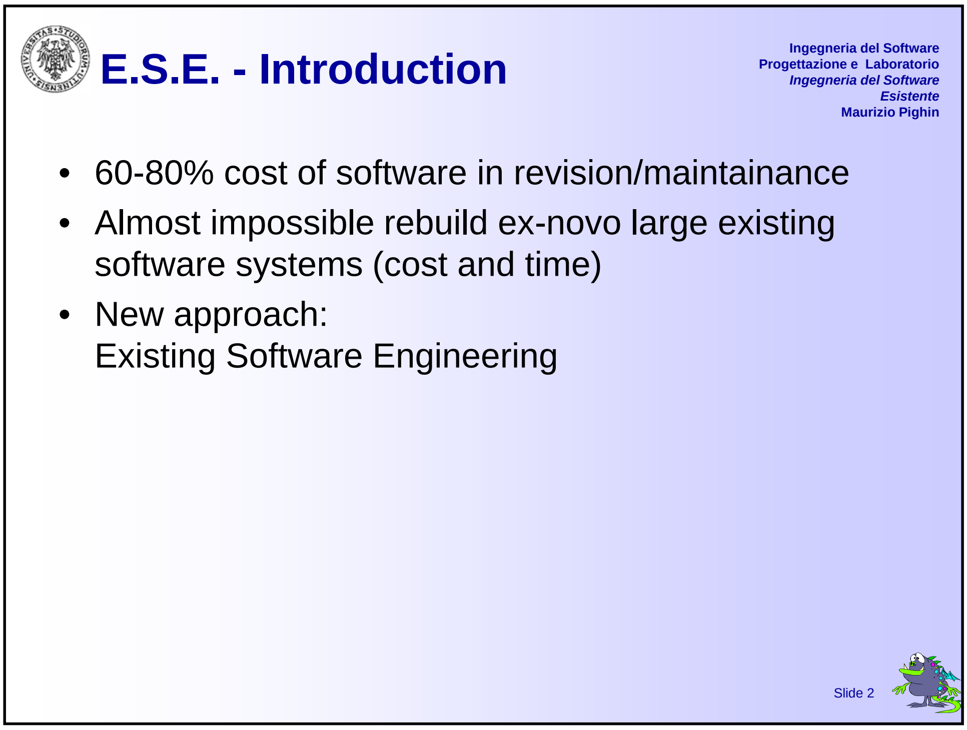
Università degli Studi di Udine

Ingegneria del Software Esistente

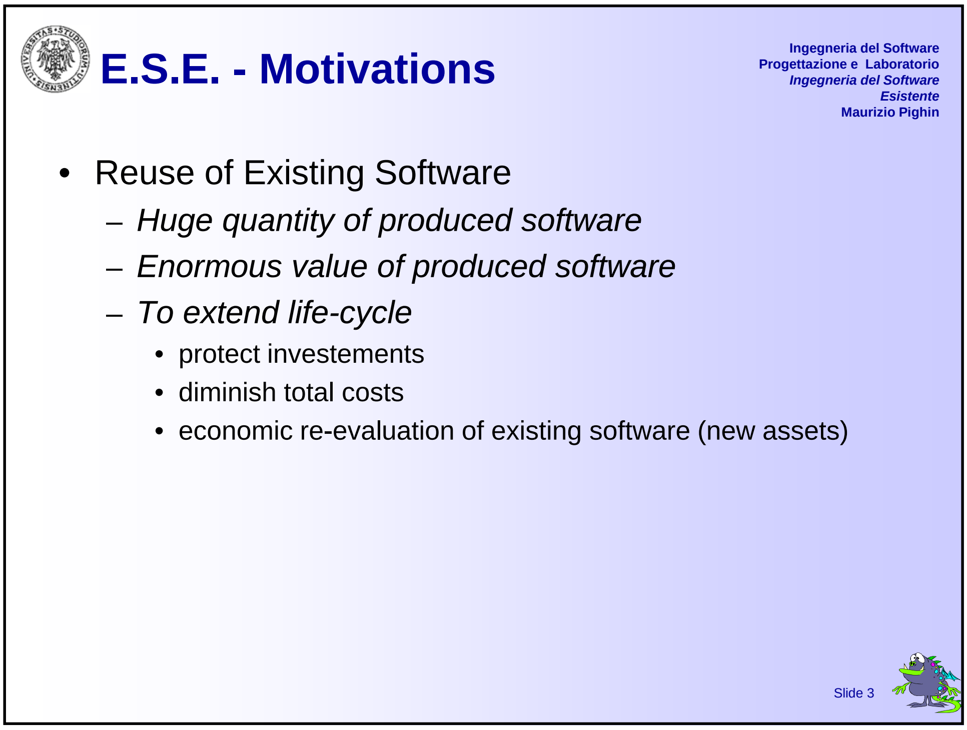
prof. Maurizio Pighin

Dipartimento di Matematica e Informatica



E.S.E. - Introduction

- 60-80% cost of software in revision/maintainance
- Almost impossible rebuild ex-novo large existing software systems (cost and time)
- New approach:
  Existing Software Engineering

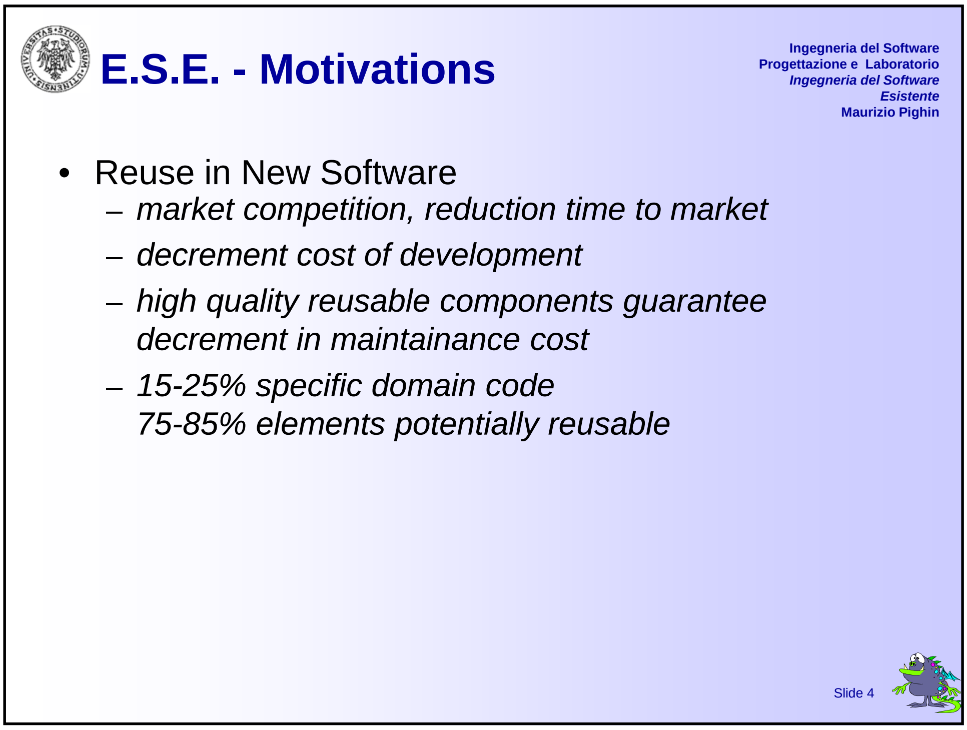


E.S.E. - Motivations

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

- Reuse of Existing Software
  - *Huge quantity of produced software*
  - *Enormous value of produced software*
  - *To extend life-cycle*
    - protect investements
    - diminish total costs
    - economic re-evaluation of existing software (new assets)

Slide 3

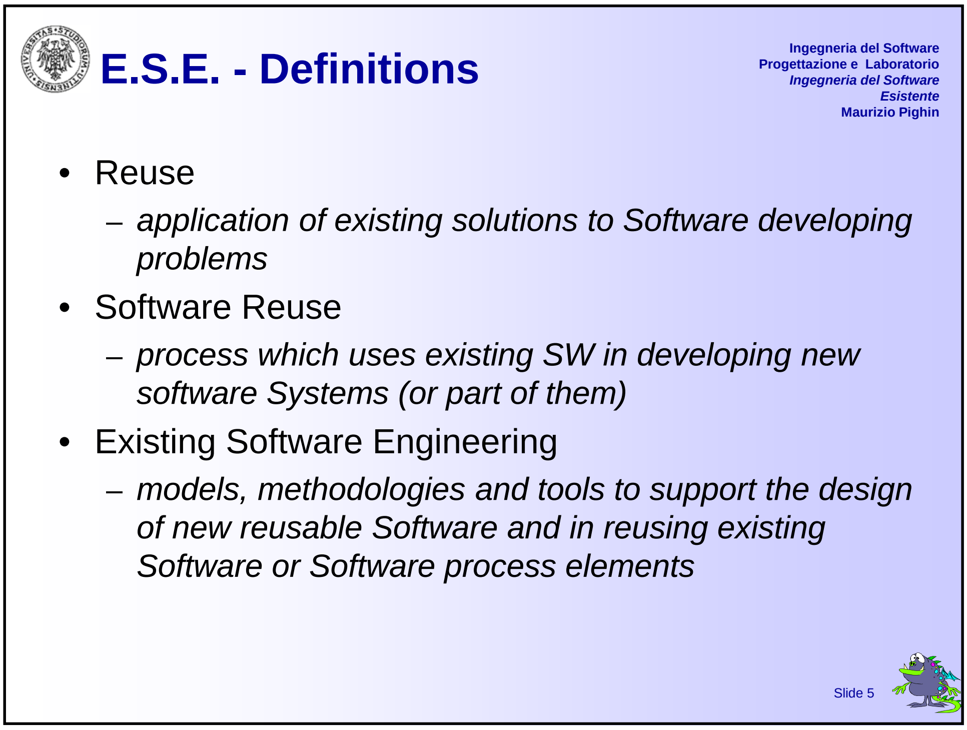


E.S.E. - Motivations

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

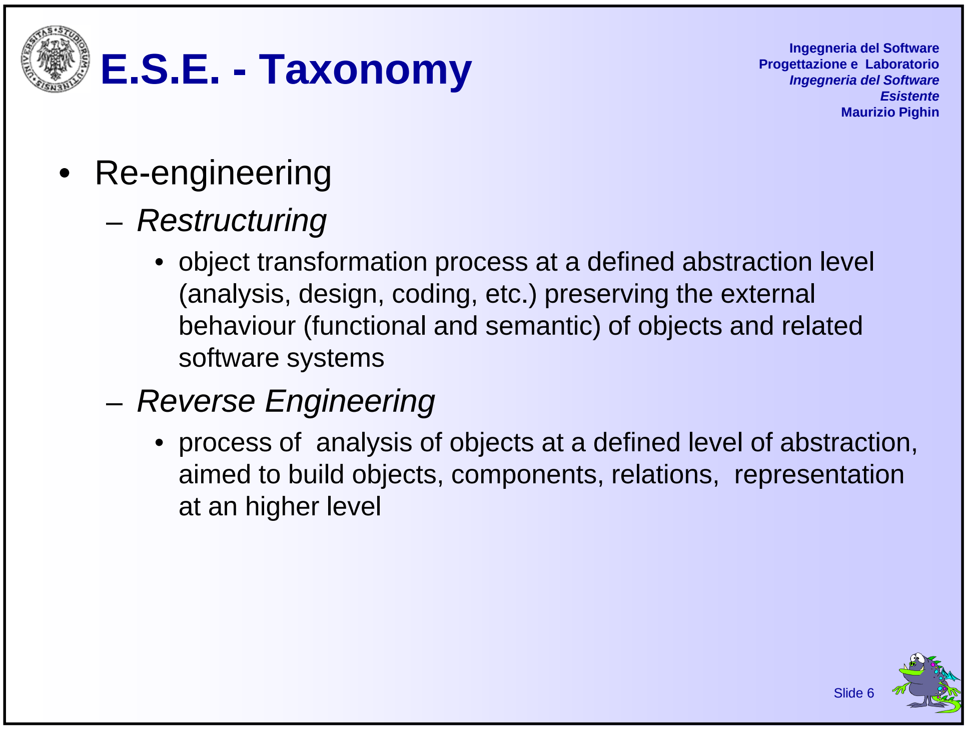
- Reuse in New Software
  - *market competition, reduction time to market*
  - *decrement cost of development*
  - *high quality reusable components guarantee decrement in maintainance cost*
  - *15-25% specific domain code*
    75-85% elements potentially reusable

Slide 4



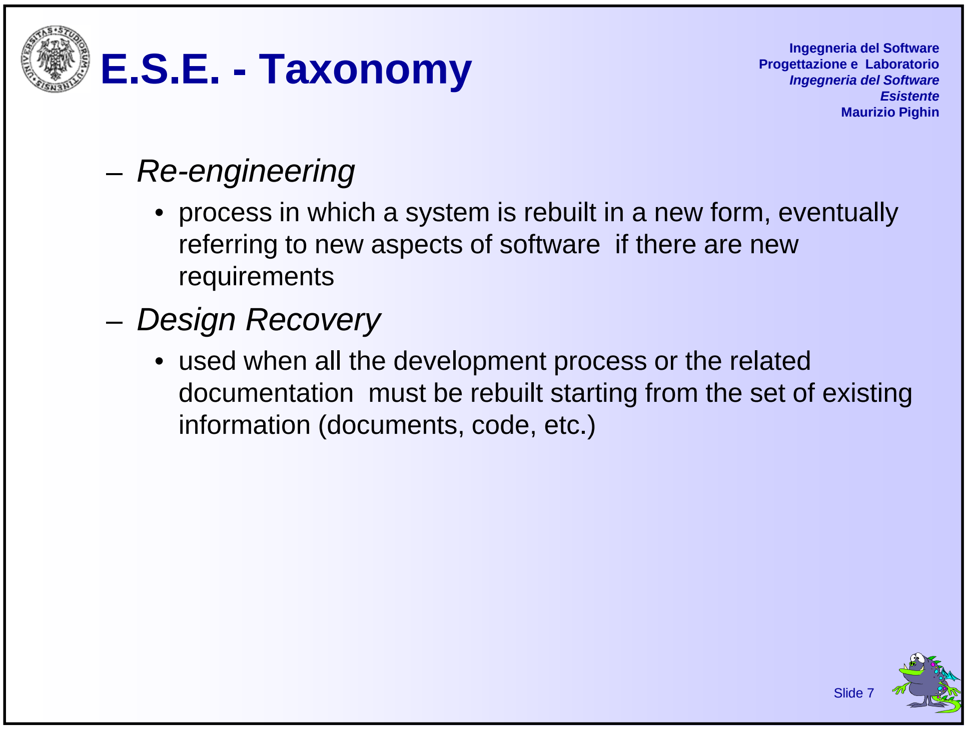
E.S.E. - Definitions

- Reuse
  - *application of existing solutions to Software developing problems*
- Software Reuse
  - *process which uses existing SW in developing new software Systems (or part of them)*
- Existing Software Engineering
  - *models, methodologies and tools to support the design of new reusable Software and in reusing existing Software or Software process elements*



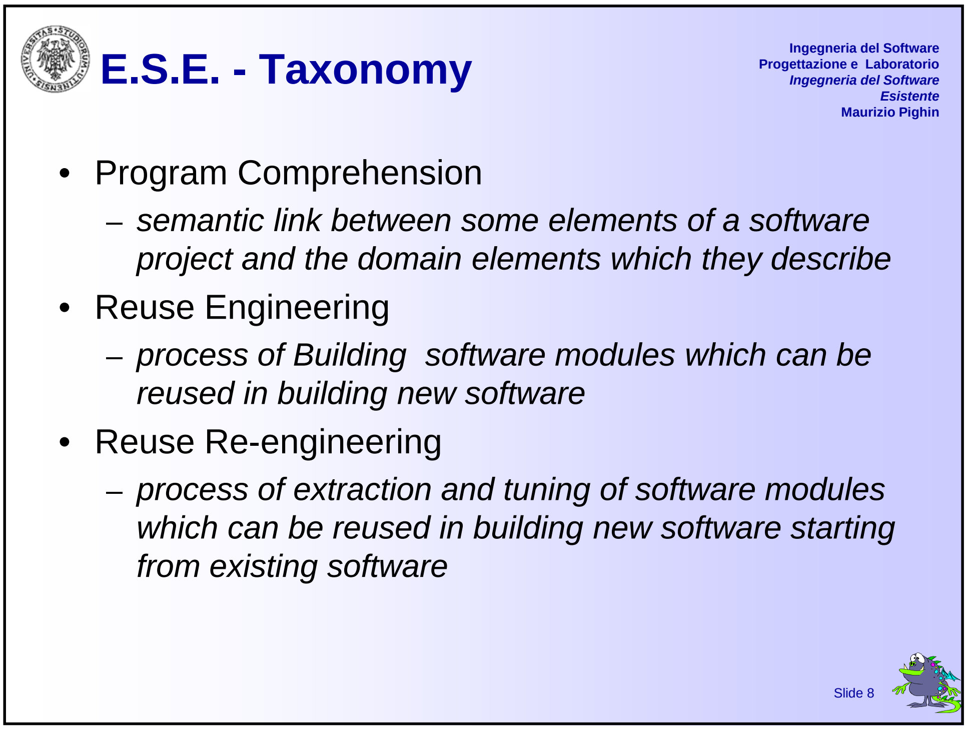
E.S.E. - Taxonomy

- Re-engineering
  - *Restructuring*
    - object transformation process at a defined abstraction level (analysis, design, coding, etc.) preserving the external behaviour (functional and semantic) of objects and related software systems
  - *Reverse Engineering*
    - process of analysis of objects at a defined level of abstraction, aimed to build objects, components, relations, representation at an higher level



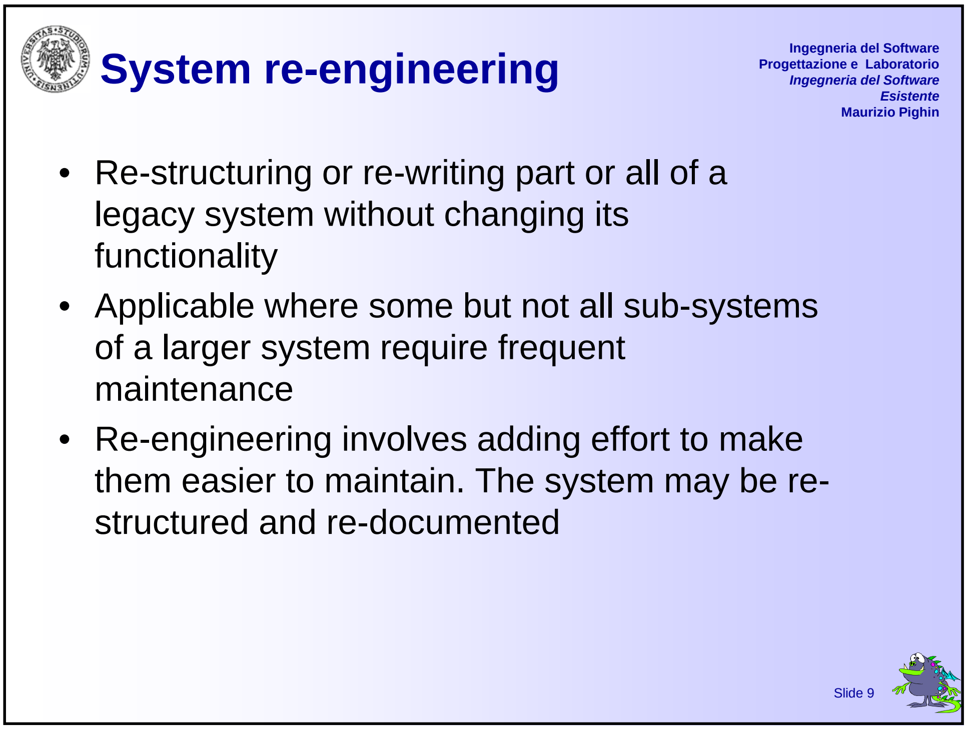
E.S.E. - Taxonomy

- *Re-engineering*
  - process in which a system is rebuilt in a new form, eventually referring to new aspects of software if there are new requirements
- *Design Recovery*
  - used when all the development process or the related documentation must be rebuilt starting from the set of existing information (documents, code, etc.)



E.S.E. - Taxonomy

- Program Comprehension
  - *semantic link between some elements of a software project and the domain elements which they describe*
- Reuse Engineering
  - *process of Building software modules which can be reused in building new software*
- Reuse Re-engineering
  - *process of extraction and tuning of software modules which can be reused in building new software starting from existing software*

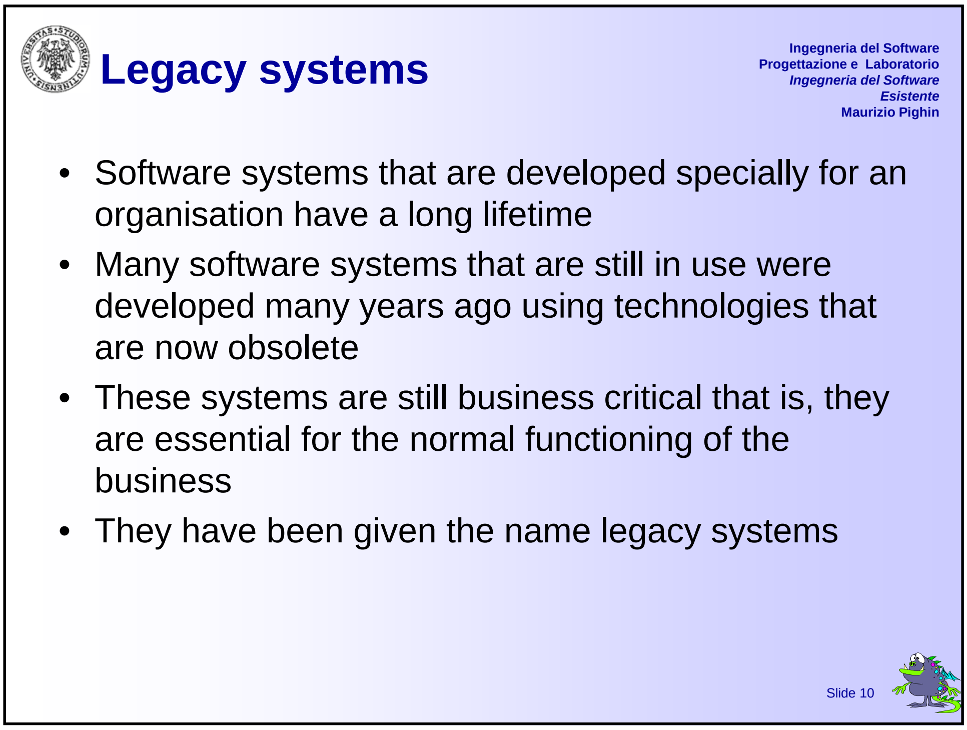


System re-engineering

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

- Re-structuring or re-writing part or all of a legacy system without changing its functionality
- Applicable where some but not all sub-systems of a larger system require frequent maintenance
- Re-engineering involves adding effort to make them easier to maintain. The system may be re-structured and re-documented

Slide 9

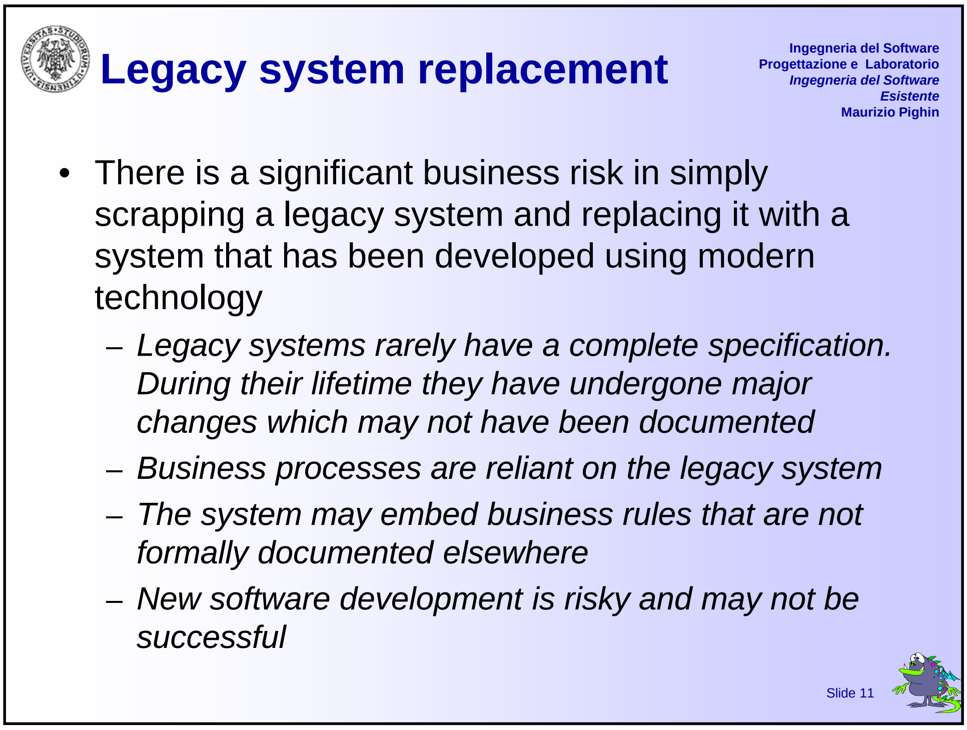


Legacy systems

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

- Software systems that are developed specially for an organisation have a long lifetime
- Many software systems that are still in use were developed many years ago using technologies that are now obsolete
- These systems are still business critical that is, they are essential for the normal functioning of the business
- They have been given the name legacy systems

Slide 10

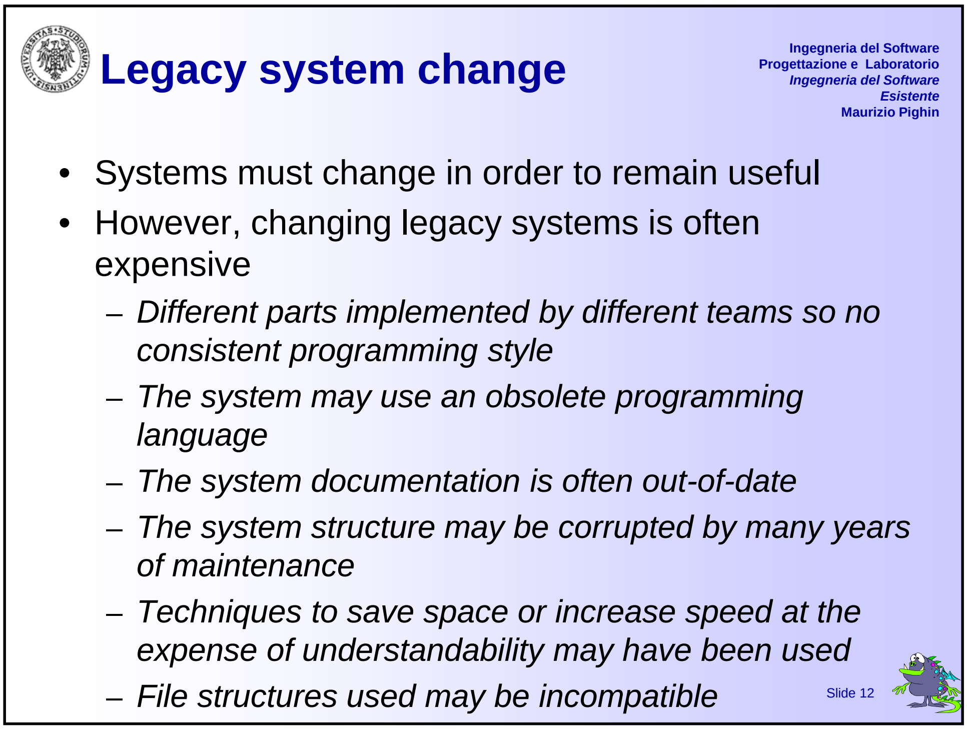


Legacy system replacement

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software Esistente
Maurizio Pighin

- There is a significant business risk in simply scrapping a legacy system and replacing it with a system that has been developed using modern technology
  - *Legacy systems rarely have a complete specification. During their lifetime they have undergone major changes which may not have been documented*
  - *Business processes are reliant on the legacy system*
  - *The system may embed business rules that are not formally documented elsewhere*
  - *New software development is risky and may not be successful*

Slide 11

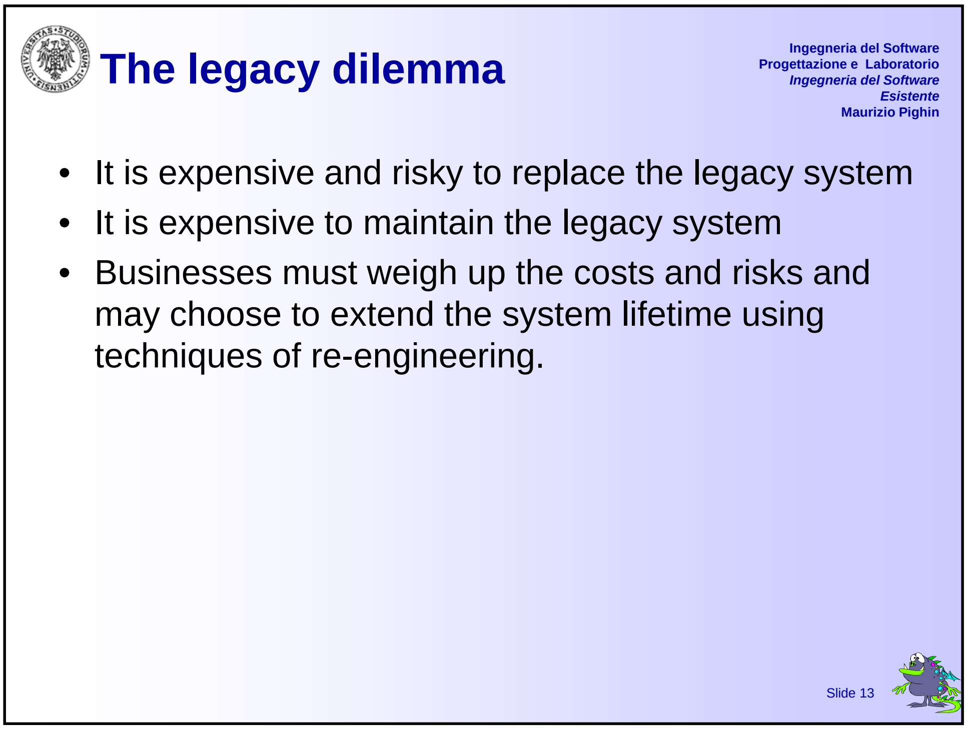


Legacy system change

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software Esistente
Maurizio Pighin

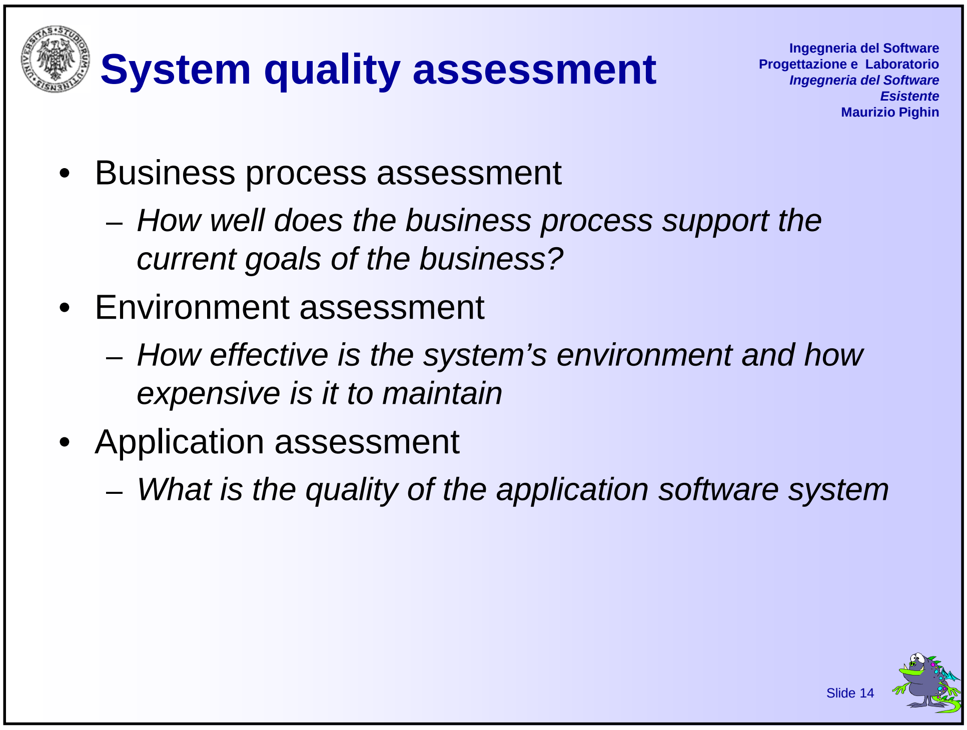
- Systems must change in order to remain useful
- However, changing legacy systems is often expensive
  - *Different parts implemented by different teams so no consistent programming style*
  - *The system may use an obsolete programming language*
  - *The system documentation is often out-of-date*
  - *The system structure may be corrupted by many years of maintenance*
  - *Techniques to save space or increase speed at the expense of understandability may have been used*
  - *File structures used may be incompatible*

Slide 12



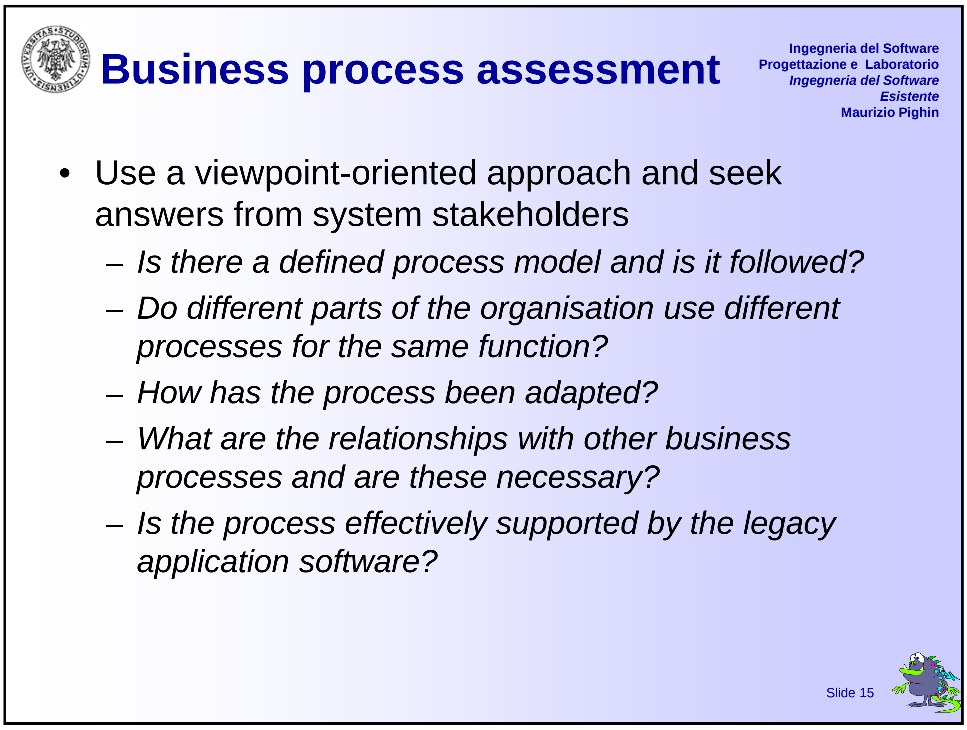
The legacy dilemma

- It is expensive and risky to replace the legacy system
- It is expensive to maintain the legacy system
- Businesses must weigh up the costs and risks and may choose to extend the system lifetime using techniques of re-engineering.



System quality assessment

- Business process assessment
  - *How well does the business process support the current goals of the business?*
- Environment assessment
  - *How effective is the system's environment and how expensive is it to maintain*
- Application assessment
  - *What is the quality of the application software system*

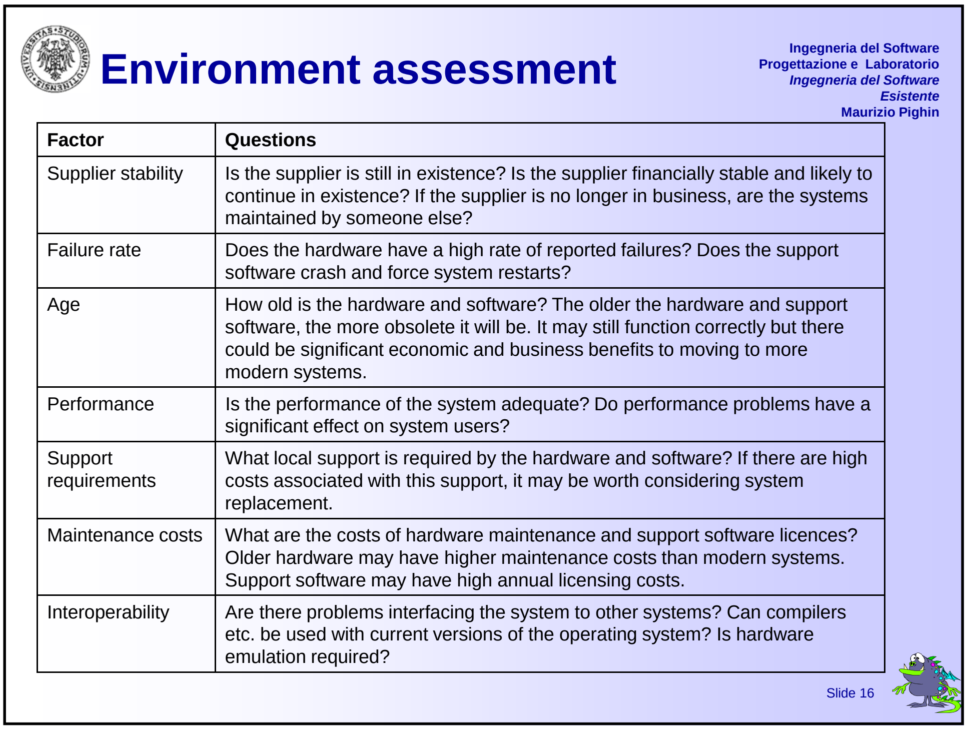


Business process assessment

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

- Use a viewpoint-oriented approach and seek answers from system stakeholders
  - *Is there a defined process model and is it followed?*
  - *Do different parts of the organisation use different processes for the same function?*
  - *How has the process been adapted?*
  - *What are the relationships with other business processes and are these necessary?*
  - *Is the process effectively supported by the legacy application software?*

Slide 15

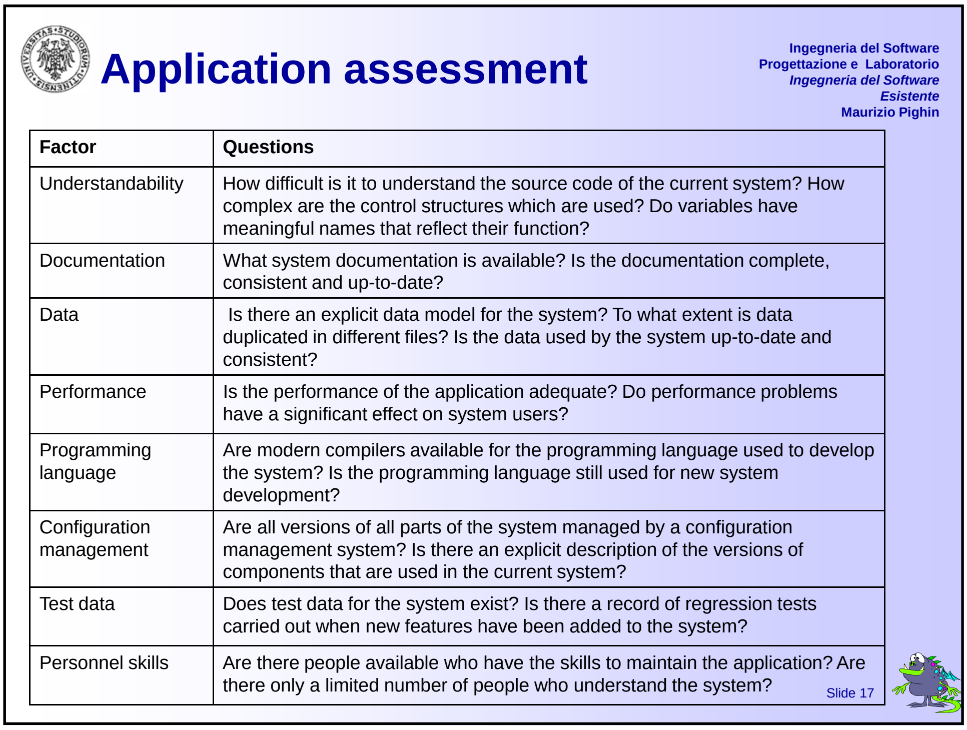


Environment assessment

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

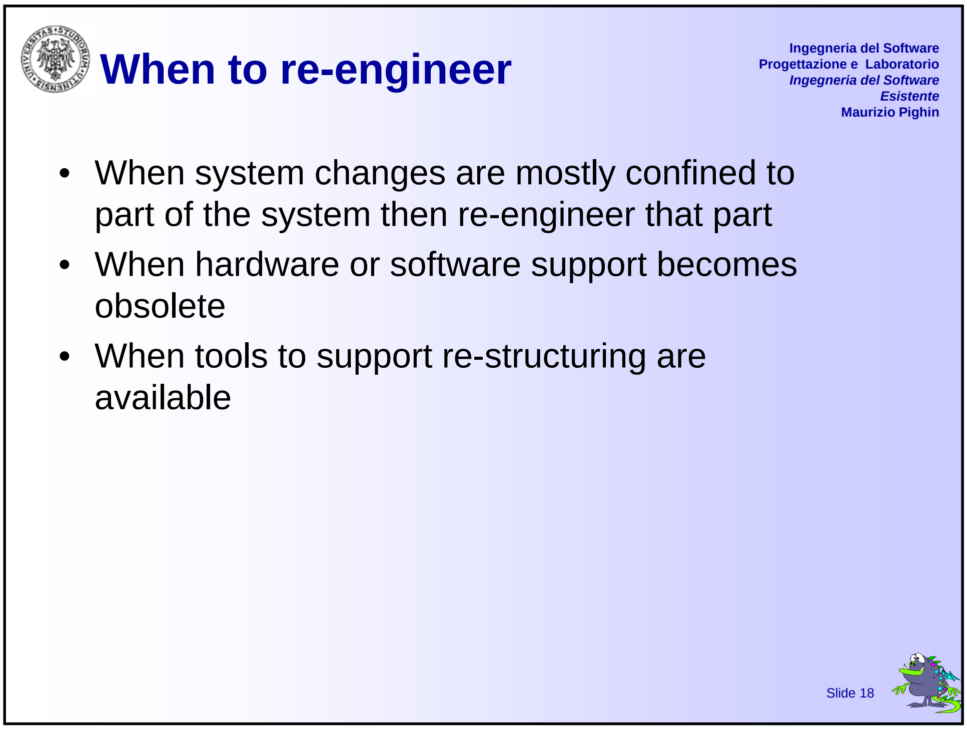
Factor	**Questions**
Supplier stability	Is the supplier is still in existence? Is the supplier financially stable and likely to continue in existence? If the supplier is no longer in business, are the systems maintained by someone else?
Failure rate	Does the hardware have a high rate of reported failures? Does the support software crash and force system restarts?
Age	How old is the hardware and software? The older the hardware and support software, the more obsolete it will be. It may still function correctly but there could be significant economic and business benefits to moving to more modern systems.
Performance	Is the performance of the system adequate? Do performance problems have a significant effect on system users?
Support requirements	What local support is required by the hardware and software? If there are high costs associated with this support, it may be worth considering system replacement.
Maintenance costs	What are the costs of hardware maintenance and support software licences? Older hardware may have higher maintenance costs than modern systems. Support software may have high annual licensing costs.
Interoperability	Are there problems interfacing the system to other systems? Can compilers etc. be used with current versions of the operating system? Is hardware emulation required?

Slide 16



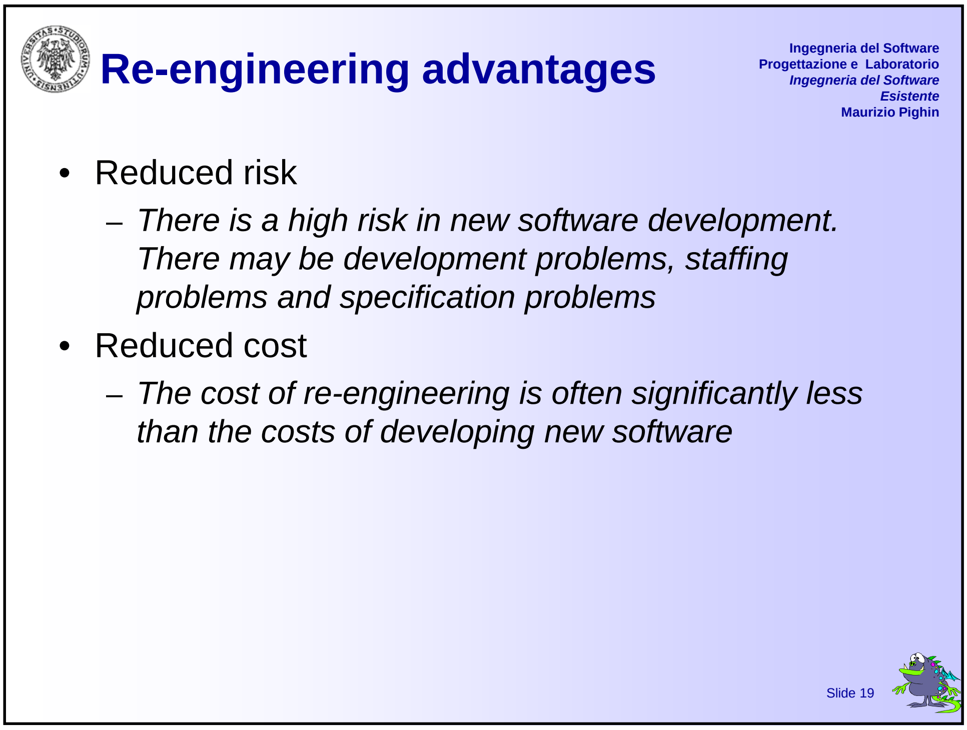
Application assessment

Factor	Questions
Understandability	How difficult is it to understand the source code of the current system? How complex are the control structures which are used? Do variables have meaningful names that reflect their function?
Documentation	What system documentation is available? Is the documentation complete, consistent and up-to-date?
Data	Is there an explicit data model for the system? To what extent is data duplicated in different files? Is the data used by the system up-to-date and consistent?
Performance	Is the performance of the application adequate? Do performance problems have a significant effect on system users?
Programming language	Are modern compilers available for the programming language used to develop the system? Is the programming language still used for new system development?
Configuration management	Are all versions of all parts of the system managed by a configuration management system? Is there an explicit description of the versions of components that are used in the current system?
Test data	Does test data for the system exist? Is there a record of regression tests carried out when new features have been added to the system?
Personnel skills	Are there people available who have the skills to maintain the application? Are there only a limited number of people who understand the system?



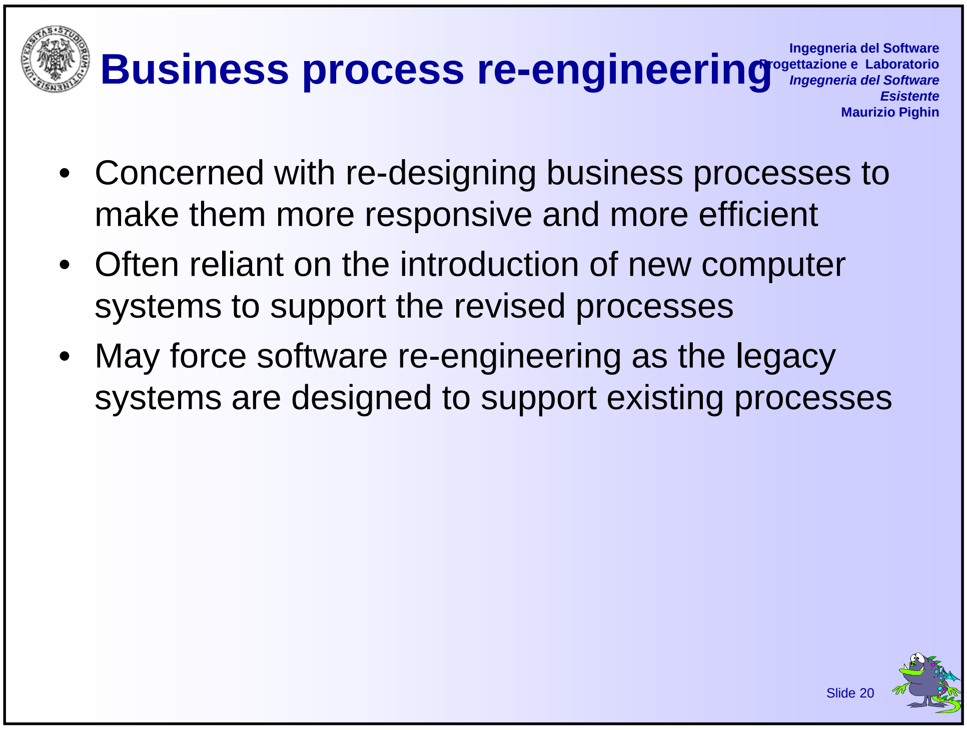
When to re-engineer

- When system changes are mostly confined to part of the system then re-engineer that part
- When hardware or software support becomes obsolete
- When tools to support re-structuring are available



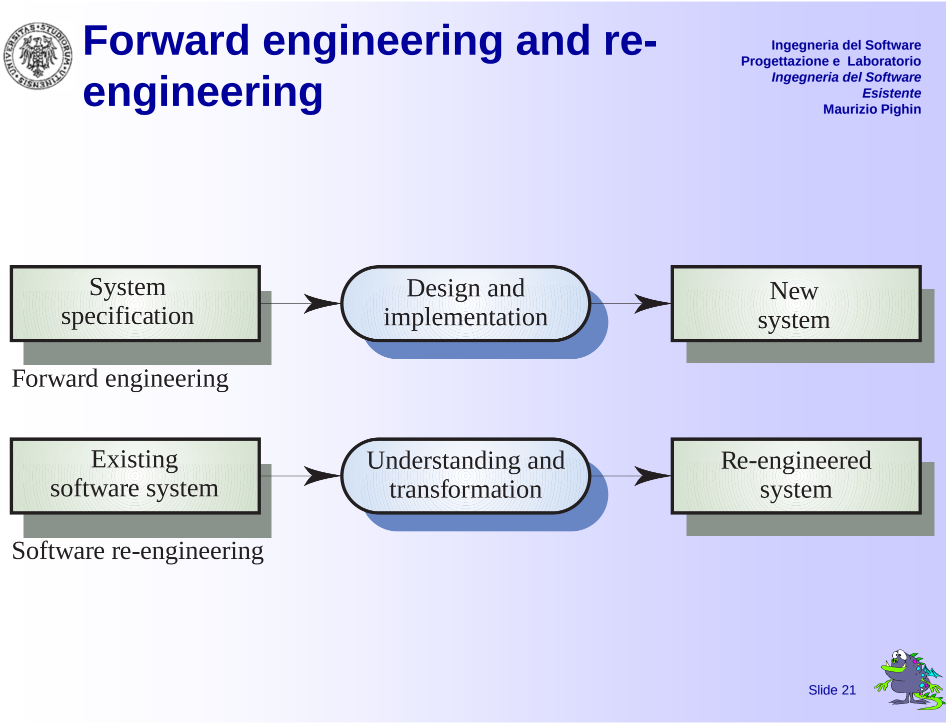
Re-engineering advantages

- Reduced risk
  - *There is a high risk in new software development. There may be development problems, staffing problems and specification problems*
- Reduced cost
  - *The cost of re-engineering is often significantly less than the costs of developing new software*



Business process re-engineering

- Concerned with re-designing business processes to make them more responsive and more efficient
- Often reliant on the introduction of new computer systems to support the revised processes
- May force software re-engineering as the legacy systems are designed to support existing processes



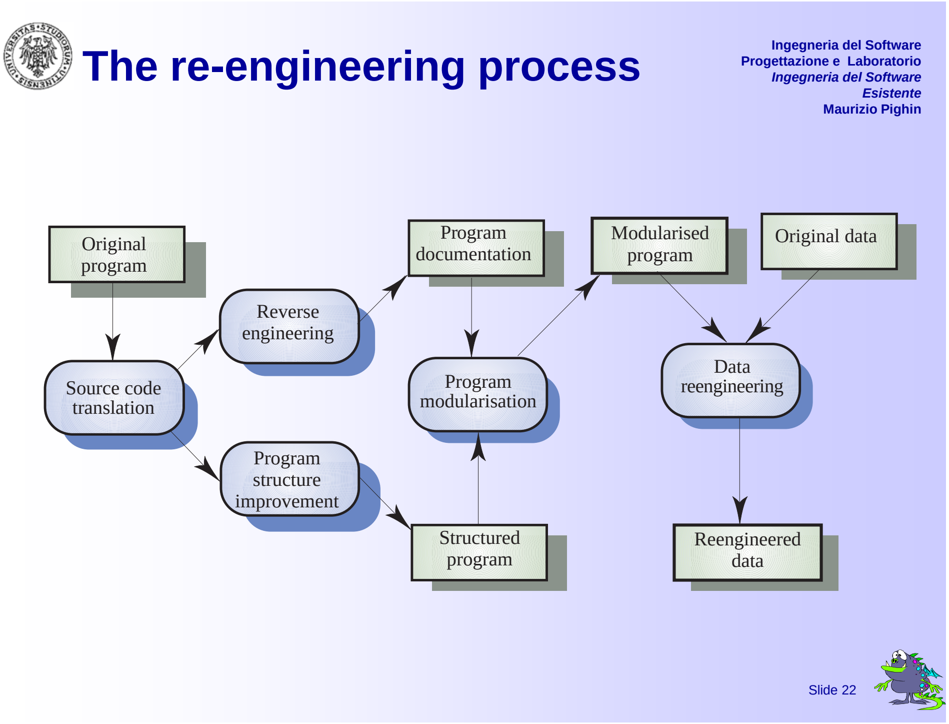
Forward engineering and re-engineering

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

System specification
Design and implementation
New system
Forward engineering

Existing software system
Understanding and transformation
Re-engineered system
Software re-engineering

Slide 21

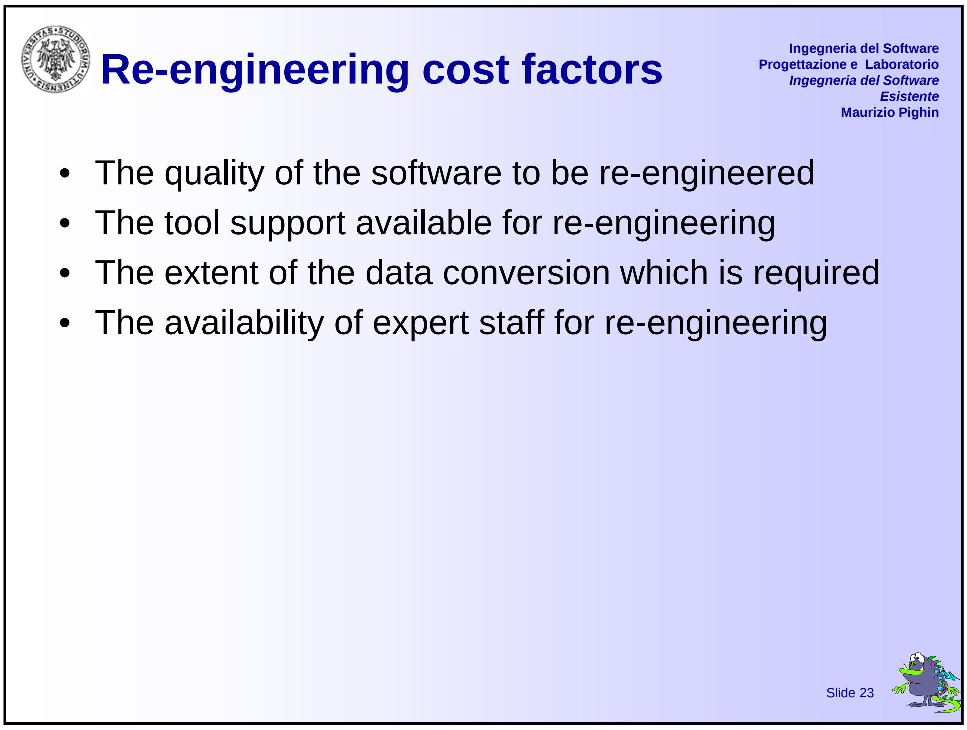


The re-engineering process

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

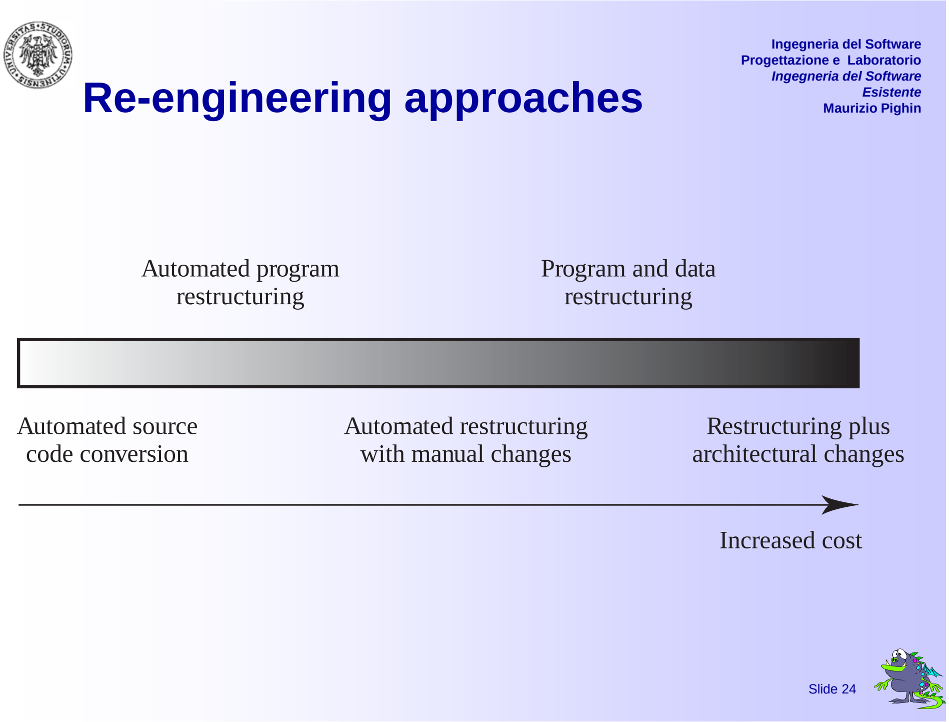
Original program
Source code translation
Reverse engineering
Program structure improvement
Program documentation
Program modularisation
Structured program
Modularised program
Original data
Data reengineering
Reengineered data

Slide 22



Re-engineering cost factors

- The quality of the software to be re-engineered
- The tool support available for re-engineering
- The extent of the data conversion which is required
- The availability of expert staff for re-engineering



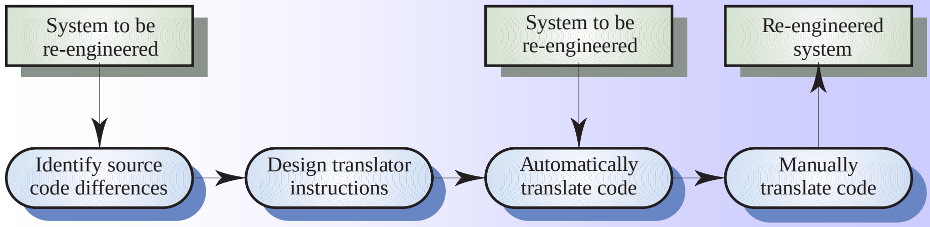
Re-engineering approaches

Automated program restructuring
Program and data restructuring
Automated source code conversion
Automated restructuring with manual changes
Restructuring plus architectural changes
Increased cost

Source code translation

- Involves converting the code from one language (or language version) to another e.g. FORTRAN to C++
- May be necessary because of:
  - *Hardware platform update*
  - *Staff skill shortages*
  - *Organisational policy changes*
- Only realistic if an automatic translator is available

The program translation process


System to be re-engineered
System to be re-engineered
Re-engineered system
Identify source code differences
Design translator instructions
Automatically translate code
Manually translate code

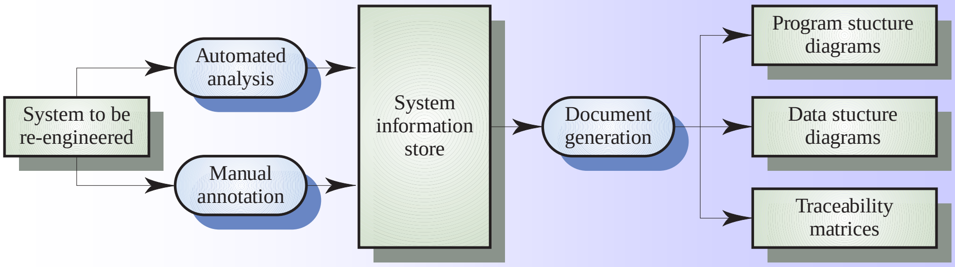
Reverse engineering

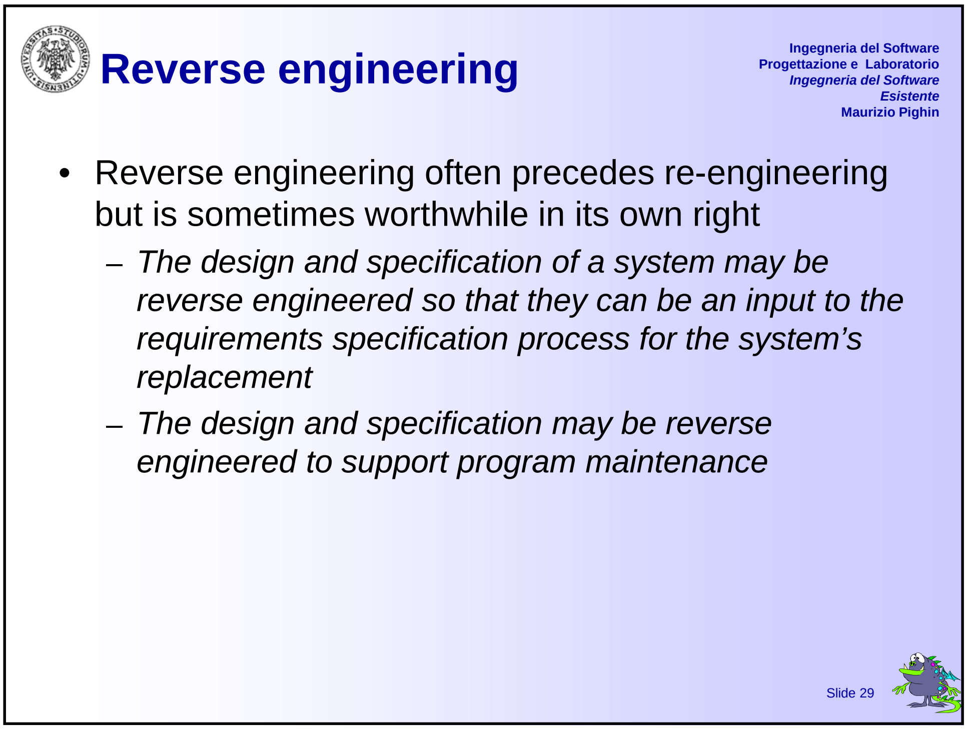
Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

- Analysing software with a view to understanding its design and specification
- May be part of a re-engineering process but may also be used to re-specify a system for re-implementation
- Builds a program data base and generates information from this
- Program understanding tools (browsers, cross-reference generators, etc.) may be used in this process

The reverse engineering process

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin



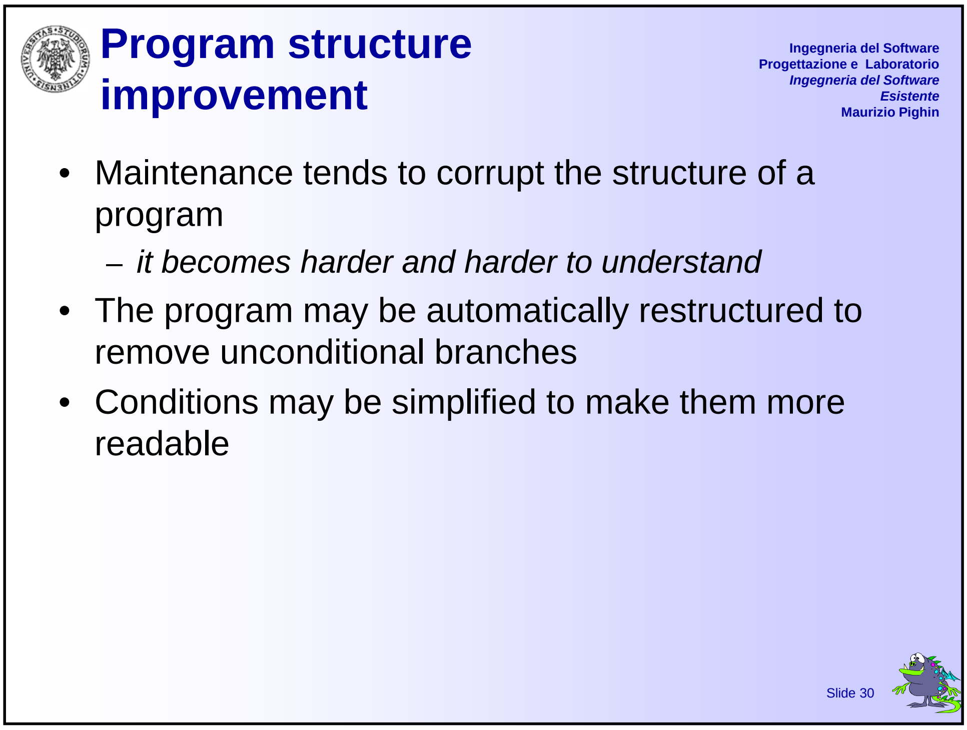


Reverse engineering

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

- Reverse engineering often precedes re-engineering but is sometimes worthwhile in its own right
  - *The design and specification of a system may be reverse engineered so that they can be an input to the requirements specification process for the system's replacement*
  - *The design and specification may be reverse engineered to support program maintenance*

Slide 29

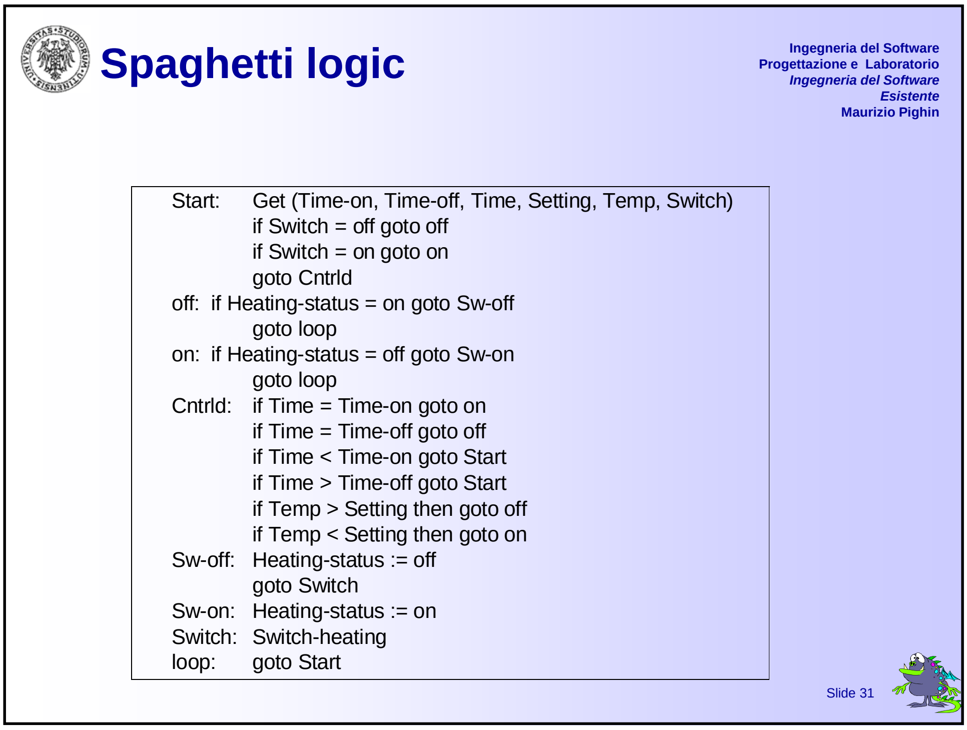


Program structure improvement

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

- Maintenance tends to corrupt the structure of a program
  - *it becomes harder and harder to understand*
- The program may be automatically restructured to remove unconditional branches
- Conditions may be simplified to make them more readable

Slide 30

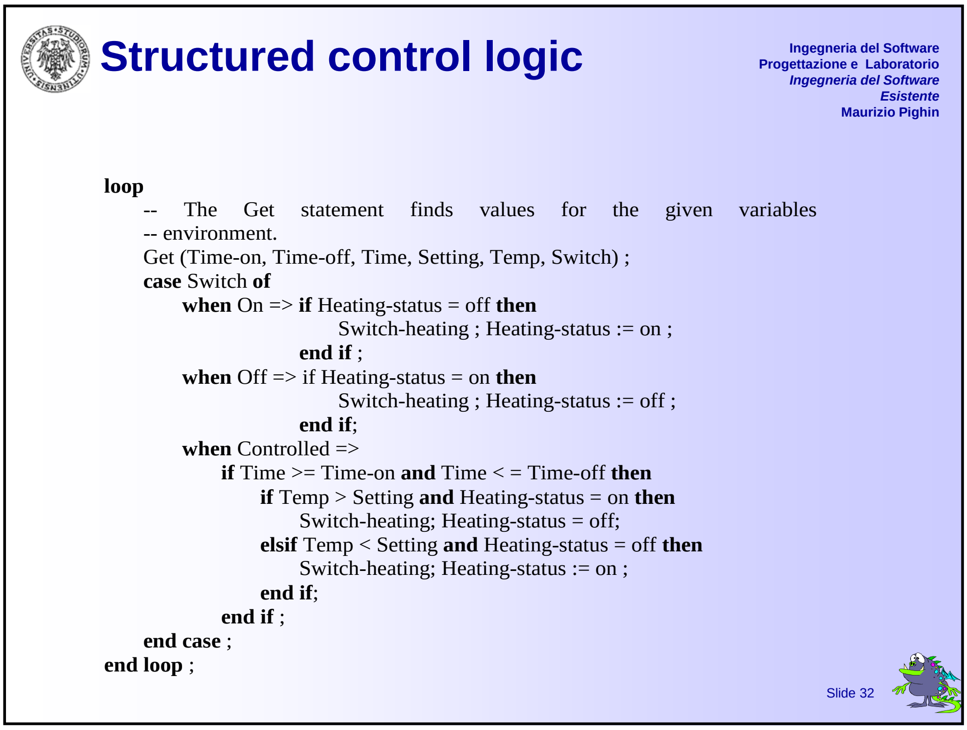


Spaghetti logic

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

```
Start:    Get (Time-on, Time-off, Time, Setting, Temp, Switch)
          if Switch = off goto off
          if Switch = on goto on
          goto Cntrld
off:  if Heating-status = on goto Sw-off
          goto loop
on:  if Heating-status = off goto Sw-on
          goto loop
Cntrld:   if Time = Time-on goto on
          if Time = Time-off goto off
          if Time < Time-on goto Start
          if Time > Time-off goto Start
          if Temp > Setting then goto off
          if Temp < Setting then goto on
Sw-off:   Heating-status := off
          goto Switch
Sw-on:    Heating-status := on
Switch:   Switch-heating
loop:     goto Start
```

Slide 31

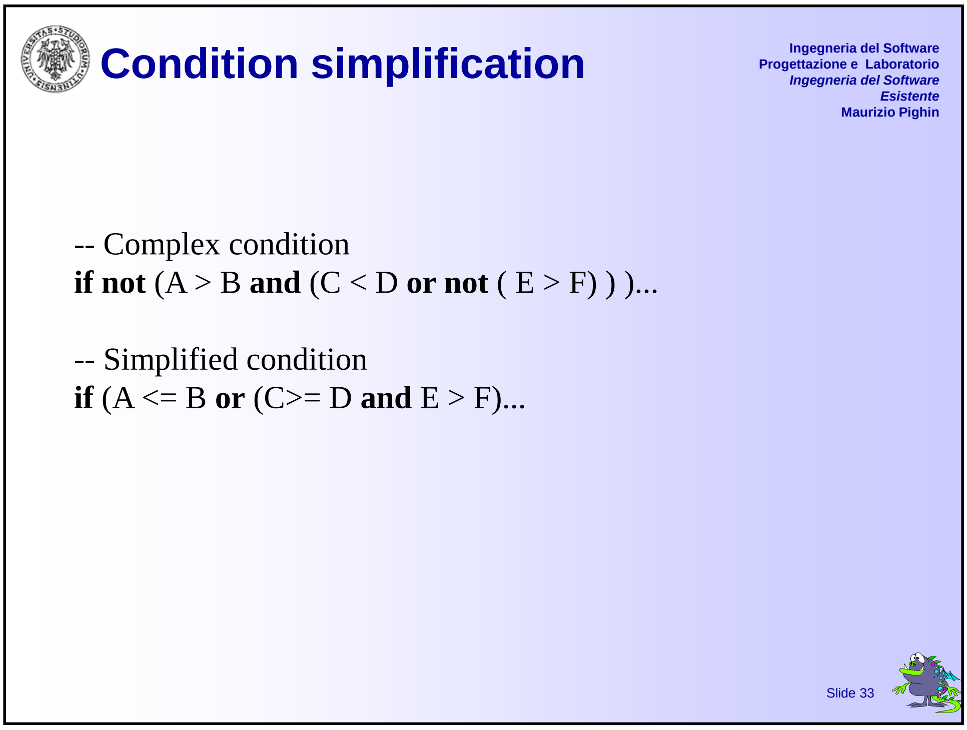


Structured control logic

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

```
loop
    -- The Get statement finds values for the given variables
    -- environment.
    Get (Time-on, Time-off, Time, Setting, Temp, Switch) ;
    case Switch of
        when On => if Heating-status = off then
                        Switch-heating ; Heating-status := on ;
                    end if ;
        when Off => if Heating-status = on then
                        Switch-heating ; Heating-status := off ;
                    end if;
        when Controlled =>
            if Time >= Time-on and Time < = Time-off then
                if Temp > Setting and Heating-status = on then
                    Switch-heating; Heating-status = off;
                elsif Temp < Setting and Heating-status = off then
                    Switch-heating; Heating-status := on ;
                end if;
            end if ;
    end case ;
end loop ;
```

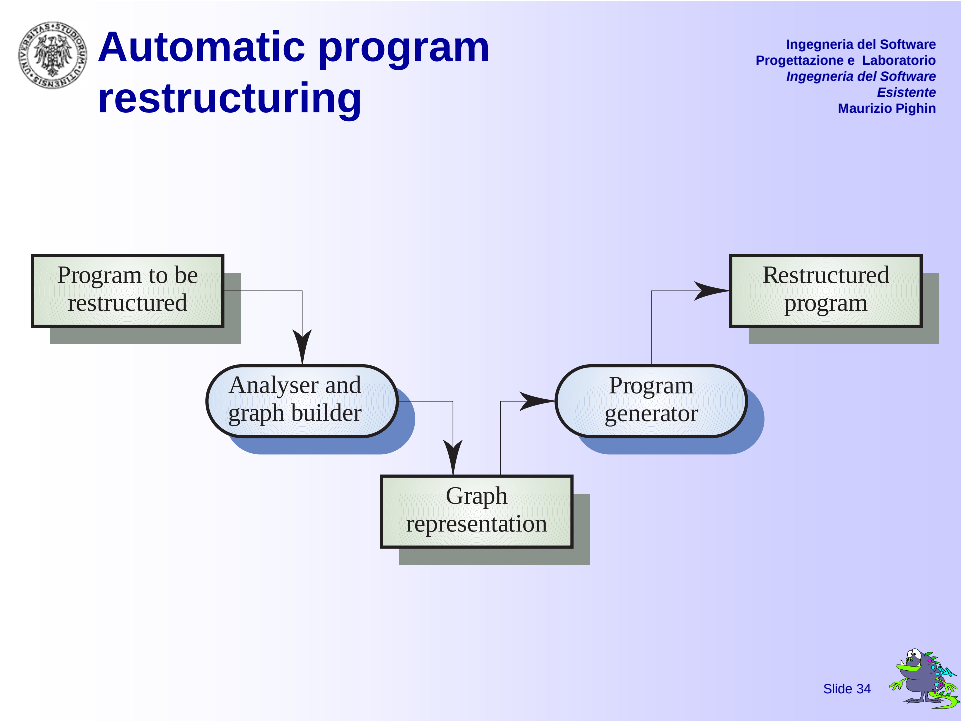
Slide 32



Condition simplification

-- Complex condition
if not (A > B **and** (C < D **or not** (E > F)))...

-- Simplified condition
if (A <= B **or** (C>= D **and** E > F)...



Automatic program restructuring

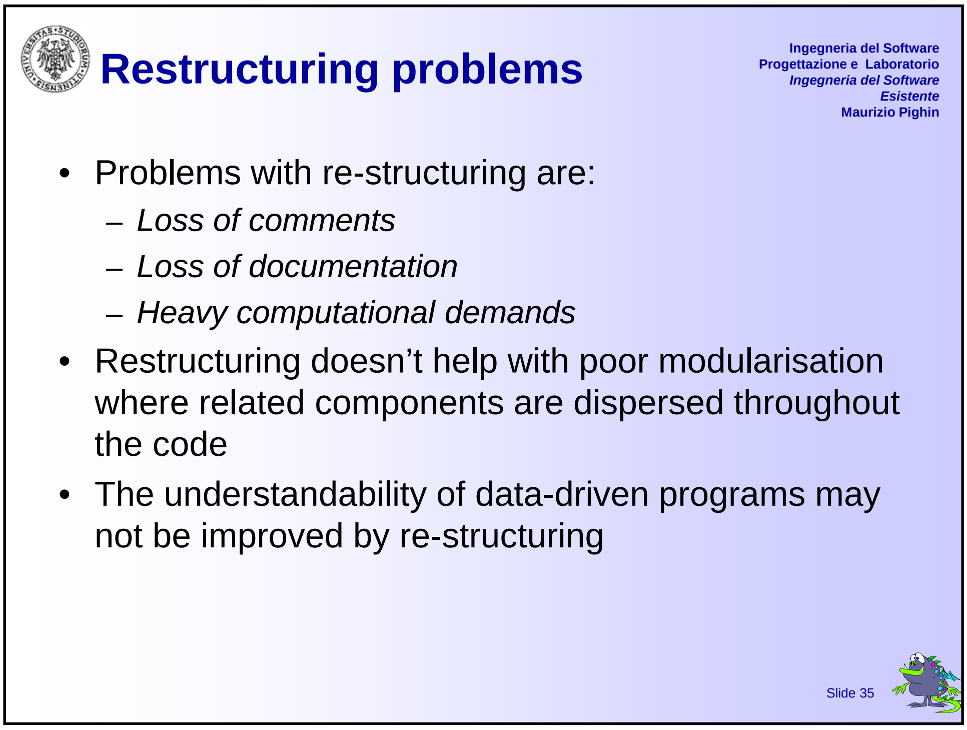
Program to be restructured

Analyser and graph builder

Graph representation

Program generator

Restructured program

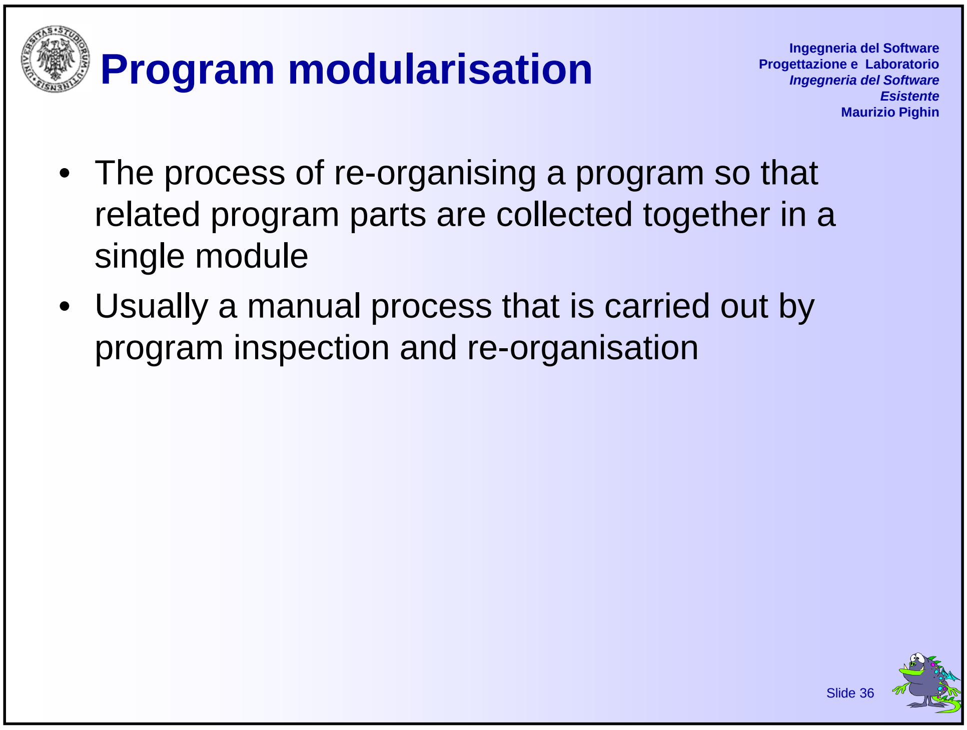


Restructuring problems

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

- Problems with re-structuring are:
  - *Loss of comments*
  - *Loss of documentation*
  - *Heavy computational demands*
- Restructuring doesn’t help with poor modularisation where related components are dispersed throughout the code
- The understandability of data-driven programs may not be improved by re-structuring

Slide 35

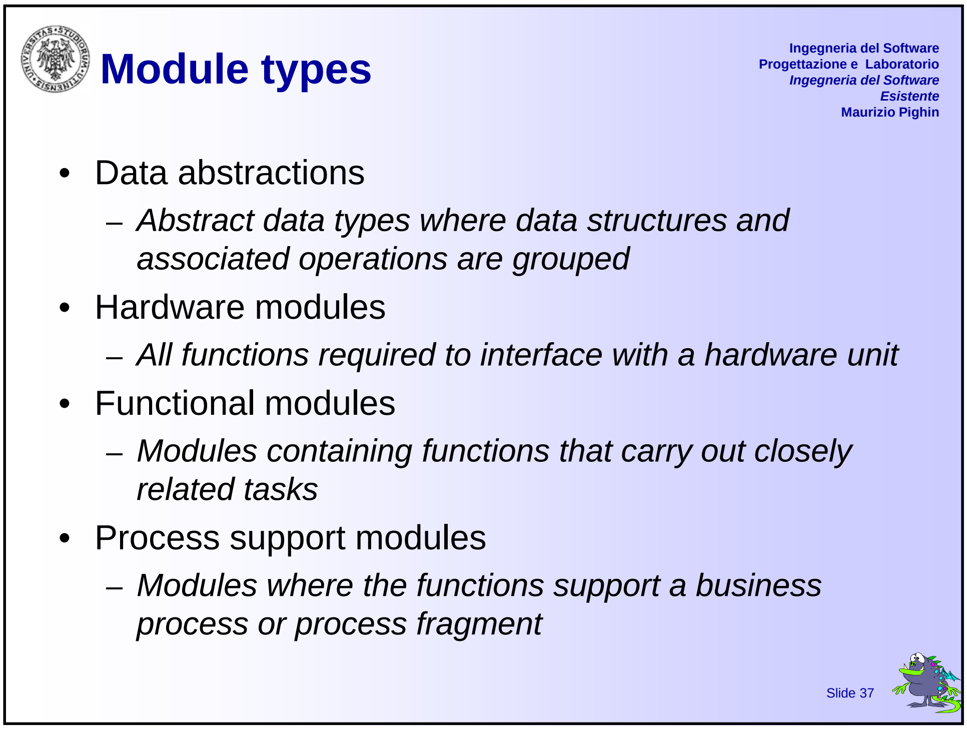


Program modularisation

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

- The process of re-organising a program so that related program parts are collected together in a single module
- Usually a manual process that is carried out by program inspection and re-organisation

Slide 36

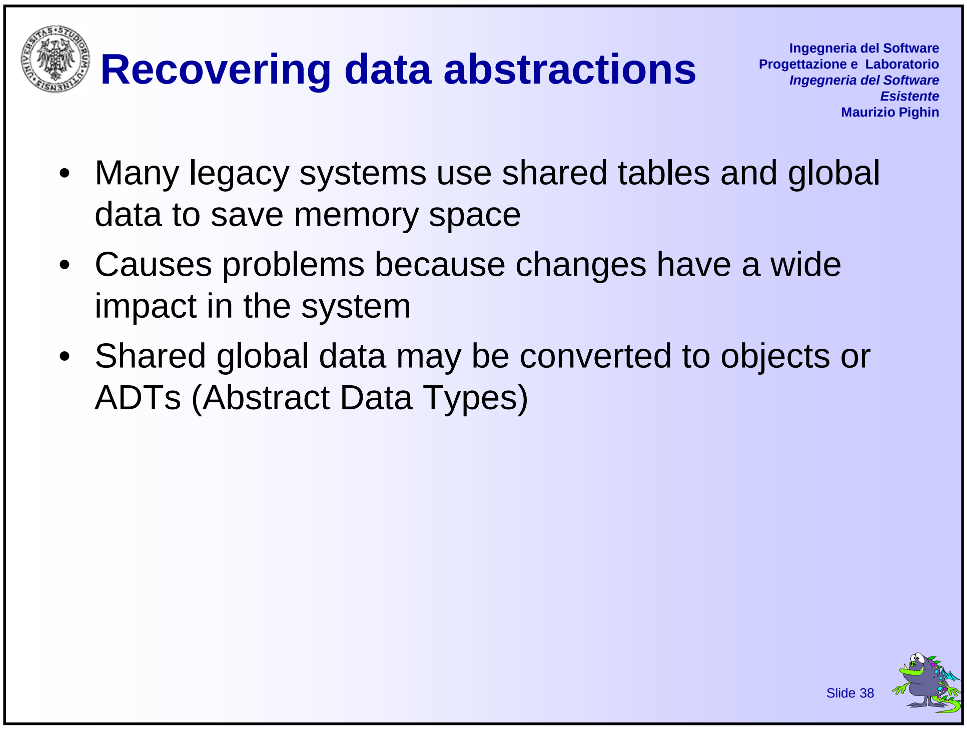


Module types

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

- Data abstractions
  - *Abstract data types where data structures and associated operations are grouped*
- Hardware modules
  - *All functions required to interface with a hardware unit*
- Functional modules
  - *Modules containing functions that carry out closely related tasks*
- Process support modules
  - *Modules where the functions support a business process or process fragment*

Slide 37

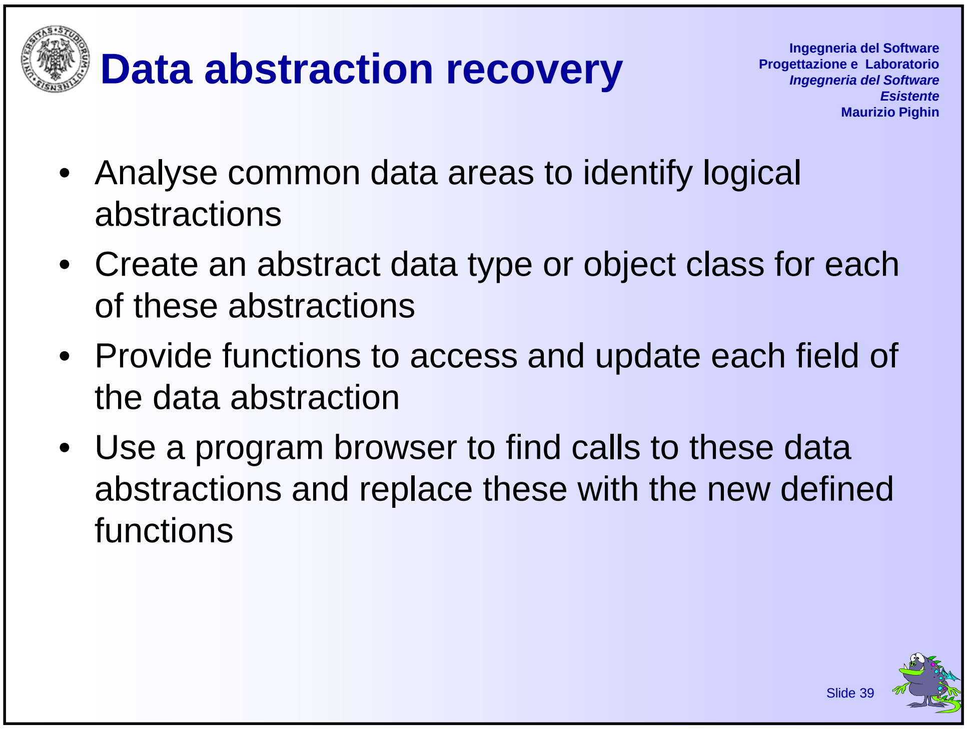


Recovering data abstractions

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

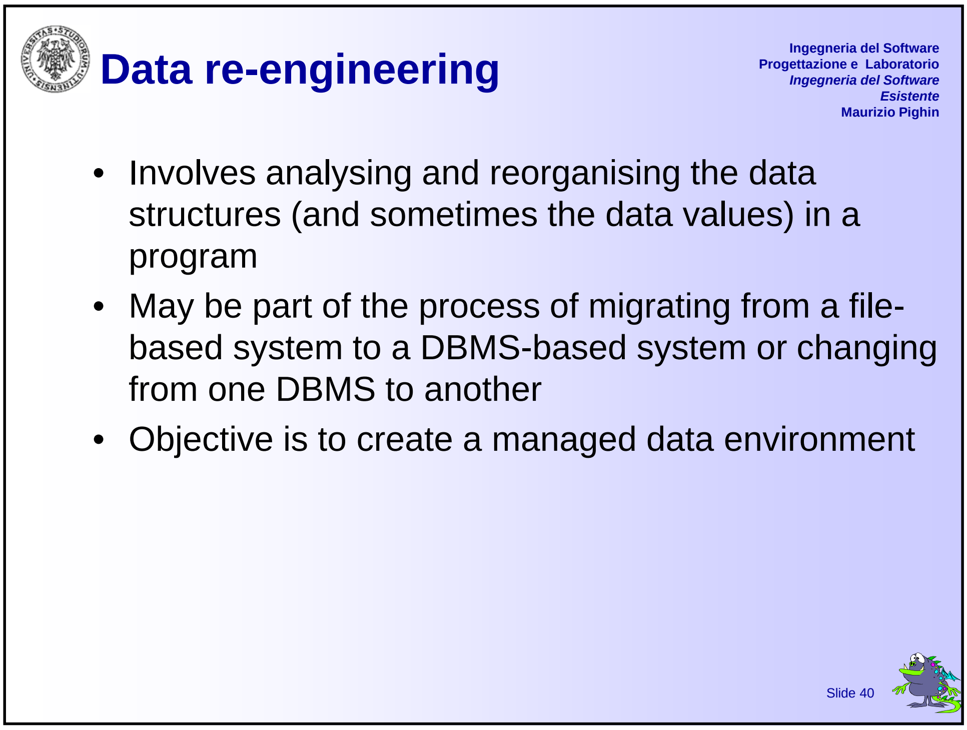
- Many legacy systems use shared tables and global data to save memory space
- Causes problems because changes have a wide impact in the system
- Shared global data may be converted to objects or ADTs (Abstract Data Types)

Slide 38



Data abstraction recovery

- Analyse common data areas to identify logical abstractions
- Create an abstract data type or object class for each of these abstractions
- Provide functions to access and update each field of the data abstraction
- Use a program browser to find calls to these data abstractions and replace these with the new defined functions



Data re-engineering

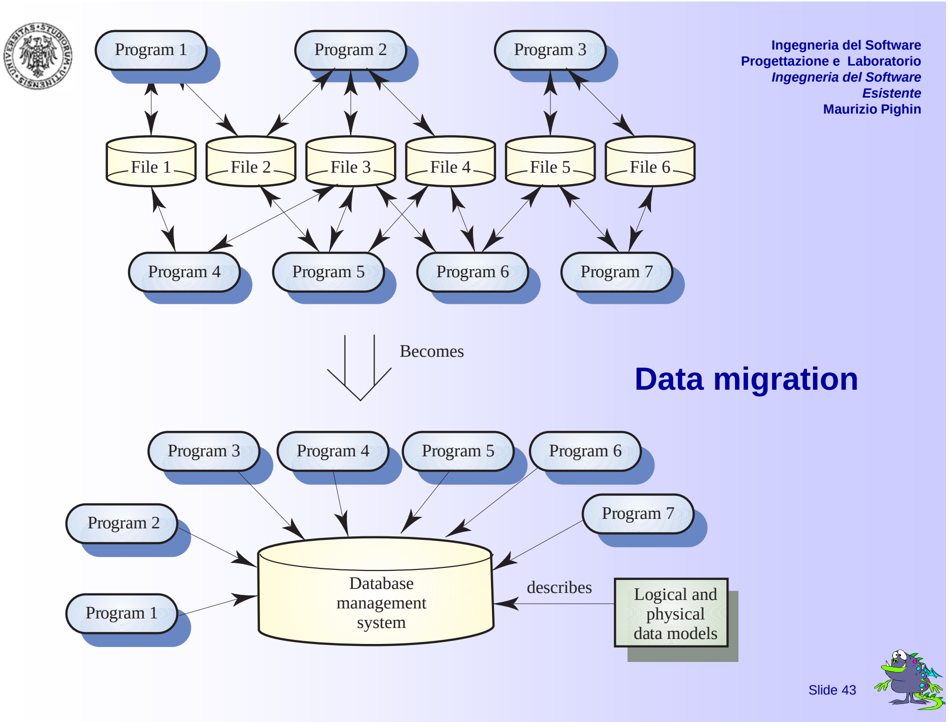
- Involves analysing and reorganising the data structures (and sometimes the data values) in a program
- May be part of the process of migrating from a file-based system to a DBMS-based system or changing from one DBMS to another
- Objective is to create a managed data environment

Approaches to data re-engineering

Approach	Description
Data cleanup	The data records and values are analysed to improve their quality. Duplicates are removed, redundant information is deleted and a consistent format applied to all records. This should not normally require any associated program changes.
Data extension	In this case, the data and associated programs are re-engineered to remove limits on the data processing. This may require changes to programs to increase field lengths, modify upper limits on the tables, etc. The data itself may then have to be rewritten and cleaned up to reflect the program changes.
Data migration	In this case, data is moved into the control of a modern database management system. The data may be stored in separate files or may be managed by an older type of DBMS.

Data problems

- End-users want data on their desktop machines rather than in a file system. They need to be able to download this data from a DBMS
- Systems may have to process much more data than was originally intended by their designers
- Redundant data may be stored in different formats in different places in the system



Data migration

Program 1 Program 2 Program 3

File 1 File 2 File 3 File 4 File 5 File 6

Program 4 Program 5 Program 6 Program 7

Becomes

Program 3 Program 4 Program 5 Program 6

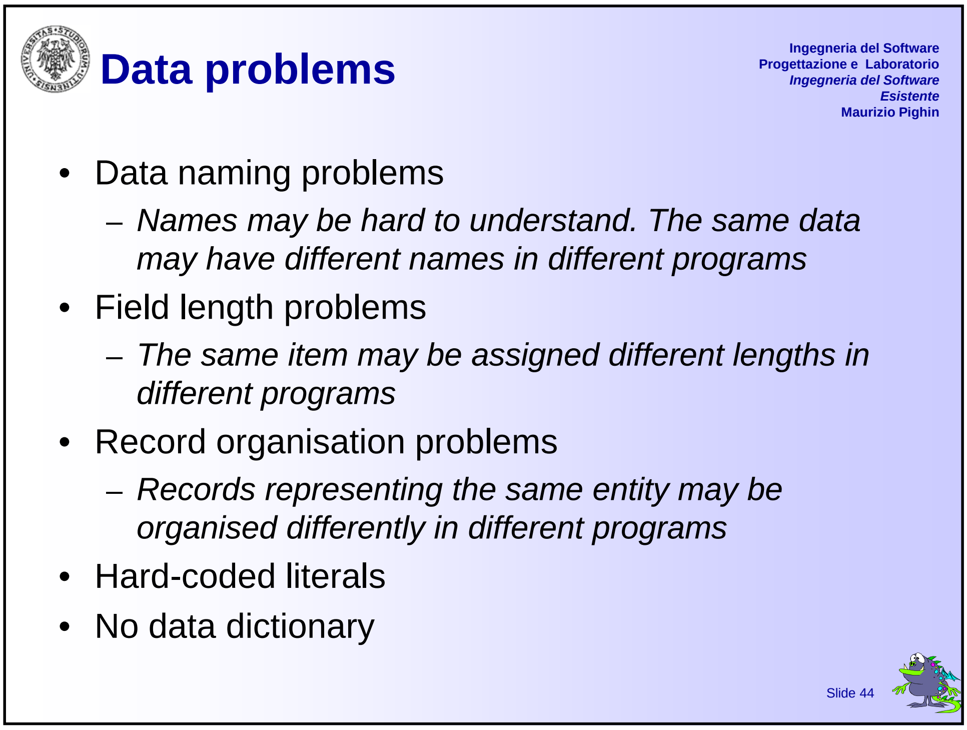
Program 2 Program 7

Program 1

Database management system

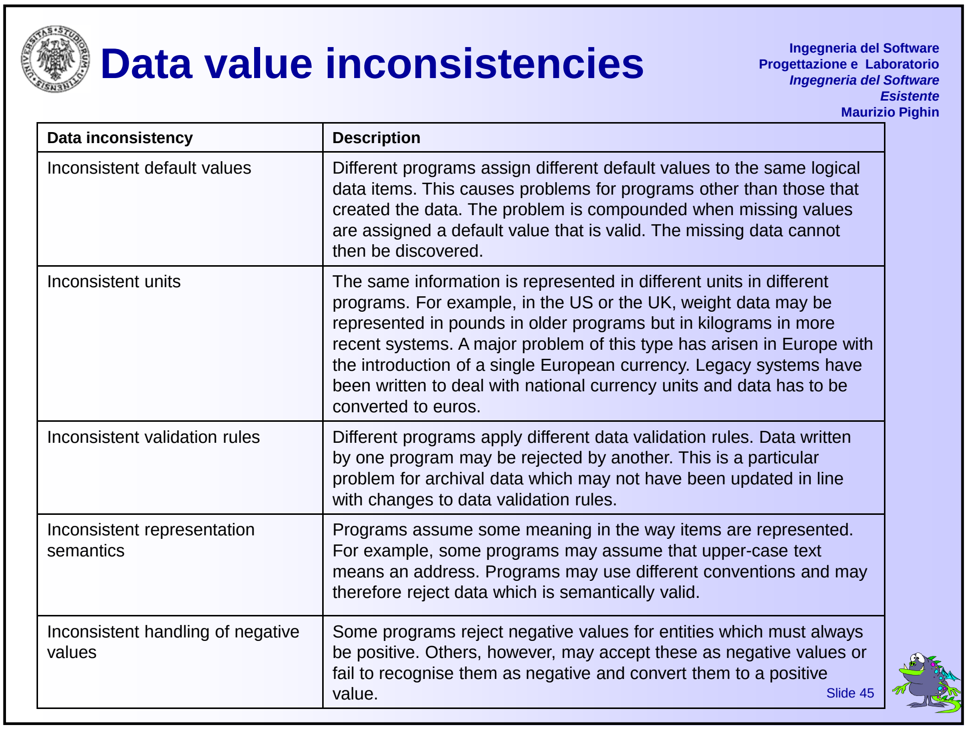
describes

Logical and physical data models



Data problems

- Data naming problems
  - *Names may be hard to understand. The same data may have different names in different programs*
- Field length problems
  - *The same item may be assigned different lengths in different programs*
- Record organisation problems
  - *Records representing the same entity may be organised differently in different programs*
- Hard-coded literals
- No data dictionary

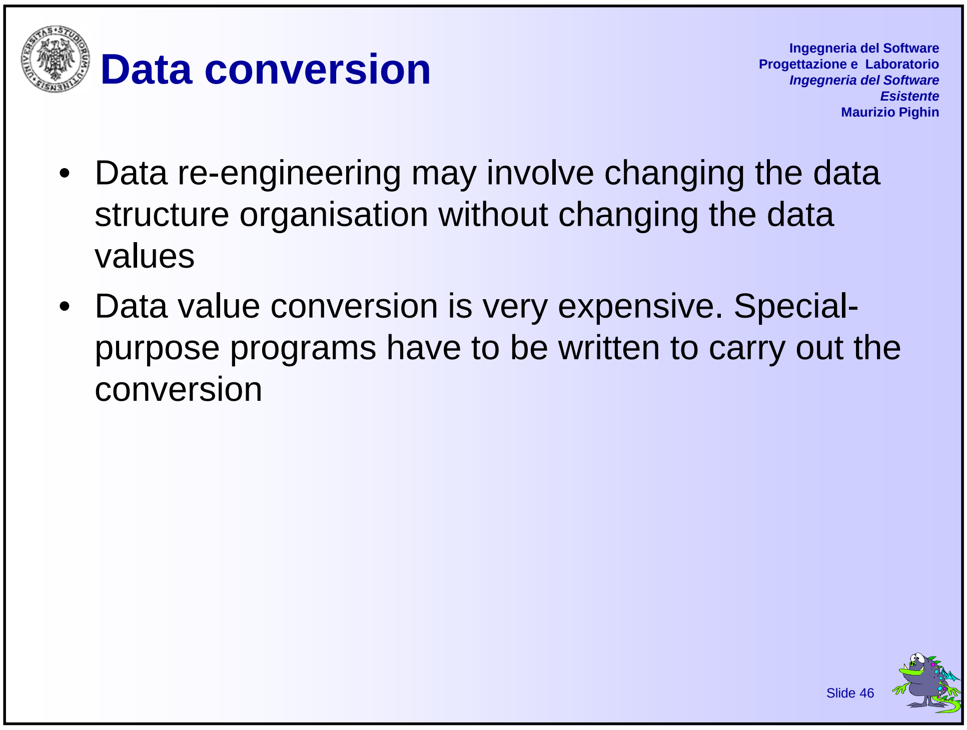


Data value inconsistencies

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

Data inconsistency	Description
Inconsistent default values	Different programs assign different default values to the same logical data items. This causes problems for programs other than those that created the data. The problem is compounded when missing values are assigned a default value that is valid. The missing data cannot then be discovered.
Inconsistent units	The same information is represented in different units in different programs. For example, in the US or the UK, weight data may be represented in pounds in older programs but in kilograms in more recent systems. A major problem of this type has arisen in Europe with the introduction of a single European currency. Legacy systems have been written to deal with national currency units and data has to be converted to euros.
Inconsistent validation rules	Different programs apply different data validation rules. Data written by one program may be rejected by another. This is a particular problem for archival data which may not have been updated in line with changes to data validation rules.
Inconsistent representation semantics	Programs assume some meaning in the way items are represented. For example, some programs may assume that upper-case text means an address. Programs may use different conventions and may therefore reject data which is semantically valid.
Inconsistent handling of negative values	Some programs reject negative values for entities which must always be positive. Others, however, may accept these as negative values or fail to recognise them as negative and convert them to a positive value.

Slide 45

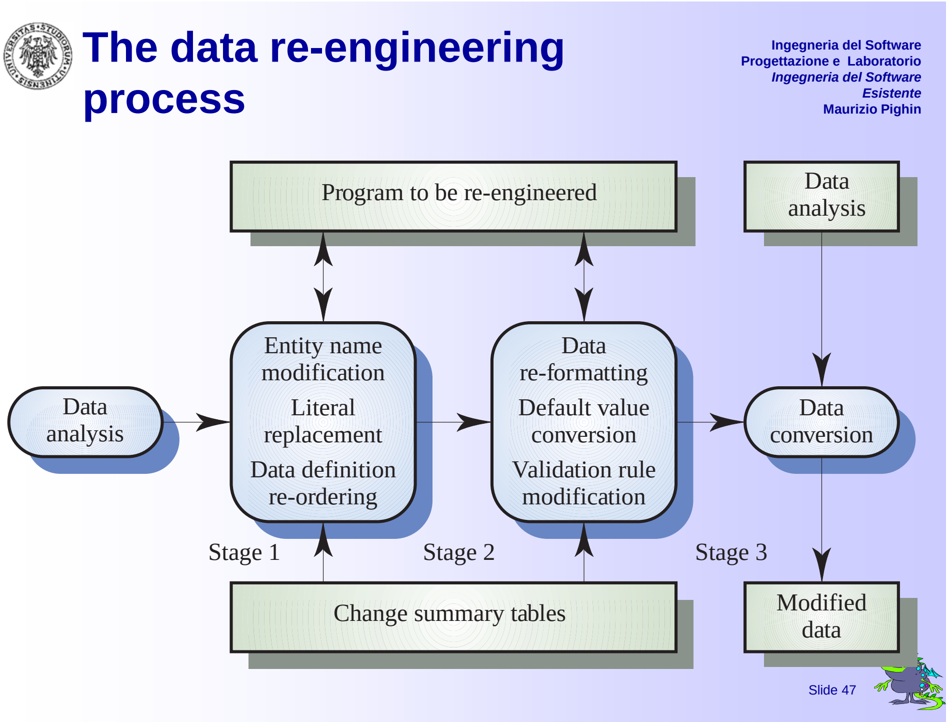


Data conversion

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

- Data re-engineering may involve changing the data structure organisation without changing the data values
- Data value conversion is very expensive. Special-purpose programs have to be written to carry out the conversion

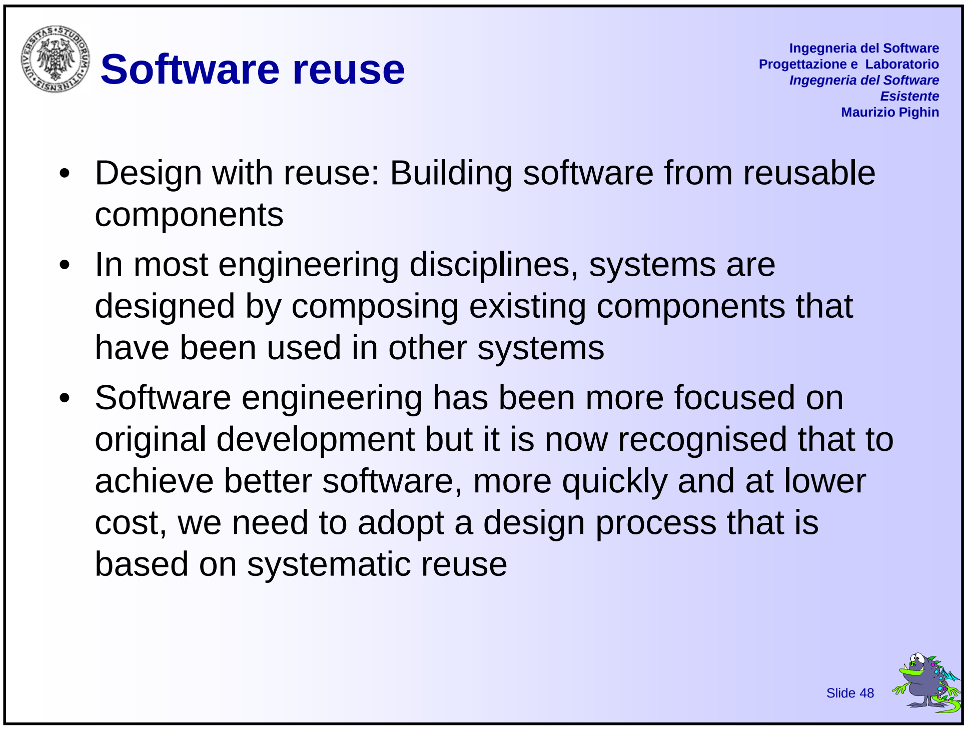
Slide 46



The data re-engineering process

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

Slide 47

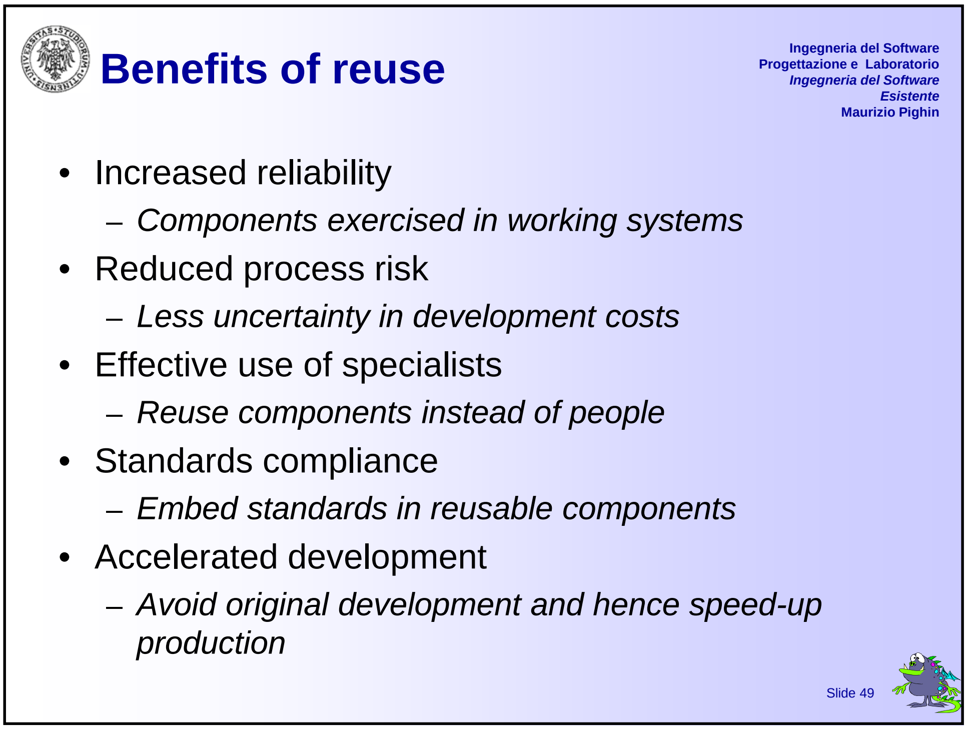


Software reuse

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

- Design with reuse: Building software from reusable components
- In most engineering disciplines, systems are designed by composing existing components that have been used in other systems
- Software engineering has been more focused on original development but it is now recognised that to achieve better software, more quickly and at lower cost, we need to adopt a design process that is based on systematic reuse

Slide 48

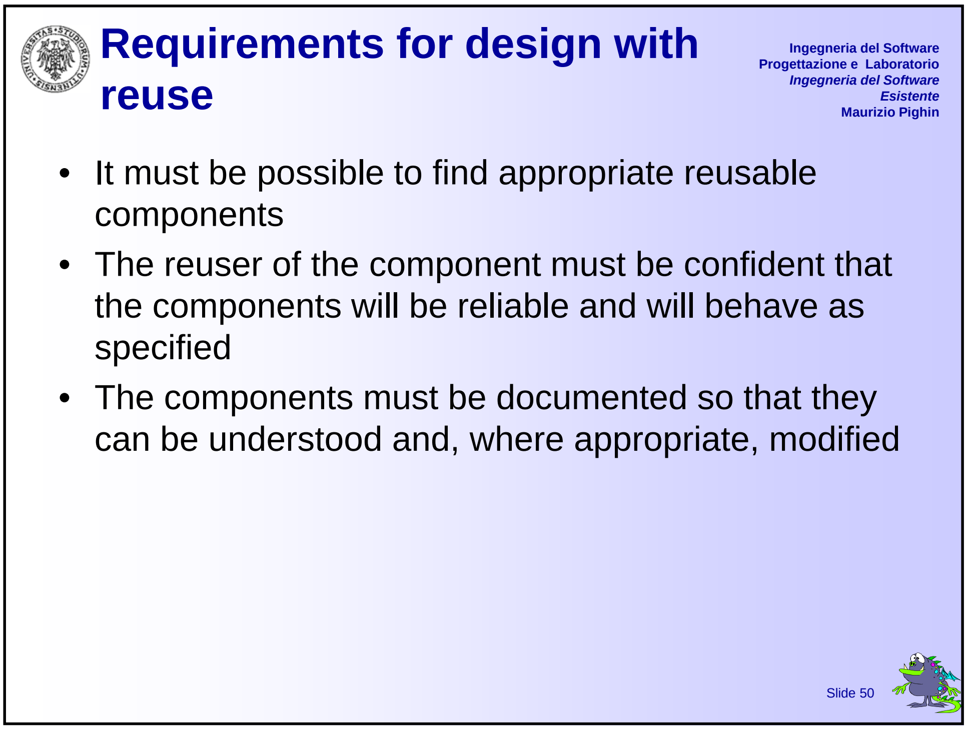


Benefits of reuse

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

- Increased reliability
  - *Components exercised in working systems*
- Reduced process risk
  - *Less uncertainty in development costs*
- Effective use of specialists
  - *Reuse components instead of people*
- Standards compliance
  - *Embed standards in reusable components*
- Accelerated development
  - *Avoid original development and hence speed-up production*

Slide 49

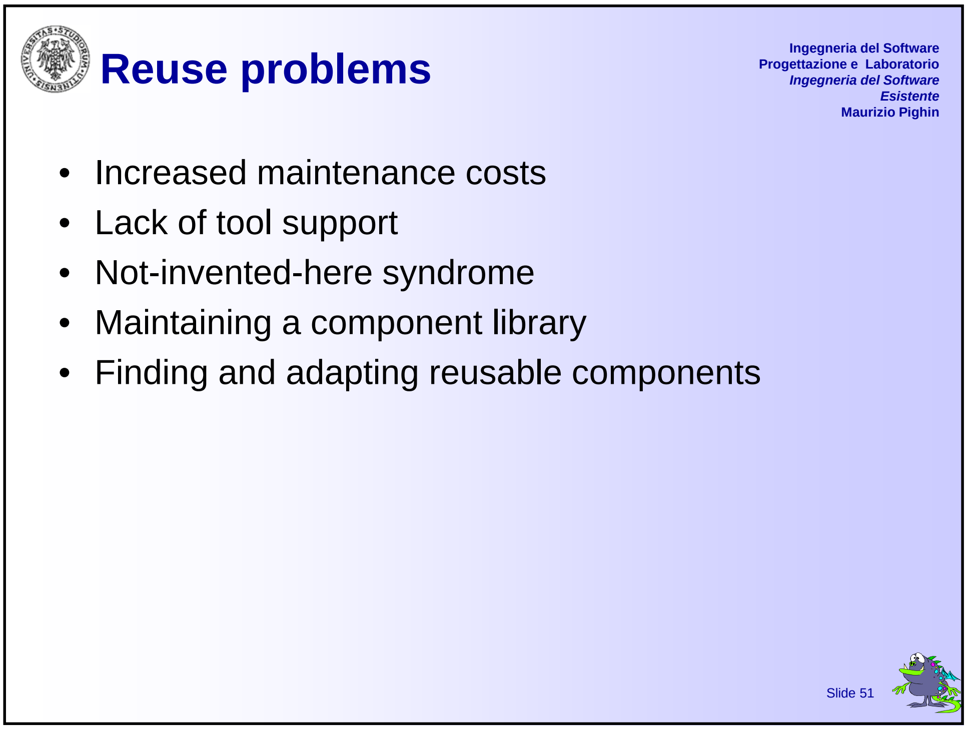


Requirements for design with reuse

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

- It must be possible to find appropriate reusable components
- The reuser of the component must be confident that the components will be reliable and will behave as specified
- The components must be documented so that they can be understood and, where appropriate, modified

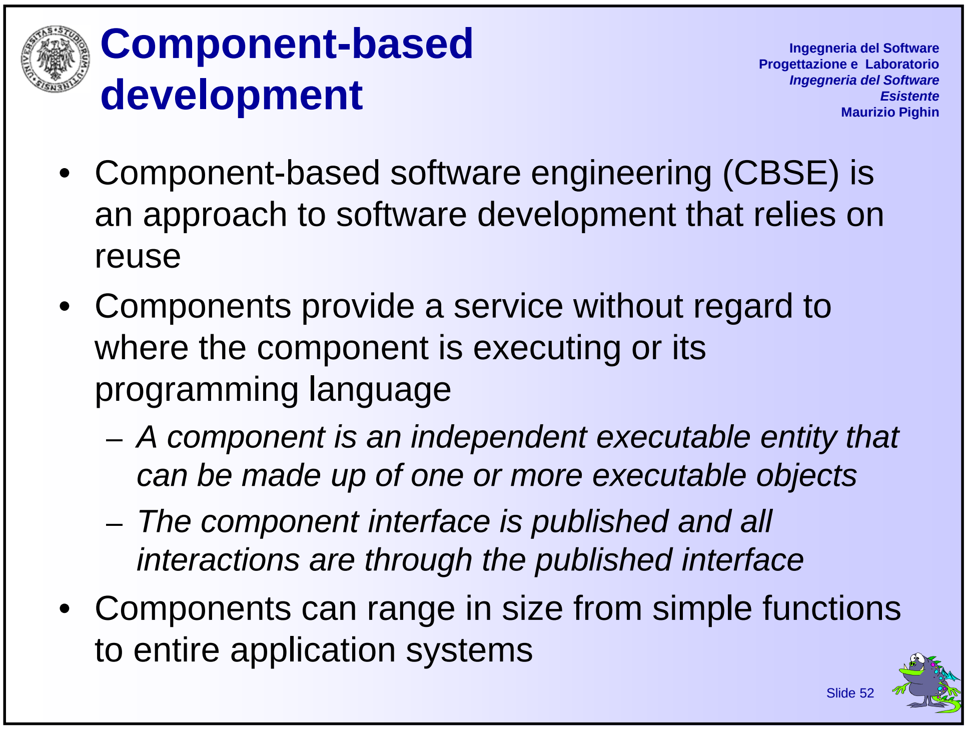
Slide 50



Reuse problems

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

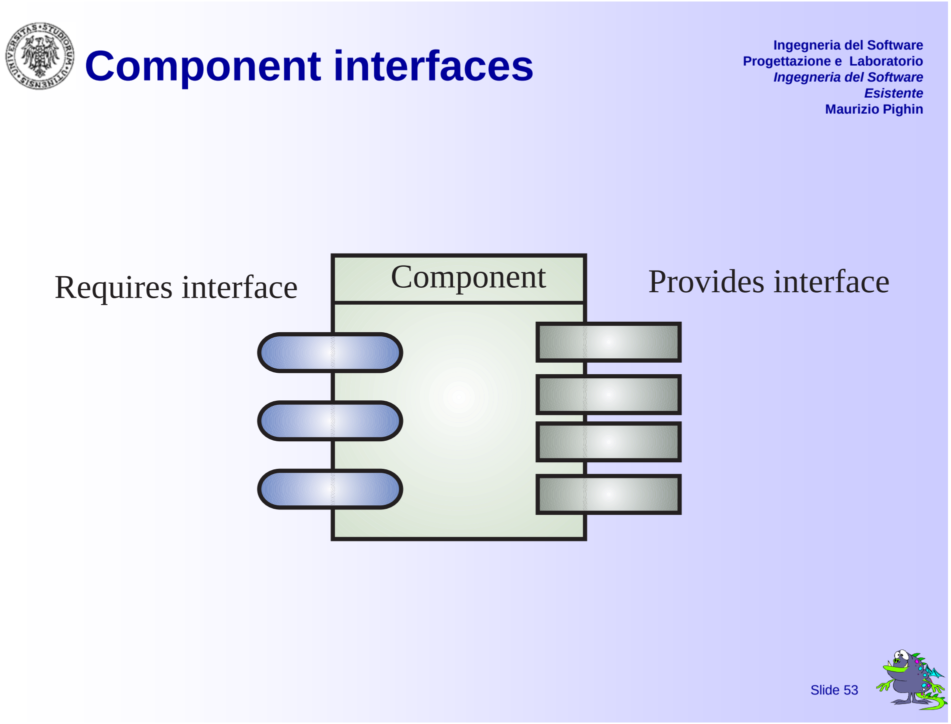
- Increased maintenance costs
- Lack of tool support
- Not-invented-here syndrome
- Maintaining a component library
- Finding and adapting reusable components



Component-based development

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

- Component-based software engineering (CBSE) is an approach to software development that relies on reuse
- Components provide a service without regard to where the component is executing or its programming language
  - *A component is an independent executable entity that can be made up of one or more executable objects*
  - *The component interface is published and all interactions are through the published interface*
- Components can range in size from simple functions to entire application systems

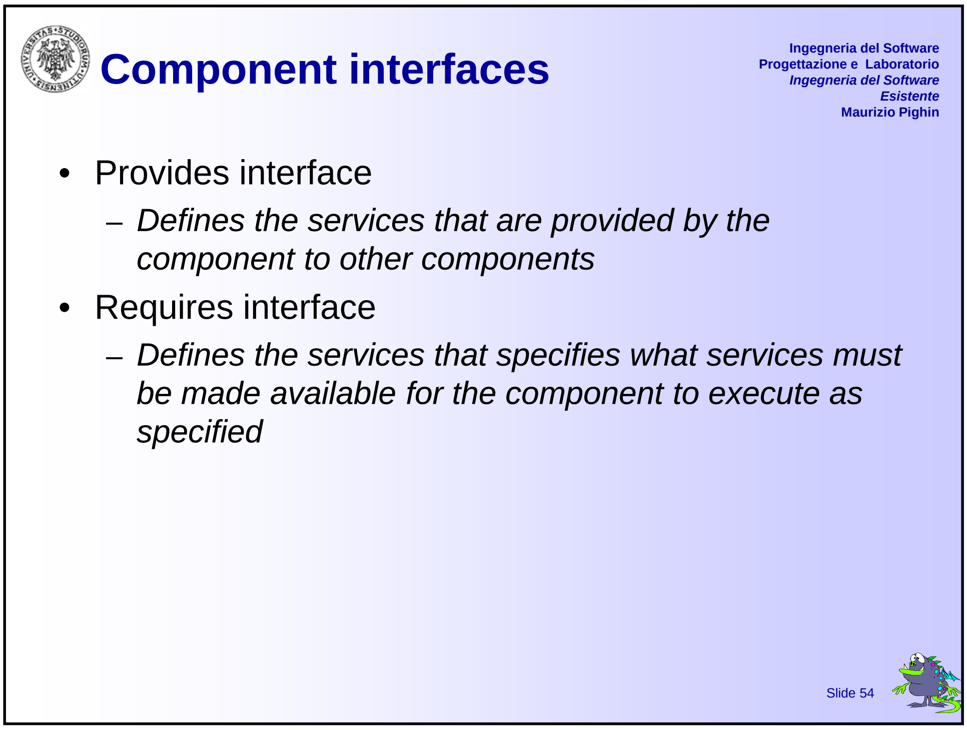


Component interfaces

Requires interface

Component

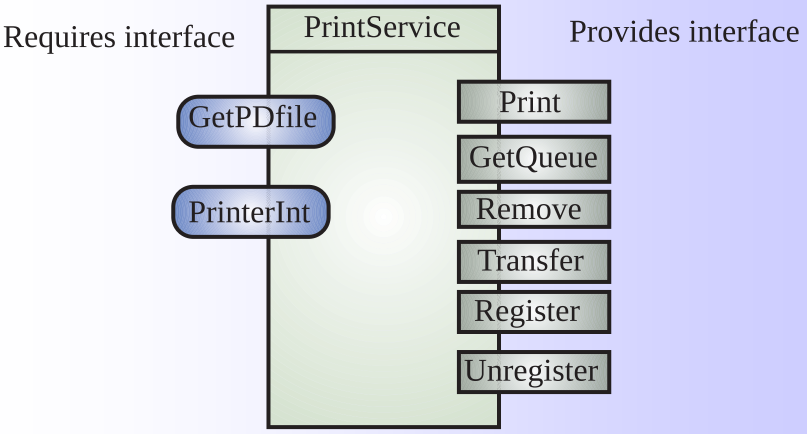
Provides interface



Component interfaces

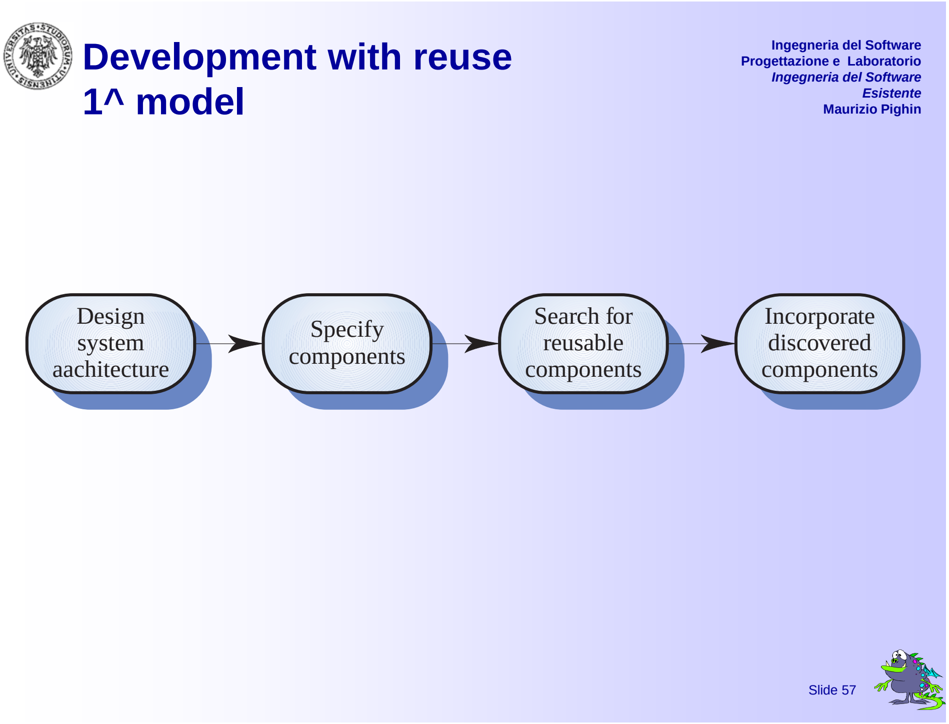
- Provides interface
  - *Defines the services that are provided by the component to other components*
- Requires interface
  - *Defines the services that specifies what services must be made available for the component to execute as specified*

Printing services component



CBSE processes

- Component-based development can be integrated into a standard software process by incorporating a reuse activity in the process
- However, in reuse-driven development, the system requirements are modified to reflect the components that are available
- CBSE usually involves a prototyping or an incremental development process with components being ‘glued together’ using a scripting language



Development with reuse
1^ model

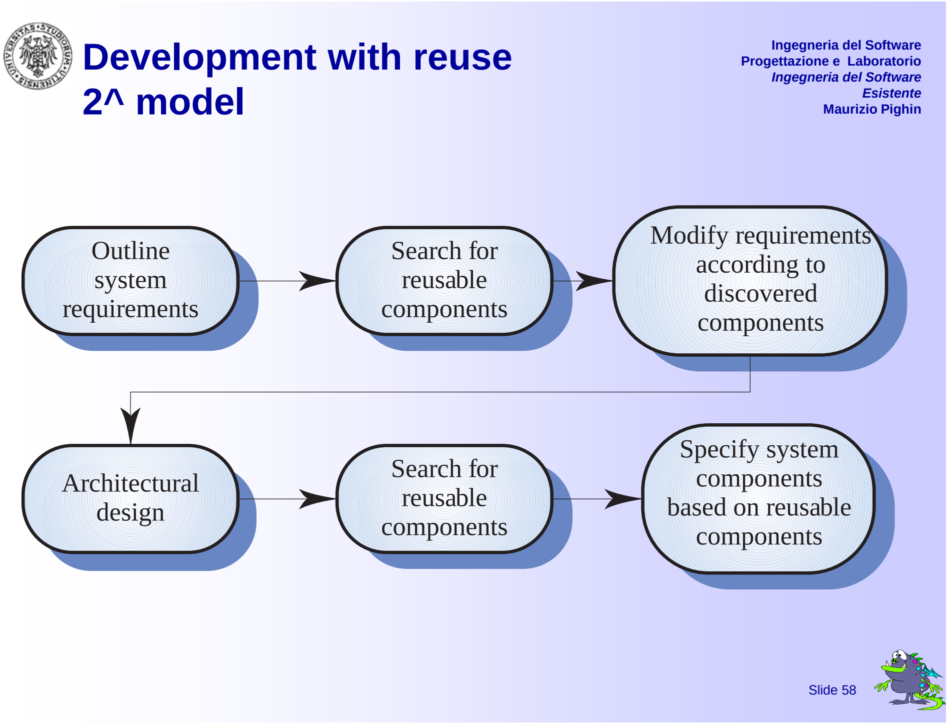
Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

Design system aachitecture

Specify components

Search for reusable components

Incorporate discovered components



Development with reuse
2^ model

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

Outline system requirements

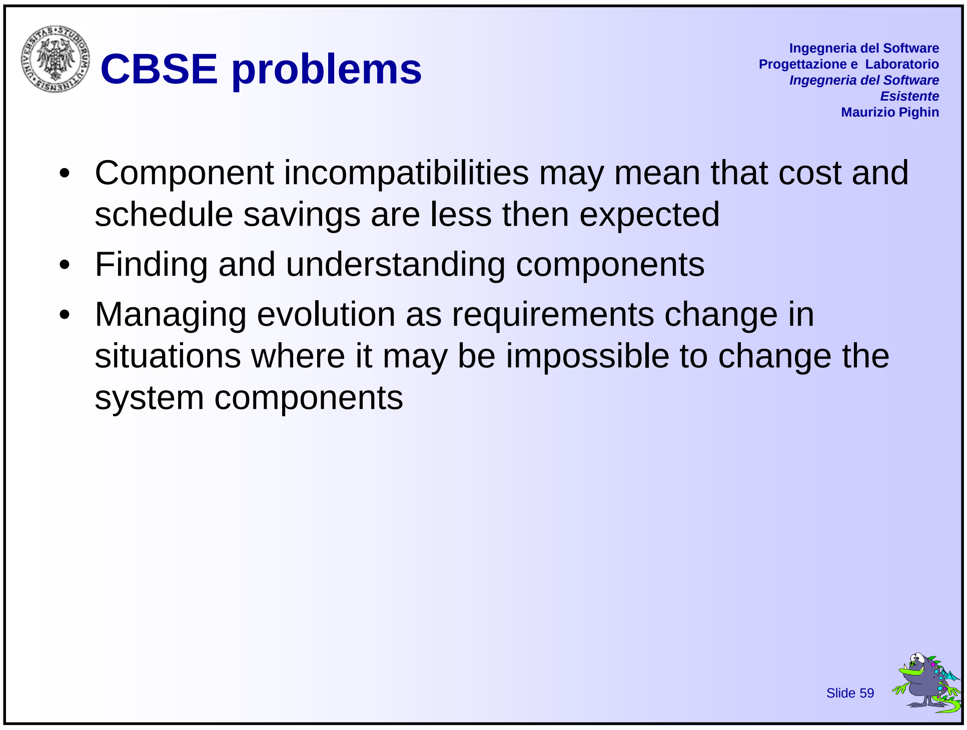
Search for reusable components

Modify requirements according to discovered components

Architectural design

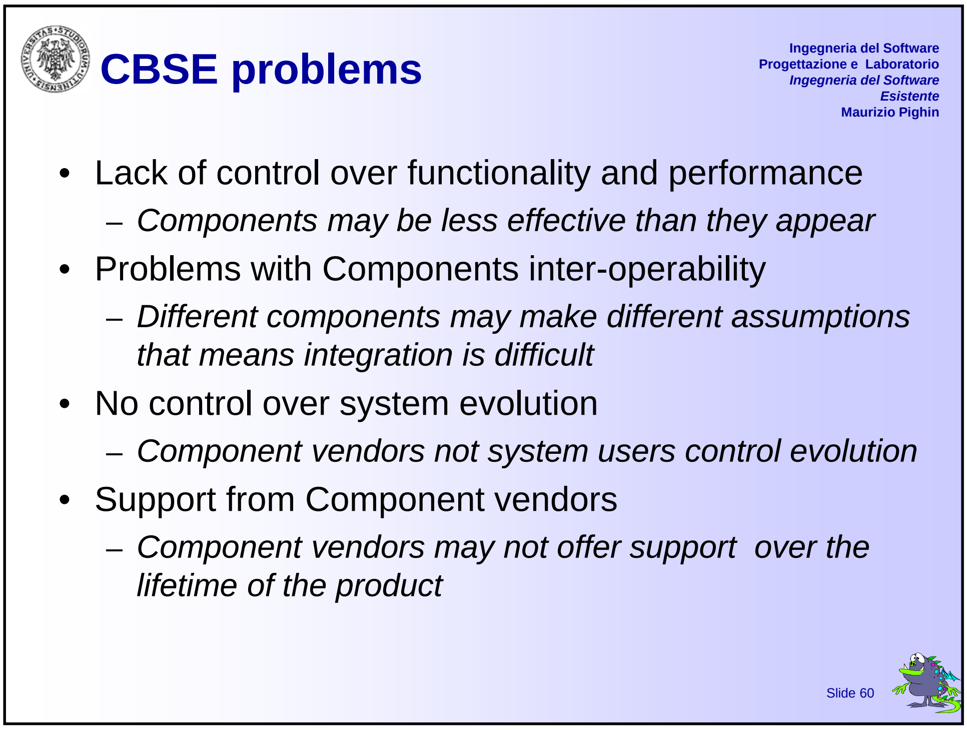
Search for reusable components

Specify system components based on reusable components



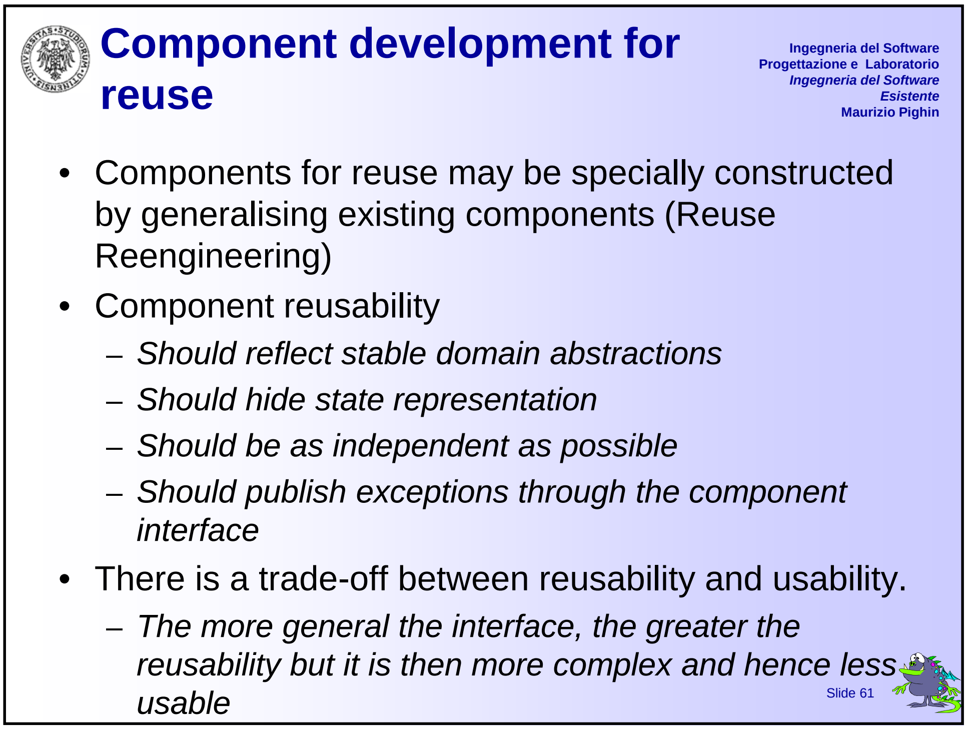
CBSE problems

- Component incompatibilities may mean that cost and schedule savings are less then expected
- Finding and understanding components
- Managing evolution as requirements change in situations where it may be impossible to change the system components



CBSE problems

- Lack of control over functionality and performance
  - *Components may be less effective than they appear*
- Problems with Components inter-operability
  - *Different components may make different assumptions that means integration is difficult*
- No control over system evolution
  - *Component vendors not system users control evolution*
- Support from Component vendors
  - *Component vendors may not offer support over the lifetime of the product*

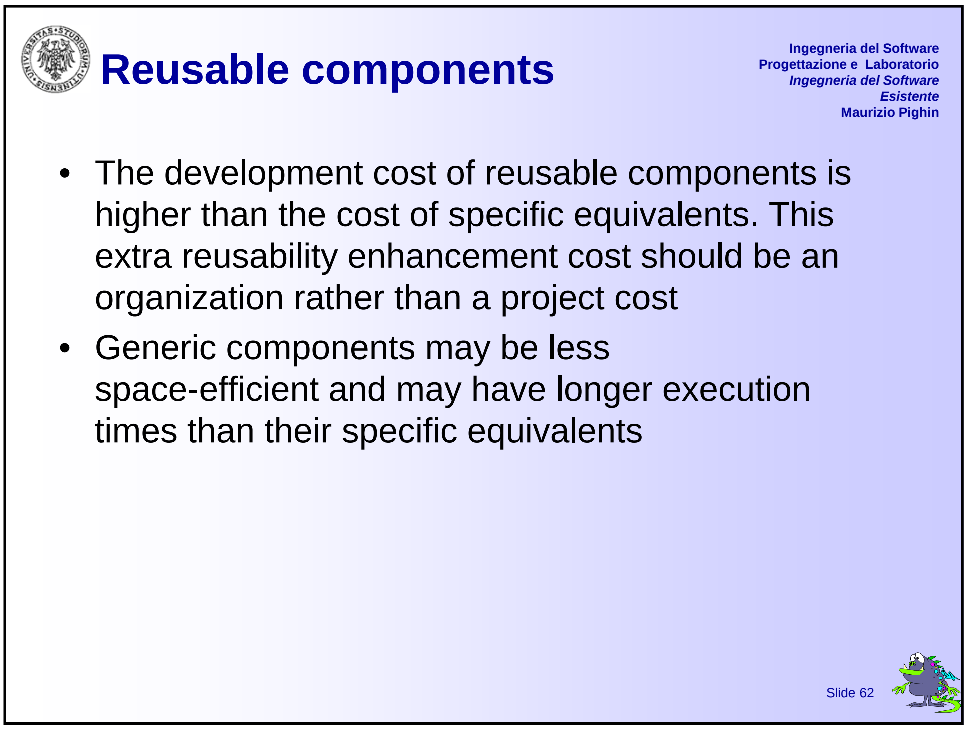


Component development for reuse

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

- Components for reuse may be specially constructed by generalising existing components (Reuse Reengineering)
- Component reusability
  - *Should reflect stable domain abstractions*
  - *Should hide state representation*
  - *Should be as independent as possible*
  - *Should publish exceptions through the component interface*
- There is a trade-off between reusability and usability.
  - *The more general the interface, the greater the reusability but it is then more complex and hence less usable*

Slide 61

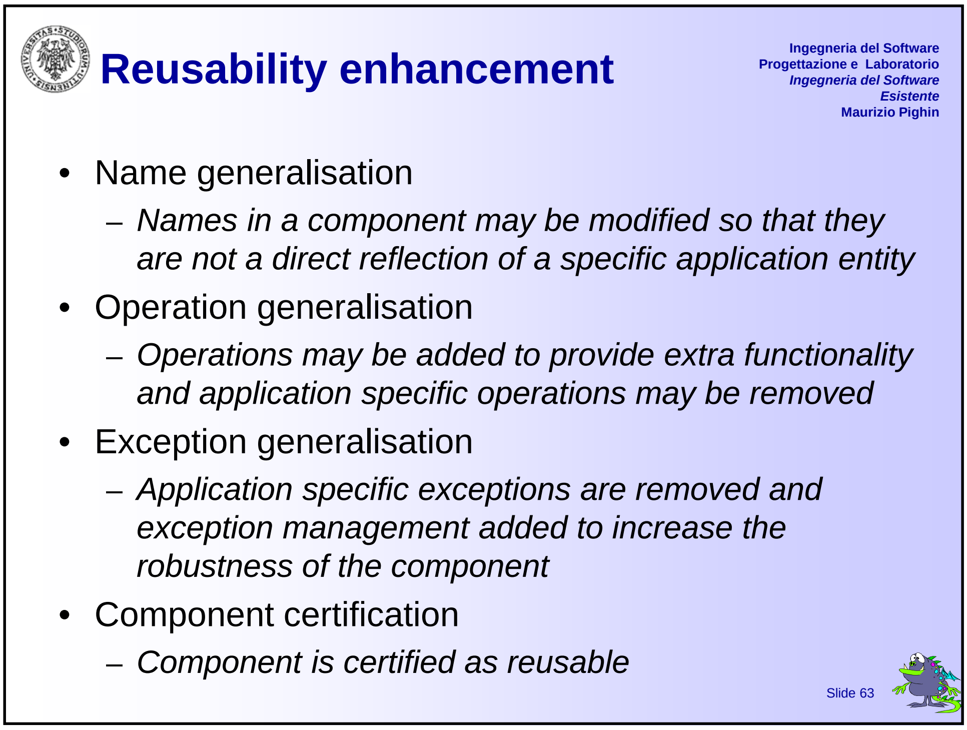


Reusable components

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

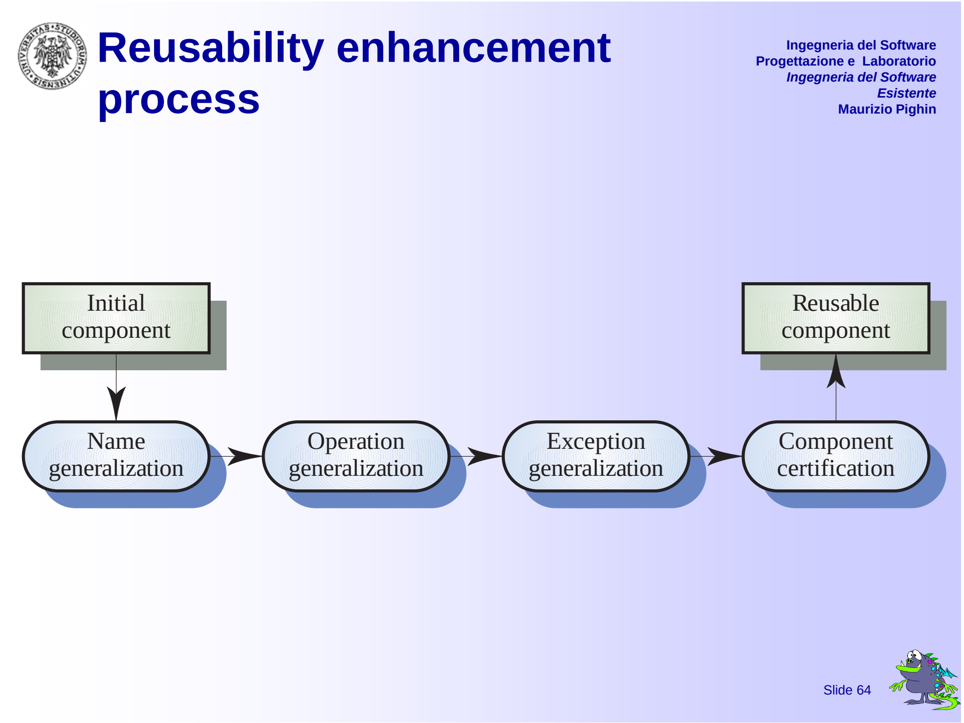
- The development cost of reusable components is higher than the cost of specific equivalents. This extra reusability enhancement cost should be an organization rather than a project cost
- Generic components may be less space-efficient and may have longer execution times than their specific equivalents

Slide 62

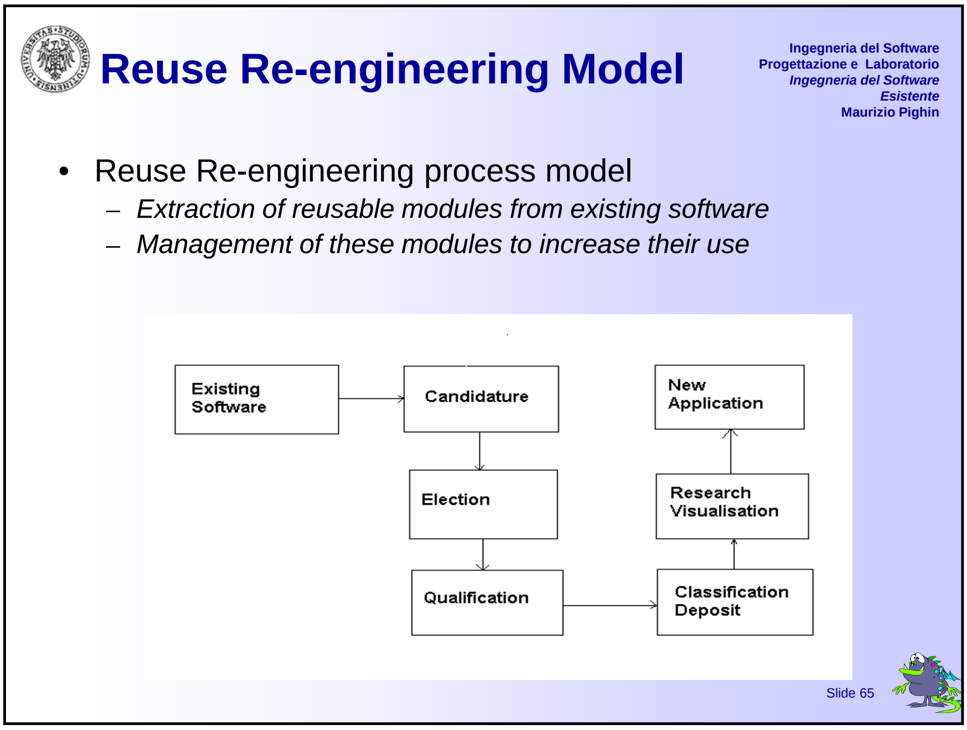


Reusability enhancement

- Name generalisation
  - *Names in a component may be modified so that they are not a direct reflection of a specific application entity*
- Operation generalisation
  - *Operations may be added to provide extra functionality and application specific operations may be removed*
- Exception generalisation
  - *Application specific exceptions are removed and exception management added to increase the robustness of the component*
- Component certification
  - *Component is certified as reusable*



Reusability enhancement process

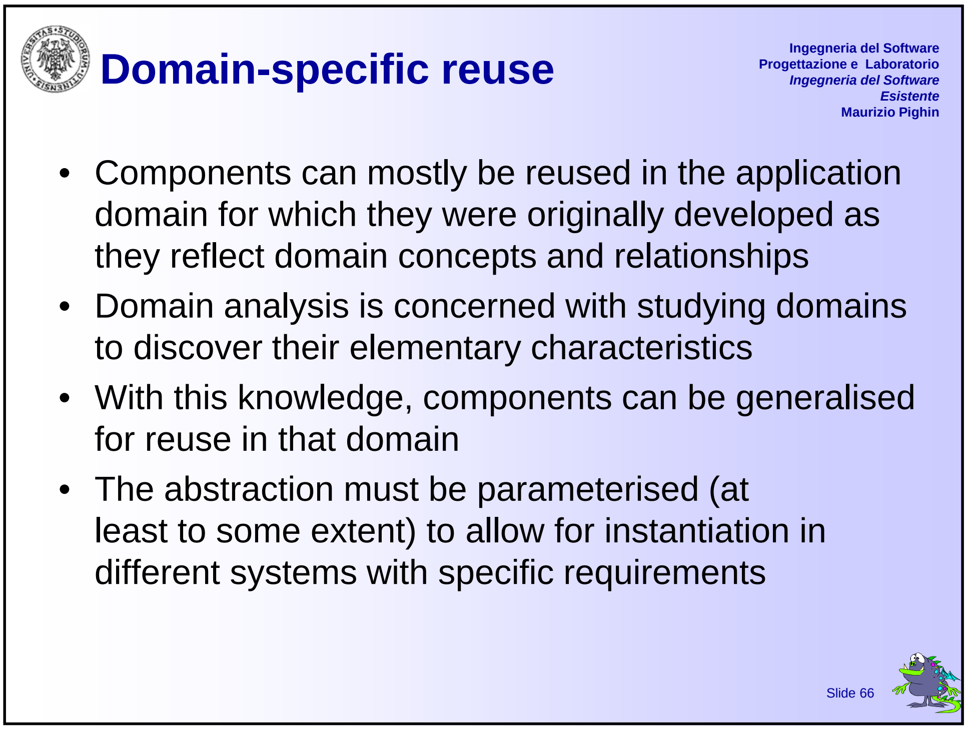


Reuse Re-engineering Model

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

- Reuse Re-engineering process model
  - *Extraction of reusable modules from existing software*
  - *Management of these modules to increase their use*

Slide 65

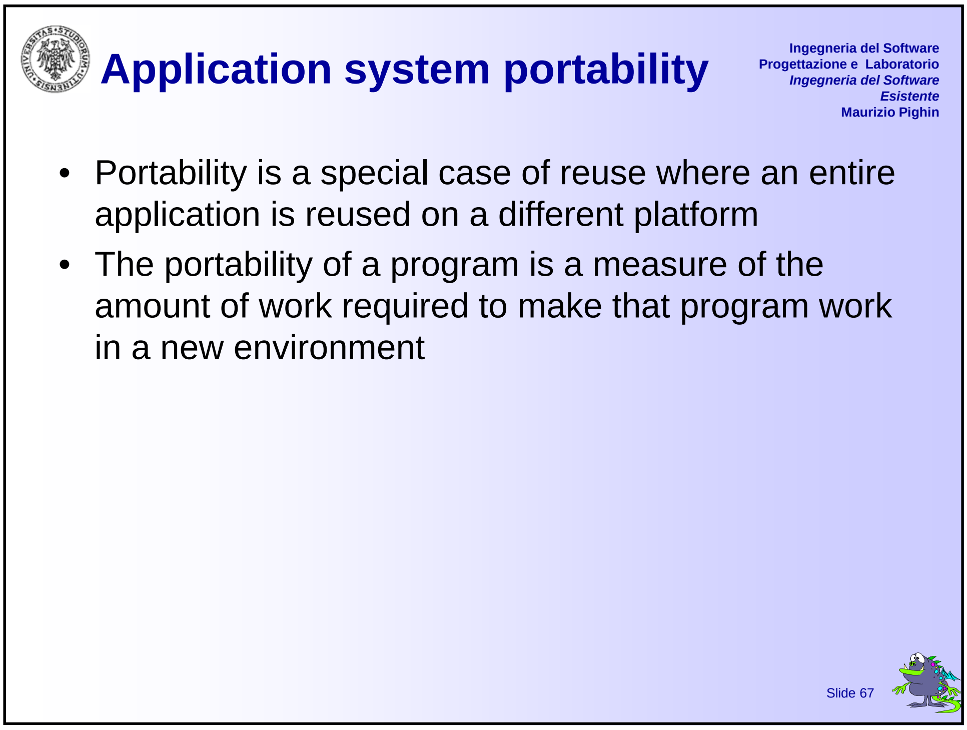


Domain-specific reuse

Ingegneria del Software
Progettazione e Laboratorio
Ingegneria del Software
Esistente
Maurizio Pighin

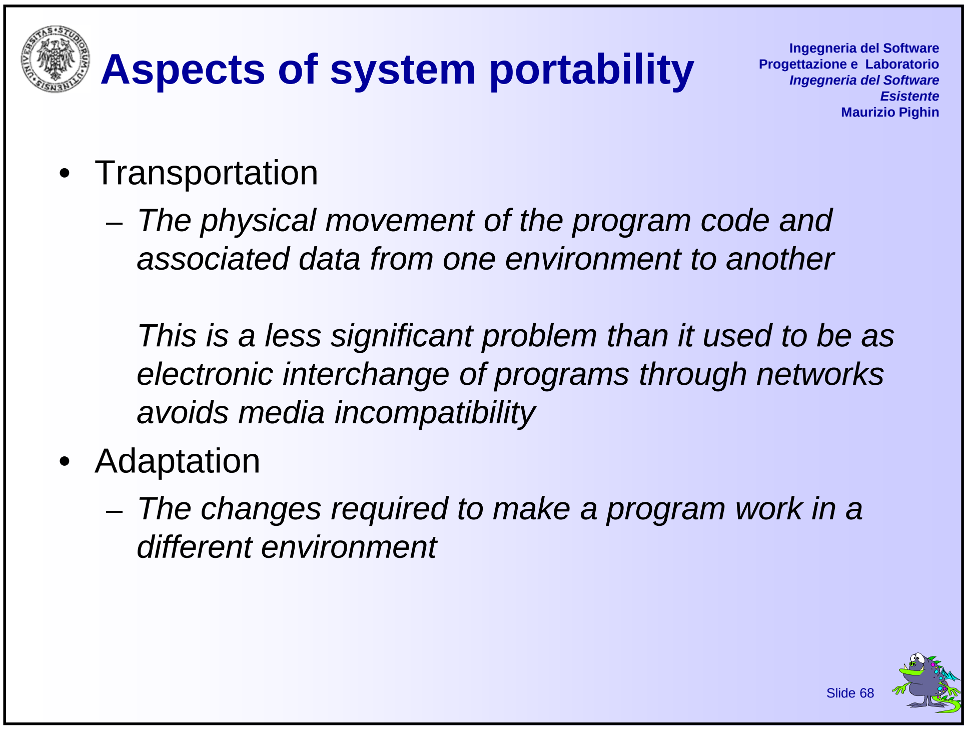
- Components can mostly be reused in the application domain for which they were originally developed as they reflect domain concepts and relationships
- Domain analysis is concerned with studying domains to discover their elementary characteristics
- With this knowledge, components can be generalised for reuse in that domain
- The abstraction must be parameterised (at least to some extent) to allow for instantiation in different systems with specific requirements

Slide 66



Application system portability

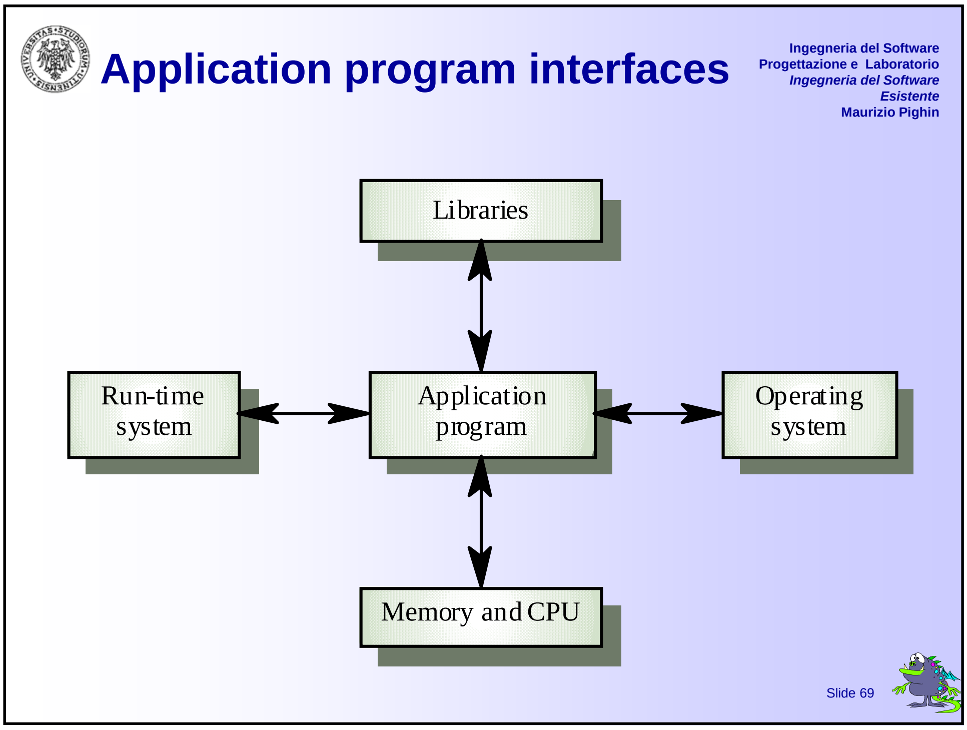
- Portability is a special case of reuse where an entire application is reused on a different platform
- The portability of a program is a measure of the amount of work required to make that program work in a new environment



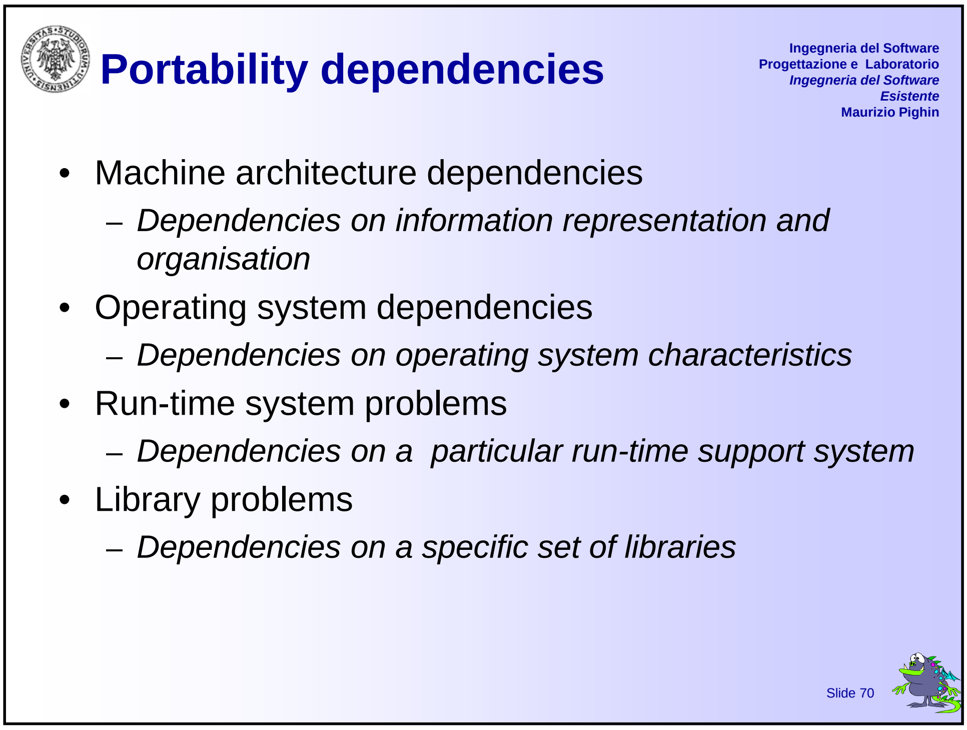
Aspects of system portability

- Transportation
  - *The physical movement of the program code and associated data from one environment to another*

    This is a less significant problem than it used to be as electronic interchange of programs through networks avoids media incompatibility
- Adaptation
  - *The changes required to make a program work in a different environment*



Application program interfaces



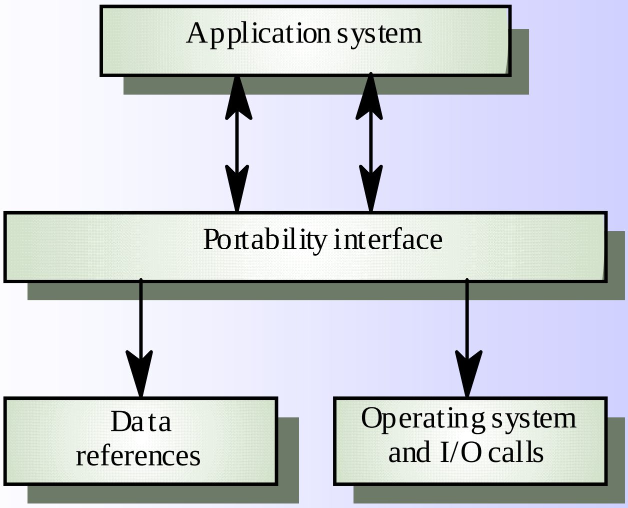
Portability dependencies

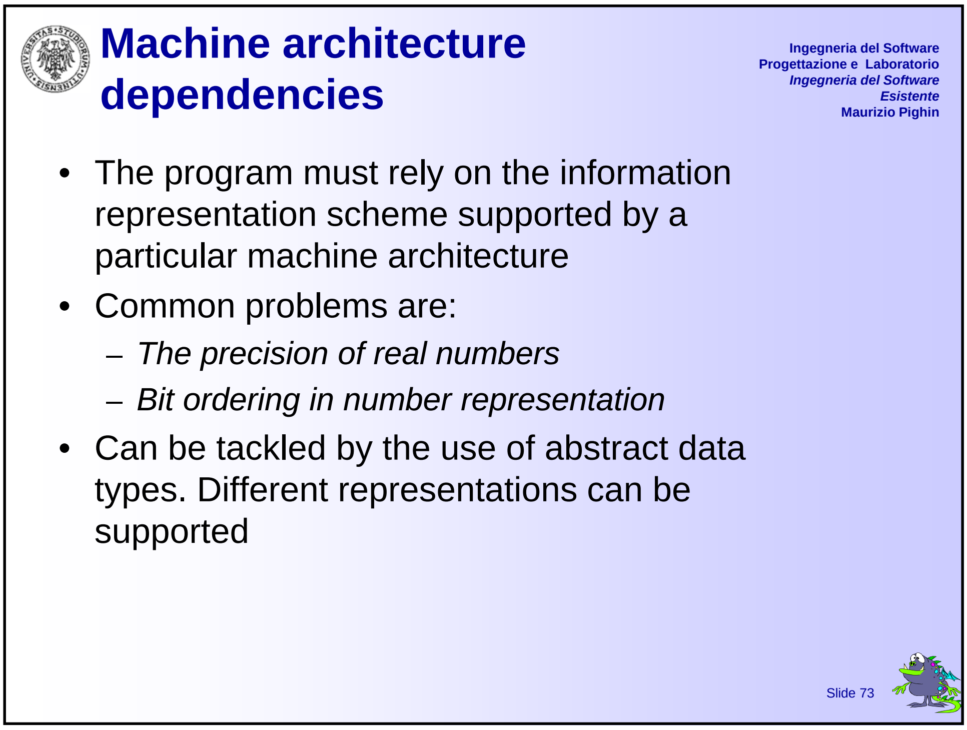
- Machine architecture dependencies
  - *Dependencies on information representation and organisation*
- Operating system dependencies
  - *Dependencies on operating system characteristics*
- Run-time system problems
  - *Dependencies on a particular run-time support system*
- Library problems
  - *Dependencies on a specific set of libraries*

Development for portability

- Isolate parts of the system which are dependent on the external program interfaces. These interfaces should be implemented as a set of abstract data types or objects
- Define a portability interface to hide machine architecture and operating system characteristics
- To port the program, only the code behind the portability interface need be rewritten

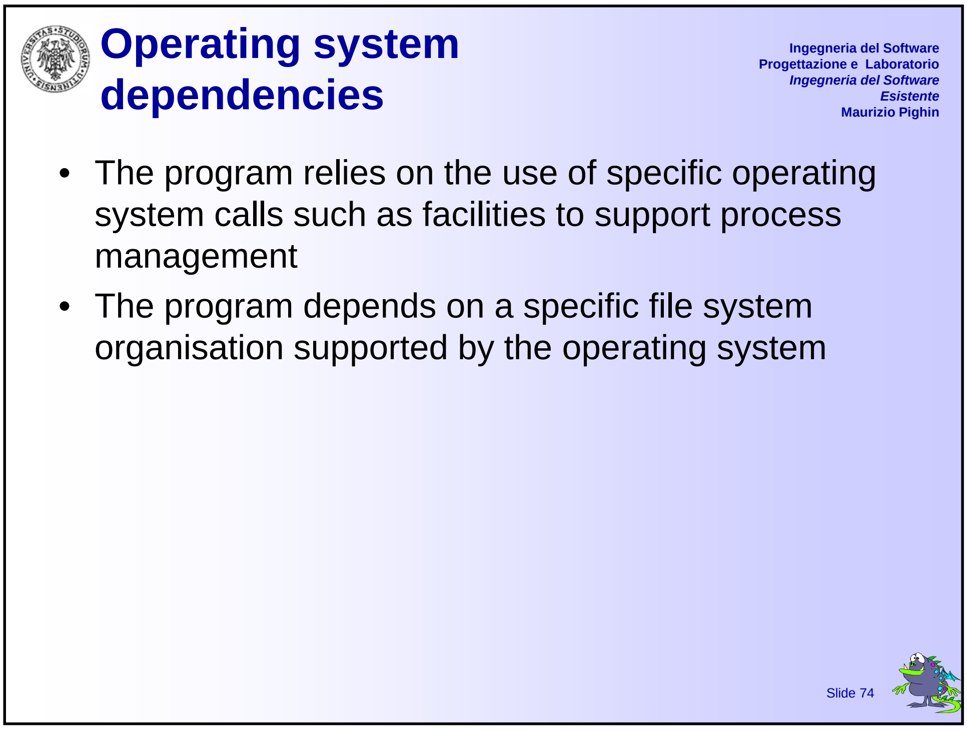
A portability interface





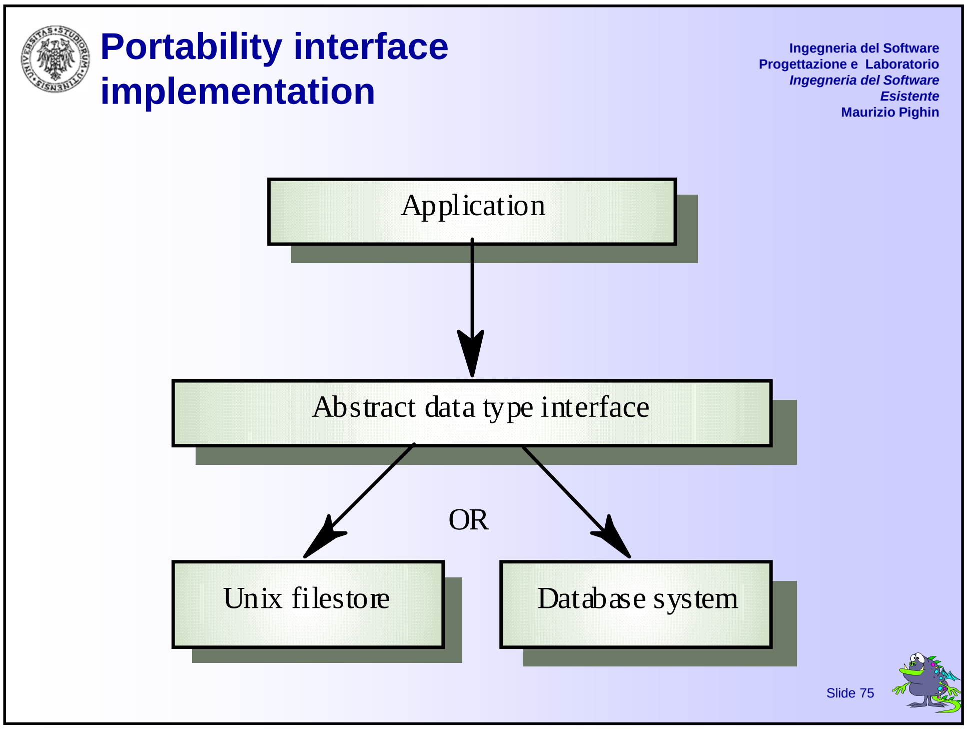
Machine architecture dependencies

- The program must rely on the information representation scheme supported by a particular machine architecture
- Common problems are:
  - *The precision of real numbers*
  - *Bit ordering in number representation*
- Can be tackled by the use of abstract data types. Different representations can be supported

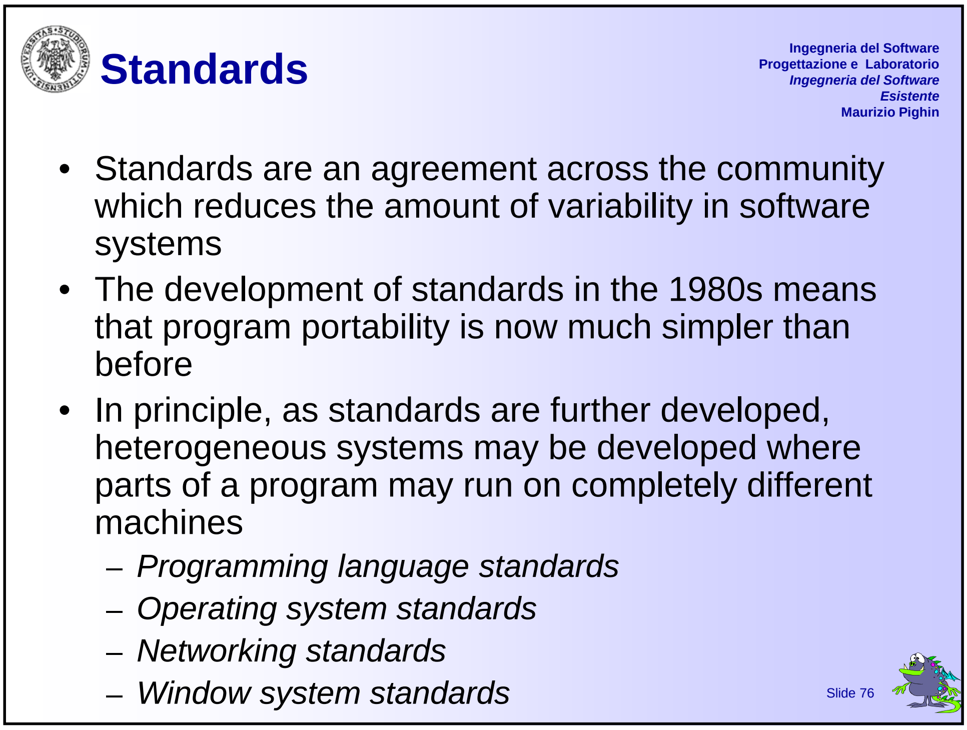


Operating system dependencies

- The program relies on the use of specific operating system calls such as facilities to support process management
- The program depends on a specific file system organisation supported by the operating system



Portability interface implementation



Standards

- Standards are an agreement across the community which reduces the amount of variability in software systems
- The development of standards in the 1980s means that program portability is now much simpler than before
- In principle, as standards are further developed, heterogeneous systems may be developed where parts of a program may run on completely different machines
  - *Programming language standards*
  - *Operating system standards*
  - *Networking standards*
  - *Window system standards*