SOFTWARE RE-ENGINEERING

Unit 4

SOFTWARE REENGINEERING STRATEGIES

- Three strategies that specify the basic steps of reengineering are:
- **rewrite**,
- **rework**, and
- **replace**.

REWRITE STRATEGY:

- This strategy reflects the principle of alteration.
- An operational system is transformed into a new system, while
- preserving the abstraction level of the original system.
- For example, the Fortran code of a system can be rewritten in the C language.
- Internal schema, business logic, flow, dependencies, internal structures are not changed.
- May have some cosmetic changes/additions to improve look and feel of the system.

(a) Rewrite

REWORK STRATEGY:

- The rework strategy applies all the three principles.
- Let the goal of a reengineering project is to:
- Replace the unstructured control flow
- Constructs new flow
- Remove expensive codes (complex)
- Remove any extra iterative and conditional flows
- Make function calls effective.

REWORK STRATEGY:

- A classical, rework strategy based approach is as follows:

  Step 1

  - Application of abstraction: By parsing the code, generate a control-flow graph (CFG) for the given system.

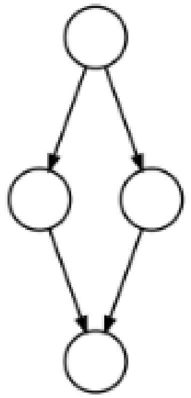
  Step 2

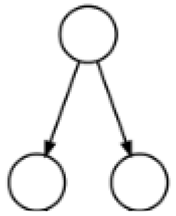
  - Application of alteration: Apply a restructuring algorithm to the control-flow graph to produce a structured control-flow graph.

  Step 3

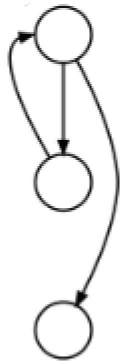
  - Application of refinement: Translate the new, structured control-flow graph back into the original programming language.

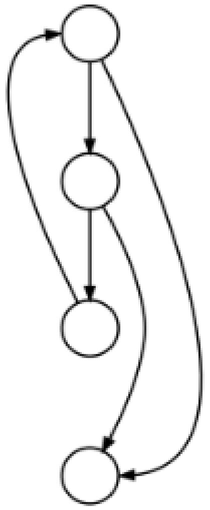
if-then-else



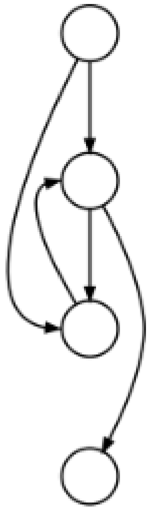


a loop

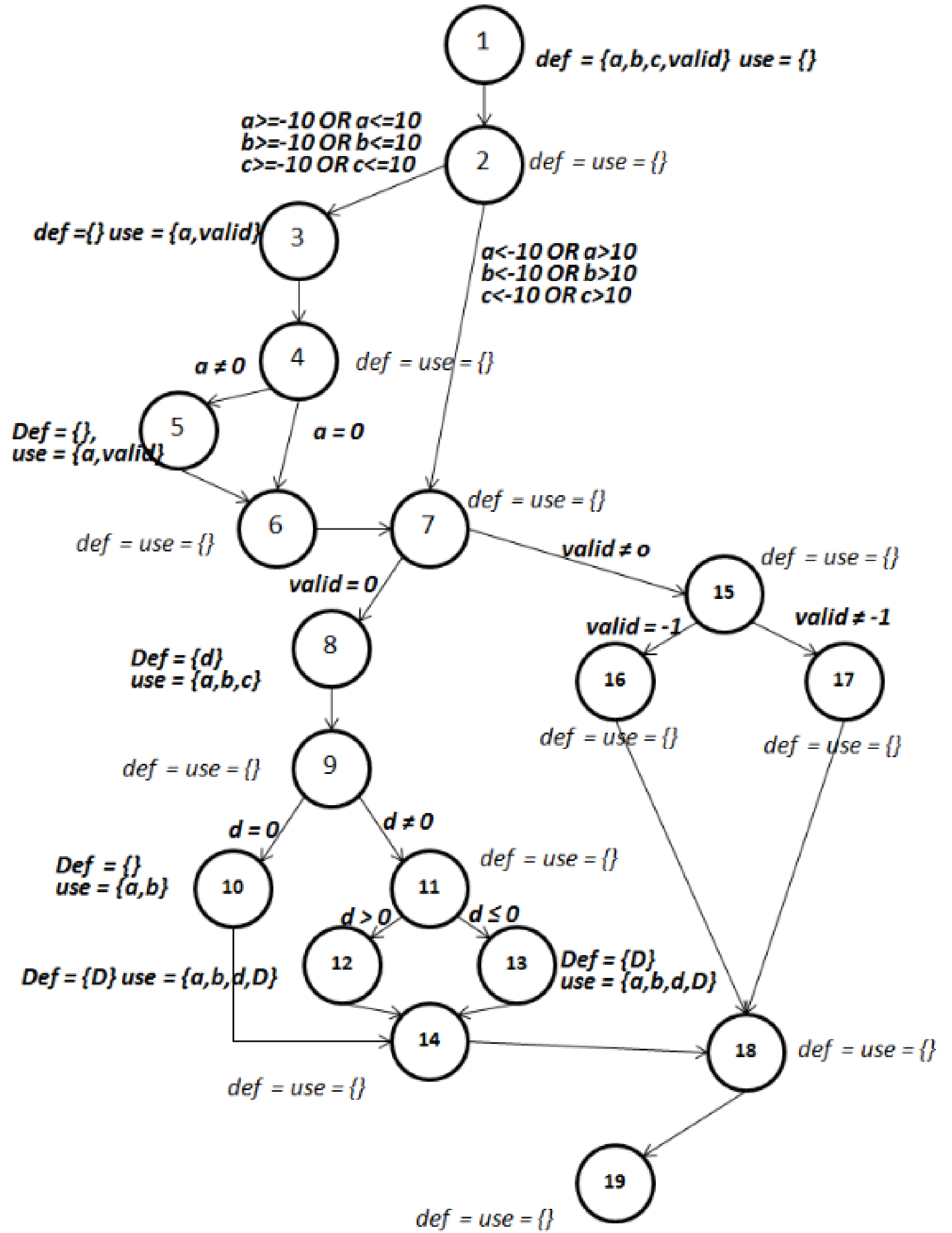


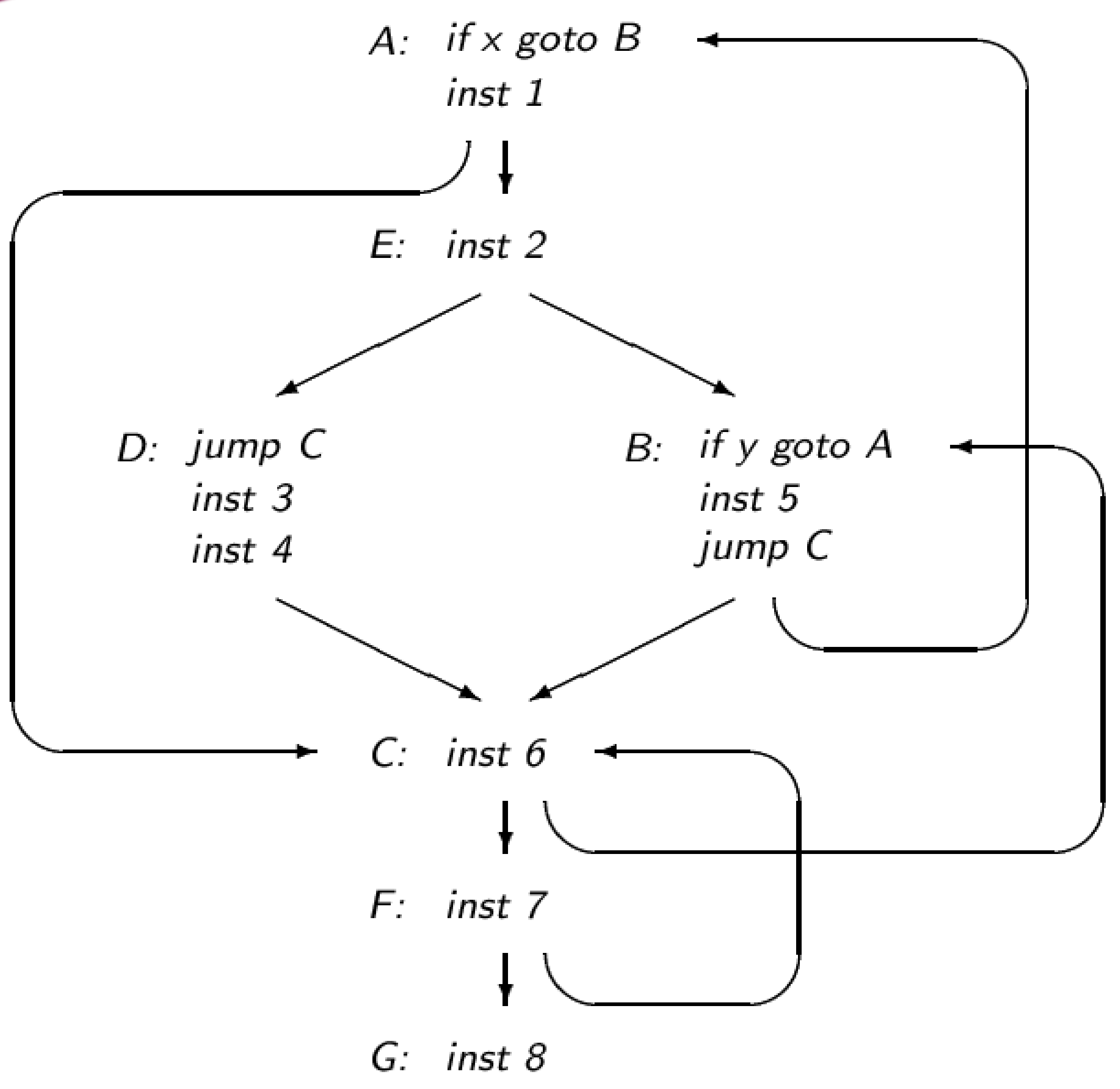


a natural loop with two exits, e.g. while with an if...break in the middle; non-structured but reducible



a loop with two entry points, e.g. goto into a while or for loop


1
def = {a,b,c,valid} use = {}
a>=-10 OR a<=10
b>=-10 OR b<=10
c>=-10 OR c<=10
2
def = use = {}
def ={} use = {a,valid}
3
a<-10 OR a>10
b<-10 OR b>10
c<-10 OR c>10
4
a ≠ 0
def = use = {}
5
Def = {},
use = {a,valid}
a = 0
6
def = use = {}
7
def = use = {}
valid ≠ o
15
def = use = {}
valid = 0
8
Def = {d}
use = {a,b,c}
valid = -1
valid ≠ -1
16
17
def = use = {}
def = use = {}
9
def = use = {}
d = 0
d ≠ 0
def = use = {}
Def = {}
use = {a,b}
10
11
d > 0
d ≤ 0
12
13
Def = {D}
use = {a,b,d,D}
Def = {D} use = {a,b,d,D}
14
18
def = use = {}
def = use = {}
19
def = use = {}


A: if x goto B
inst 1
E: inst 2
D: jump C
inst 3
inst 4
B: if y goto A
inst 5
jump C
C: inst 6
F: inst 7
G: inst 8

CFG

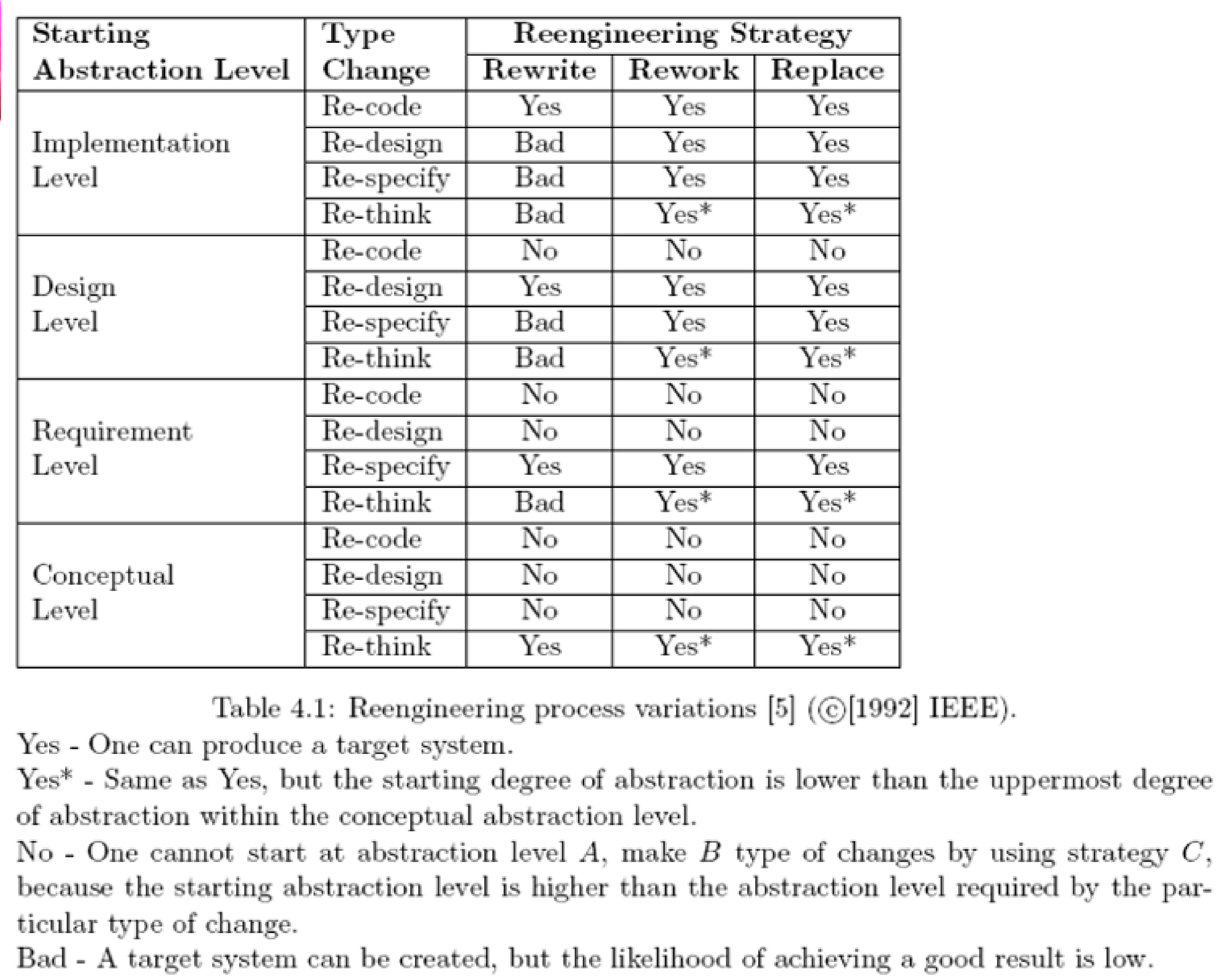
- A **control-flow graph** (**CFG**) is a representation, using graph notation, of all paths that might be traversed through a program during its execution.
- The CFG is use as a static-analysis tools.

- he CFG can be obtained, by starting from the program's (full) flow graph.
- The graph in which every node represents an individual instruction.

- Example :

```
0: t0 = read_num
1: if t0 mod 2 == 0
2: print t0 + " is even."
3: goto 5
4: print t0 + " is odd."
5: end program
```

```
int main(){
int t0;
Scanf("%d", & t0);
if (t0 % 2 == 0)
    printf("%d is even.", t0);
Printf(" %d is odd.", t0);
}
```



Starting Abstraction Level	Type Change	Reengineering Strategy		
		Rewrite	Rework	Replace
Implementation Level	Re-code	Yes	Yes	Yes
	Re-design	Bad	Yes	Yes
	Re-specify	Bad	Yes	Yes
	Re-think	Bad	Yes*	Yes*
Design Level	Re-code	No	No	No
	Re-design	Yes	Yes	Yes
	Re-specify	Bad	Yes	Yes
	Re-think	Bad	Yes*	Yes*
Requirement Level	Re-code	No	No	No
	Re-design	No	No	No
	Re-specify	Yes	Yes	Yes
	Re-think	Bad	Yes*	Yes*
Conceptual Level	Re-code	No	No	No
	Re-design	No	No	No
	Re-specify	No	No	No
	Re-think	Yes	Yes*	Yes*

Table 4.1: Reengineering process variations [5] (©[1992] IEEE).

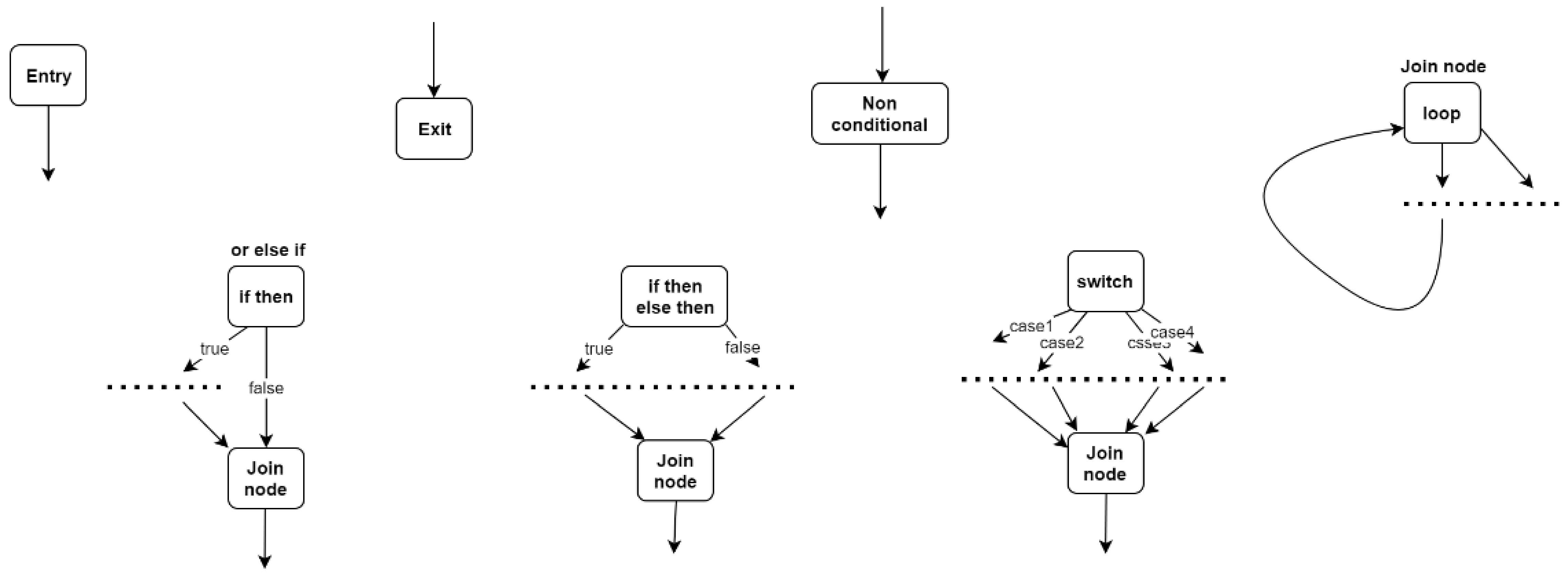
Yes - One can produce a target system.

Yes* - Same as Yes, but the starting degree of abstraction is lower than the uppermost degree of abstraction within the conceptual abstraction level.

No - One cannot start at abstraction level *A*, make *B* type of changes by using strategy *C*, because the starting abstraction level is higher than the abstraction level required by the particular type of change.

Bad - A target system can be created, but the likelihood of achieving a good result is low.

CHEAT SHEET


Entry
Exit
Non conditional
Join node
loop
or else if
if then
true
false
Join node
if then else then
true
false
Join node
switch
case1
case2
csse3
case4
Join node

REENGINEERING PROCESS

- An ordered set of activities designed to perform a specific task is called a process.
- For ease of understanding and communication, processes are described by means of process models.
- For example, in the software development domain, the Waterfall process model is widely used in developing well-understood software systems.
- Process models are used to comprehend, evaluate, reason about, and improve processes.

REENGINEERING PROCESS

- Process models are described by means of important relationships among data objects, human roles, activities, and tools.
- We will discuss five process models for software reengineering.
- The five approaches are different in two aspects:

(i) the extent of reengineering performed, and

(ii) the rate of substitution of the operational system with the new one.

BIG BANG APPROACH

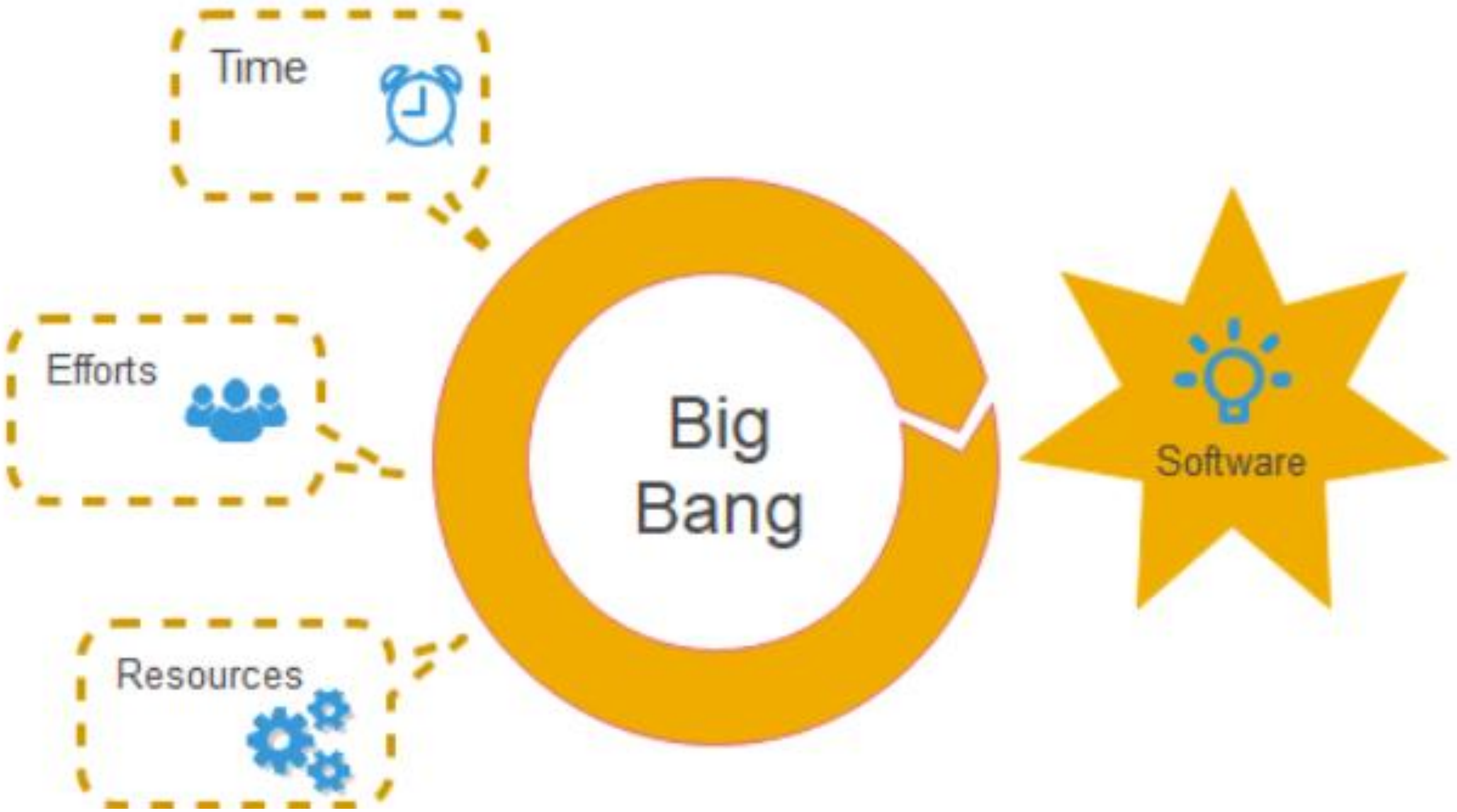
- The "**Big Bang**" approach replaces the whole system at once.
- Once a reengineering effort is initiated, it is continued until all the objectives of the project are achieved and the target system is constructed.
- This approach is generally used if reengineering cannot be done in parts.
- For example, if there is a need to move to a different system architecture, then all components affected by such a move must be changed at once.

DISADVANTAGE

- The consequent advantage is that the system is brought into its new environment all at once.

- The disadvantage of Big Bang is that the reengineering project becomes a monolithic task, which may not be desirable in all situations.

- In addition, the Big Bang approach consumes too much resources at once for large systems, and takes a long stretch of time before the new system is visible.


Time
Efforts
Resources
Big Bang
Software

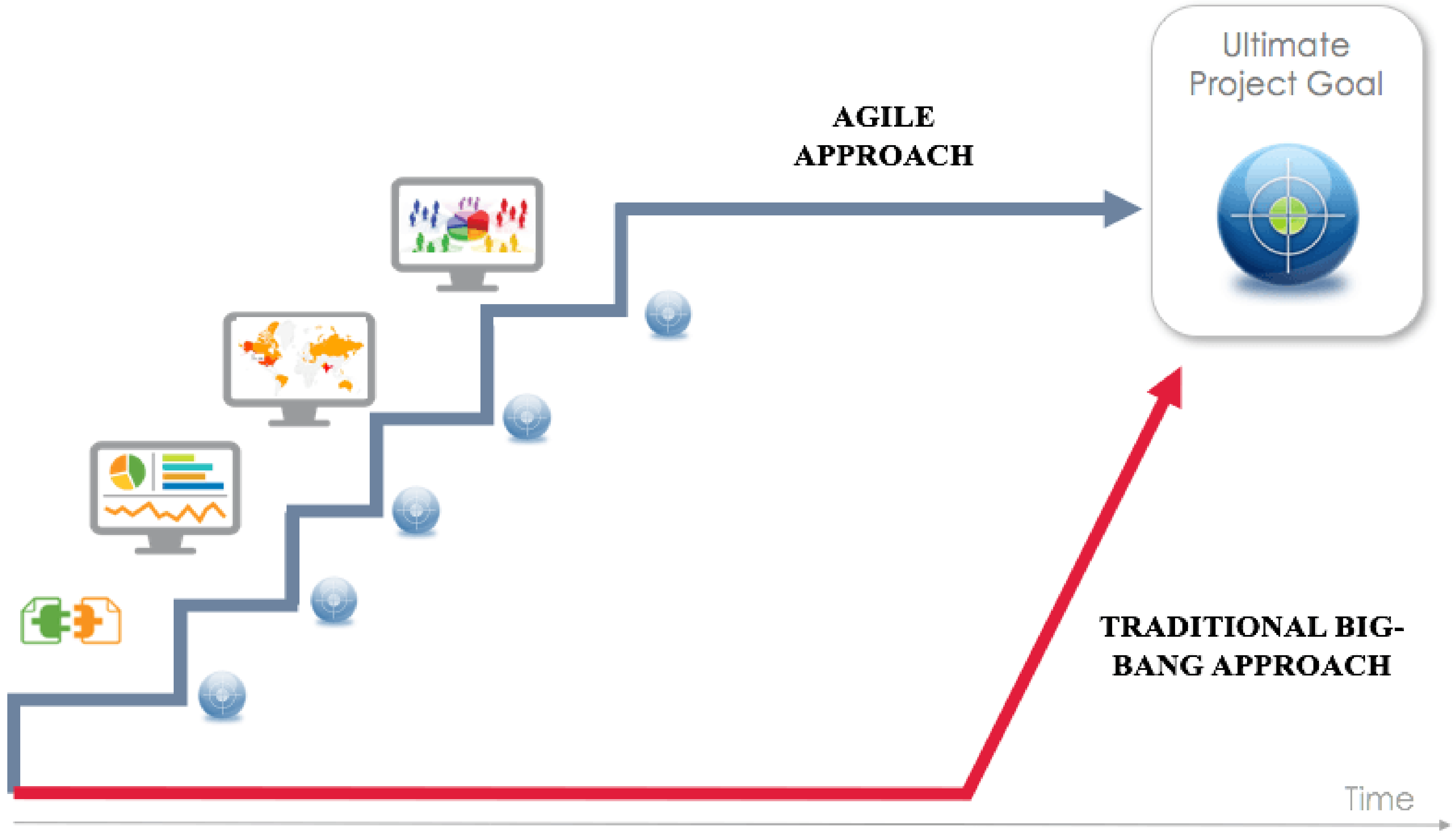
INCREMENTAL APPROACH

- In this approach a system is reengineered gradually, one step closer to the target system at a time.
- For a large system, several new interim versions are produced and released.
- Successive interim versions satisfy increasingly more project goals than their preceding versions.

- The advantages of this approach are as follows:
  (i) locating errors becomes easier, because one can clearly identify the newly added components, and
  (ii) It becomes easy for the customer to notice progress, because interim versions are released.

- The disadvantages of the incremental approach are as follows:
  (i) with multiple interim versions and their careful version controls, the incremental approach takes much longer to complete, and
  (ii) even if there is a need, the entire architecture of the system cannot be changed.

INCREMENTAL VS BIG BANG APPROACH



PARTIAL APPROACH

- In this approach, only a part of the system is reengineered and then it is integrated with the non-engineered portion of the system.
- One must decide whether to use a "Big Bang" approach or an "Incremental" approach for the portion to be reengineered.
- The following three steps are followed in the partial approach:
  - In the first step, the existing system is partitioned into two parts: one part is identified to be reengineered and the remaining part to be not reengineered.
  - In the second step, reengineering work is performed using either the "Big Bang" or the "Incremental" approach.
  - In the third step, the two parts, namely, the not-to-be-reengineered part and the reengineered part of the system, are integrated to make up the new system.

- The partial approach has the advantage of reducing the scope of reengineering that is less time and costs less.

- A disadvantage of the partial approach is that modifications are not performed to the interface between the portion modified and the portion not modified.
- Compatibility or performance issue may be faced.

ITERATIVE APPROACH

- The reengineering process is applied on the source code of a few procedures at a time, with each reengineering operation lasting for a short time.

- This process is repeatedly executed on different components in different stages.

- During the execution of the process, ensure that the four types of components can coexist:
  - old components not reengineered,
  - components currently being reengineered,
  - components already reengineered, and
  - new components added to the system.

- There are two advantages of the iterative reengineering process:
  (i) it guarantees the continued operation of the system during the execution of the reengineering process, and
  (ii) the maintainers' and the users' familiarities with the system are preserved.

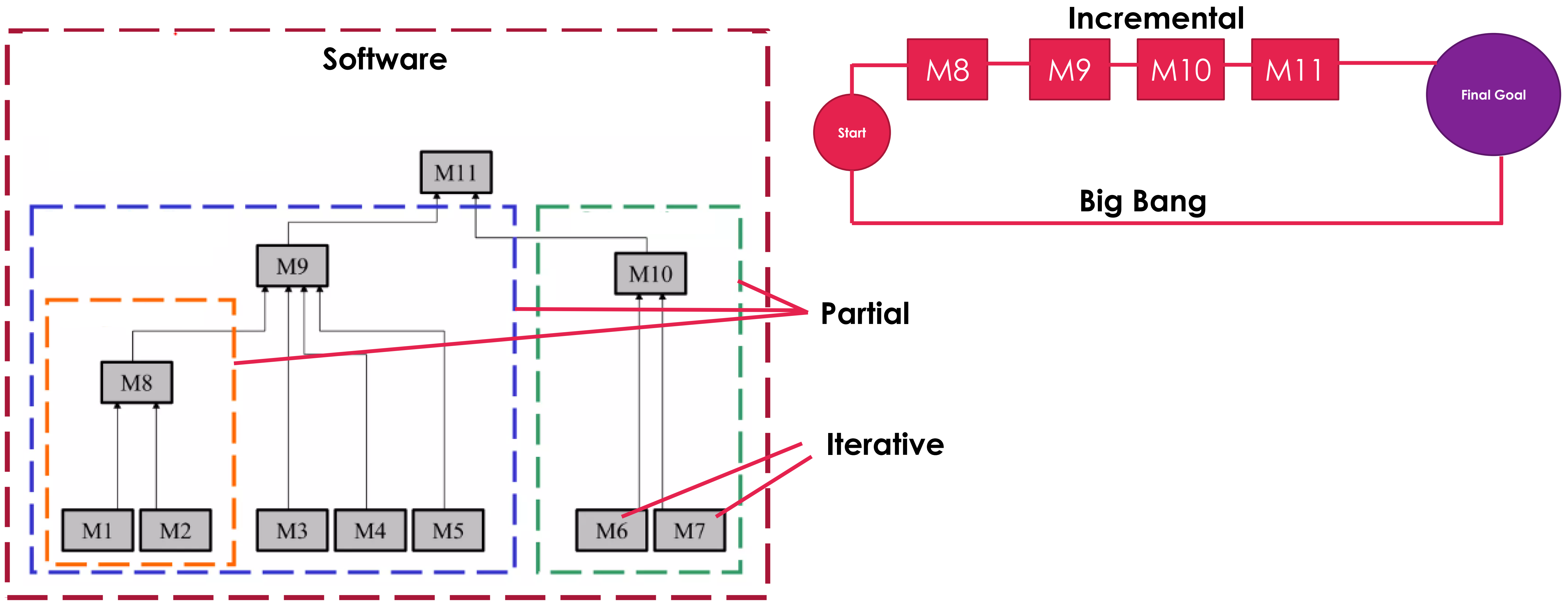
- The disadvantage of this approach is the need to keep track of the four types of components during the reengineering process.
- In addition, both the old and the newly reengineered components need to be maintained.

EVOLUTIONARY APPROACH

- In the "Evolutionary approach" components of the original system are substituted with re-engineered components.
- In this approach, the existing components are grouped by functions and reengineered into new components.
- Software engineers focus their reengineering efforts on identifying functional objects irrespective of the locations of those components within the current system.
- As a result, the new system is built with functionally cohesive components as needed.

- There are two advantages of the “Evolutionary” approach:
  (i) the resulting design is more cohesive, and
  (ii) the scope of individual components is reduced.

- A major disadvantage:
  (i) all the functions with much similarities must be first identified throughout the operational system.
  (ii) next, those functions are refined as one unit in the new system


Software
M11
M9
M10
M8
M1
M2
M3
M4
M5
M6
M7
Partial
Iterative
Incremental
M8
M9
M10
M11
Start
Final Goal
Big Bang

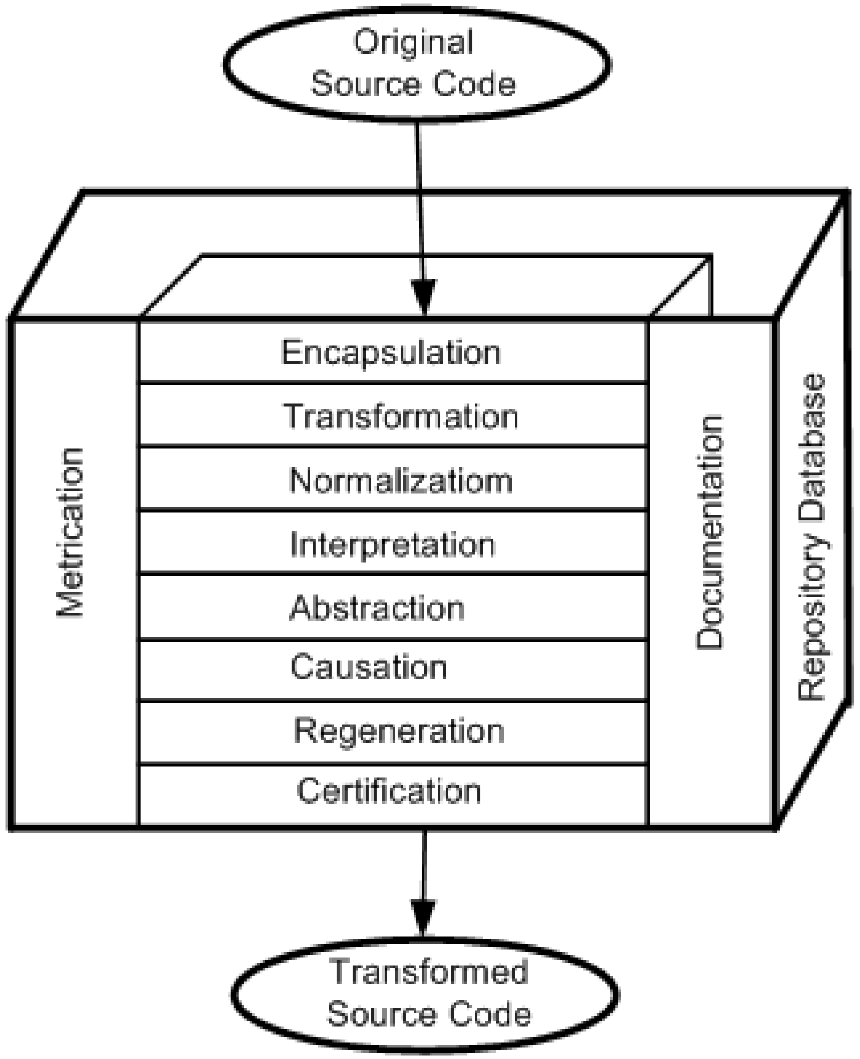
SOURCE CODE REENGINEERING REFERENCE MODEL

- The SCORE/RM model was proposed by Colbrook, Smythe and Darlison.

- The framework consists of four kinds of elements:
    - **function,**
    - **documentation,**
    - **repository database, and**
    - **metrication.**

FUNCTION ELEMENT

- The function element is divided into eight layers, namely:
  - Encapsulation,
  - Transformation,
  - Normalization,
  - Interpretation,
  - Abstraction,
  - Causation,
  - Regeneration, and
  - Certification.



- The eight layers provide a detailed approach to:

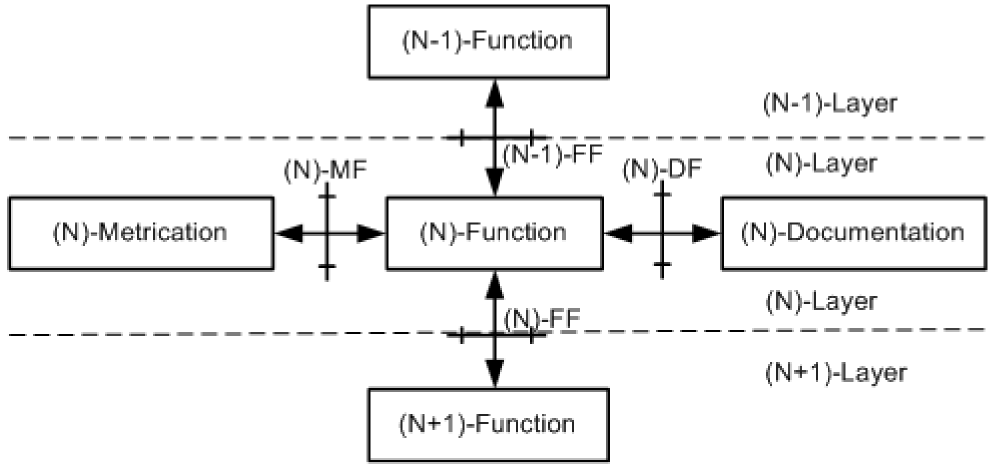
    (i) rationalizing the system to be reengineered by removing redundant data and altering the control flow,

    (ii) comprehending the software's requirements, and

    (iii) reconstructing the software according to established practices.

- The first six of the eight layers together constitute a process for reverse engineering, and the final three a process for forward engineering.
- Improvements in the software as a result of reengineering is quantified by means of the metrication element.
- The metrication element is described in terms of the relevant software metrics before executing a layer and after executing the same layer.
- The repository database is the information store for the entire reengineering process, containing the following kinds of information:
  - metrication,
  - documentation, and
  - both the old and the new source code.

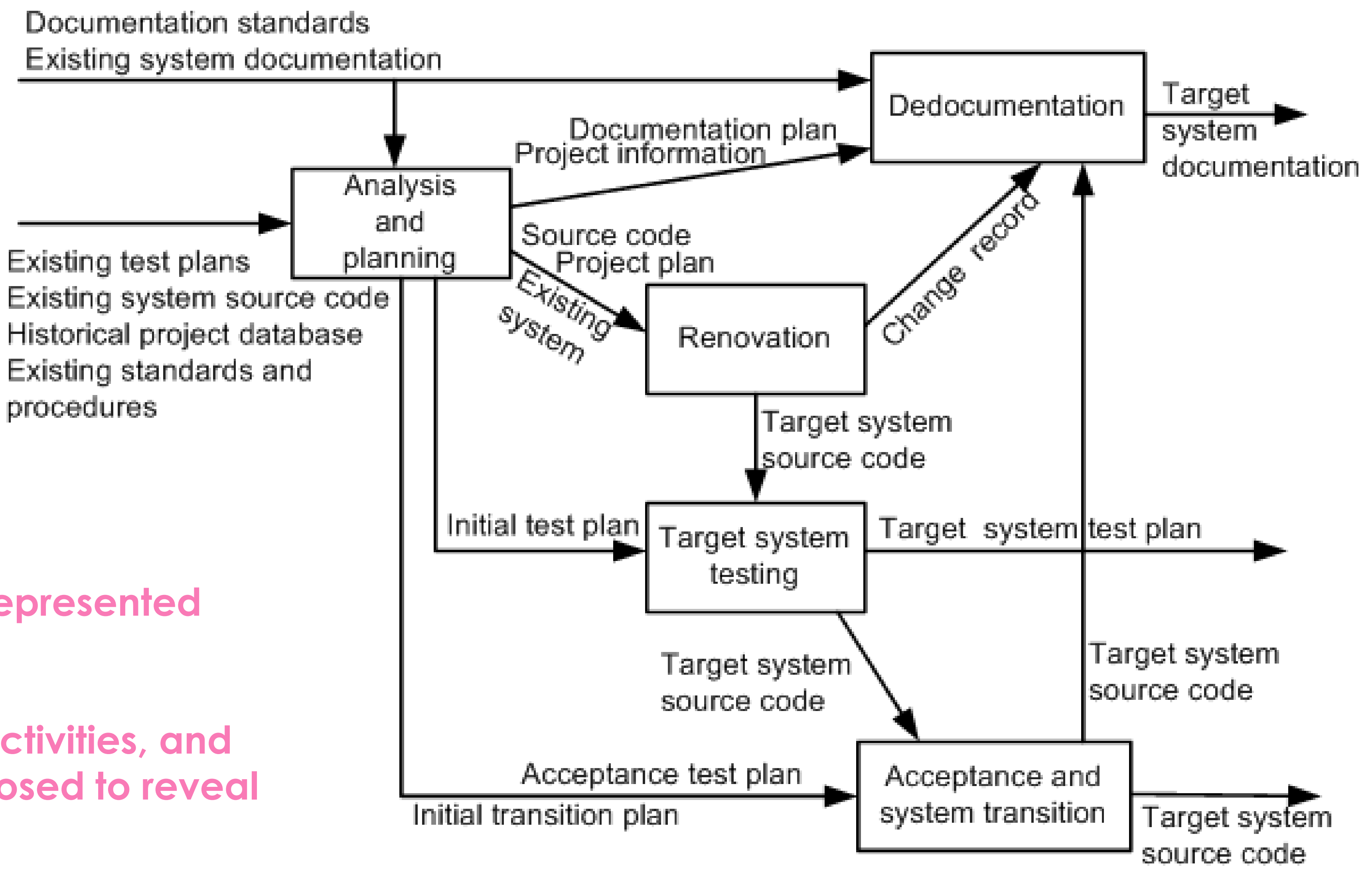
- The interfaces among the elements are shown in Figure.
- For simplicity, any layer is referred to as (N)− layer, while its next lower and next higher layers are referred to as (N − 1)− layer and the (N + 1)− layer, respectively.
- The three types of interfaces are explained as follows:
  - Metrication/Function: (N)-MF – the structures describing the metrics and their values.
  - Documentation/Function: (N)-DF – the structures describing the documentation.
  - Function/Function: (N)-FF – the representation structures for source code passed between the layers.



The interface nomenclature

PHASE REENGINEERING MODEL

- The phase model of software reengineering was originally proposed by Byrne and Gustafson.
- The model comprises five phases: analysis and planning, renovation, target system testing, redocumentation, and acceptance testing and system transition, as depicted in Figure on next slide.
- The labels on the arcs denote the possible information that flows from the tail entities of the arcs to the head entities.

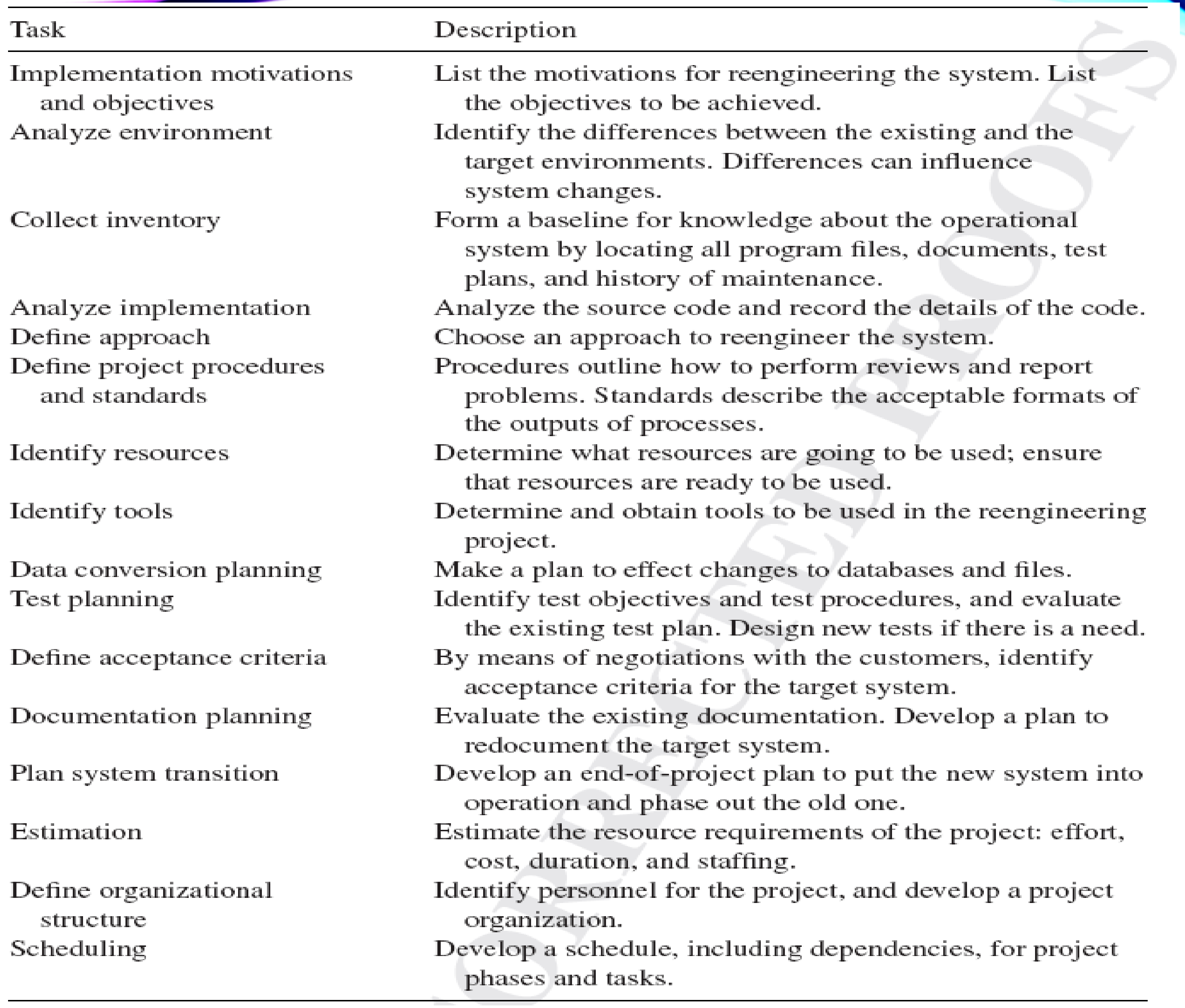


- A major process activity is represented by each phase.
- Tasks represent a phase's activities, and tasks can be further decomposed to reveal the detailed methodologies.

ANALYSIS AND PLANNING

- Analysis addresses three technical and one economic issue.
  - The first technical issue concerns the present state of the system to be reengineered and understanding its properties.
  - The second technical issue concerns the identification of the need for the system to be reengineered.
  - The third technical issue concerns the specification of the characteristics of the new system to be produced.
- The economic issue concerns a cost and benefit analysis of the reengineering project.
- Planning includes:
  - understanding the scope of the work;
  - identifying the required resources;
  - identifying the tasks and milestones;
  - estimating the required effort; and
  - preparing a schedule.

- Task Analysis & Planning Phase



Task	Description
Implementation motivations and objectives	List the motivations for reengineering the system. List the objectives to be achieved.
Analyze environment	Identify the differences between the existing and the target environments. Differences can influence system changes.
Collect inventory	Form a baseline for knowledge about the operational system by locating all program files, documents, test plans, and history of maintenance.
Analyze implementation	Analyze the source code and record the details of the code.
Define approach	Choose an approach to reengineer the system.
Define project procedures and standards	Procedures outline how to perform reviews and report problems. Standards describe the acceptable formats of the outputs of processes.
Identify resources	Determine what resources are going to be used; ensure that resources are ready to be used.
Identify tools	Determine and obtain tools to be used in the reengineering project.
Data conversion planning	Make a plan to effect changes to databases and files.
Test planning	Identify test objectives and test procedures, and evaluate the existing test plan. Design new tests if there is a need.
Define acceptance criteria	By means of negotiations with the customers, identify acceptance criteria for the target system.
Documentation planning	Evaluate the existing documentation. Develop a plan to redocument the target system.
Plan system transition	Develop an end-of-project plan to put the new system into operation and phase out the old one.
Estimation	Estimate the resource requirements of the project: effort, cost, duration, and staffing.
Define organizational structure	Identify personnel for the project, and develop a project organization.
Scheduling	Develop a schedule, including dependencies, for project phases and tasks.

RENOVATION

- An operational system is modified into the target system in the renovation phase.
- Two main aspects of a system are considered in this phase:

  (i) representation of the system.

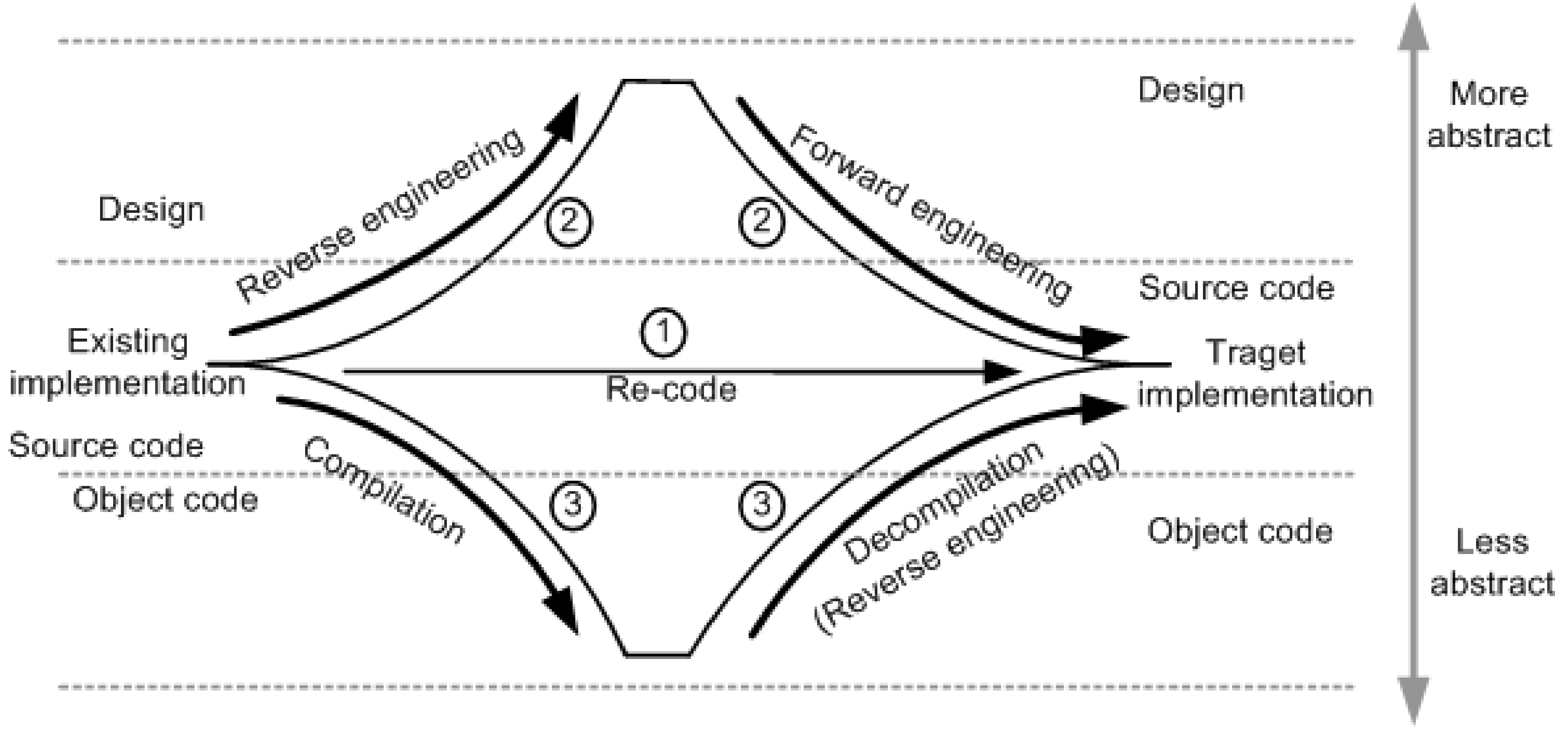
  It refers to source code, but it may include the design model and the requirement specification of the existing system.

  (ii) representation of external data.

  It refers to the database and/or data files used by the system. Often the external data are reengineered, and it is known as data reengineering.
- An operational system can be renovated in many ways, depending upon the objectives of the project, the approach followed, and the starting representation of the system.
- It may be noted that the starting representation can be source code, design, or requirements.
- Table 4.1 discussed earlier recommends several alternatives to renovate a system.

RENOVATION: EXAMPLE

- A project in which the objective is to re-code the system from Fortran to C.
- Figure 4.9 shows the three possible replacement strategies.
- First, to perform source-to-source translation, program migration is used.
- Second, a high-level design is constructed from the operational source code, say, in Fortran, and the resulting design is re-implemented in the target language, C in this case.
- Finally, a mix of compilation and decompilation is used to obtain the system implementation in C


Design
More abstract
Design
Reverse engineering
Forward engineering
2
2
Source code
1
Existing implementation
Traget implementation
Re-code
Source code
Compilation
Decompilation (Reverse engineering)
Object code
3
3
Object code
Less abstract

TARGET SYSTEM TESTING

- In this phase for system testing, faults are detected in the target system.
- Those faults might have been introduced during reengineering.
- Fault detection is performed by applying the target system test plan on the target system.
- The same testing strategies, techniques, methods, and tools that are used in software development are used during reengineering.
- For example, apply the existing system-level test cases to both the existing and the new system.
- Assuming that the two systems have identical requirements, the test results from both the scenarios must be the same.

RE-DOCUMENTATION

- In the re-documentation phase, documentations are rewritten, updated, and/or replaced to reflect the target system.
- Documents are revised according to the re-documentation plan.
- The two major tasks within this phase are:
  (i) analyze new source code, and
  (ii) create documentation.
- Documents requiring revision are:
  - requirement specification.
  - design documentation.
  - a report justifying the design decisions, assumptions made in the implementation. configuration.
  - user and reference manuals.
  - on-line help.
  - document describing the differences between the existing and the target system.

ACCEPTANCE AND SYSTEM TRANSITION

- In this final phase, the reengineered system is evaluated by performing acceptance testing.
- Acceptance criteria should already have been established in the beginning of the project.
- Should the reengineered system pass those tests, preparation begins to transition to the new system.
- On the other hand, if the reengineered system fails some tests, the faults must be fixed; in some cases, those faults are fixed after the target system is deployed.
- Upon completion of the acceptance tests, the reengineered system is made operational, and the old system is put out of service.
- System transition is guided by the prior developed transition plan.