

Introduction



Motivation

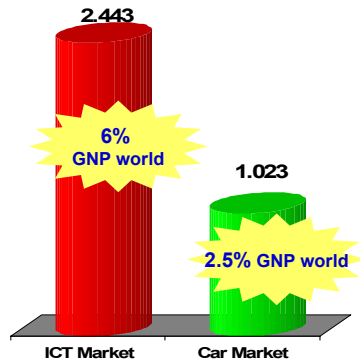
Summary

- Software – if engineered – consists of products at various levels of abstraction, ranging from code over designs to requirements. Each product is useful ONLY, if stakeholders (people, roles) are defined using these products, and if it is traceable (up/down) to related products. In industry, a large variety of “real” products (terminology & contents) and processes by which these products are created exist. The common denominator, however, are essential contents. Therefore, in this lecture, a so-called Virtual Product Model is used as a reference model.
- In real-world development environments each real physical document contains part, one or more of these virtual products. For example, “System analysis document” may contain problem description and user requirements. In any event, lack of any virtual product has to be justified!

Software and the economy

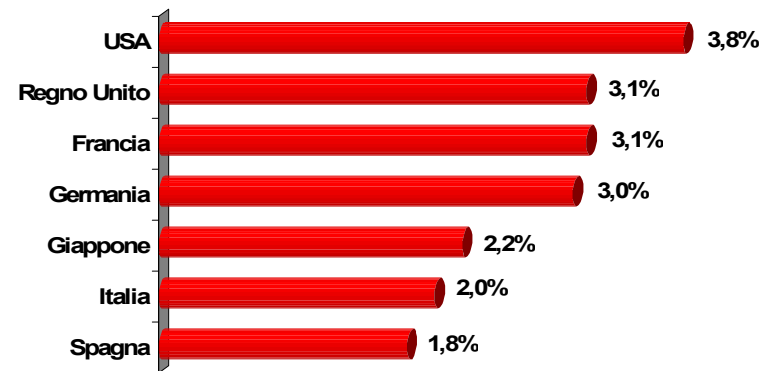
- The economies of ALL developed nations are dependent on software.
- More and more systems are software controlled
- Expenditure on software represents a significant fraction of GNP in all developed countries.
- Software engineering: how to develop software

ICT Market - world - 2004



SoftEng
http://softeng.polito.it

IT - GNP (2003)

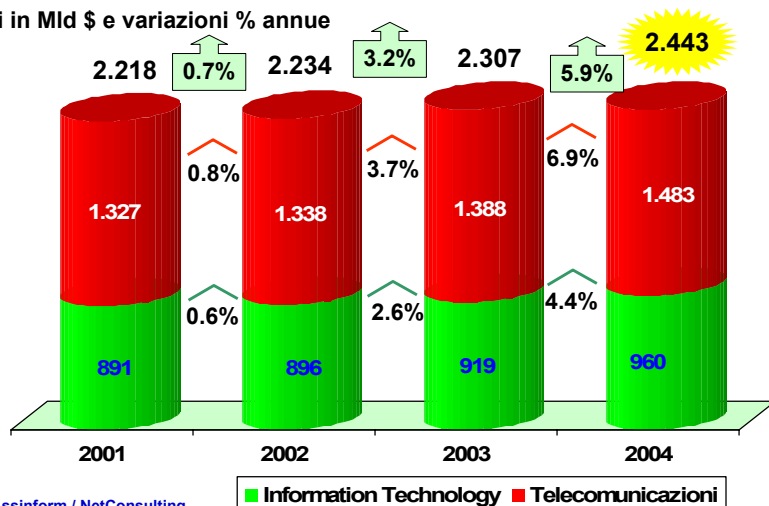


Fonte: Assinform / NetConsulting

SoftEng
http://softeng.polito.it

CT - world (2001-2004)

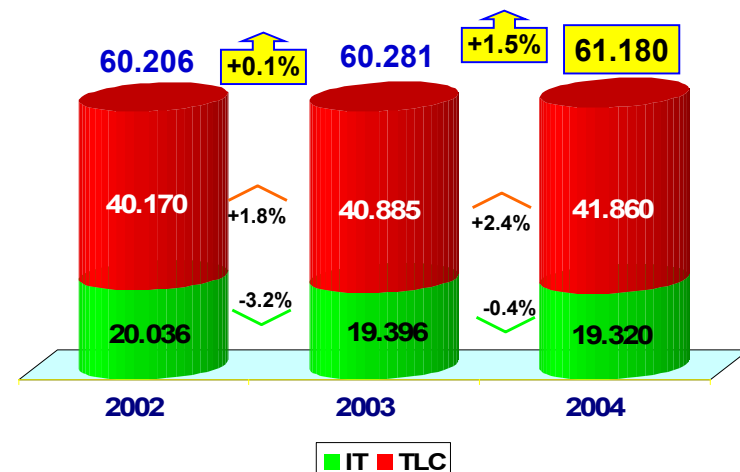
Valori in Mld \$ e variazioni % annue



Fonte: Assinform / NetConsulting

SoftEng
http://softeng.polito.it

ICT - Italy (2002-2004)



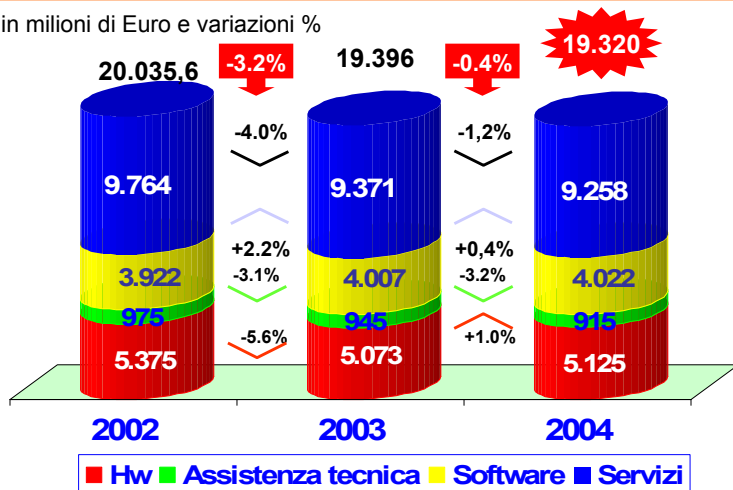
Fonte: Assinform / NetConsulting

SoftEng
http://softeng.polito.it

Valori in Milioni di Euro e in %

IT Italy (2002-2004)

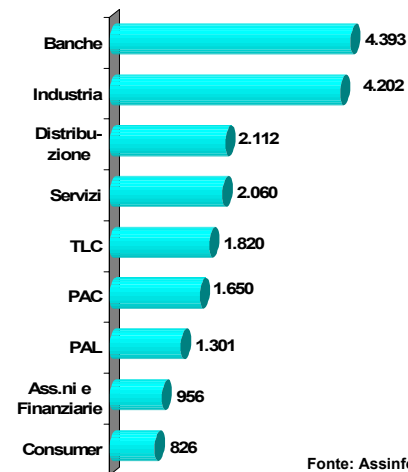
Valori in milioni di Euro e variazioni %



Fonte: Assinform / NetConsulting



IT per settore (2002-4)



Fonte: Assinform / NetConsulting



Main ICT companies (Italy)

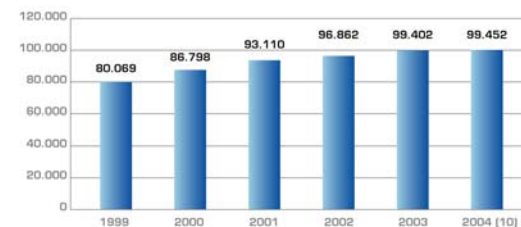
- 1 IBM ITALIA SPA
- 2 IT TELECOM SPA
- 3 GLOBAL VALUE SERVICES & SOLUTIONS
- 4 GRUPPO FINSIEL
- 5 ATOS ORIGIN SPA - SEMA SPA
- 6 GETRONICS SOLUTIONS ITALIA SPA
- 7 GRUPPO ENGINEERING
- 8 ELSAG SPA
- 9 ORACLE ITALIA SRL
- 10 GRUPPO SIEMENS BUSINESS SERVICES
- 11 SAP ITALIA SPA
- 12 DATAMAT SPA
- 13 GRUPPO CEDACRI
- 14 T-SYSTEMS ITALIA SPA
- 15 CSI - PIEMONTE
- 16 GRUPPO INFRACOM
- 17 COMPUTER ASSOCIATES S.P.A.
- 18 ZUCCHETTI SRL
- 19 INFOCAMERE
- 20 C.S.E. SPC. CONS. A.R.L.
- 21 NCR ITALIA SPA
- 22 ETNOTEAM SPA
- 23 FINMATICA SPA
- 24 REPLY SPA
- 25 GRUPPO PRIDE
- 26 SEC SERVIZI
- 27 SAS INSTITUTE SRL
- 28 MET SOGEDA
- 29 C.T.O.S.P.A.
- 30 GRUPPO FORMULA SPA

Classifiche di Databank Consulting e Data Manager delle aziende di software e servizi 2003



Number ICT companies (I)

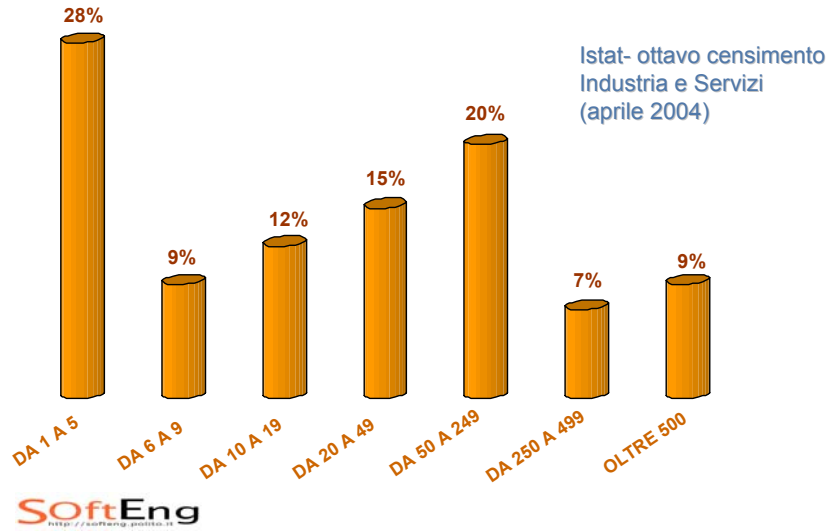
LE AZIENDE ICT IN ITALIA



Federcomin/DIT – Osservatorio Permanente della società dell'informazione – ottobre 2004



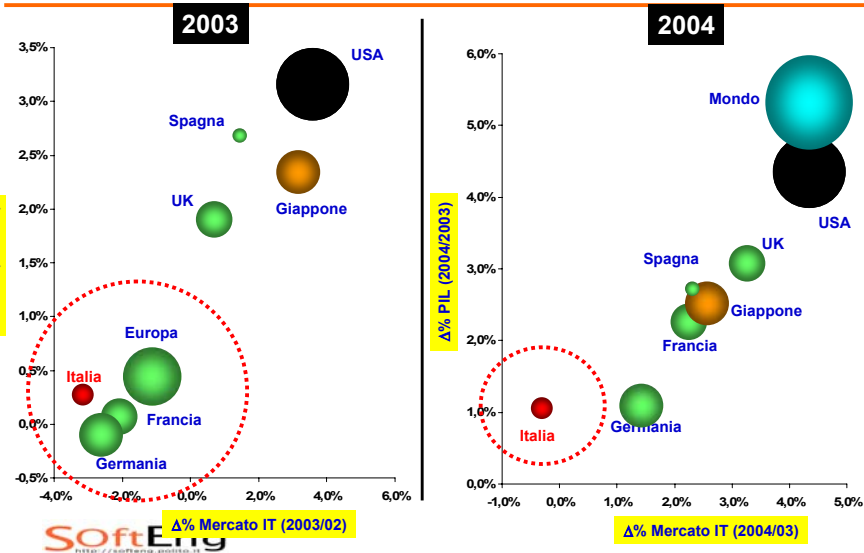
Employees



Software, innovation, development

- Evidence of correlation between ICT diffusion and wealth
 - ♦ Positive correlation IT usage and per capita GDP
 - ♦ Positive correlation productivity increase and ICT usage

Development - IT investment



Development and ICT usage

- 1 Singapore
- 2 Iceland
- 3 Finland
- 4 Denmark
- 5 USA
- 6 Sweden
- 7 Honk Kong
- 8 Japan
- 9 Switzerland
- 10 Canada
- 11 Australia
- 12 UK
- 13 Norway
- 14 Germany
- 15 Taiwan

- 16 Netherlands
- 19 Austria
- 20 France
- 29 Spain
- 39 India
- 41 China
- 45 Italy

Classifica WEF
(world economic forum)
Global IT report 2004-200
www.weforum.org

WEF - Global competitiveness

- 1 Finland
- 2 USA
- 3 Sweden
- 4 Denmark
- 5 Taiwan
- 6 Singapore
- 7 Iceland
- 8 Switzerland
- 9 Norway
- 10 Australia
- 11 Netherlands
- 12 Japan
- 13 UK
- 14 Canada
- 15 Germany

Classifica WEF
(world economic forum)
Global competitiveness
report 2004-2005
www.weforum.org

SoftEng 47 Italy
<http://softeng.polito.it>

Global competitiveness

- Institutions
- Infrastructure
- Macroeconomy
- Health + primary education
- Higher education
- Market efficiency
- Technological readiness
- Business sophistication
- Innovation

SoftEng
<http://softeng.polito.it>

Table 4: Top performers in the nine pillars of the Global Index

Country	Institutions	Infrastructure	Macroeconomy	Health and primary education	Higher education and training	Market efficiency	Technological readiness	Business sophistication	Innovation
Singapore	1	5	9	69	8	4	1	20	9
Denmark	2	1	16	23	3	5	2	4	10
Taiwan	27	34	1	25	42	24	36	31	41
Norway	26	9	93	1	16	16	17	1	2
Canada	3	10	10	10	1	12	12	12	4
United States	16	8	62	47	2	1	5	3	1

Definitions

Software

- **Software is a collection of computer programs, procedures, rules, associated documentation and data.**
 - ♦ software development is more than merely the development of programs
 - ♦ software incorporates documents describing various views for various stakeholders (e.g. users, developers)
- **For a given problem, software is approximately 10 times more expensive to produce than a simple program [Brooks75: The Mythical Man Month]**
 - ♦ Average: 10 to 50 LoC per person day
 - ♦ About 7 LoC in critical systems

Software – types

- stand alone
 - ♦ word processor,
- embedded
 - ♦ ABS, digital camera, mobile phone, ..
- process support
 - ♦ production process (things): industrial automation
 - ♦ business process (information): management automation

Software – criticality

- safety critical
 - ♦ aerospace, military, medical, ..
- mission critical
 - ♦ banking, logistics, industrial production, ..
- other
 - ♦ games, ..

Software – complexity

- Complexity: Parts and interactions among parts
 - [H Simon, The sciences of the artificial 1969]
 - ♦ car: 30.000 parts/components
 - ♦ airplane: 100.000 parts/components
 - ♦ cell phone, printer driver: 1M Lines of code
 - ♦ cellular network, operating system: several Millions
- software systems are probably the most complex human artifacts

Shames ..

- Safety critical
 - ♦ Therac25 – 3 casualties (1985)
 - ♦ crash KAL 901 – 225 casualties
- Mission critical
 - ♦ Ariane V (1996)
 - ♦ Mars Polar Lander (2000)
 - ♦ ATT switching system (1990)

Diffusion

- local
 - ♦ 1945 – 1980: scientific community, military, banks, large private organizations
- global
 - ♦ 1985 – today: ‘free’ hardware, huge diffusion of computing, impact on everyday’s life

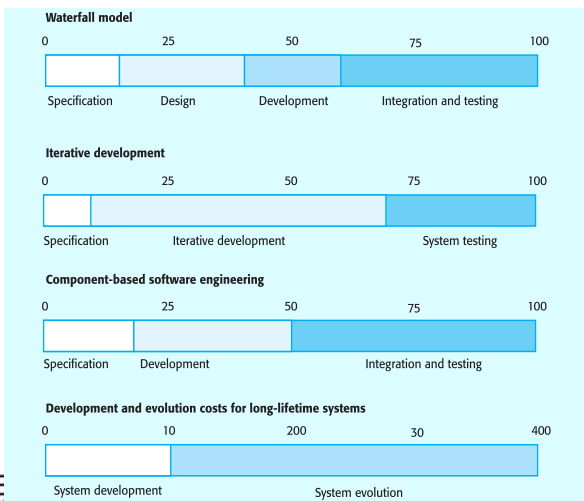
Software costs

- Software costs often dominate computer system costs. The costs of software on a PC are often greater than the hardware cost.
- Software costs more to maintain than it does to develop. For systems with a long life, maintenance costs may be several times development costs.

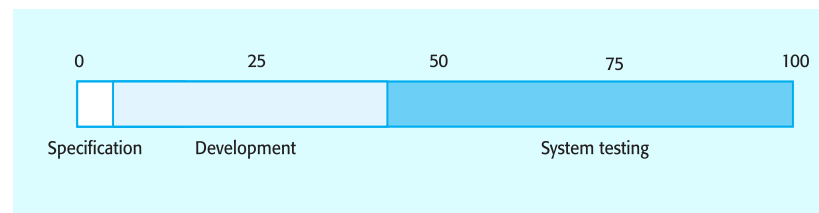
What are the costs of sw engineering?

- ♦ Roughly 60% of costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs.
- ♦ Costs vary depending on the type of system being developed and the requirements of system attributes such as performance and system reliability.
- ♦ Distribution of costs depends on the development model that is used.

Activity cost distribution



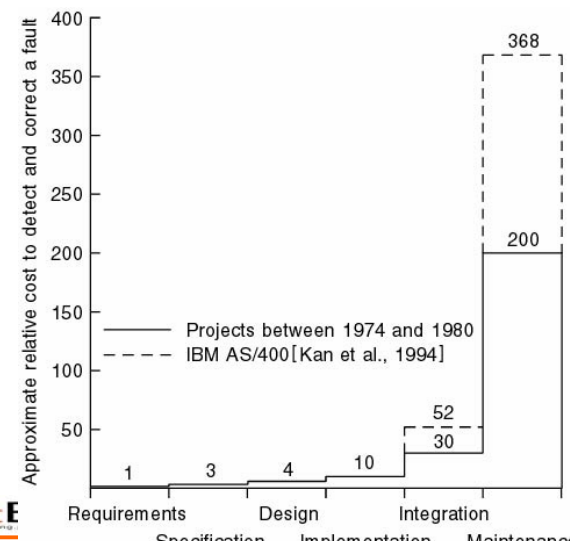
Product development costs



Quality

- Most defects injected in requirements and design
- The earlier a defect injected, the more costly to correct it
 - ♦ Because defects are usually found in operation

Defect correction cost



Functional vs. non functional

- Functional characteristics of software
 - ♦ “Add two integer numbers”
- Non functional properties
 - ♦ User interface usable by not computer expert
 - ♦ Precision
 - relative error $< 10^{-9}$
 - absolute error $< 10^{-8}$
 - ♦ Reliability
 - sum must be correct 99,999999% times
 - ♦ Performance, efficiency
 - Sum must be performed $< 0,01$ millisec
 - Sum must use < 10 kbytes ram memory

Functional vs. non functional

- Non functional properties sometimes harder to express
- Harder to design into software
 - ♦ They are *emerging* properties
 - Depend on the whole system, i.e. reliability, performance

Myths

Software is inexpensive

- ♦ Add-on to engineering products, as product often free?
- ♦ Very labor intensive --
assuming
 - Productivity = 40 - 8000 LOC per person month
 - Personal cost = : \$ 8.000 per person month
→ \$1 to \$200 per LOC
- a medium sized project with 50.000 LOC costs between \$50.000 and \$10.000.000 in personnel

Myths

Software does not break as it ages

- Failures do not occur due to material fatigue (as with hardware) but due to the execution of logical faults
 - ♦ hardware reliability concepts don't work
- All software faults can be removed before execution
- Software changes due to requirements changes, platform changes (and defect corrections)

Myths

Software is produced

- **Software is not mass produced (like machines)**
 - ♦ replication (manufacturing) is almost effortless
- **Software is developed**
 - ♦ the description of the solution is the product
 - ♦ Non-deterministic process due to human involvement
 - ♦ Controllable in a probabilistic manner only
 - ♦ Quality focus/assurance has to be part of the development phase

Typical software problems

- **Too expensive (up to a factor of 10).**
- **Delivered too late (up to a factor of 2).**
- **Does not live up to user expectations (e.g., reliability)**
- **High-Profile Example Failures**
 - ♦ Ariane-5 accident (? System process problem ?)
 - ♦ Year-2000 problem (? System architecture/documentation problem ?)

Solo programming vs. engineering

Solo-Programming characteristics (Dave Parnas):

- **Small programs**
 - ♦ how are complex programs structured?
 - ♦ --> modularization
- **Programs specified in computer science jargon**
 - ♦ how does one communicate with customers?
 - ♦ --> need adequate language
- **Focus on "correctness" as the measure of quality**
 - ♦ what other quality attributes might be important?
 - ♦ --> ex safety, performance
- **Activities performed Individually**
 - ♦ how is teamwork supported?
 - ♦ --> precise interfaces, roles, process model

- **Quality properties (e.g. correctness) are demonstrable**
 - ♦ usually independently of testing
 - ♦ --> not true for all non functional requirements
- **Quality properties are externally visible**
 - ♦ how does one communicate with customers?
 - ♦ --> specification, measurable
- **High understandability and changeability**
 - ♦ How does one guarantee ease of modification & quality of modified system?
 - ♦ --> traceability of documentation to code & (divide & conquer)

Engineering – solo programming

- The main difference between solo programming and software engineering is not really the result of a project (in terms of code / executable system)

BUT

- The way the system is
 - ♦ developed (e.g., according to eng. principles like early defect detection and state-of-the-art methods)
 - ♦ architected (e.g., modularized for ease of change)
 - ♦ documented (e.g., tractable doc for ease of change & notation suitable for stake-holders)
 - ♦ trusted (e.g., verification & validation is documented)

- | | |
|---|--------------------------------------|
| ▪ Software large | ▪ Software small |
| ▪ team / teams | ▪ One person |
| ▪ Requirements defined by customer, contract signed | ▪ Requirements defined by programmer |
| ▪ Many deliverables | ▪ One deliverable |
| ▪ Many changes, long lifespan | ▪ Few changes, short lifespan |
| ▪ costly | ▪ free |

SE

- Focus on the development of large/complex systems
- Deal with systems that satisfy third party requirements
- Consider Team-based Development
- Make software maintainable (Software Systems can outlive their developers, and can be used by many people)

Large complex systems

- Based on the “divide and conquer” principle
- Structuring concepts for intellectually managing complex systems include
 - ♦ Modularization (horizontal division)
 - ♦ Explicit interfaces
 - ♦ Documentation (vertical division)

3rd party requirements

- **Communication between as well as with developers must be supported**
 - ♦ appropriate languages
- **Wide range of quality requirements on software systems**
 - ♦ more than just correctness
- **The assurance of quality requirements must be demonstrable**
 - ♦ certification
- **The effects of possible failures must be limited**
 - ♦ avoidance of catastrophic failures

Team based

- **The exact behavior of a software system and its units must be precisely defined**
 - ♦ formally or informally
- **The independent performance and coordination of activities by different team members must be supported**

Long lived, multiuser

- **A system should be developed to support**
 - ♦ change
 - ♦ extension
 - ♦ reuse of individual parts
 - ♦ deployment in various configurations and on various platforms

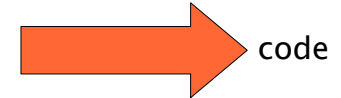
Software

- **In summary**
 - ♦ large
 - ♦ complex
 - ♦ expensive
 - ♦ diffused everywhere
 - ♦ long lived
 - ♦ mission or safety critical
 - ♦ “not” solo programming

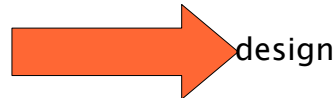
Getting there

From the bottom up

- We need the final thing
 - ♦ Executable code
- But we do not write the executable
 - ♦ Source code



-
- But the source code is large
 - ♦ Several physical units
 - Files and directories
 - ♦ Several logical units
 - Functions
 - classes
 - Packages
 - Subsystems
 - So, what units? How do we define and organize them?



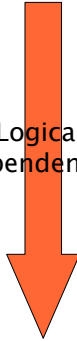
-
- But, exactly, what the software should do?
 - ♦ Add numbers, count cars, forecast weather, control mobile phone, support administration of company?



The production activities

- Requirement engineering
 - ♦ What the software should do
- Architecture and design
 - ♦ What units and how organized
- Implementation
 - ♦ Write source code, (executable code)
 - ♦ Integrate units

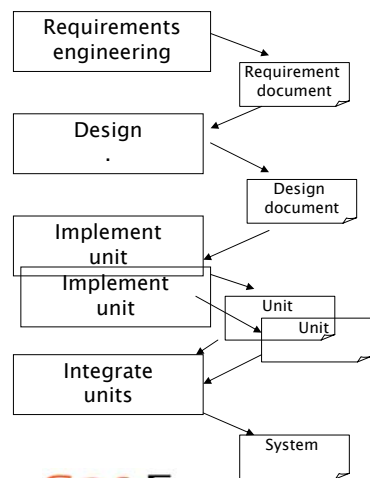
Logical dependencies



The production activities (2)

- Logically, each activity depends on the previous ones
 - ♦ To design, one must know the requirements
 - ♦ To implement, one must know the design and the requirements
- First approach is to do these activities in sequence
 - ♦ See waterfall model later
- In practice feedbacks and recycles must be provided
- Requirements and design are written down in documents

Production activities

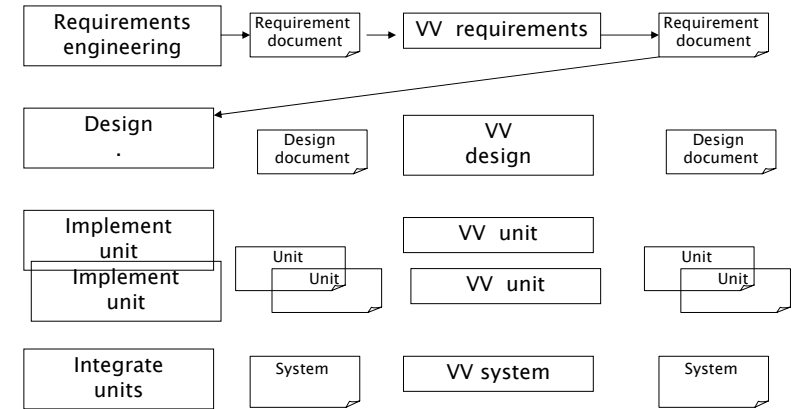


- Ok, we did it
 - ♦ Does it work?
 - ♦ Is it doing what it should do?
 - Or
 - ♦ Did we understand the requirements correctly?
 - ♦ Did we implement the requirements correctly?

The V & V activities

- V & V = verification and validation
- Control that the requirements are correct
 - ♦ Externally: did we understand what the customer/user wants?
 - ♦ Internally: is the document consistent?
- Control that the design is correct
 - ♦ Externally: is the design capable of supporting the requirements
 - ♦ Internally: is the design consistent?
- Control that the code is correct
 - ♦ Externally: is the code capable of supporting the requirements and the design?
 - ♦ Internally: is the code consistent (syntactic checks)

Production + VV activities

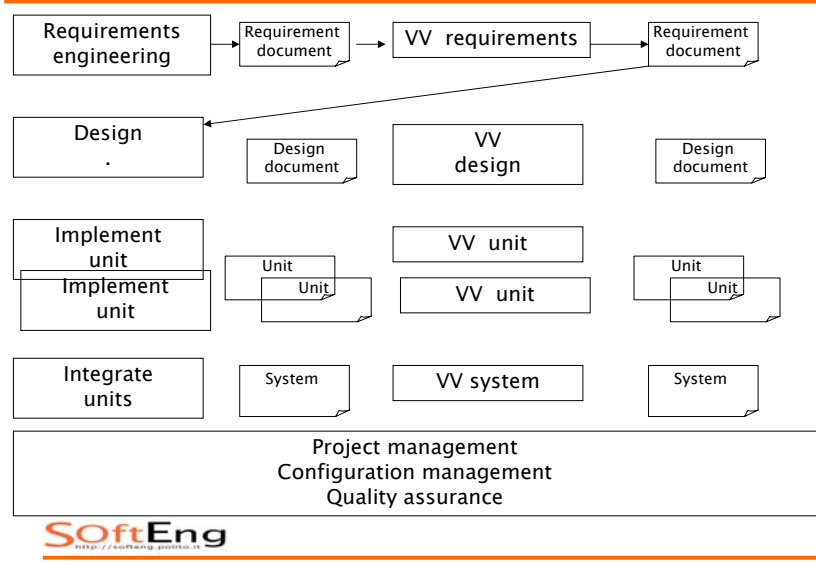


- Well, seems a lot of work
 - ♦ Who does what, when?
 - ♦ With what resources?
 - ♦ How much will it cost, when will we finish?
 - ♦ Where are the documents and units? Who can modify what?
 - ♦ Are we doing it state of the art?

The management activities

- Project management
 - ♦ Assign work and monitor progress
 - ♦ Estimate and control budget
- Configuration management
 - ♦ Identify, store documents and units
 - ♦ Keep track of relationships and history
- Quality assurance
 - ♦ Define quality goals
 - ♦ Define how work will be done
 - ♦ Control results

The whole picture



The whole picture

- Not new
- Just applying engineering approach to software production
- What do aeronautics engineers do?

Production + test activities

- ◆ Requirement definition (“what”)
 - airplane, civil usage
 - capacity > 400 people
 - range > 12000km,
 - Noise level < xdB, consumption < .., acquisition cost < y\$, operation cost < w \$/year
- ◆ high level design (“how”)
 - Blueprints of the airplane
 - Definition of subsystems
 - Avionics, structure, engines
 - Mathematical models
 - Structural (wings and frame)
 - Thermodynamic (engines)

- ◆ low level design
 - Further definition of subsystems
 - In several cases subcontracted or acquired (engine)
- ◆ implementation
 - Implementation of each subsystem
- ◆ unit test
 - Verification that subsystem complies to its specification

- ◆ Integration
 - Put subsystems together (ex. wing + frame)
- ◆ Integration test
 - Test the assemblies
- ◆ Acceptance test
 - Does it fly?
- ◆ Certification
 - FAA or other tests that it flies and issues a certificate
 - (a defined and long list of checks)

Management activities

- ◆ project management
 - project planning
 - project tracking
 - budgeting, accounting
- ◆ configuration management
 - Parts and assemblies
 - change control
- ◆ Quality management
 - Quality handbook
 - Quality plan
 - roles

Is there a difference?

Traditional engineering	Software engineering
<ul style="list-style-type: none"> ▪ Hundreds year old ▪ Theory from physics or other hard science, laws and mathematical models ▪ Maturity of customers and managers 	<ul style="list-style-type: none"> ▪ 50 years old ▪ Limited theories and laws. More a social science? ▪ Variable maturity of customers and managers

SE in one slide

- Activities
 - ◆ Production, VV, management
- Documents (and code)
 - ◆ To share and control information, decisions
- Techniques
 - ◆ To support activities
- Languages
 - ◆ To write documents (UML), code
- Models
 - ◆ To guide, support activities and the whole
 - ◆ CMM and CMM-I, ISO 9000-3, ISO 15504, ISO 12207, ISO 9126, IEEE, ..

Three basic approaches to SE

- Cow boy programming
Just code, all the rest is time lost and real programmers don't do it
- 1. Document based, semiformal, UML
Semiformal language for documents (UML), hand (human) based transformations and controls
- 2. Formal/model based
Formal languages for documents, automatic transformations and controls
- 3. Agile
Limited use of documents

Approaches, diffusion

- Cow boy programming
Not un-applied ..
- 1. Document based, semiformal, UML
Standard industrial practice, especially on large projects and mature companies/domains
- 2. Formal
Limited application in critical domains, small part of projects, does not scale up in large projects
- 3. Agile
Latest approach, debated, limited but increasing usage

Approaches

- This course is focused on approach 1
- Specific lectures on approach 2 and 3

A more detailed model of activities and documents

Activities

The engineering of software is organized in three levels of activities:

- **application engineering:** understanding the expectations of customers and/or users for a software system to be developed and translating these expectations into precise requirements for the software development team as well as deciding on the acceptability of the software system delivered by the development team based on these requirements
- **system engineering:** developing a solution for given requirements based on accepted well-specified software units as well as validating software systems constructed from such units
- **unit engineering:** developing software units according to specification

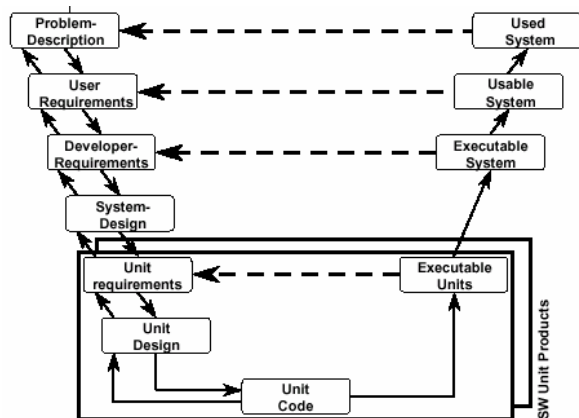
Activities

This separation helps to distinguish between the fundamentally different challenges for these three levels. In industry, these different levels can be performed by one company or distributed across different companies

Example

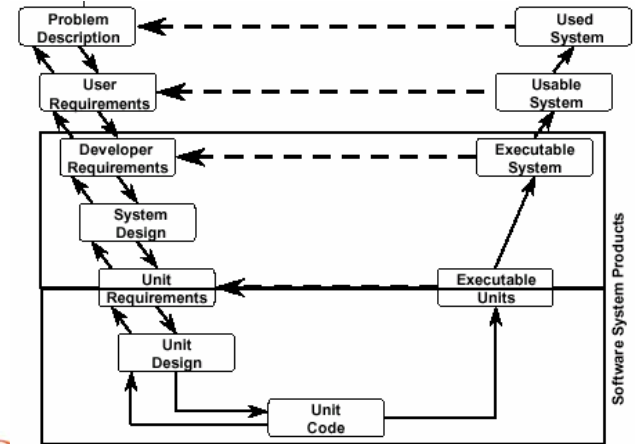
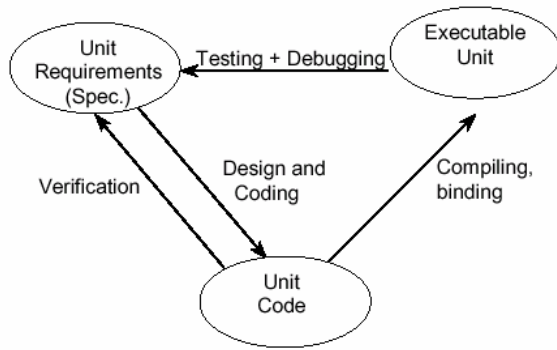
- Telecom Italia Mobile operates mobile networks (customer/user)
- IT Telecom performs application engineering for network monitoring;
- CSC performs the system engineering and integrates units developed by specialized software companies
- Nokia networks develops one unit, Ericsson another unit

Software unit engineering



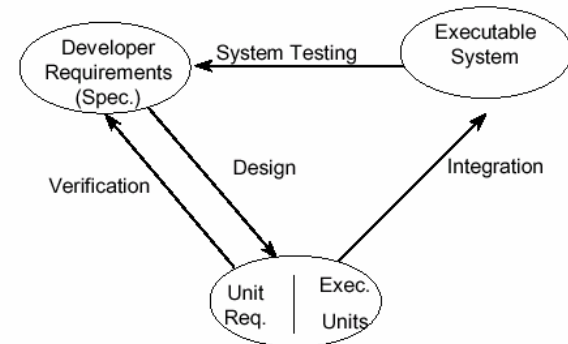
Definition

- **Input:** unit requirements = program specification
- **Goal:** the creation of a unit (e.g. small programs), consistent with the requirements, by means of a single person
- **Properties:** development according to well-defined techniques, methods and tools
 - ♦ compiling, linking, testing, debugging...
 - ♦ verification against unit designs
 - ♦ validation against unit requirements
- **Output:** verified and validated, executable unit

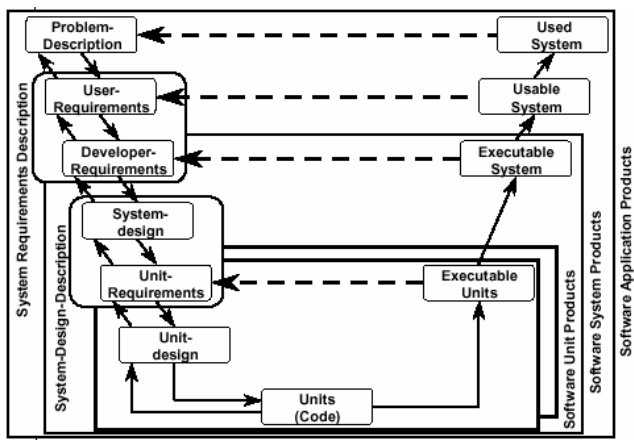


Definition

- **Input:** developer requirements
- **Goal:** development of a system or unit from the developer requirements
- **Properties:**
 - higher complexity due to scale of the system
 - more focus on teams
 - verification of system design against developer reqmts
 - verification of unit requirements against system design
 - validation against developer requirements
- **Output:** executable program or system

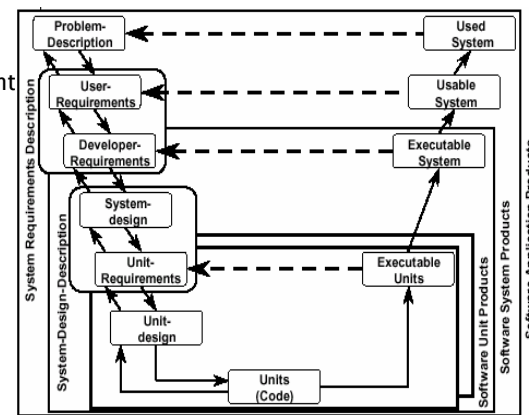


Virtual Product Model



Documents in VPM

- Problem description
- User requirements
- Developer requirement
- System design
- Unit requirements
- Unit design and code
- Executable units
- Executable system
- Usable system
- Used system



Problem description

- Definition of the problem to be solved
- Can be based on
 - ♦ market analysis
 - ♦ user interviews
 - ♦ feedback from users

User requirements

- Definition of the functional and non-functional requirements from the perspective of the user
- Facilitate communication between users and developers
- Defines the behavior of the system/ software and the interface to the users
- Verified against the problem description

Developer requirements

- **Definition of the functional and non-functional requirements from the perspective of the designer and/or maintainer**
- **Facilitates communication within the design and/or maintenance team**
 - ♦ often more formal than user requirements
- **Typically derived from the user requirements in order to ensure traceability**
- **Verified against the user requirements**

System design

- **Decomposition of the system into units**
 - ♦ Unit requirements defined in terms of exported features, imported features and behavior
- **Question: How can the system be built in a way that satisfies the developer requirements?**
- **Generally speaking**
 - ♦ requirements documents describe the system in terms of the problem
 - ♦ design documents describe system in terms of the solution
- **Verified against developer requirements**

Unit requirements

- **Units are logical, semi-autonomous parts of a system**
- **Come in various forms**
 - ♦ functions
 - ♦ data
 - ♦ abstract data types
 - ♦ classes
 - ♦ components
 - ♦ subsystems
- **Unit requirements describe the interface of the unit.**

Unit design and code

- **Involves design of internal data structures and algorithms**
- **Verified against unit requirements**
- **Through “step-wise refinement”, the realization of a unit is described in ever more detail until an executable form is reached**
- **Verified against unit design**

Executable units

- Created from unit implementations by the usual process of compilation and binding etc.
- May involve the simulation of missing units
- Validated through tests against unit requirements

Executable system

- Constructed from executable units according to system design
- May be constructed recursively (subsystem, system)
- Validated through tests against developer requirements

Usable system

- Involves the installation and adaptation of the system to the environment of the customer
- Validated against the user requirements (testing based on realistic usage scenarios)

Used system

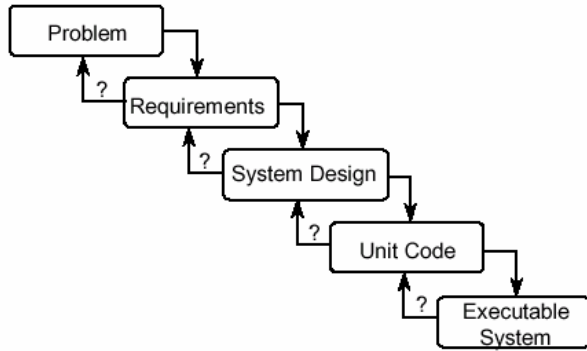
- System in actual use
- “Validated” against the problem description by customer’s use of the system
- Experience with the performance of the system serves as the foundation for improvement.

Process models

Process models

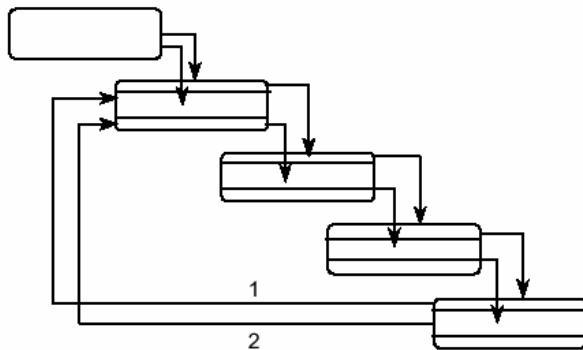
- We have introduced activities and documents
- They can be executed in a variety of ways
- Process models define temporal guidelines on activities
 - ◆ Waterfall
 - ◆ Iterative
 - ◆ Prototyping

Waterfall



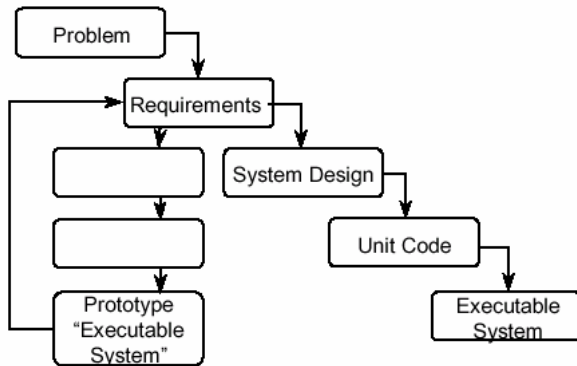
- Starting from the description of the problem, all development steps are performed sequentially
- Pre-requisites
 - ♦ requirements must be complete, stable and understood
 - ♦ teams needs experience with development technology
- Advantages
 - ♦ no coordination or integration problems
- Disadvantages
 - ♦ subsequent change of requirement very difficult
 - ♦ learning from experience within a project is difficult

Iterative



- Starting from a partitioned problem description, development steps are applied sequentially to each part until the whole system has been incrementally developed
- Pre-requisites
 - ♦ partitionable requirements
- Advantages
 - ♦ facilitates learning within projects
 - ♦ early access to product parts for customers
- Disadvantages
 - ♦ system integration can be problematic

Prototyping

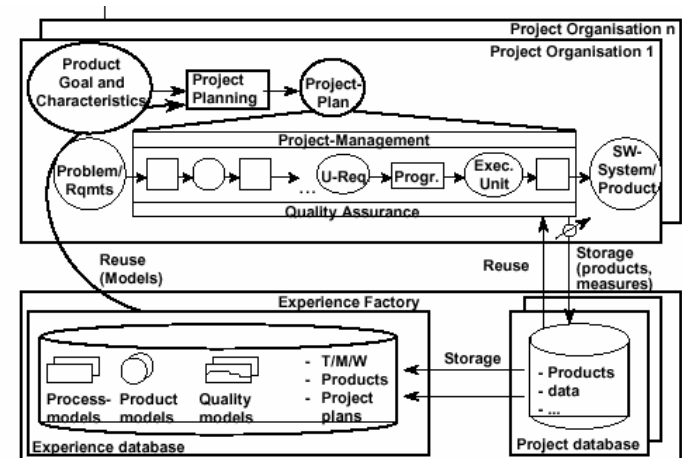


- Starting from the problem description, an executable system is created to validate unclear or complex aspects as quickly as possible
- Pre-requisites
 - knowledge of the critical aspects of the system reqmts
 - technology support for rapid prototype development
- Advantages
 - requirements can be clarified before development begins
- Disadvantages
 - danger that prototype is regarded as the final product

Projects and process model

- In practice
 - Each project in a company can follow a specific process model

Complete model



- Software Engineering (SE) is the sub-discipline of computer science
 - ♦ *defining models, techniques, methods and tools to support the development of large software systems based on sound engineering principles*
 - ♦ *defining models, techniques, methods and tools to manage software development projects and organizations*
 - ♦ *empirically evaluating the effectiveness of models, techniques, methods and tools in specific contexts*

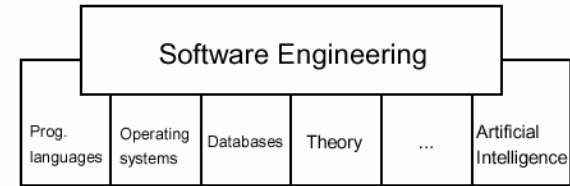
1.6 SE: the Discipline

SE

- *A discipline that deals with the building of software systems which are so large that they are built by a team or teams of engineers [Ghezzi, Jazayeri, Mandrioli]*
- *Multi person construction of multi version software [Parnas]*

- *A discipline whose aim is the production of fault-free software, delivered on time and within budget, that satisfies the user needs. Furthermore, the software must be easy to modify when the user needs change. [Schach]*

- *Software engineering is an engineering discipline that is concerned with all aspects of software production. Software engineers should adopt a systematic and organised approach to their work and use appropriate tools and techniques depending on the problem to be solved, the development constraints and the resources available. [Sommerville]*



- Software engineering builds on the foundations of other computer science disciplines
- but has also influenced their development
 - ♦ strong links in both directions

▪ Programming languages

- formal languages to describe reqmts and designs
- modularity concepts in new programming languages (e.g. Modula, C++, Ada)

▪ Operating Systems

- first experience with large systems (principles such as virtual machines, layers ...)
- new operating systems (e.g. UNIX) contain simple development environments

▪ Databases

- manipulation of complex data structures
- SE- data base technologies (OO)

▪ Theory

- FSM-model for specification and verification
- theory of abstract data types, reliability models

▪ Artificial Intelligence

- Explorative Processes (e.g. Prolog for prototyping)
- Expert systems – provide practical SE assistance (ie. “Development Assistants”)

The beginnings

- 1940's - **individual use of computers for solo-programming**
 - development of the electronic computer
 - no operating systems -> programs loaded from punch cards
 - typically used for numerical applications
- 1950's - **use of multiple computer-supported applications**
 - single-user operating systems
 - high-level programming languages (e.g. Fortran, Cobol)

SoftEng
<http://softeng.polito.it>

1.7 SE: the History

SoftEng
<http://softeng.polito.it>

- 1960's - **extensive use of computer supported solutions**
 - ♦ more powerful computers, more cheaply available
 - ♦ multi-programmer operating systems
 - ♦ applications becoming more numerous, complex, critical

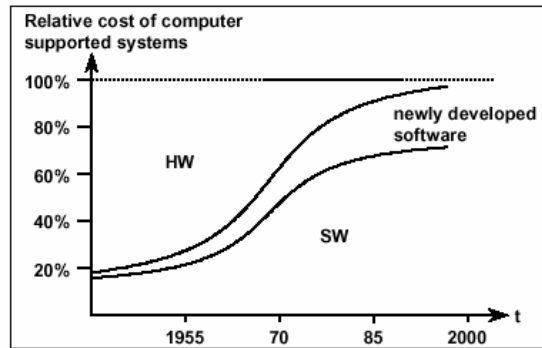
SoftEng
<http://softeng.polito.it>

1968, software crisis

- ♦ term coined at NATO conference in Garmish Partenkirchen
 - software crisis: the problem
 - SE: the solution

SoftEng
<http://softeng.polito.it>

Economic importance of software



(Boehm 1976)

1968 – today

- Economic importance of software
 - ♦ absolute software costs (estimated)
 - ♦ 1985 : \$140 billion
 - ♦ 1995 : \$450 billion
 - ♦ 2000 : \$3000 billion
- Causes:
 - ♦ growing expectations on software systems
 - ♦ many new technologies (e.g. languages) and tools for software development
 - ♦ no qualitative break through
 - from art form to engineering discipline

Trends – development

- Component based SE
 - ♦ Buy + integrate vs. build
 - ♦ Open source or commercial
- Offshoring
- Outsourcing
- Agile

Trends – business models

- ASP – pay per use
 - ♦ software is run on the provider's machines. Users use it through a network (Internet or Extranet). Users pay for using the software rather than purchasing it. E.g., mySAP.com.
- Freeware and pro versions
 - ♦ a light version of the software is distributed free of charge. The professional version is charged. E.g., RealPlayer.
- Shareware: software is distributed freely to facilitate trial use. Users pay for it if they decide to keep it and use it. E.g., WinZIP.
- Adware: the software is free. The interface show advertisement banners refreshed via Internet. E.g., Eudora

Core concepts and definitions

- **Software Engineering**

- ♦ Consequent application of engineering principles to the development of software.

- **Engineering principles**

- ♦ Systematic procedure
 - product, process and organization are well structured
 - Quality (and other) goals are explicitly (and measurably) defined
 - Guided by engineering principles
- ♦ Adherence to existing body of knowledge, norms, quality/cost/time requirements
- ♦ Use of models
- ♦ Use of formal techniques as indicated by project requirements
- ♦ Use of empirical validation (e.g., testing) when formal models/techniques are lacking

- **Software Engineering (SE) is the sub-discipline of computer science**

- ♦ defining models, techniques, methods and tools to support the development of large software systems based on sound engineering principles
- ♦ defining models, techniques, methods and tools to manage software development projects and organizations
- ♦ empirically evaluating the effectiveness of models, techniques, methods and tools in specific contexts

- **Software Application Engineering (SW Engineering)**

- ♦ the engineering-based creation of a software system for solving an application problem using undefined techniques/methods/tools

- **Software System Engineering (SW Development)**

- ♦ the engineering based creation of a software system according to the defined (developer) requirements using defined techniques/methods/tools

- **Software Unit Engineering (Programming)**

- ♦ the engineering-based creation of a software unit according to the predefined (unit) requirements using predefined techniques/methods/tools

▪ Principles

- ♦ foundations guiding the approaches taken
- ♦ e.g. structuring (top-down, information hiding), documentation, problem solutions

▪ Techniques

- ♦ steps for describing a particular product or carrying out a particular process
- ♦ (e.g. black-box test techniques, programming languages)

▪ Methods

- ♦ planned application of established techniques for reaching predefined goals
- ♦ what, how and, under what circumstances how well
- ♦ e.g. test methods

▪ Tools

- ♦ computer support for techniques and methods

▪ Software (System/Unit)

- ♦ an executable program or unit

▪ (Software) Documentation

- ♦ atomic/complex information entities created during the course of an engineering-based development project (organized according to the Virtual Product Model)

▪ (Software) Product

- ♦ a software system, unit or document

▪ Product Model

- ♦ describes the characteristic of a class of products

▪ Software (Process)

- ♦ every atomic or complex activity related to the engineering-based creation of software (e.g. management, development, unit development, test data creation, waterfall process etc.)

▪ Process Model

- ♦ describes the characteristics of a class of processes

▪ Life Cycle Model

- ♦ A comprehensive process model including the set of processes needed for a type of project, augmented with information about the relationships among the individual processes

▪ Project Plan

- ♦ A life cycle model augmented with information about resource allocation, costs and other goals for a specific project

▪ Quality

- ♦ every attribute of a product or process
- ♦ e.g. reliability, effort

▪ Quality Model

- ♦ describes a quality property for classes of contexts
- ♦ e.g. effort distribution in a project to be developed by a particular firm according to the waterfall model.