

Introduction



Motivation

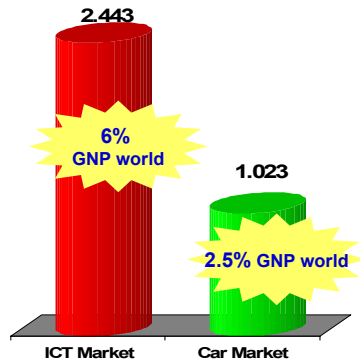
Summary

- 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

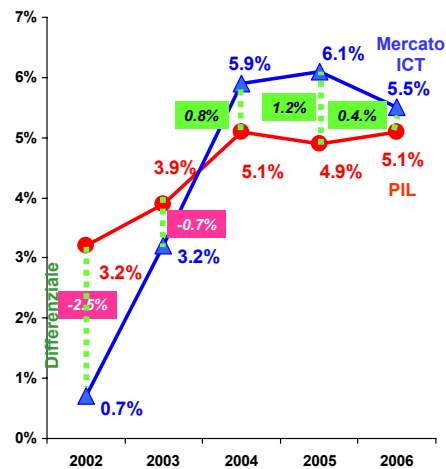
ICT Market - world - 2004



SoftEng

ICT - World market

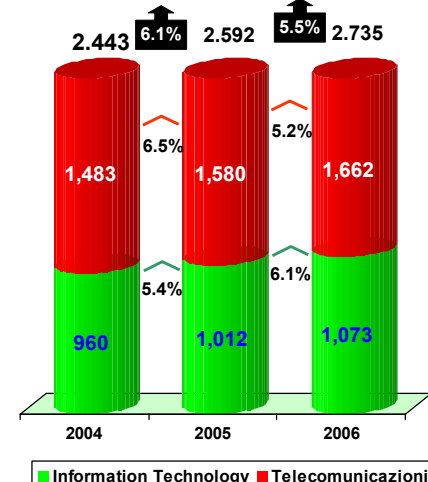
Variazioni % annue



Fonte: AlTech - Assinform / NetConsulting

SoftEng

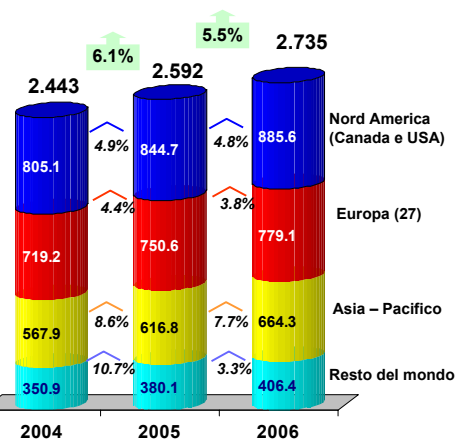
Valori in Mld\$ e variazioni % annue



Information Technology Telecomunicazioni

ICT market, per area

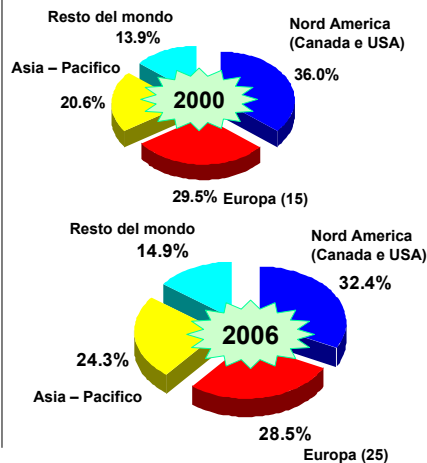
Valori in Mld\$ e variazioni % annue



Fonte: AlTech - Assinform / NetConsulting

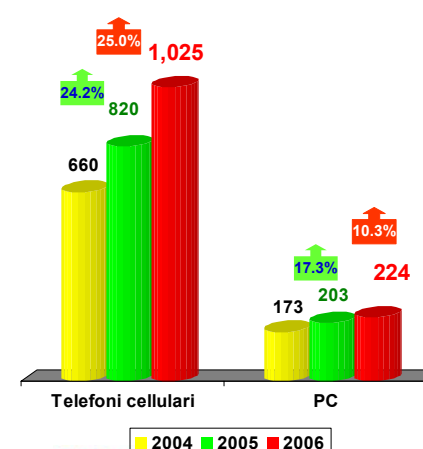
SoftEng

Quote % delle varie aree sul mercato mondiale dell'ICT



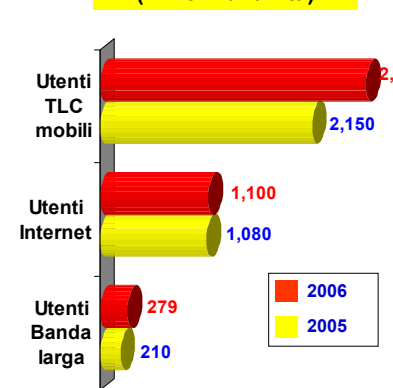
ICT Diffusion, world

Unità vendute nel mondo (2004 vs 2006 - milioni di unità)



SoftEng

Parco 2006 nel mondo (milioni di unità)

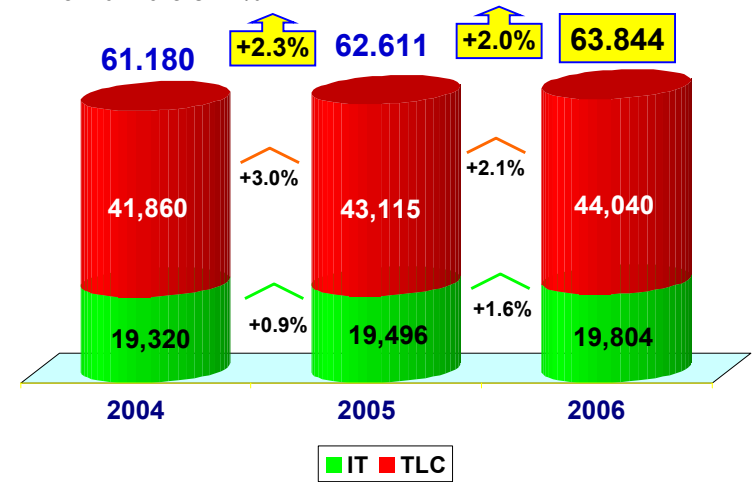


Fonte: AlTech - Assinform / NetConsulting

Italy

ICT Market

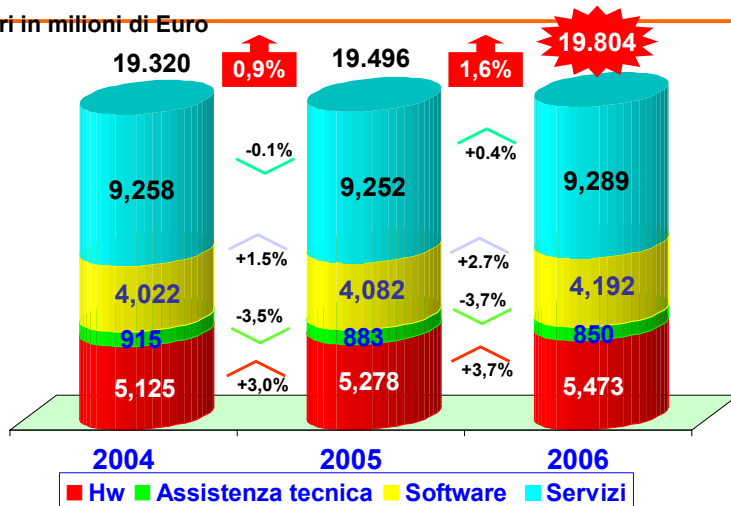
Valori in Milioni di Euro e in %



Fonte: AlTech - Assinform / NetConsulting

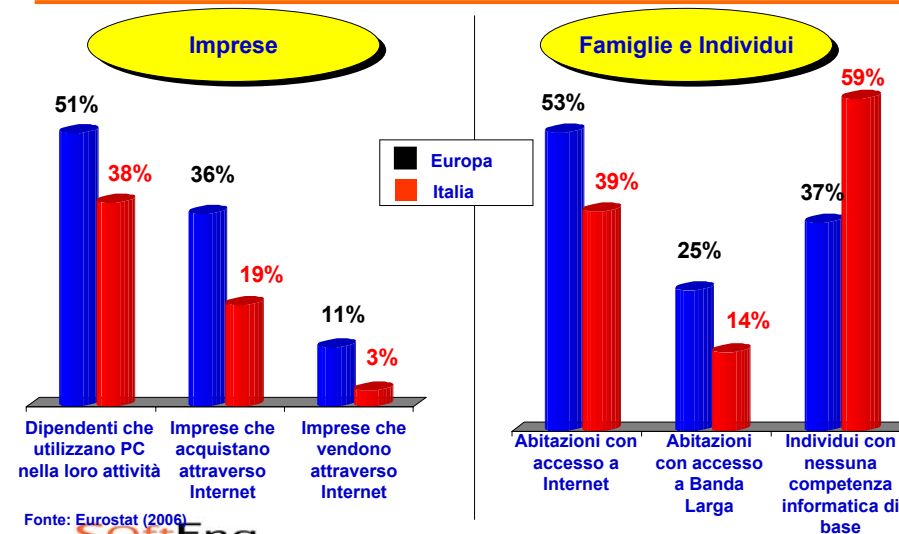
IT Market

Valori in milioni di Euro



Fonte: AlTech - Assinform / NetConsulting

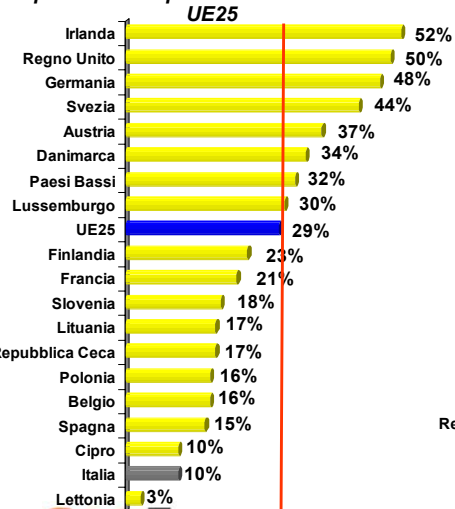
Diffusion



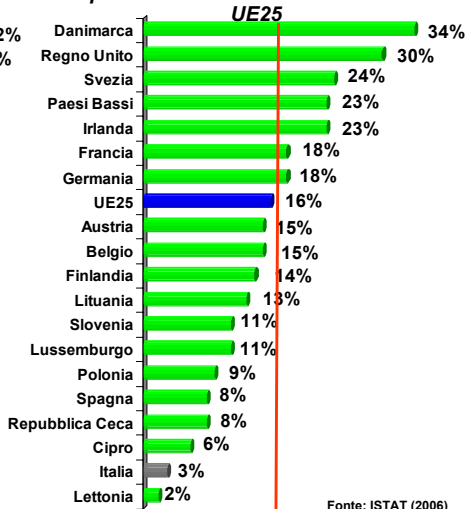
Fonte: Eurostat (2006)

Diffusion

Imprese che acquistano on-line nei Paesi UE25



Imprese che vendono on-line nei Paesi UE25



Fonte: ISTAT (2006)

IT (2006)

Valori in Euro

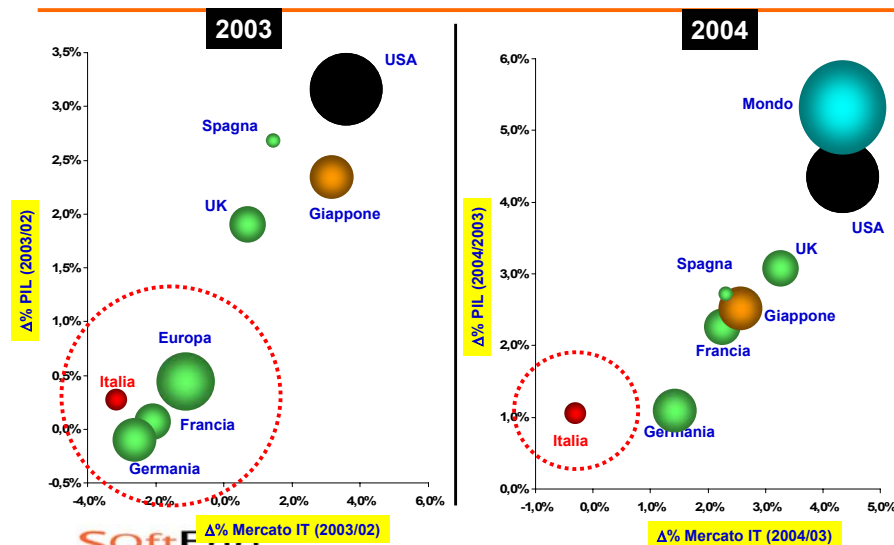
	Spesa IT / PIL	Spesa IT procapite	Spesa IT / occupato
USA	3,9%	1.408	2.945
Giappone	2,3%	878	1.765
Germania	3,1%	812	1.837
Regno Unito	3,1%	983	2.095
Francia	3,2%	839	2.050
Italia	1,9%	341	878
Spagna	1,9%	372	748

SoftEng

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

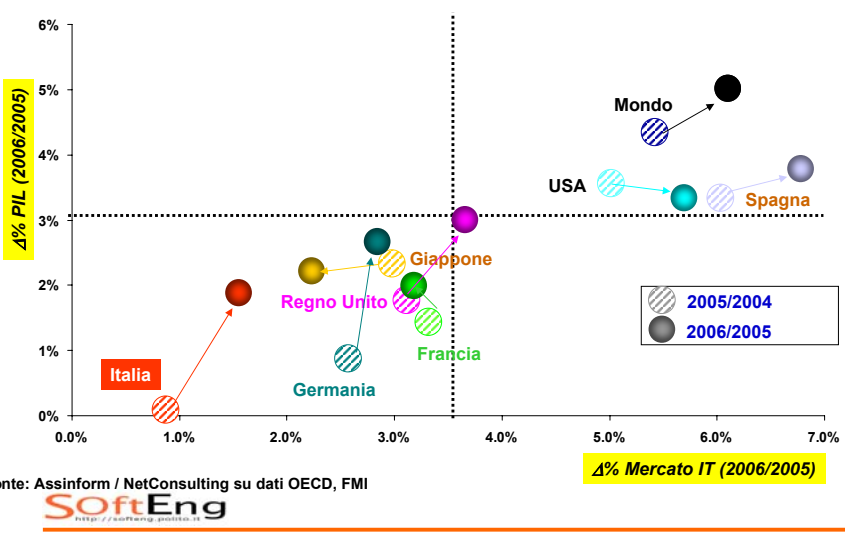


SoftEng

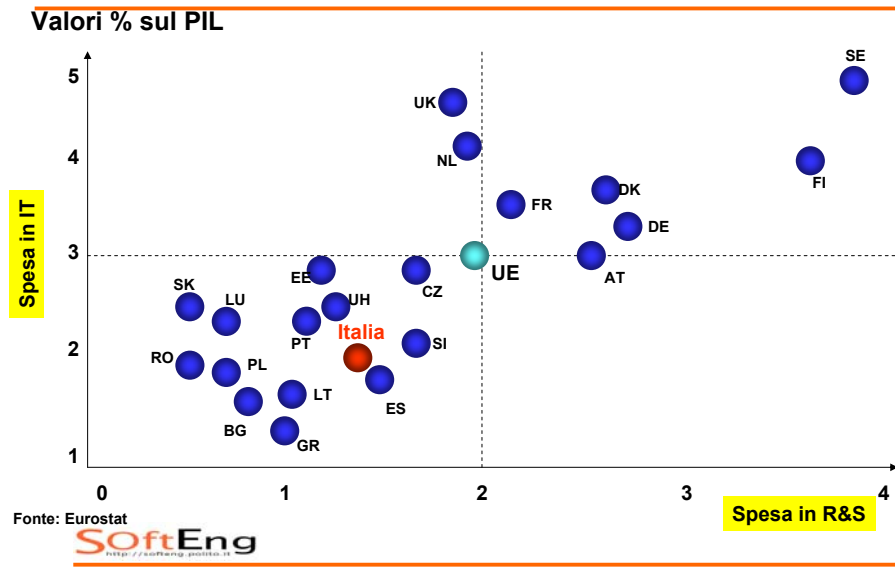
Fonte: Anisform / NetConsulting su dati OECD, EMI

SoftEng

Development - IT Investment



R&D investment vs. IT investment



WEF - ICT development of nations

1 - Denmark
Sweden
Singapore
Finland
Switzerland
Netherlands
United States
Iceland
United Kingdom
Norway
Canada
Hong Kong SAR
Taiwan, China
Japan
Australia
Germany
Austria
Israel
Korea, Rep.
Estonia
Ireland
New Zealand
France
Belgium

Ranking WEF
(world economic forum)
Global IT report 2006-2007
www.weforum.org

Malaysia
Malta
Portugal
United Arab Emirates
Slovenia
Chile
Spain
Hungary
Czech Republic
Tunisia
Qatar
Thailand
40 - Italy

WEF - ICT development

- ICT conducive environment
 - ♦ Regulatory aspects, soft + hard infrastructure
- ICT readiness
 - ♦ Individuals, business, government
- ICT usage
 - ♦ Individuals, business, government

WEF - Global competitiveness

1- Switzerland
Finland
Sweden
Denmark
Singapore
United States
Japan
Germany
Netherlands
United Kingdom
Hong Kong SAR
Norway
Taiwan, China
Iceland
Israel
Canada
Austria
France
Australia
Belgium
Ireland
Luxembourg
New Zealand
Korea, Rep.
Estonia

World Economic Forum- Global competitiveness report 2006-2007
www.weforum.org

Malaysia
Chile
Spain
Czech Republic
Tunisia
Barbados
United Arab Emirates
Slovenia
Portugal
Thailand
Latvia
Slovak Republic
Qatar
Malta
Lithuania
Hungary
45 - Italy

Global competitiveness

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

SoftEng
http://softeng.polito.it

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.

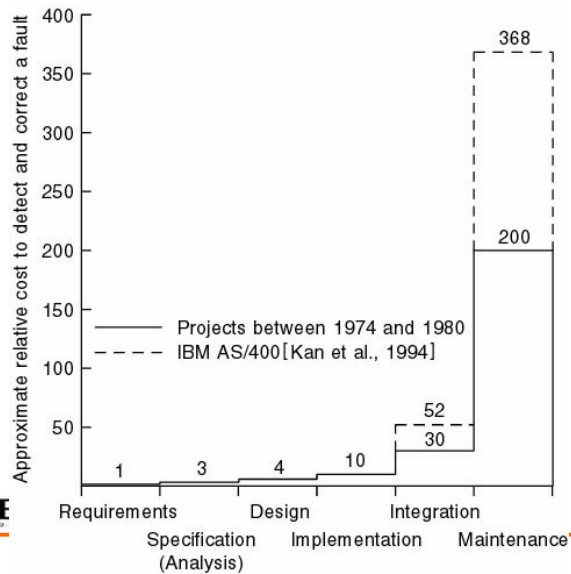
What are the costs of sw engineering?

- ◆ 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.

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 = 200 – 1000 LOC per person month
 - Personal cost = : \$ 8.000 per person month
 - \$8 to \$40 per LOC
- a medium sized project with 50.000 LOC costs between \$400.000 to \$1.600.000 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

Software can be perfect

- All software faults may not be removed before execution

Software is stable

- 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 ?)

Engineering – solo programming

- | | |
|---|--------------------------------------|
| ▪ 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 |

Engineering vs. Solo programming

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)

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

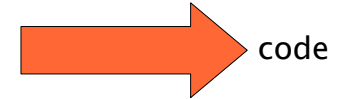
Software

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

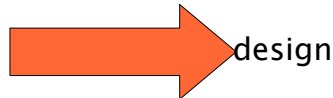
Getting there

From the bottom up

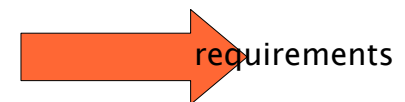
- We need the final thing
 - ♦ Executable code
- But we do not write the executable
 - ♦ Source code



-
- 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?



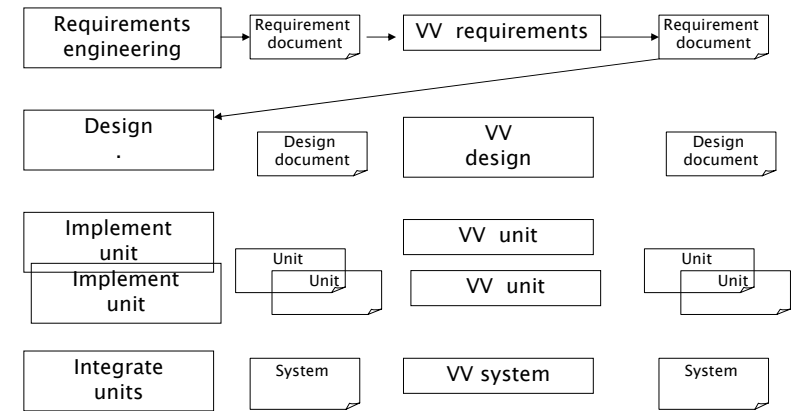
-
- But, exactly, what the software should do?
 - ♦ Add numbers, count cars, forecast weather, control mobile phone, support administration of company?



The 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

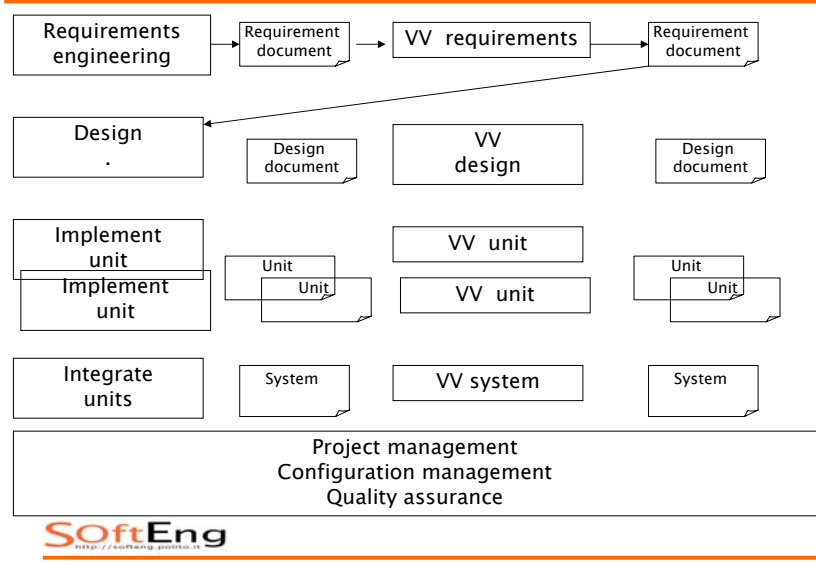


- Well, seems a lot of work
 - ♦ Who does what, when?
 - ♦ With what resources?
 - ♦ How much will it cost, when will we finish?
 - ♦ Where are the documents and units? Who can modify what?
 - ♦ Are we doing it state of the art?

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

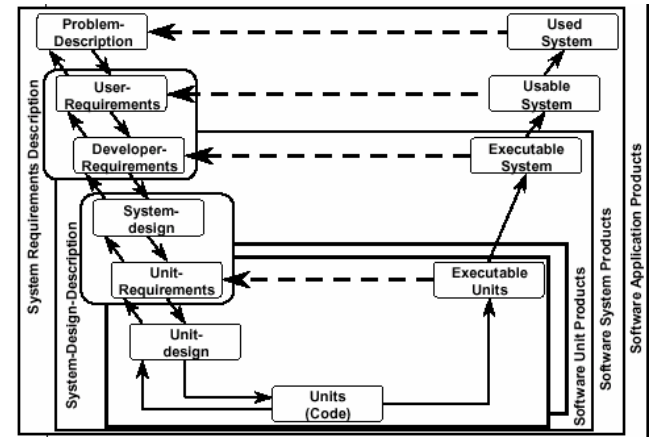
- This course is focused on approach 1
- Specific lectures on approach 2 and 3

A more detailed model of activities and documents

Documents: Product models

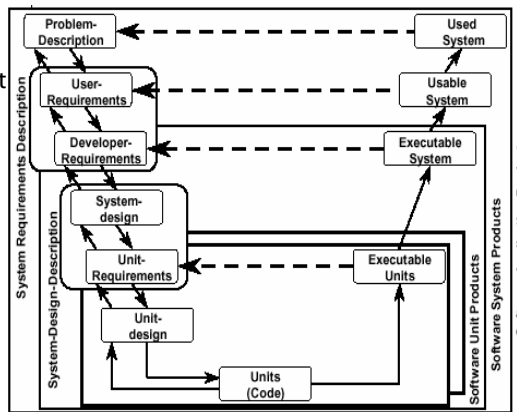
- The large variety of process or life cycle models differ according to the
 - ♦ number and type of resulting physical products (e.g. documents)
 - ♦ number, type and relationships of the development steps
- However, there are similarities in the number and type of information (contents OR virtual products) which are needed for the complete documentation of a system
 - ♦ the Virtual Product Model is the reference for all Real software product models in industry

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.

1.6 SE: the Discipline

SE – Definition, Rombach

- 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*

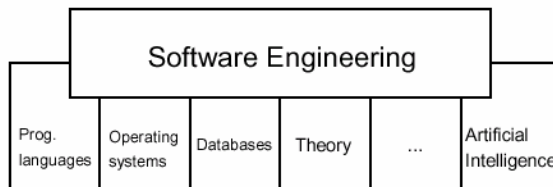
SE

- *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]*

SE and CS



- 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”)

1.7 SE: the History

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)

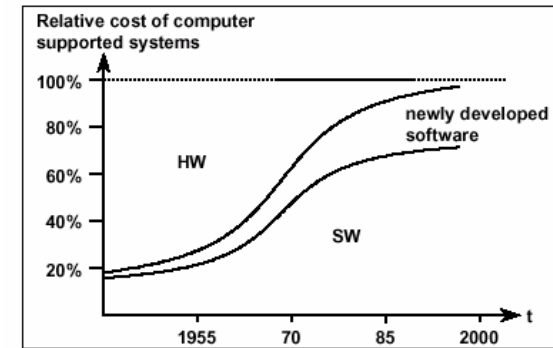
▪ 1960's - extensive use of computer supported solutions

- ♦ more powerful computers, more cheaply available
- ♦ multi-programmer operating systems
- ♦ applications becoming more numerous, complex, critical

1968, software crisis

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

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