handbooks consisting of thousands of pages with solid and proven knowledge and patterns
- Consolidated by experts in the domain…

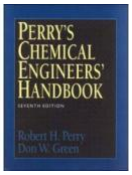


1800 pages

1600 pages



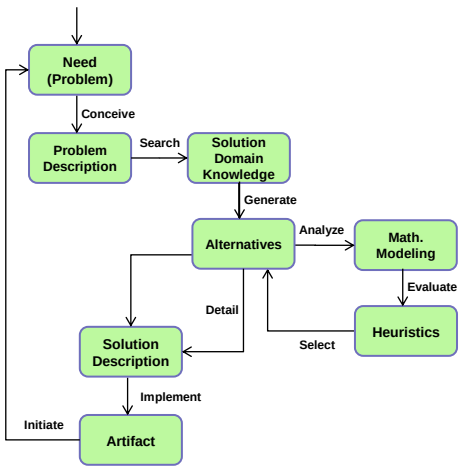
3672 pages



2640 pages

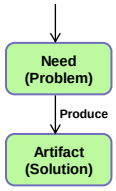
Control-driven Problem Solving - Engineering

- The development of mathematical modeling supported the control of the alternatives selection.
- Much later, this has led to *automation,* which is first applied in manufacture.
- Machines were built with automatic-control mechanisms that include a feedback control system providing the capacity for self-correction.
- Further, the advent of the computer has greatly supported the use of feedback control systems in manufacturing processes.



Primitive Problem Solving - SE

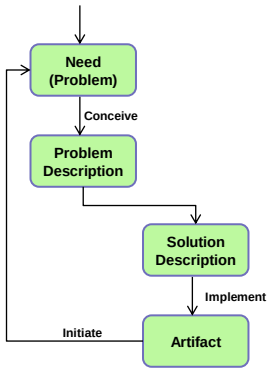
- first programs were expressed in machine code, and later in assembler
- First programming languages: ALGOL, FORTRAN, LISP, COBOL focusing basically on implementing algorithms



- Problems are simple (algorithmic)
- Direct mapping of problem to software
- No design, no systematic domain knowledge, no mathematical analysis of alternatives

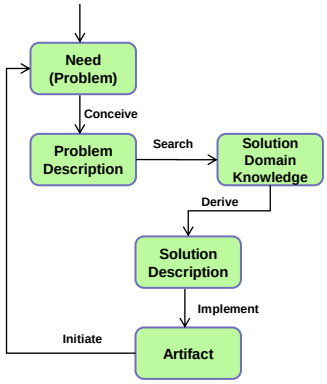
Design-Driven Problem Solving - SE

- To cope with the increasing complexity structured design methods including design notations (cp. drafting in engineering) were introduced in the 1970s.
- CASE tools were introduced in the mid 1980s to provide automated support for structured software development methods
- At the start of the 1990s several object-oriented analysis and design methods were introduced to fit the existing object-oriented language abstractions and new object-oriented notations were introduced.
- Inspired from architecture design, *design patterns* have been introduced as a way to cope with recurring design problems in a systematic way.
- *Software architectures* have been introduced to approach software development from the overall system structure.



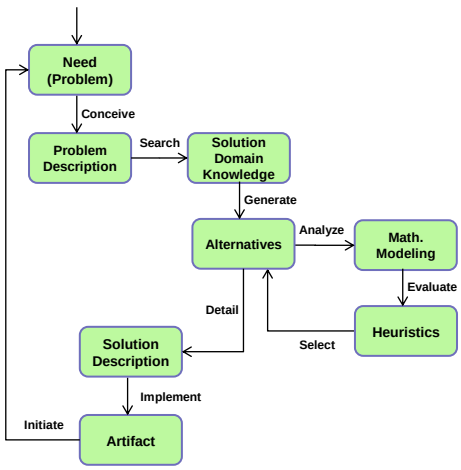
Domain-Driven Problem Solving - SE

- The basic scientific knowledge, on which software engineering relies, is mainly computer science that has developed over the last decades.
- , such aProgress is largely made in isolated parts algorithms and abstract data types
- One of the interesting developments is the increasing size of pattern knowledge.
- The software engineering body of knowledge has evolved in the last four decades.
- but is still quite meager wrt traditional engineering.
- The available handbooks of software engineering are still not comparable to the standard handbooks of mature engineering disciplines.
- Moreover, on many fundamental concepts in software engineering consensus among experts has still not been reached yet and research is ongoing.



Control-Driven Problem Solving - SE

- Mathematical modeling is more and more integrated in software design.
- *Empirical software engineering* started in mid 1990s to devise experiments on software, in collecting data from the experiments, and in devising laws and theories from this data.
- To analyze software systems, metrics are being developed and tested.
- Unfortunately, the level of control does not equal that of mature engineering in which quantitative analysis is widely applied to optimize the derived solutions
- Using mathematical optimization techniques for deriving solutions is not common practice

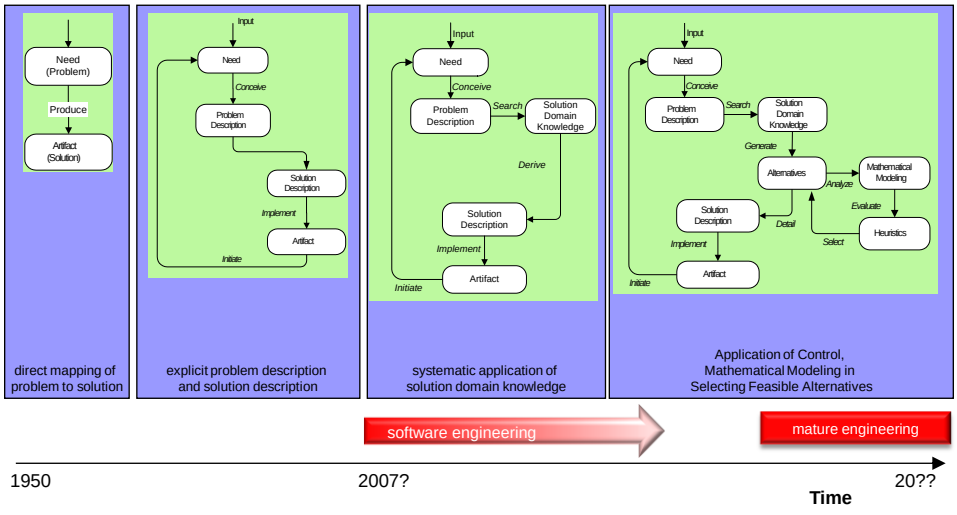


- B. Tekinerdogan & M. Akşit & F. Introducing the Concept of Synthesis in the Software Architecture Design Process, Journal of Integrated Design and Process Science, Vol. 10, No. 1, pp.45-56, 2006.
- B. Tekinerdogan and M. Akşit. Synthesis Based Software Architecture Design, in Software Architectures and Component Technology: The State of the Art in Research and Practice, M. Akşit (Ed.), Boston: Kluwer Academic Publishers, pp. 143 - 173, 2001.
- B. Tekinerdogan. Synthesis-Based Software Architecture Design, PhD Thesis, Dept. of Computer Science, University of Twente, March 23, 2000.

Comparison

	Mature engineering	Software Engineering
Technical Problem Analysis	Explicit problem description specified with quantified metrics. Well-defined problems.	Usually implicitly defined as part of the requirements and usually no quantification of required solution. Ill-defined problems.
Availability of Domain Knowledge	Very extensive solution domain knowledge compiled in different handbooks.	Basically knowledge for isolated domains in computer science. Increasing number of pattern catalogs
Application of Domain Knowledge	Explicit domain analysis process for deriving abstractions from solution domain.	Solution domain analysis not a common practice. In general applied in case reuse is required.
Solution Description	Rich set of stable notations for different problems.	Various design notations that change from time to time. Still lack of global standards.
Application of Heuristics	Explicitly specified in handbooks as a complementary means to mathematical techniques for defining feasible solutions.	Implicit in software development methods.
Alternative Analysis	Explicit alternative space analysis; optimization techniques for defining the feasible alternatives; empirical research	Implicit. Almost no systematic support for alternative space analysis.
Quality Measurement	Explicit quality concerns both for development and evaluation.	Quality is usually implicit. No systematic support for measuring quality in common software practices

Evolution of Problem Solving Concepts



Education…

Our knowledge on SE has increased

- SE is rapidly approaching towards becoming a mature engineering discipline
- The last decades our knowledge on software engineering has increased dramatically…
- Together software has become important in industry; Software has become even mission critical for many organizations who never envisioned themselves in the software business.
- As such there is an urgent need for graduates on SE!
- But…

What kind of knowledge is available?
What is important?
What should be taught? How?

Software Growth

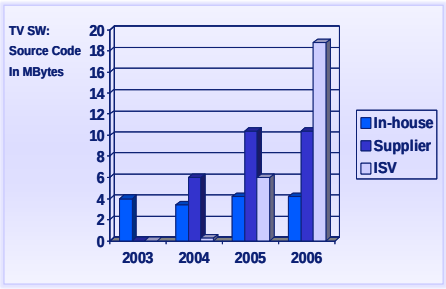
- Software is growing in size and complexity…
- Who is going to design and implement this…?!

→ More software engineers are needed

• Software project team in 1978: 1 person
Productivity: 1 Kbyte per person-year
Production: 1 KByte in one year

•Software project team in 2004: 100+ people
Productivity: 10 Kbyte per person-year
Production: 1 Mbyte in one year

• Required code-size in 2006: 32 MByte
Who is going to design/create this? (effort 2500 person years!)

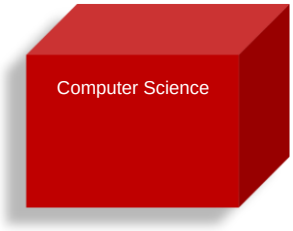


Philips Semiconductors (NXP)

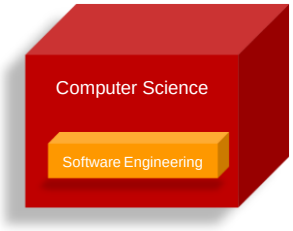
Software engineering vs. Computer Science

Conventional views:

- Software engineering is not a discipline by itself but is applied in different subdisciplines in computer science

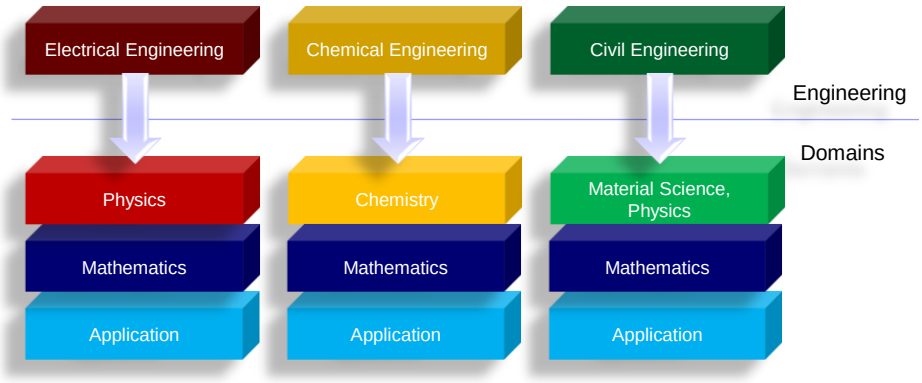


- Software engineering is a subfield of Computer Science



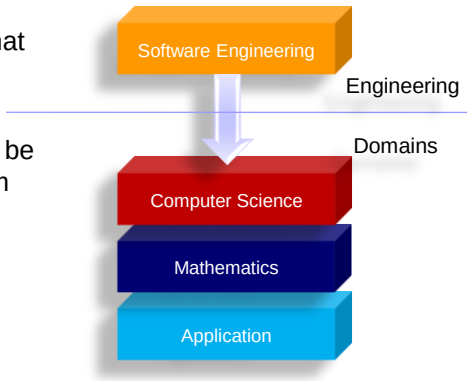
Other engineering disciplines

- Separation between engineering and its utilized domains (scientific basis)



Domains for software engineering

- **Domain**: An area of knowledge or activity characterized by a set of concepts and terminology Understood by practitioners in that area.
- Software Engineering need also be considered as being distinct from Computer Science



SWEBOK

- Software Engineering Body of Knowledge, IEEE, 2004
  - "the sum of knowledge within the profession of software engineering"
- SWEBOK Guide
  - The guide to use the body of knowledge
- Software engineers need to know:
  - Body of knowledge
  - How to apply that knowledge
  - How to apply broader domain of knowledge to build products (application domain)

The SWEBOK Knowledge Areas

Software requirements
Software design
Software construction
Software testing
Software maintenance
Software configuration management
Software engineering management
Software engineering process
Software engineering tools and methods
Software quality

Related disciplines

Computer engineering
Computer science
Management
Mathematics
Project management
Quality management
Software ergonomics
Systems engineering

http://www.swebok.org/

Possible MSc Software Engineering program

Courses shared with other engineering disciplines

General Chemistry for Engineering.
Engineering Mathematics Ia (linear systems, matrices, complex numbers).
Engineering Mathematics Ib (continuation of above).
Calculus for Engineering I.
Introductory Mechanics.
Engineering Design and Communication.
Safety Training (1 unit).
Calculus for Engineering II.
Waves, Electricity and Magnetic Fields.
Engineering Mathematics IIa (differential equations, transforms).
Engineering Mathematics IIb (vector calculus, coordinate systems).
Introductory Programming for Engineers.
Engineering Economics.

Courses introducing other engineering areas to software engineers

Introduction to the Structure and Properties of Engineering Materials.
Introduction to Dynamics and Control of Physical Systems.
Digital System Principles and Logic Design for Software Engineers.[8]
Architecture of Computers and Multi-Processors.[9]
Introduction to Thermodynamics and Heat Transfer.

Applied Mathematics

Applications of Mathematical Logic in Software Engineering.
Applications of Discrete Mathematics in Software Engineering.
Statistical Methods for Software Engineers.

Software Courses

Software Design I – Programming To Meet Precise Specifications.
Software Design II – Structure and Documentation of Software.
Design & Selection of Computer Algorithms and Data Structures.
Machine-Level Computer Programming.
Design and Selection of Programming Languages.
Communication Skills – Explaining Software.
Software Design III – Designing Concurrent and Real-Time Software.
Computational Methods for Science and Engineering.
Optimization Methods, Graph Models, Search and Pruning Techniques.
Data Management Techniques.
Software and Social Responsibility.[11]
Design of Real-Time Systems and Computerized Control Systems.
Fundamentals of Computation.
Performance Analysis of Computer Systems.
Design of Human Computer Interfaces.
Design of Parallel/Distributed Computer Systems and Computations.

D. Parnas. Software Engineering Programmes are not Computer Science Programmes, 1998

Example SE Program – Univ. of Twente

University of Twente, Dept. of Computer Science, Software Engineering

Core Courses MSc CS-SE

Curriculum: MSc. Computer Science - track SE
Start level: Bachelor
Credits: 120

Program (Curriculum Courses)

Quarter	Course code	Course name	Credits
1	158075	Deterministic Models in Operations Research	5.0
	211133	Design of Software Architectures	5.0
2	152075	Graph theory	5.0
	232080	Problem analysis and software requirements	5.0
	233030	Specification of Information Systems	5.0
3	211109	Advanced logic	5.0
	214012	System validation	5.0
4	151139	Algebra	5.0
	211123	Component en aspect oriented programming	5.0
	217001	Testing techniques	5.0
	234004	Software management	5.0

You have to select at least 6 core courses. The course Sofware Management (234004) is a compulsory core course. Your selection should have enough depth and breadth. This means in general that the courses Design of Software Architectures (211133), Specification of Information Systems (233030) and System Validation (214012) are part of the study program. The selection of courses in the study program - including core courses and specialization courses - has to be approved by the study advisor.

Advanced Courses

10 credits each
Minimum 2 out of 4

- Advanced Requirements Engineering
- Advanced Design of Software Architectures
- Advanced Programming Concepts
- Modeling & Analysis of Concurrent Systems

Elective Courses

- Advanced Courses from other CS tracks
- Patterns for Software Development (266100)
- Research Project
- Master Courses related to Graduation Project
- Maximum 30 credits external

Graduation Project / MSc Thesis

30 Credits

Software Engineering Courses

- Data abstractions in programming
- Object-oriented programming
- Software Engineering (General Course)
- Software Requirements Engineering
- Software Architecture Design
- Aspect-Oriented Software Development
- Model-Driven Software Development
- Component-Oriented Software Development
- Object-Oriented Design Patterns
- Object-Oriented Analysis and Design
- Web Services and Application Integration
- Software Product Line Engineering
- Software Project Management
- Formal Methods
- Software Testing
- Internship SE (3 months ☺)
- MSc SE
- PhD à la Europe… ☺

Impact of Education on Industry

- SE education's purpose should be to provide highly qualified engineers that can be effective in industry and/or academy
- This on its turn will be of benefit to society…

What to do in Turkey…

Make a plan for the coming 5-10 years:

- A lot more (quality) software engineering courses *should be* introduced
  - BSc and MSc courses
  - PhD research in software engineering topics
- Form -*real*- MSc software engineering programmes
  - if needed with multiple universities
  - exchange programs with other universities abroad
- Aim for hiring the coming years basically academic personnel with a clear software engineering background…
  - Some academicians might need to consider to make the move to software engineering…
- …

Responsibility of all stakeholders… (government, university, industry and other organisatons)

Research Approaches…

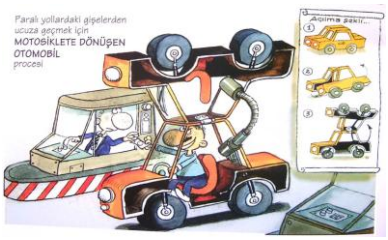
Software Research Crisis…?

- Software engineering is becoming rapidly a mature engineering discipline
- However, even a mature software engineering might fail to influence industrial practice and the quality of resulting software…
- The problem stems from treating research and the involvement of industry in applying the research as separate, sequential activities.

What should be the research and how to set this up?

C. Potts, Software Engineering Research Revisited, IEEE Software, September 1993

Useless Inventions… ☺

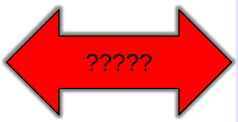


Porof. Zihni Sinir (Irfan Sayar)
http://www.zihnisinir.com/

No Research…

Application-Problem Domain (Industry)

- Problem (version 1)
- Problem (version 2)
- Problem (version 3)
- Problem (version 4)



Research-Solutions Domain (Academia)

Research-but-no-Transfer

Application-Problem Domain (Industry)

- Problem (version 1)
- Problem (version 2)
- Problem (version 3)
- Problem (version 4)

?????

Research-Solutions Domain (Academia)

- Research Solution (version 1)
- Research Solution (version 1)
- Research Solution (version 1)
- Research Solution (version 1)

Research-then-Transfer

Application-Problem Domain (Industry)

- Problem (version 1)
- Problem (version 2)
- Problem (version 3)
- Problem (version 4)

Wide gulf bridged by indirect, anecdotal knowledge

Technology transfer Gap bridged by hard, but frequently inappropriate technology

Research-Solutions Domain (Academia)

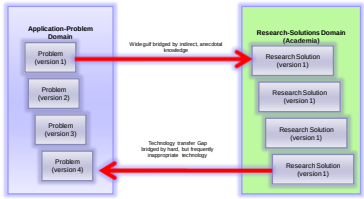
- Research Solution (version 1)
- Research Solution (version 1)
- Research Solution (version 1)
- Research Solution (version 1)

Problem evolves invisibly to the research community

Incremental Refinement of research solutions

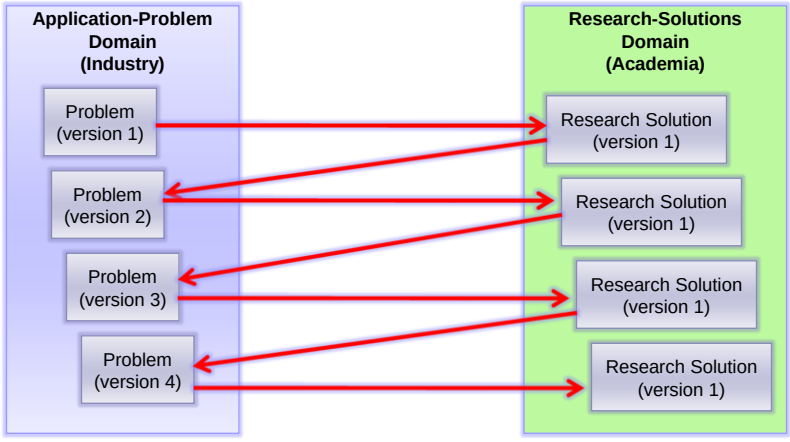
Research-then-Transfer Problems

- weak connections even from the start
- lack of interaction to define the problem
- this causes both research and practice to evolve separately
- Concentration in research is on technical refinement of research solution - but lacks industrial need as focus and as such may be misplaced.
- Delay in evaluation; no early evaluation
- Evaluation is difficult as research solutions may use technology that is not commonly used in industry
- Transfer is difficult because industry has little basis for confidence in proposed research solution.

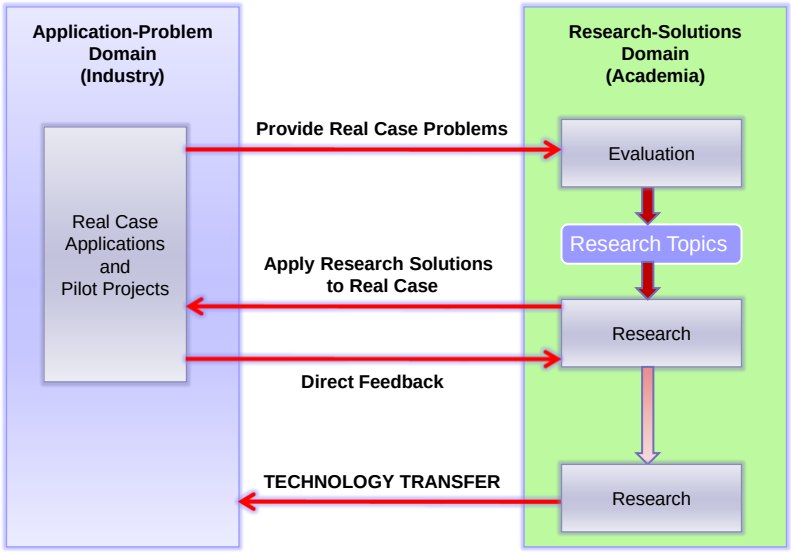


- No real winners…

Industry-as-Laboratory

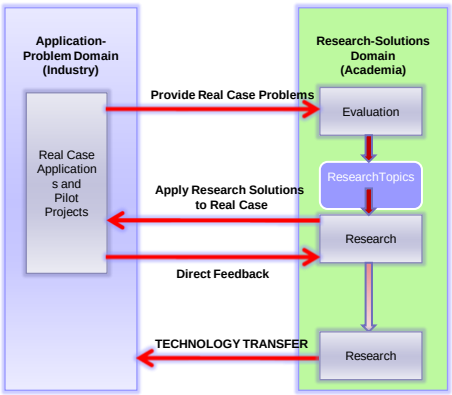


Industry-as-Laboratory



Industry as Laboratory - Advantages

- Problem is acquired from the real practitioners in industry, (e.g industrial partners in a research project).
  - Stronger connection at start
  - solve problems that really do matters to practitioners
- Constant interaction by practitioners and researchers
- Early evaluation and usage by industry
- optimal reduction of Technology Transfer Gap.



Emphasis on Real Case Studies

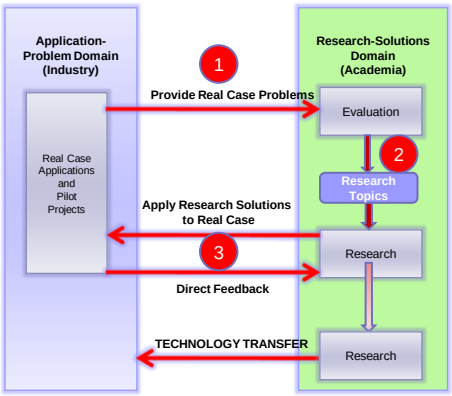
Industry-as-Laboratory emphasizes Real Case Studies:

- **Scale and complexity**
  - small and simple cases are avoided
  - large scale of real cases lead to changes in the types of problems that become most significant and in the solutions that must be adopted .
- **Unpredictability**
  - implicit assumptions are eliminated as researchers learn more about real problems
- **Dynamism**
  - an invented, fictitious case study cannot capture project dynamics
  - system evolution and changes makes a real-world case study much more vital

What about publications…?

1. Industry-as-Laboratory emphasizes Real Case Studies
2. The real-world complications of industrial case studies lead to more representative problems
3. Evaluation of the research solutions are done in a real-world setting

→ These are ingredients for high quality publications!



Some concerns…

- **Overemphasis on the short term**
  - delving into too much detail of the real-case…
- **Yo-yoing between details and abstractions**
  - how to interpet and abstract the details?
- **Discomfort with lack of control**
  - researchers are to some extent dependent on the events in the project
- **Tendency to consider technology transfer as happening automatically**
  - industry-as-laboratory approach facilitates technology transfer
  - however it is still a difficult process and needs careful planning
- **Loss of privacy**
  - Companies are reluctant to have their systems and projects discussed in public
  - Confidentiality must be preserved

Shifting towards industry-as-laboratory

- The various forms of research ideally complement one another.
  - Too industrially focused research may lack adequate theory
  - Academically focused research may miss the practice
- There is and should still be a place for innovative, purely speculative research in software engineering,
- However, research which studies real problems in partnership with industry needs to be given a higher profile.

Example Industry-as-Laboratory Project - Trader Project

Trader Project

Industry-as-laboratory project

- **T**elevision **R**elated **A**rchitecture **D**esign to **E**nhance **R**eliability (Trader)
- Context: Embedded Systems and in particular consumer electronics (Digital TV)
- Period: Sept.2004-Aug. 2009
- 10 partners (industrial and academic)
- 22 fte/yr, 7 PhDs, 2 Postdocs,
- around 3-4 miljon euro
- funded by The Netherlands Ministry of Economical Affairs under the Bsik programme

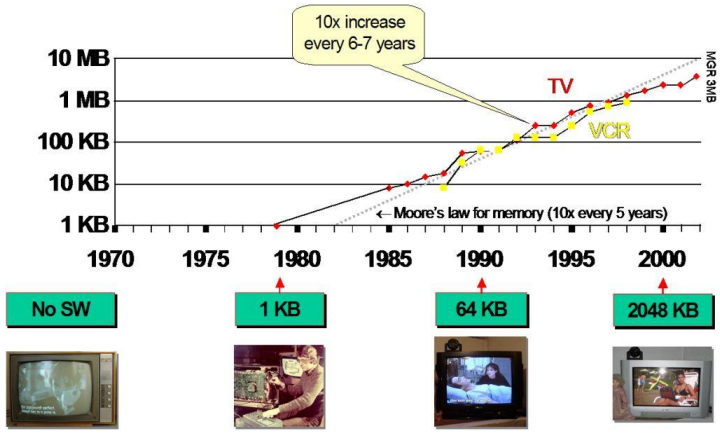
Embedded Systems INSTITUTE

ESI, Eindhoven: project location

NXP founded by Philips

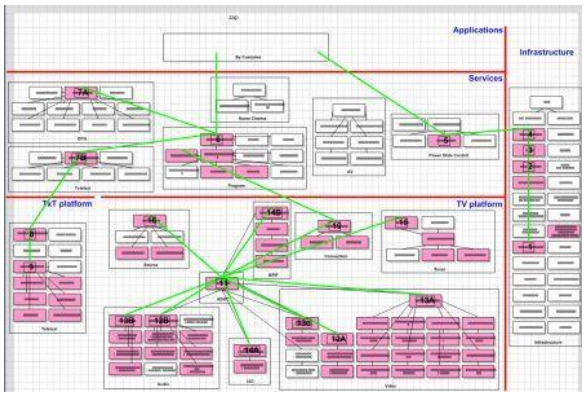
Carrying Industrial Partner

Software that controls the system…



Slide from: B. Pronk, NXP (Philips Semiconductors)

Complexity of DTV



DTV Architecture

Trader Context

- The universal trend of the increased functionality, openness and shift from hardware to software puts serious challenges on quality factors such as performance, availability and reliability
- In fact, embedded systems are now largely defined and controlled by software as such
- the required quality factors are likewise more dependent on the quality of the adopted software and to a lesser degree on the hardware.
- …it is expected that the risk of failures in embedded systems can increase to a mission critical level.
- …to minimize this risk it is required that appropriate reliability analysis and design techniques are provided so that potential failures can be predicted or corrected in time.

DTV Reliability Problems

Aspect Ratio Problems

Mode Confusion

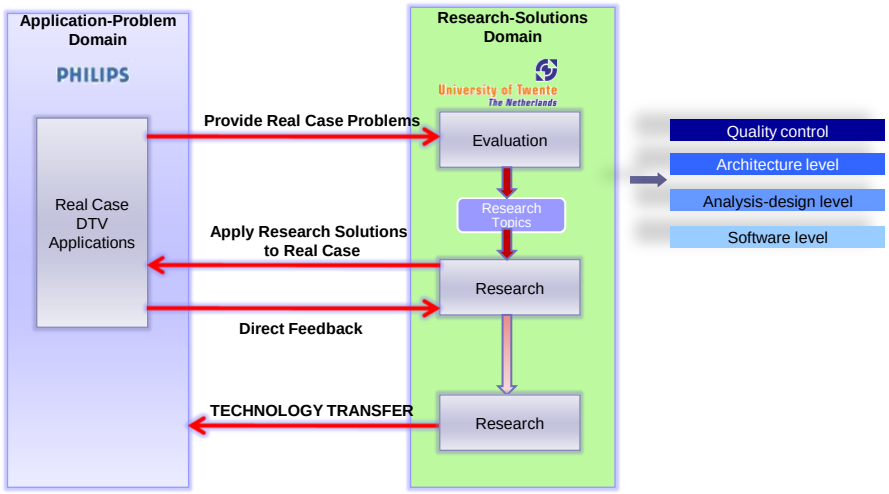
Decreased Resolution

H. Sözer, C. Hofmann, B. Tekinerdogan, & M. Akşit. Detecting Mode Inconsistencies in Component-Based Embedded Software, in proceedings of workshop on Architecting Dependable Systems, Edinburgh, Scotland, 2007.

Trader Research Areas

- **User-centric design for reliability**:
  - e.g. what is the reliability impact on the user and on the business?
  - What are methods to minimize this impact?
- **Software architectures and implementation**:
  - e.g. what is an appropriate software architectures that prevent faults from contaminating the system?
  - What are run-time mechanisms and constructs that when applied, result in more reliable behavior of the system?
- **Analysis and detection of imminent product failures**:
  - e.g. reliability modeling, hardware/software interaction, checking techniques.
  - How to apply these techniques at design time, test-time, and run-time.
- **Run-time prevention of product failures**:
  - e.g. what are end-user constraints and acceptable recovery strategies?
  - How to implement fault tolerance?

Industry-as-Laboratory



Example Research Questions…

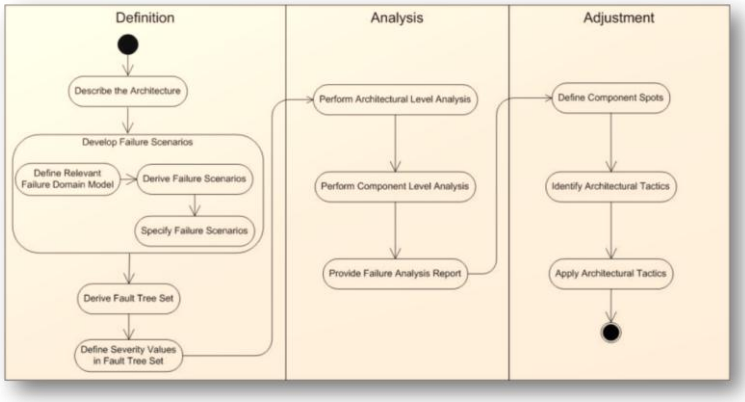
- What is the process for sensitivity analysis wrt reliability in the early phase of the software life cycle?
- How to document recovery design decisions?
- How to optimize system recovery with respect to performance and availability
- What is the impact of concurrency on failures?

Some Early Results…

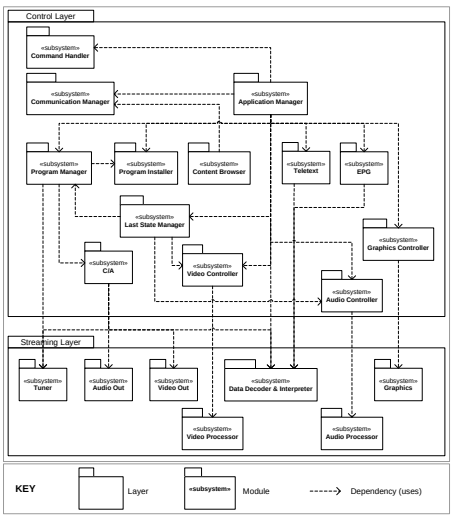
- Reliability Analysis using Failure Scenarios
  - B. Tekinerdogan, H. Sözer, & M. Akşit. Software Architecture Reliability Analysis using Failure Scenarios, submitted to Elsevier Journal of Software and Systems, 2007.
  - H. Sözer, B. Tekinerdogan, & M. Akşit. Extending Failure Modes and Effects Analysis Approach for Reliability Analysis at the Software Architecture level, accepted for publication as book chapter, Springer LNCS, 2007.
  - B. Tekinerdogan, H. Sözer, M. Akşit. Software Architecture Reliability Analysis using Failure Scenarios, 5th Working IEEE/IFIP Conference on Software Architecture (WICSA), Working Session, Pittsburgh, US, November, 6-10, 2005.
- Architectural Guidelines for Local Recoverability of Embedded Systems
  - H. Sözer, C. Hofmann, B. Tekinerdogan, & M. Akşit. Detecting Mode Inconsistencies in Component-Based Embedded Software, in proceedings of workshop on Architecting Dependable Systems, Edinburgh, Scotland, 2007.
  - B. Tekinerdogan, F. Scholten, C. Hofmann, & M. Akşit. Concern-Oriented Analysis and Refactoring of Software Architectures using Dependency Structure Matrices, submitted to IEEE WICSA Conference, 2008.
- Architectural Recovery Style for Modeling and Analyzing System Recovery
  - H. Sözer & B. Tekinerdogan. Introducing Recovery Style for Modeling and Analyzing System Recovery. submitted to IEEE WICSA Conference, 2008.
- Failure detection based on concurrency control schemes…
  - [to be written].

SARAH Process

- Software Architecture Reliability Analysis Approach
  - Systematic and generic approach for reliability analysis at the architecture design level
  - Before the system has been implemented

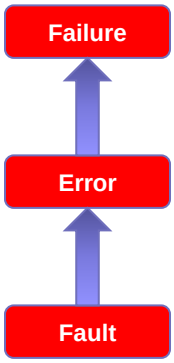


Conceptual Architecture DTV



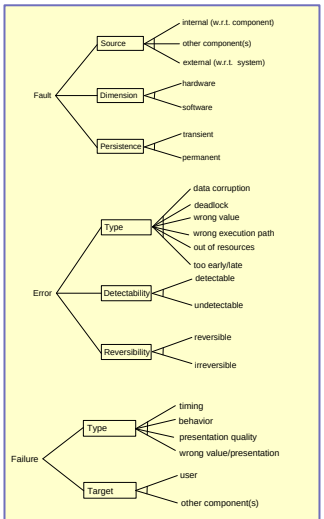
Reliability

- The probability that a system will continue to function without **failure** for a specified period in a specified environment.
  - **Failure**
    - an event that occurs when the delivered service of a system deviates from a correct service
  - **Error**
    - system state that is liable to lead to a failure
  - **Fault**
    - the cause of an error



ANSI/IEEE, "Standard Glossary of Software Engineering Terminology," STD-729-1991, ANSI /IEEE, 1991.

Deriving Failure Scenarios from Domain Models



Domain Model for Fault, Error and Failure

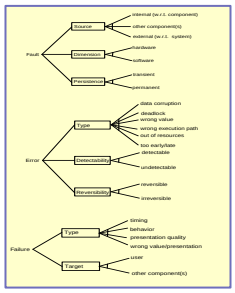
- **FID:** Failure ID
- **CID:** Component ID
- **Fault:** Description of the cause and its features.
- **Error:** Description of how the component fails or the component state that leads to the failure together with its features.
- **Failure:** Description of the effect of the failure and its features.

Template for defining failure scenarios

FID	CID	Fault	Error	Failure
F3	CB	*description:* Can not acquire information. *source:* AMR(F2) *dimension:* software *persistence:* transient	*description:* Information can not be presented due to lack of information. *type:* too early/late *detectability:* detectable *reversibility:* irreversible	*description:* Can not present content of the connected device. *type:* behavior *target:* user

Failure Scenario

SARAH: Deriving Failure Scenarios



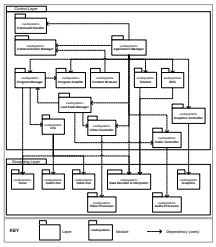
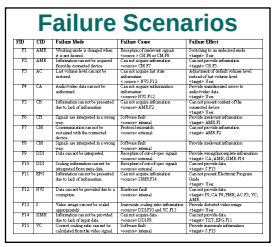


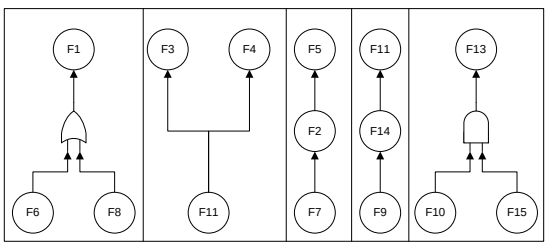
Table 2. Selected concrete failure scenarios derived from general scenarios for analysis of DTV architecture

FID	CID	Fault	Error	Failure
F1	AMR	*description:* Reception of irrelevant signals. *source:* CH(F4) OR CMR(F6) *dimension:* software *persistence:* transient	*description:* Working mode is changed when it is not desired. *type:* wrong path *detectability:* undetectable *reversibility:* reversible	*description:* Switching to an undesired mode. *type:* behavior *target:* user
F2	AMR	*description:* Can not acquire information. *source:* CMR(F5) *dimension:* software *persistence:* transient	*description:* Information can not be acquired from the connected device. *type:* too early/late *detectability:* detectable *reversibility:* irreversible	*description:* Can not provide information. *type:* timing *target:* CB(F3)
F3	CB	*description:* Can not acquire information. *source:* AMR(F2) *dimension:* software *persistence:* transient	*description:* Information can not be presented due to lack of information. *type:* too early/late *detectability:* detectable *reversibility:* irreversible	*description:* Can not present content of the connected device. *type:* behavior *target:* user
F4	CH	*description:* Software fault. *source:* internal *dimension:* software *persistence:* permanent	*description:* Signals are interpreted in a wrong way. *type:* wrong value *detectability:* undetectable *reversibility:* reversible	*description:* Provide irrelevant information. *type:* wrong value/presentation *target:* AMR(F1)
F5	CMR	*description:* Protocol mismatch. *source:* external *dimension:* software *persistence:* permanent	*description:* Communication can not be sustained with the connected device. *type:* too early/late *detectability:* detectable *reversibility:* irreversible	*description:* Can not provide information. *type:* timing *target:* AMR(F2)
F6	CMR	*description:* Software fault. *source:* internal *dimension:* software *persistence:* permanent	*description:* Signals are interpreted in a wrong way. *type:* wrong value *detectability:* undetectable *reversibility:* reversible	*description:* Provide irrelevant information. *type:* wrong value/presentation *target:* AMR(F1)
F7	DDI	*description:* Reception of out-of-spec signals. *source:* external *dimension:* software *persistence:* transient	*description:* Scaling information can not be interpreted from meta-data. *type:* wrong value *detectability:* detectable *reversibility:* reversible	*description:* Can not provide data. *type:* wrong value/presentation *target:* IC(F8)
F8	IC	*description:* Inaccurate scaling ratio information. *source:* DDI(F7) AND VC(F9) *dimension:* software *persistence:* transient	*description:* Video image can not be scaled appropriately. *type:* wrong value *detectability:* undetectable *reversibility:* reversible	*description:* Provide distorted video image. *type:* presentation quality *target:* user
F9	VC	*description:* Software fault. *source:* internal *dimension:* software *persistence:* permanent	*description:* Correct scaling ratio can not be calculated from the video signal. *type:* wrong value *detectability:* detectable *reversibility:* reversible	*description:* Provide inaccurate information. *type:* wrong value/presentation *target:* IC(F8)

SARAH: Deriving Fault Tree Set

- Fault Tree Set
  - Each node represents a failure
  - Root node: target = User
  - Logic gates for inferring fault propagation

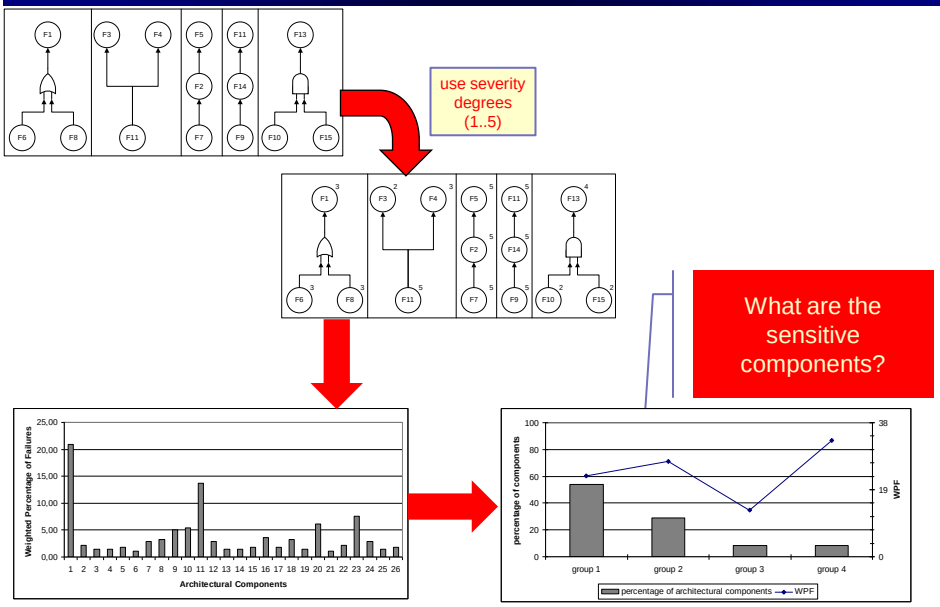




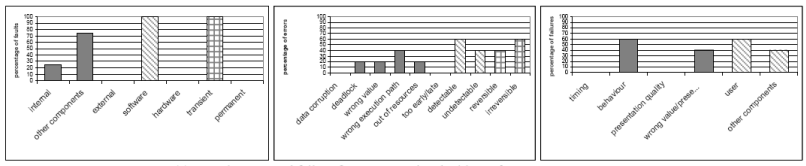
Defining severity of failures based on user-perception

Severity	Description	Annoyance Description
1.	very low	user hardly perceives failure
2.	low	failure is perceived but not really annoying
3.	moderate	annoying performance degradation is perceived
4.	high	user perceives serious loss of functions
5.	very high	basic user-perceived functions fail. System locks up and does not respond

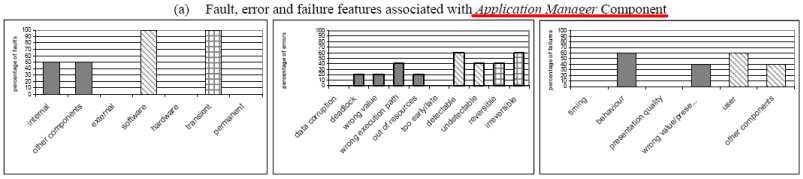
Defining Severities



SARAH: Component-Level Analysis



(a) Fault, error and failure features associated with *Application Manager* Component



(b) Fault, error and failure features associated with *Teletext* Component

What are the critical error types per (critical) component?

Lessons Learned

- The development of our approach was triggered by real problems in industry
- Method is general but applied to real case study (DTV)
- Intermediate evaluations and feedback by domain experts in industry and peer researchers in the project
- Identified several important research problems
  - What should be the units of recovery? How to analyze the optimal recovery decomposition? How to model recovery view?
  - How to define a user perception model? How to analyze severity?
  - How to analyze for failures derived from specific concerns such as concurrency, synchronization?
  - How to simulate failure scenarios? How to do model-checking?

Acknowledgments

- Thanks to the TRADER project which is under the responsibility of the Embedded Systems Institute. This project is partially supported by the Netherlands Ministry of Economic Affairs under the Bsik program.

Further thanks to my colleagues in the project:

- Mehmet Aksit (University of Twente)
- Rob Golsteijn (Philips Semiconductors (NXP))
- Christian Hofmann (University of Twente)
- Jozef Hooman (Embedded Systems Institute)
- Paul L. Janson (Philips Research)
- Pierre van de Laar(Philips Research)
- Iulian Nitescu (Philips TASS)
- Ben Pronk (Philips Semiconductors (NXP))
- Hasan Sözer (University of Twente)
- Teun Hendriks (Embedded Systems Institute)

Conclusion

Conclusion

- Software engineering is in essence a problem-solving process and to understand software engineering it is necessary to understand problem-solving.
- It appears that software engineering follows the same path of evolution of problem solving concepts as in traditional engineering
- With respect to the developments in other engineering disciplines, our study shows a higher pace of the evolution of problem-solving concepts in software engineering
- For increasing maturity level basically the three processes of *problem analysis, solution domain analysis* and evaluation (*alternatives analysis)* need to be completed

Conclusion

- The importance of software has dramatically increased in society
- Together with the increased knowledge on software engineering
- this has resulted in an urgent need for graduates in software engineering
- Software engineering education programmes are not computer science programmes and need to be developed as such.
- To keep with the developments in the world a vision needs to be formed and a mission statement developed for the coming years… asap…

Conclusion

- A mature software engineering and education can still fail in the sense that it does not impact industry
- Different research approaches can be identified
- Unfortunately, very often SE research is alltogether lacking or following a research-no-transfer or research-then transfer approach
- These are less effective than the industry-as-laboratory approach in providing industry-relevant solutions
- The industry-as-laboratory approach provides a win-win situation
- As such, the focus in software research should shift towards Industry-as-laboratory approach

ita est….

Şimdi yeni şeyler söylemek lazım…

Hergün bir yerden göçmek ne iyi
Bulanmadan donmadan akmak ne hoş
Hergün bir yere konmak ne güzel
Dünle beraber gitti cancağızım
Ne kadar söz varsa düne ait
Şimdi yeni şeyler söylemek lazım

-Mevlana

Questions…