Chapter 28

Software Re-engineering

Software re-engineering

- Reorganising and modifying existing software systems to make them more maintainable

Objectives

- To explain why software re-engineering is a cost-effective option for system evolution
- To describe the activities involved in the software re-engineering process
- To distinguish between software and data re-engineering and to explain the problems of data re-engineering

Topics covered

- Source code translation
- Reverse engineering
- Program structure improvement
- Program modularisation
- Data re-engineering

System re-engineering

- 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

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

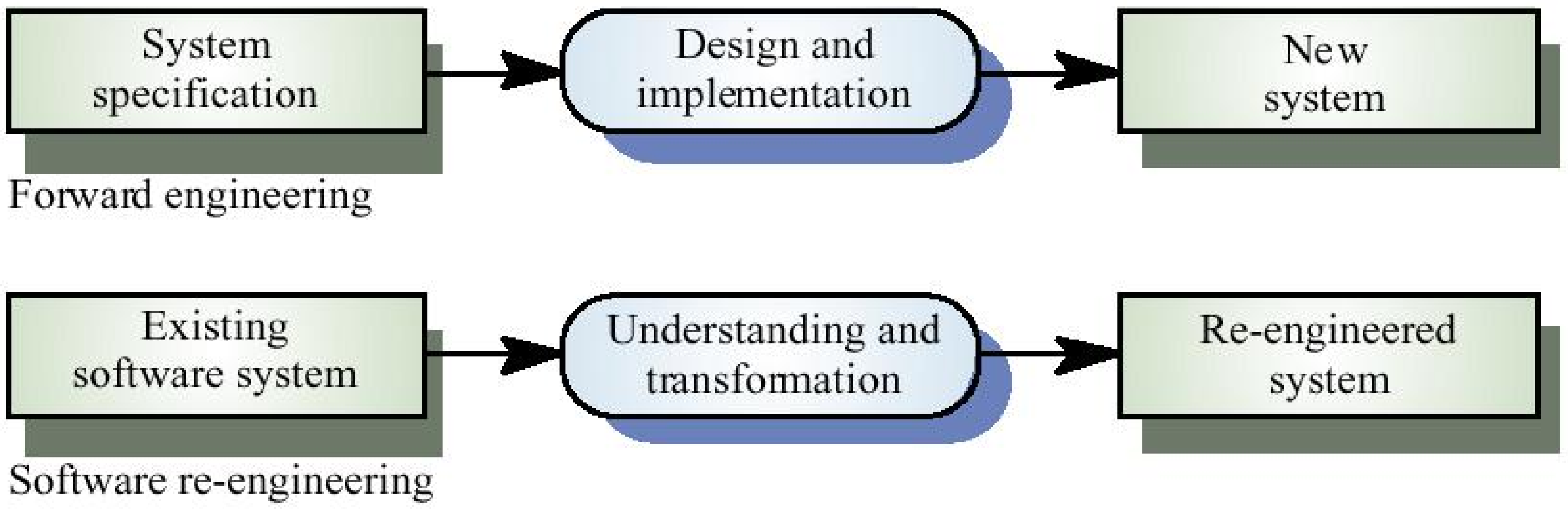
Re-engineering advantages

- 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

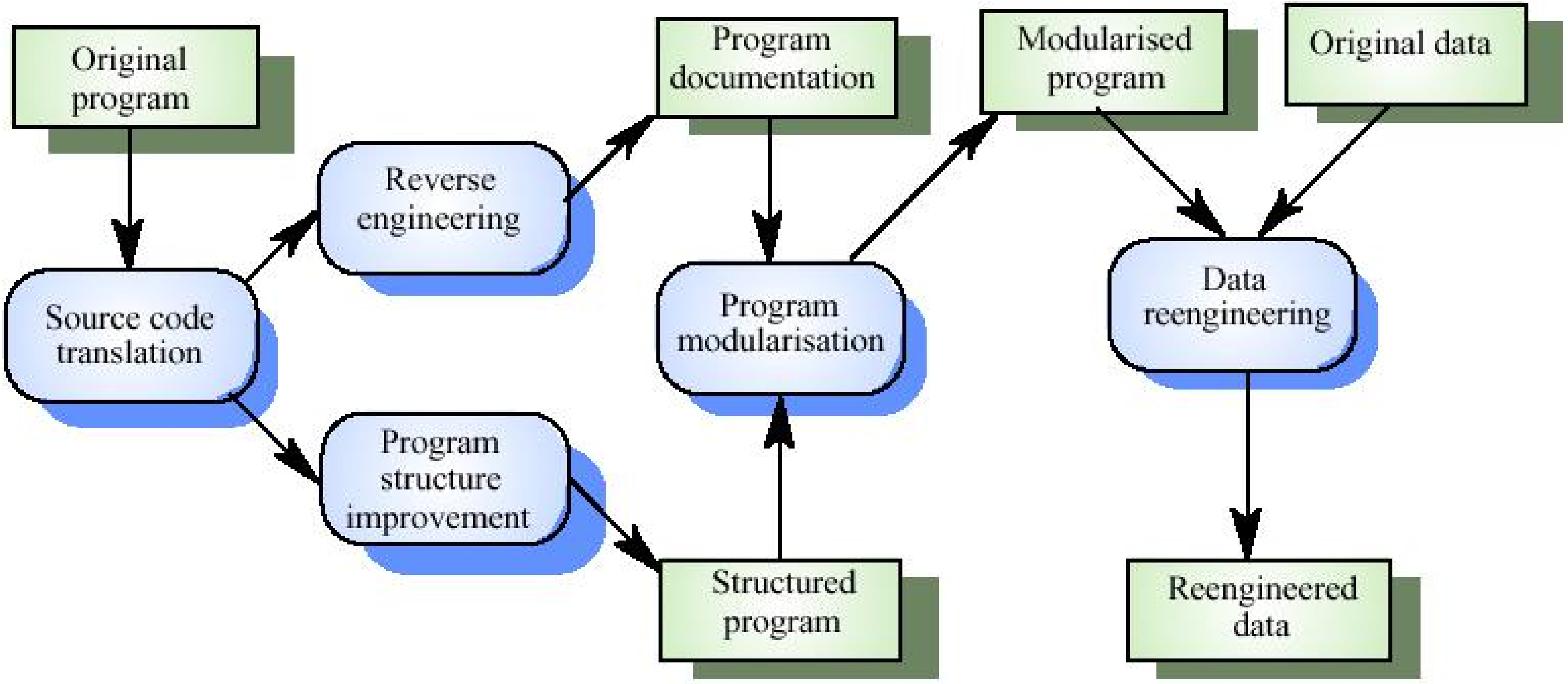
Business process re-engineering

- 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



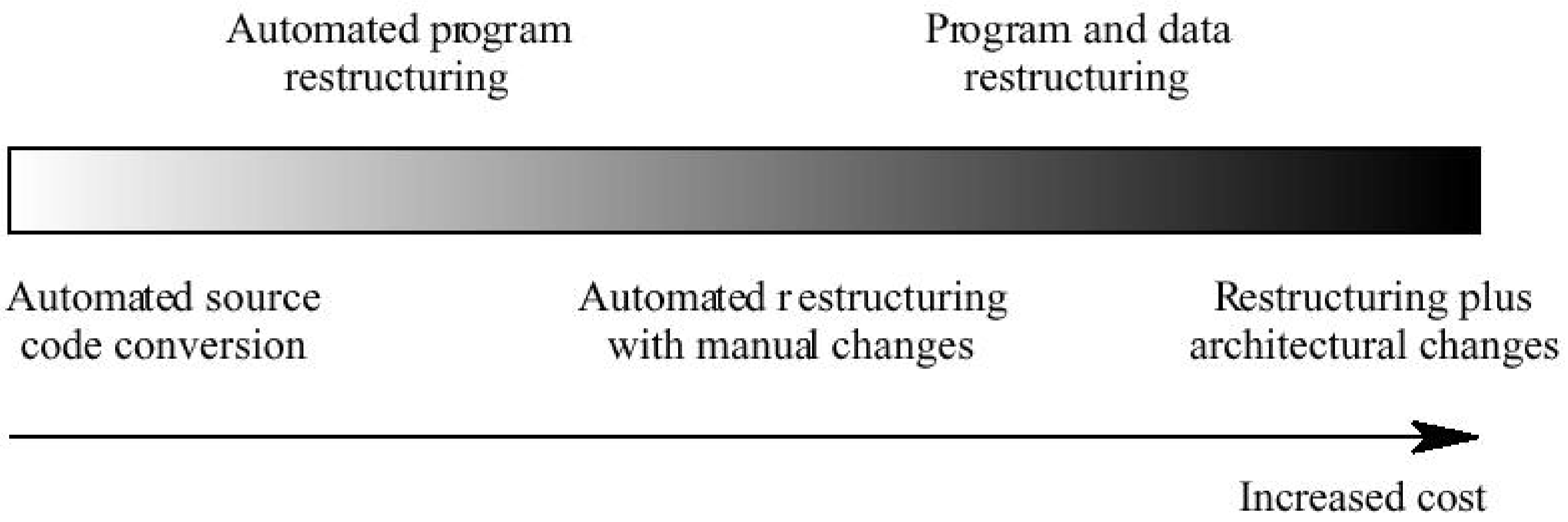
The re-engineering process



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

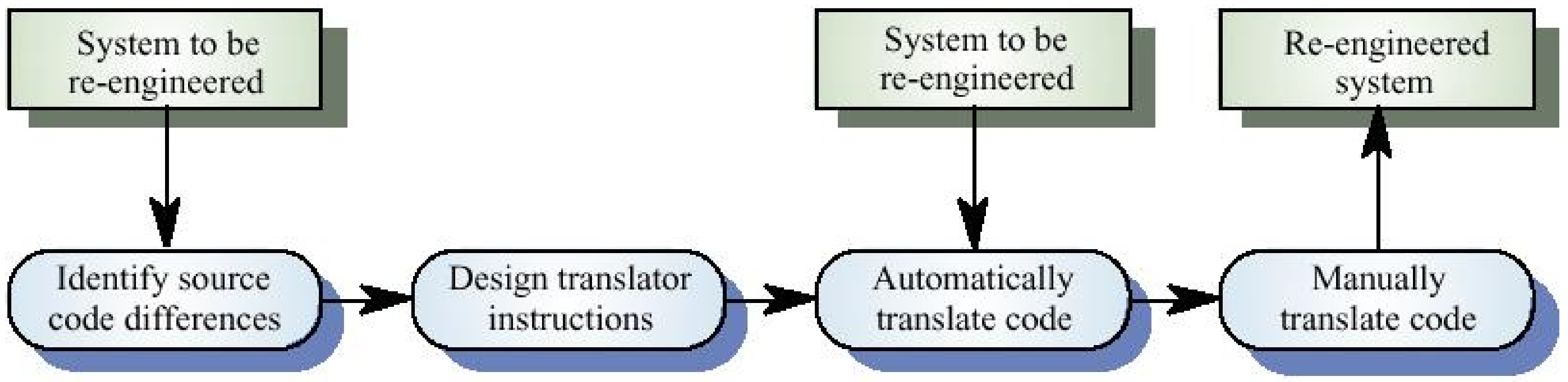
Re-engineering approaches


Automated program restructuring
Program and data restructuring
Automated source code conversion
Automated r estructuring 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

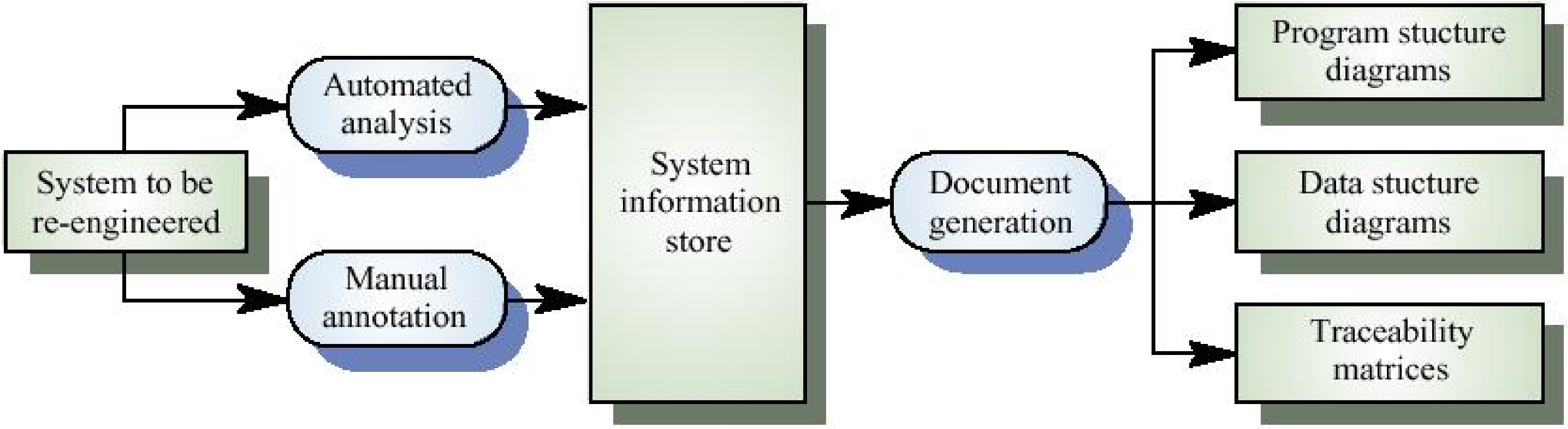
The program translation process


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

- 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



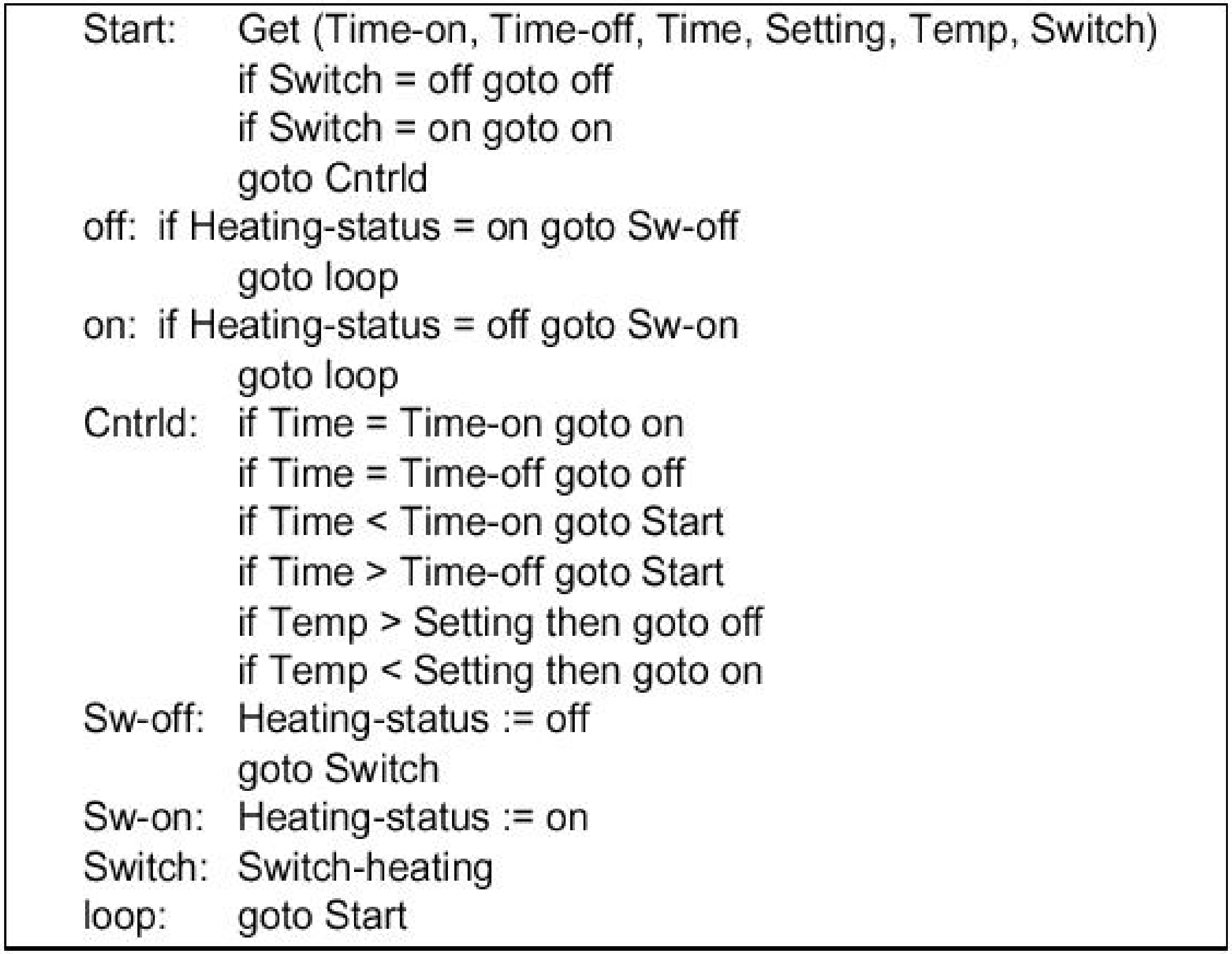
Reverse engineering

l 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

Program structure improvement

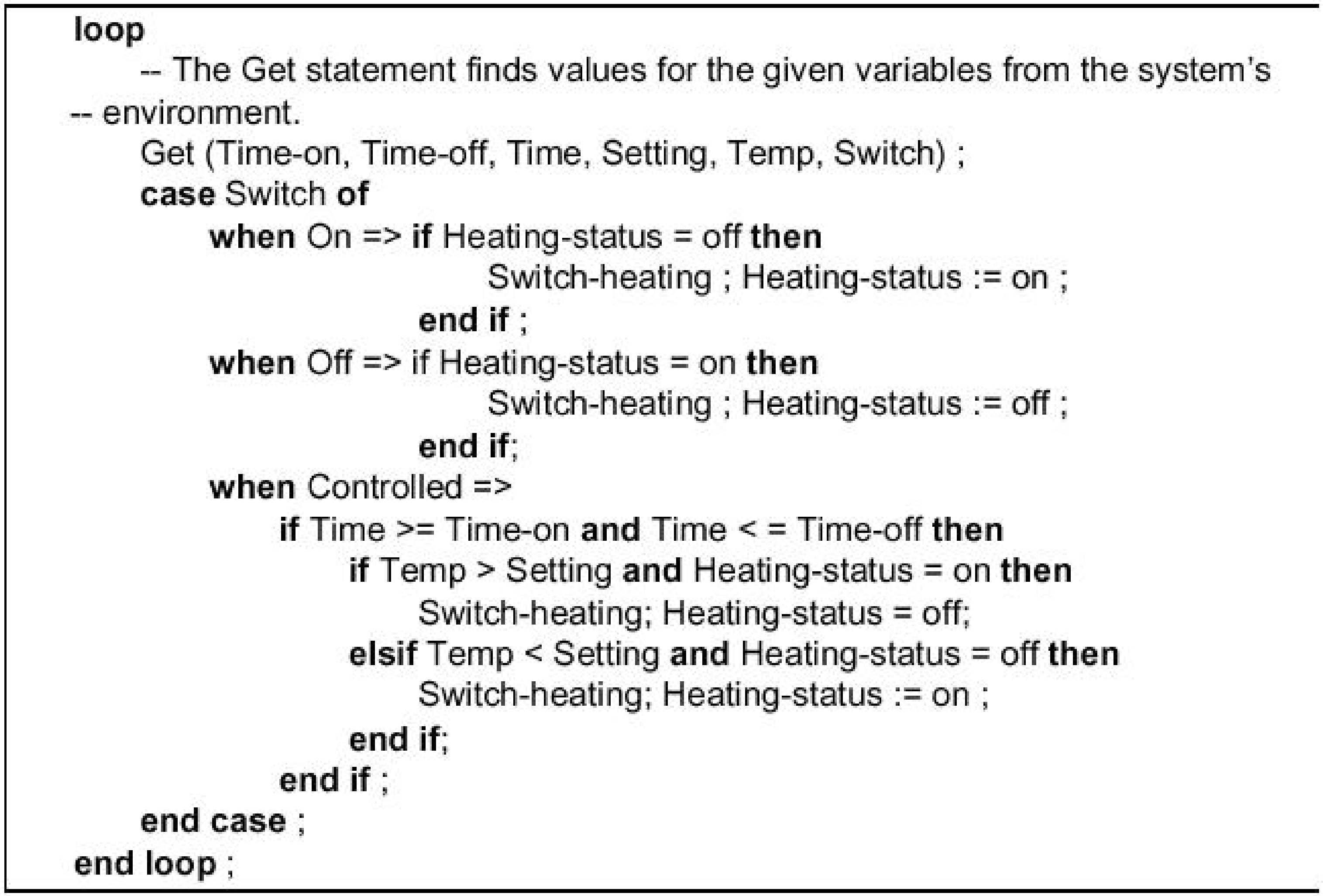
- Maintenance tends to corrupt the structure of a program. It becomes harder and harder to understand
- The program may be automatically restructured to remove unconditional branches
- Conditions may be simplified to make them more readable

Spaghetti logic



```
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
```

Structured control logic



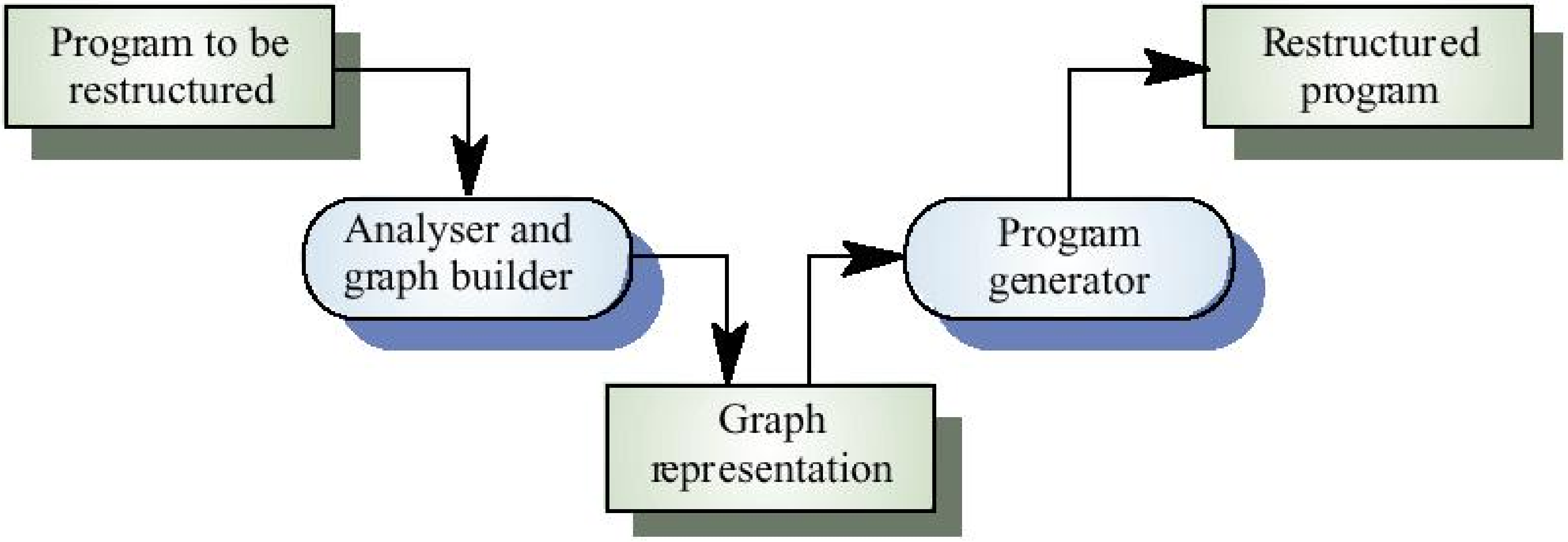
```
loop
        -- The Get statement finds values for the given variables from the system's
-- environment.
        Get (Time-on, Time-off, Time, Setting, Temp, Switch) ;
        case Switch of
                when On => if Heating-status = off then
                                        Switch-heating ; 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 ;
```

Condition simplification

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

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

Automatic program restructuring



Restructuring problems

- 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

Program modularisation

- 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

Module types

- Data abstractions
  - Abstract data types where datastructures 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

Recovering data abstractions

- 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
  - Analyse common data areas to identify logical abstractions
  - Create an ADT or object for these abstractions
  - Use a browser to find all data references and replace with reference to the data abstraction

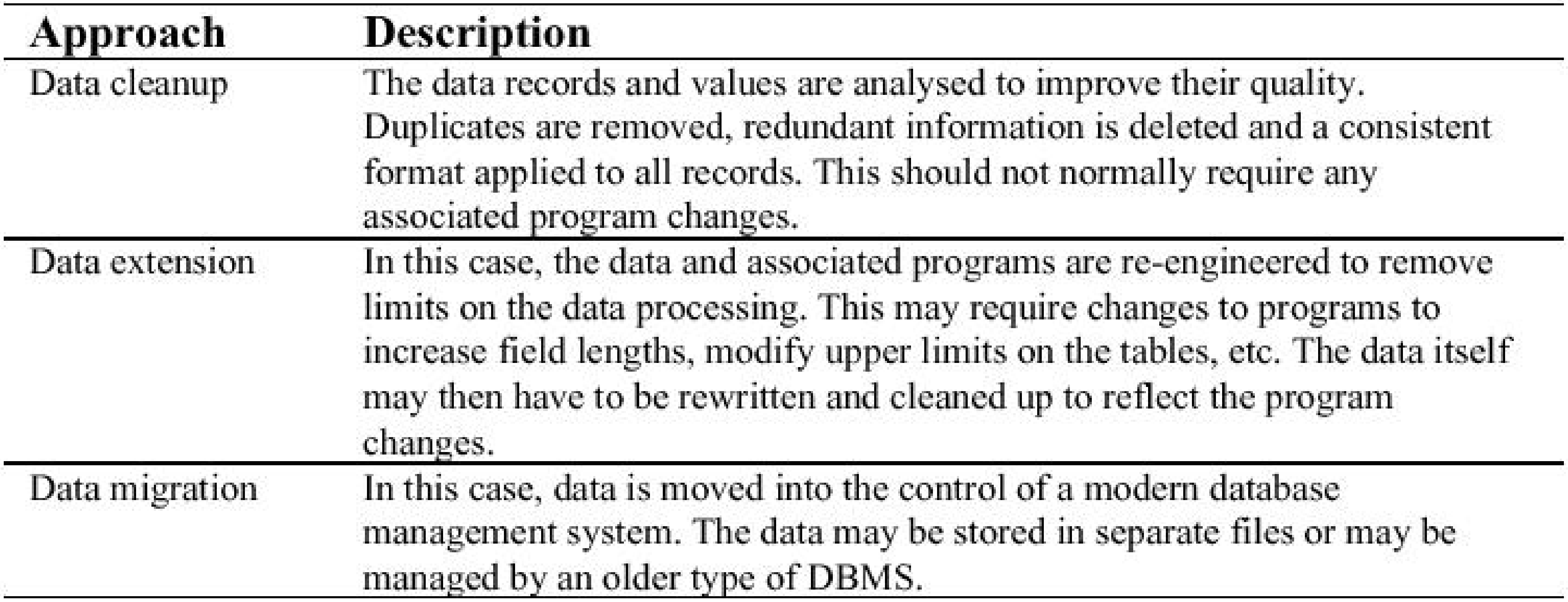
Data abstraction recovery

- 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

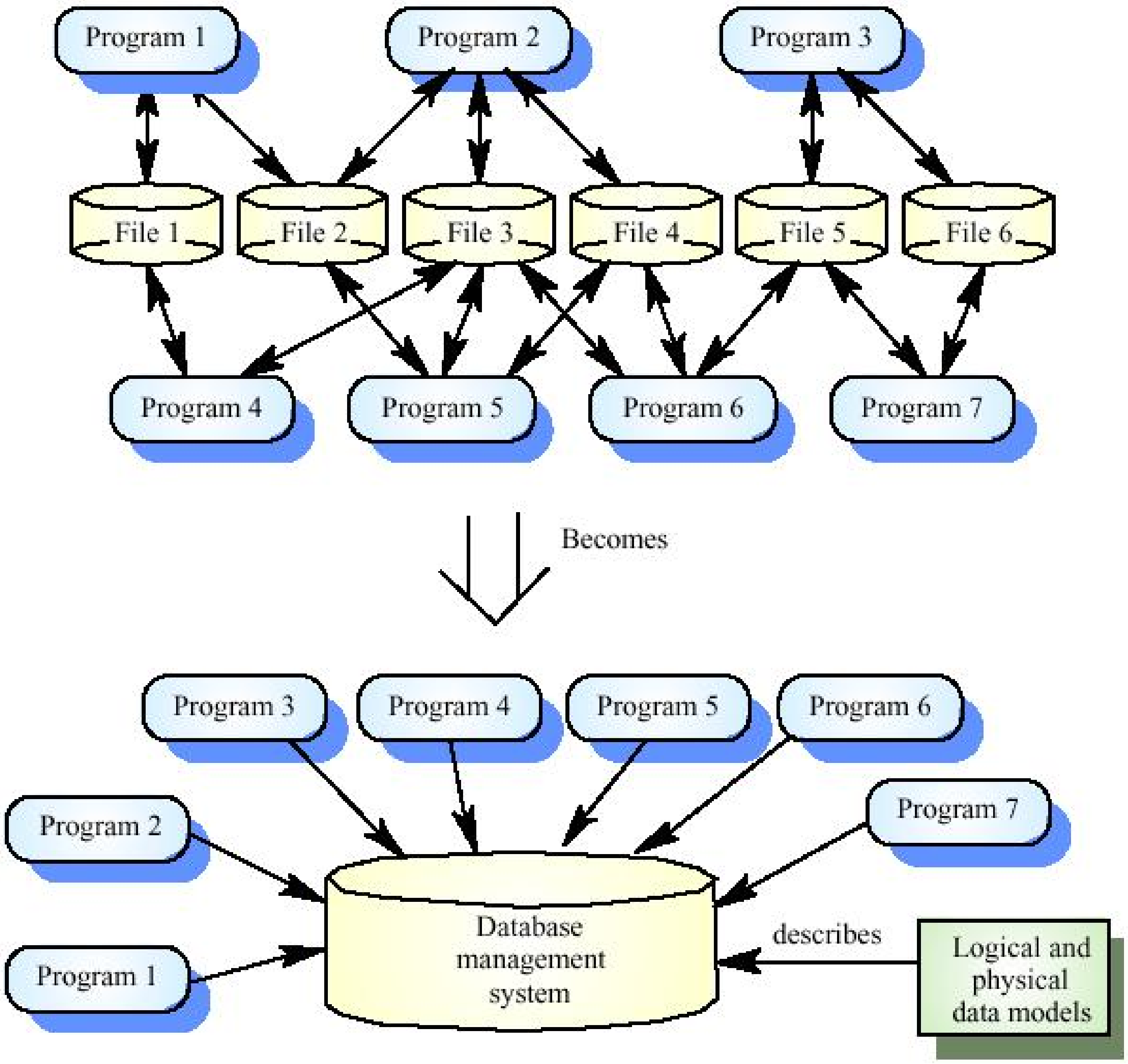
Approaches to data re-engineering



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

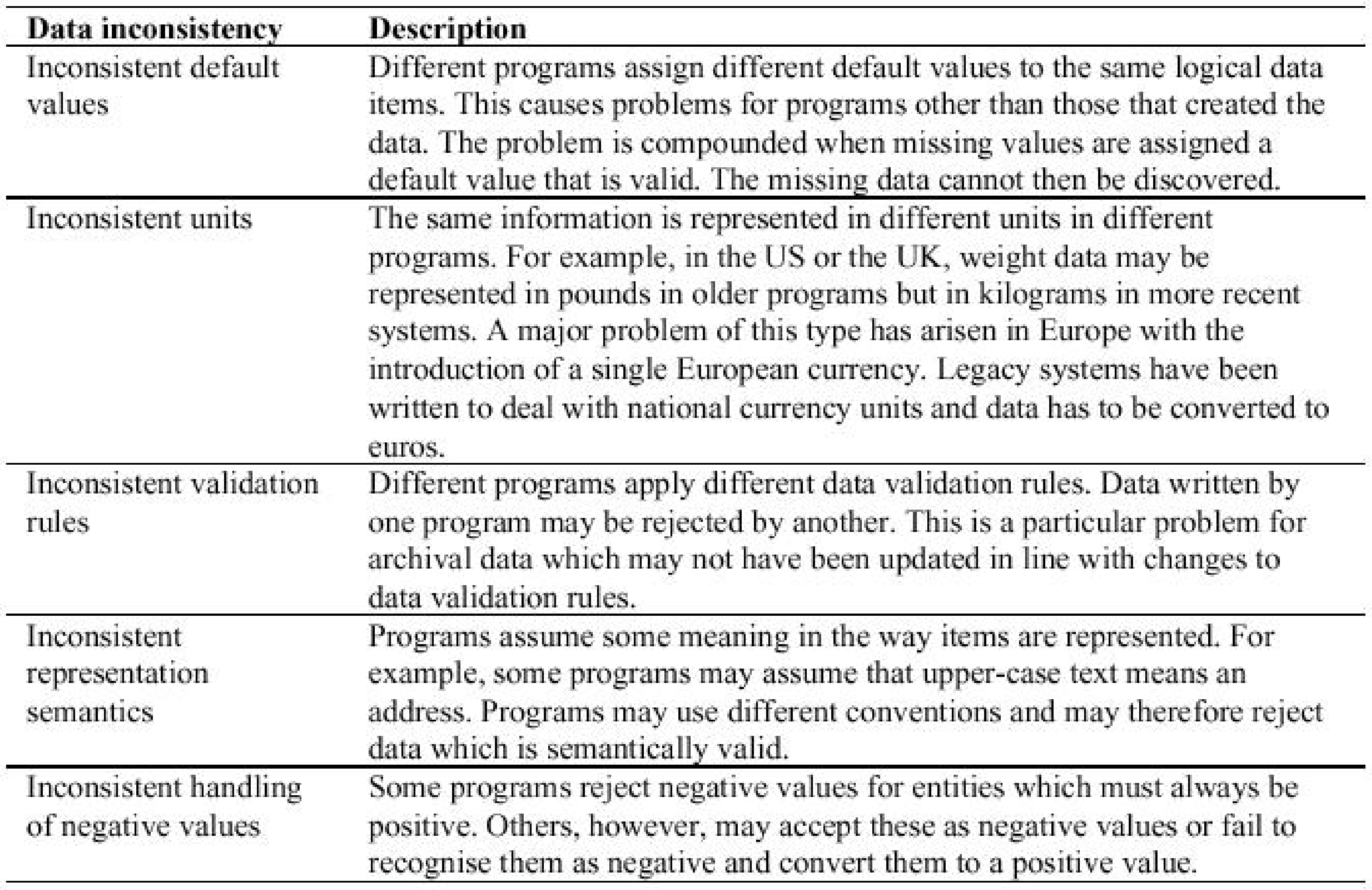


Data migration

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

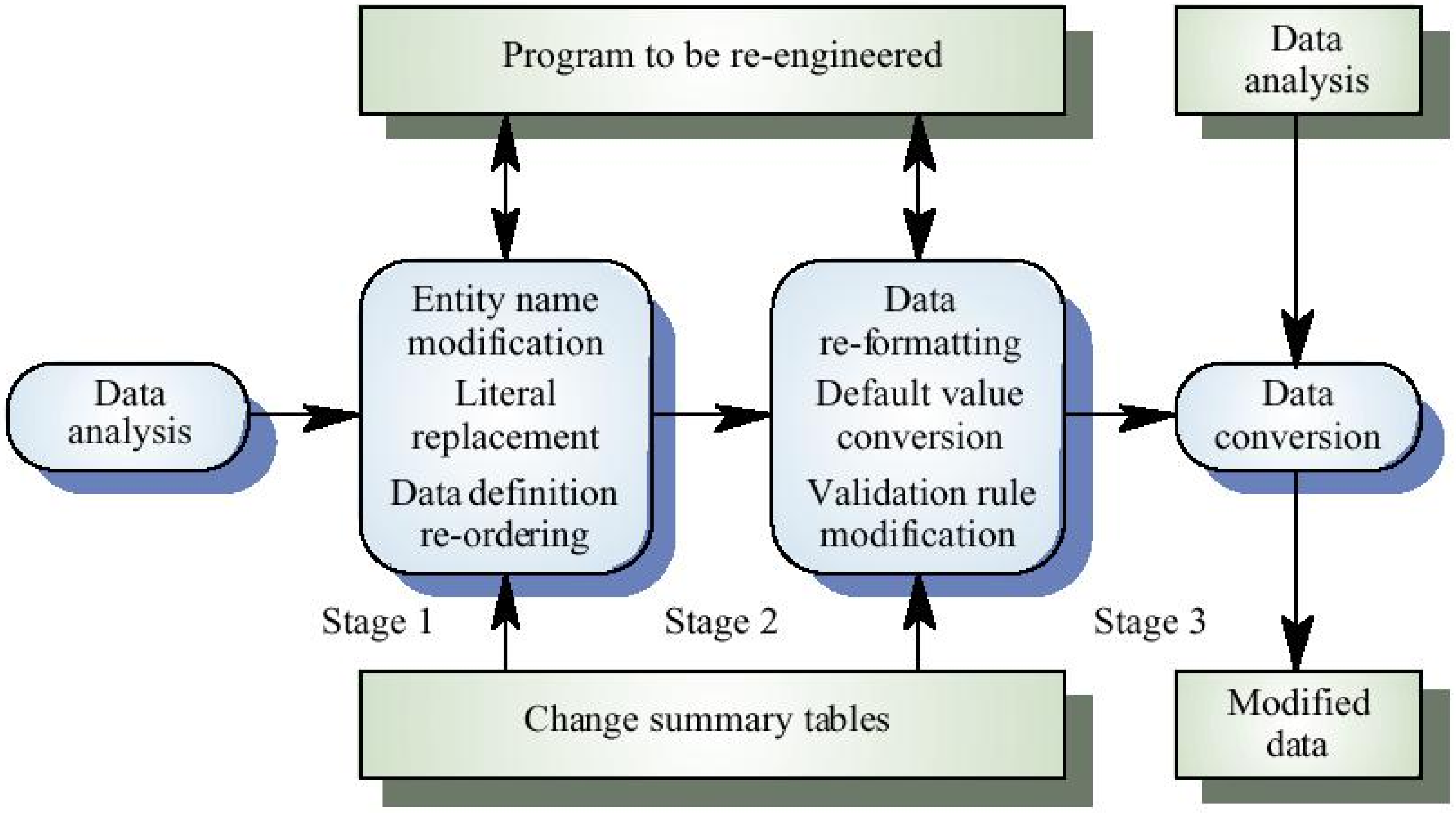


Data inconsistency	Description
Inconsistent default values	Different programs assign different default values to the same logical data items. This causes problems for programs other than those that created the data. The problem is compounded when missing values are assigned a default value that is valid. 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.

Data conversion

- 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

The data re-engineering process



Key points

- The objective of re-engineering is to improve the system structure to make it easier to understand and maintain
- The re-engineering process involves source code translation, reverse engineering, program structure improvement, program modularisation and data re-engineering
- Source code translation is the automatic conversion of of program in one language to another

Key points

- Reverse engineering is the process of deriving the system design and specification from its source code
- Program structure improvement replaces unstructured control constructs with while loops and simple conditionals
- Program modularisation involves reorganisation to group related items
- Data re-engineering may be necessary because of inconsistent data management