软件工程概论
Software Engineering

刘伟

liuwei@xidian.edu.cn

88204608

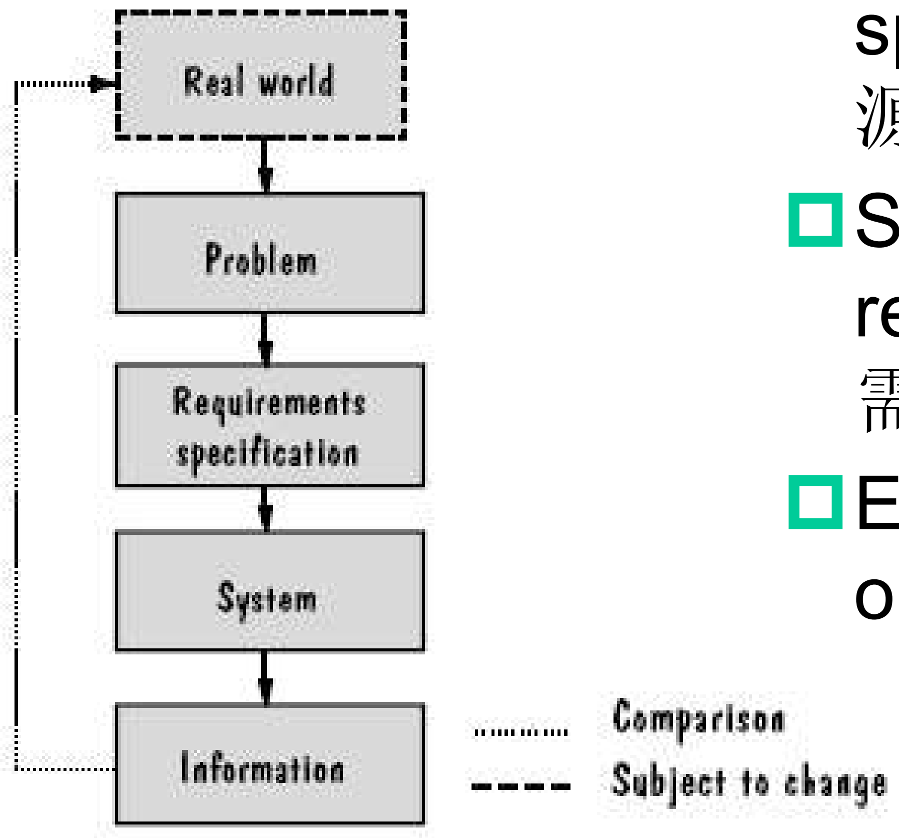
CH11. Maintaining the Systems

Content

Lehman’s system types系统类型

- Software system are evolutionary.软件系统是演化的
  - A customer makes a decision to do something a different way.顾客以不同的方式做某件事
  - The nature of the system itself changes.系统自身的属性发生了改变
- Types of Systems, in terms of the way it is related to the environment in which it operates系统类型，用系统和系统运行的环境来描述
  - S-systems
  - P-systems
  - E-systems

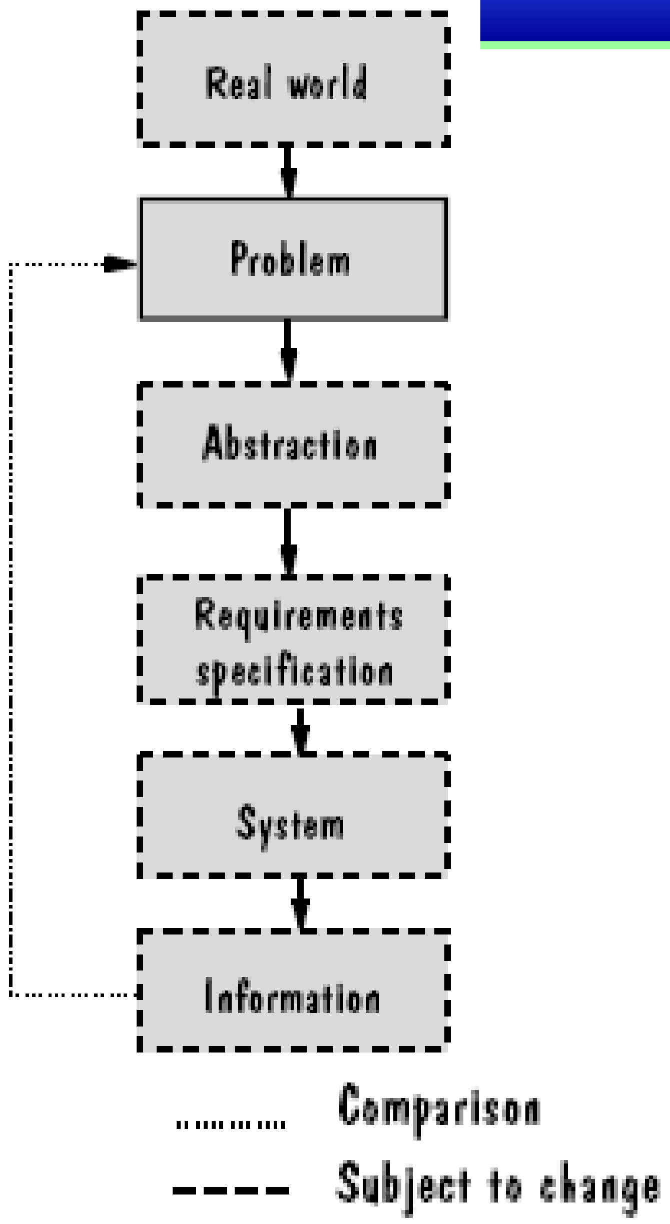
S-systems



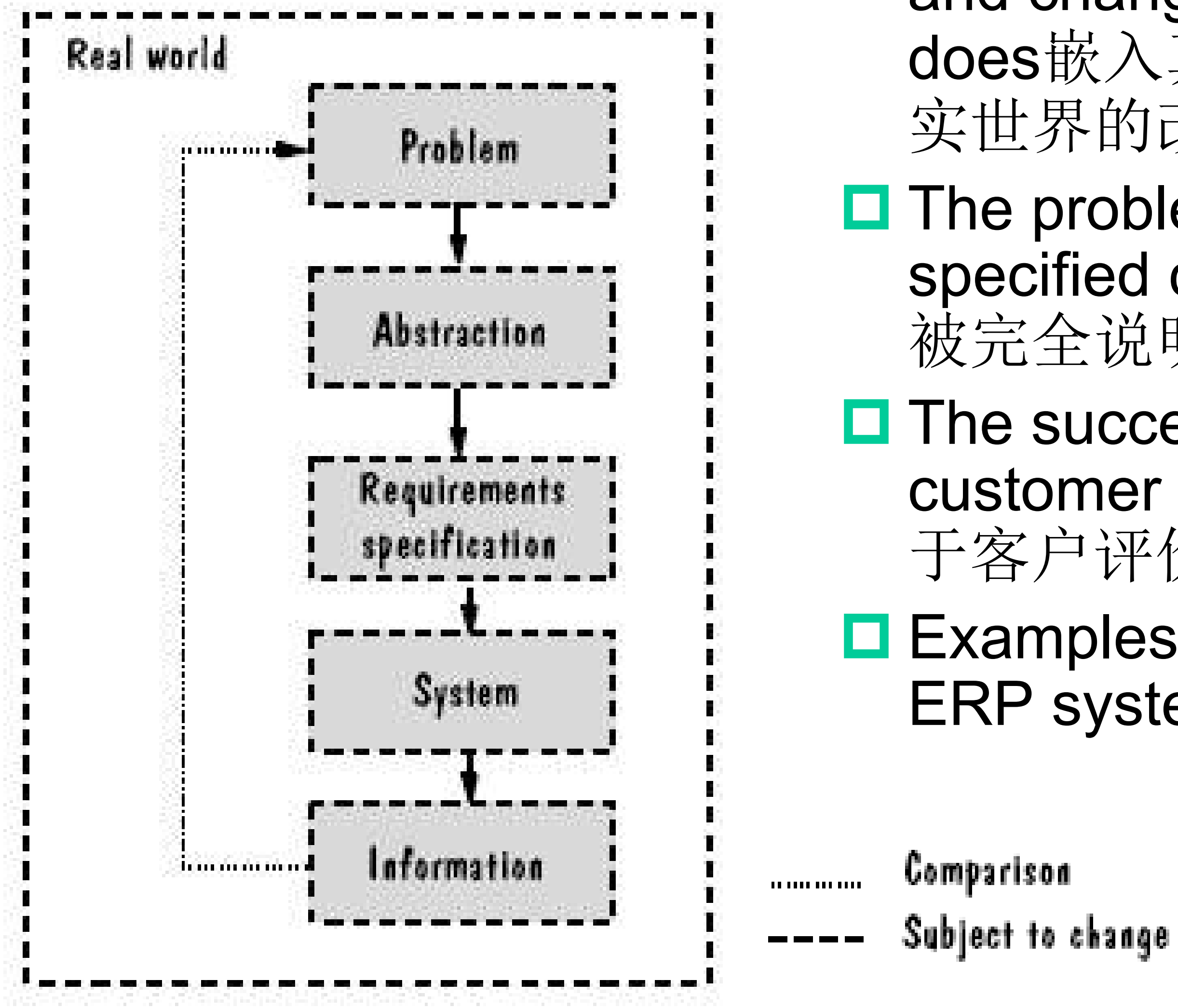
- Formally defined, derivable from a specification形式化定义、源自说明描述
- Static system, no change required静态系统，无改变需要
- Examples: mathematical operations

P-systems

- The system is based on a practical abstraction of the problem rather then on a completely defined specification系统基于问题的一个可行的抽象，而不是一个完全定义好的说明
- Abstraction and solution may change抽象与解决方案可能变化
- Examples: pattern recognition systems模式识别系统



E-systems



- Embedded in the real world and changes as the world does嵌入真实世界中并随着真实世界的改变而改变
- The problem cannot be specified completely问题不能被完全说明
- The success depends on the customer evaluation成功取决于客户评价
- Examples: banking systems, ERP systems

软件开发中变动的例子

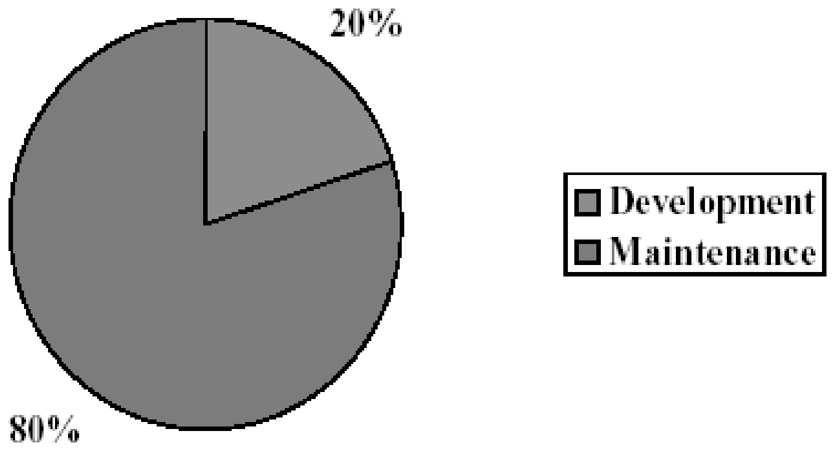
Table 11.1. Examples of change during software development.

Activity from which initial change results	*Artifacts requiring consequent change*
Requirements analysis	Requirements specification
System design	Architectural design specification
	Technical design specification
Program design	Program design specification
Program implementation	Program code
	Program documentation
Unit testing	Test plans
	Test scripts
System testing	Test plans
	Test scripts
System delivery	User documentation
	Training aids
	Operator documentation
	System guide
	Programmer guide
	Training classes

Development Time vs. Maintenance Time

- Typical development project takes between 1 and 2 years, but ......
- requires an additional 5 to 6 years of maintenance time!

- *80-20 Rule*
  - Twenty percent of the effort is in development and eighty percent is in maintenance.



System evolution vs. decline系统演化和系统衰退

- Is the cost of maintenance too high维护的成本太高吗?
- Is the system reliability unacceptable系统的可靠性可以接受吗?
- Can the system no longer adapt to further change, and within a reasonable amount of time在一个合理的时间内，系统不能再适应进一步的变化了吗?
- Is system performance still beyond prescribed constraints系统性能仍旧超出预先规定的约束条件吗?
- Are system functions of limited usefulness系统功能的作用有限吗?
- Can other systems do the same job better, faster or cheaper其他的系统能更好、更快、更廉价地做同样的工作?
- Is the cost of maintaining the hardware great enough to justify replacing it with cheaper, newer hardware维护硬件的成本高得足以用更便宜、更新的硬件来取代吗?

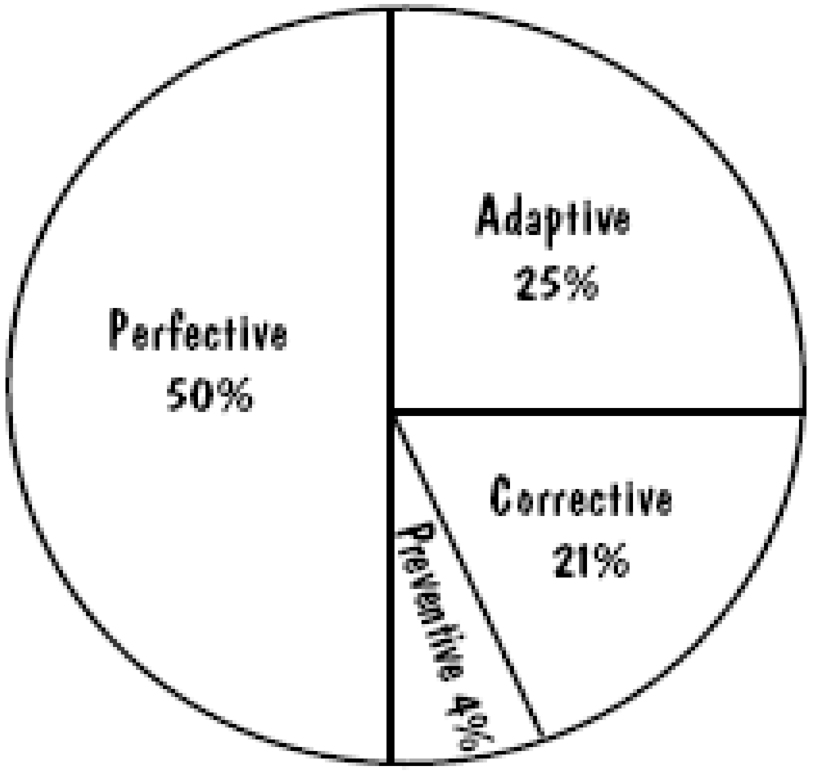
Laws of software evolution软件演化法则

- Continuing change连续的变化：leads to less utility 导致无用
- Increasing complexity递增的复杂性：structure deteriorates使结构恶化
- Fundamental law of program evolution程序演化的基本法则：program obeys statistically-determined trends and has invariants程序服从统计确定趋势和恒定性
- Conservation of organizational stability组织稳定性的守恒：global activity rate is invariant总体活动统计上是不变的
- Conservation of familiarity熟悉程度的守恒: release content is invariant版本发布内容是不变的

Types of maintenance维护类型

- Corrective改正性维护：maintaining control over day-to-day functions维持控制日常功能
- Adaptive适应性维护：maintaining control over system modifications维持控制系统修改
- Perfective完善性维护：perfecting existing functions完善现有的功能
- Preventive预防性维护：preventing system performance from degrading to unacceptable levels

Use of maintenance time



- Perfective完善性维护: 50%
- Adaptive适应性维护: 25%
- Corrective改正性维护: 21%
- Preventive预防性维护: 4%

Maintenance team responsibilities 维护小组的职责

- understanding the system理解系统
- locating information in system documentation定位系统文档中的信息
- keeping system documentation up-to-date保持系统文档更新
- extending existing functions to accommodate new or changing requirements扩展现有功能以容纳新的或变动的需求
- adding new functions to the system给系统增加新的功能
- finding the source of system failures or problems找出系统故障或问题的根源

Maintenance team responsibilities 维护小组的职责

- locating and correcting faults定位和纠正故障
- answering questions about the way the system works回答有关系统运行方式的问题
- restructuring design and code components重构设计和代码组件
- rewriting design and code components重新写设计和代码组件
- deleting design and code components that are no longer useful删除不再有用的设计和代码组件
- managing changes to the system as they are made进行系统改动

Maintenance problems与维护有关的问题

- Staff problems人员问题
  - Limited understanding理解的局限性
  - Management priorities管理的优先级
  - Morale士气
- Technical problems技术问题
  - Artifacts and paradigms工件和范例
  - Testing difficulties测试困难

Factors affecting maintenance effort

- Application type应用类型
- System novelty系统新奇度
- Turnover and maintenance staff ability
- System life span系统生命期
- Dependence on a changing environment
- Hardware characteristics硬件特性
- Design quality设计质量
- Code quality代码质量
- Documentation quality文档质量
- Testing quality测试质量

Modeling maintenance effortd对维护工作量建模

Belady and Lehman公式

$$M = p + K^{c-d}$$

其中：M是一个系统花费的所有维护工作量

p表示分析、评价、设计、编码以及测试的总工作量

K是个常量，通过与实际工作量比较而定

c是由于缺乏结构化和文档引起的复杂度

d表示维护小组对软件的熟悉程度，d削弱了c

COCOMO II软件理解的评分

Table 11.2. COCOMO II rating for software understanding.

	Very low	*Low*	*Nominal*	*High*	*Very high*
Structure结构	Very low cohesion, high coupling, spaghetti code	Moderately low cohesion, high coupling	Reasonably well-structured; some weak areas	High cohesion, low coupling	Strong modularity, information-hiding in data and control structures
Application Clarity 应用程序清晰度	No match between program and application world views	Some correlation between program and application	Moderate correlation between program and application	Good correlation between program and application	Clear match between program and application world views
Self-Descriptiveness 自描述	Obscure code; documentation missing, obscure or obsolete	Some code commentary and headers; some useful documentation	Moderate level of code commentary, headers, documentation	Good code commentary and headers; useful documentation; some weak areas	Self-descriptive code; documentation up-to-date, well-organized, with design rationale
SU增值 SU increment	50	40	30	20	10

COCOMO Ⅱ 评估和修改的评分

Table 11.3. COCOMO II ratings for assessment and assimilation effort.

Assessment and assimilation increment	*Level of assessment and assimilation effort*
0	None
2	Basic component search and documentation
4	Some component test and evaluation documentation
6	Considerable component test and evaluation documentation
8	Extensive component test and evaluation documentation

Measuring maintainability度量维护性1

- Necessary data（用平均修复时间度量所）必须的数据
  - time at which problem is reported问题报告的时间
  - time lost due to administrative delay由于行政管理延迟浪费的时间
  - time required to analyze problem分析问题需要的时间
  - time required to specify which changes are to be made确定进行哪种改动需要的时间
  - time needed to make the change进行改动需要的时间
  - time needed to test the change测试变动需要的时间
  - time needed to document the change记录变动需要的时间

Measuring maintainability度量维护性2

- Desirable data:可能有用的数据
  - ratio of total change implementation time to total number of changes implemented实现改动的总时间与实现的改动的总数目之比
  - number of unresolved problems未解决问题的数目
  - time spent on unresolved problems花在未解决问题上的时间
  - percentage of changes that introduce new faults引入新故障的改动的百分比
  - number of components modified to implement a change为实现一个改动而修改的组件数

Example for calculating cyclomatic number

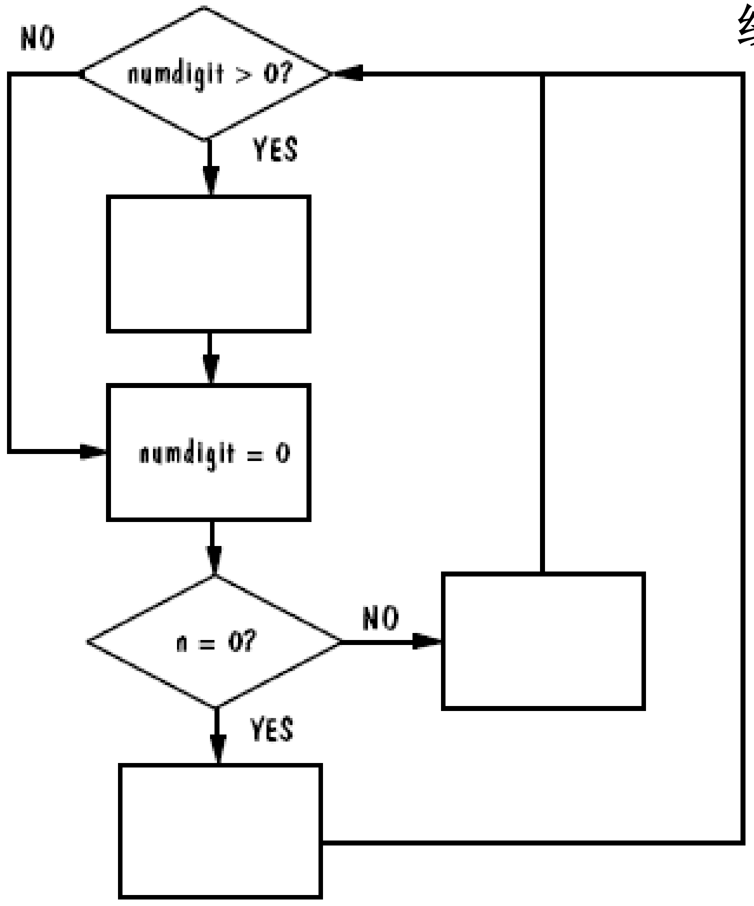
环路数＝代码中判定语句数＋1

```
    Scoreboard::drawscore(int n)
    {
            while(numdigits-- > 0} {
                    score[numdigits]->erase();
            }
            // build new score in loop, each time update position
            numdigits = 0;
            // if score is 0, just display "0"
            if (n == 0) {
                    delete score[numdigits];
                    score[numdigits] = new Displayable(di gits[0]);
                    score[numdigits]->move(Point((700-numdigits*18),40));
                    score[numdigits]->draw();
                    numdigits++;
    }
    while (n) {
                    int rem = n % 10;
                    delete score[numdigits];
                    score[numdigits] = new Displayable(digits[rem]);
                    score[numdigits]->move(Point(700-numdigits*18),40));
                    score[numdigits]->draw();
                    n /= 10;
                    numdigits++;
            }
}
```

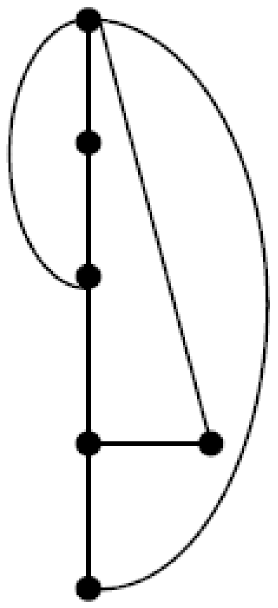
环路数计算的例子

环路数＝划分平面数

线性无关路径＝**e – n + 2**



CONTROL FLOW GRAPH



EQUIVALENT GRAPH

Fog index可读性度量指标

$$F = 0.4 \times \frac{\text{number of words}}{\text{number of sentences}} + \text{percentage of words of 3 or more syllables}$$

$$\mathbf{F = 0.4} \times \frac{\text{单词数}}{\text{句子数}} + \mathbf{3}\text{个以上音节的词的百分比}$$

Maintenance Techniques and Tools

- Configuration Management配置管理
  - Configuration Control Board配置管理委员会, pg. 489
  - Change Control变动控制
- Impact Analysis后果分析
  - Evaluation of the many risks associated with the change, including estimates of effects on resources, effort and schedule对与变动有关的多项风险的评估，包括对资源、工作量和进度影响的评估
- Automated Maintenance Tools自动维护工具

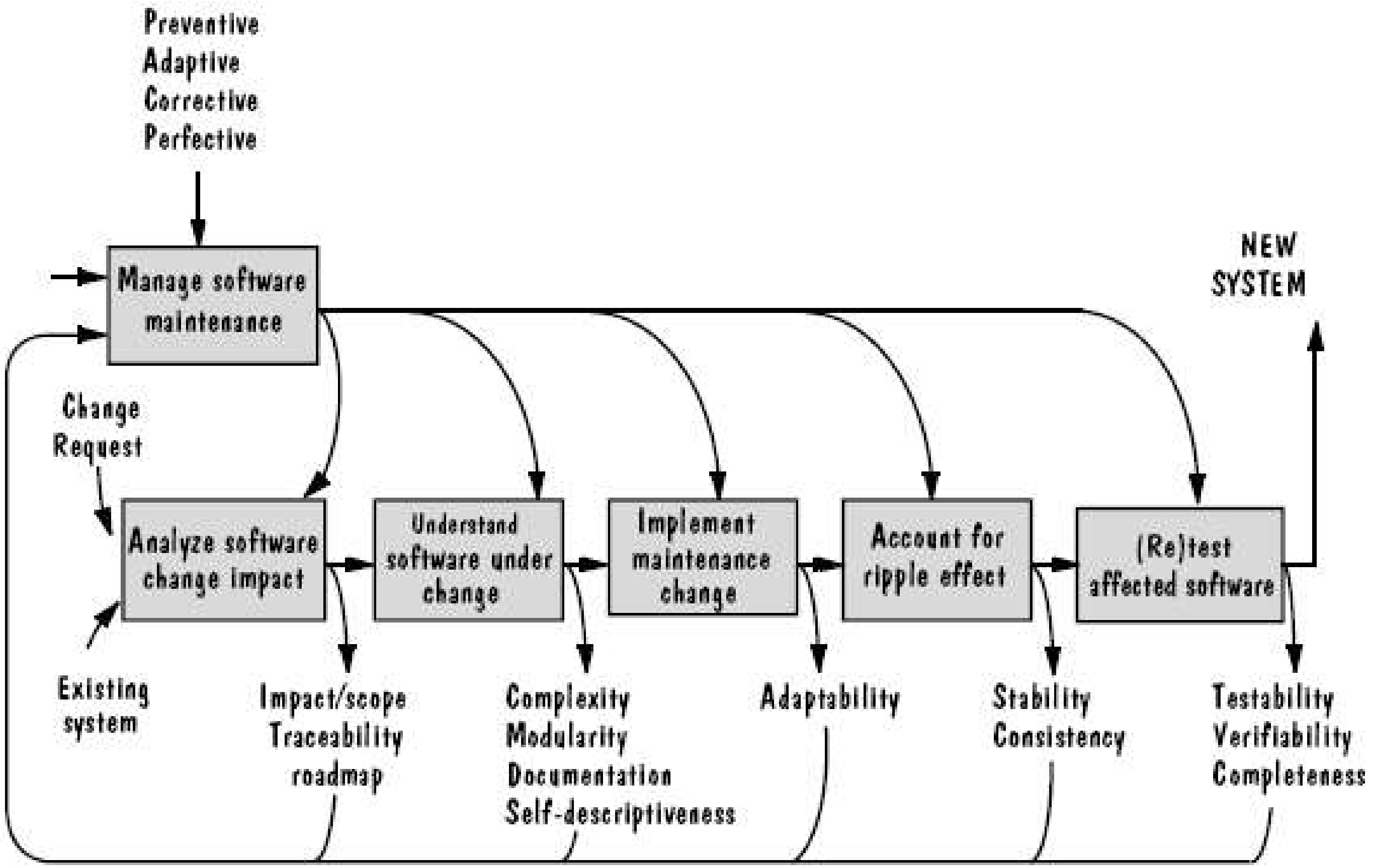
Change control issues变动控制

- *Synchronization同步*: When was the change made?
- *Identification标识*: Who made the change?
- *Naming命名*: What components of the system were changed?
- *Authentication鉴定*: Was the change made correctly?
- *Authorization授权*: Who authorized that the change be made?
- *Routing发送*: Who was notified of the change?
- *Cancellation取消*: Who can cancel the request for change?
- *Delegation委托*: Who is responsible for the change?
- *Valuation评价*: What is the priority of the change?

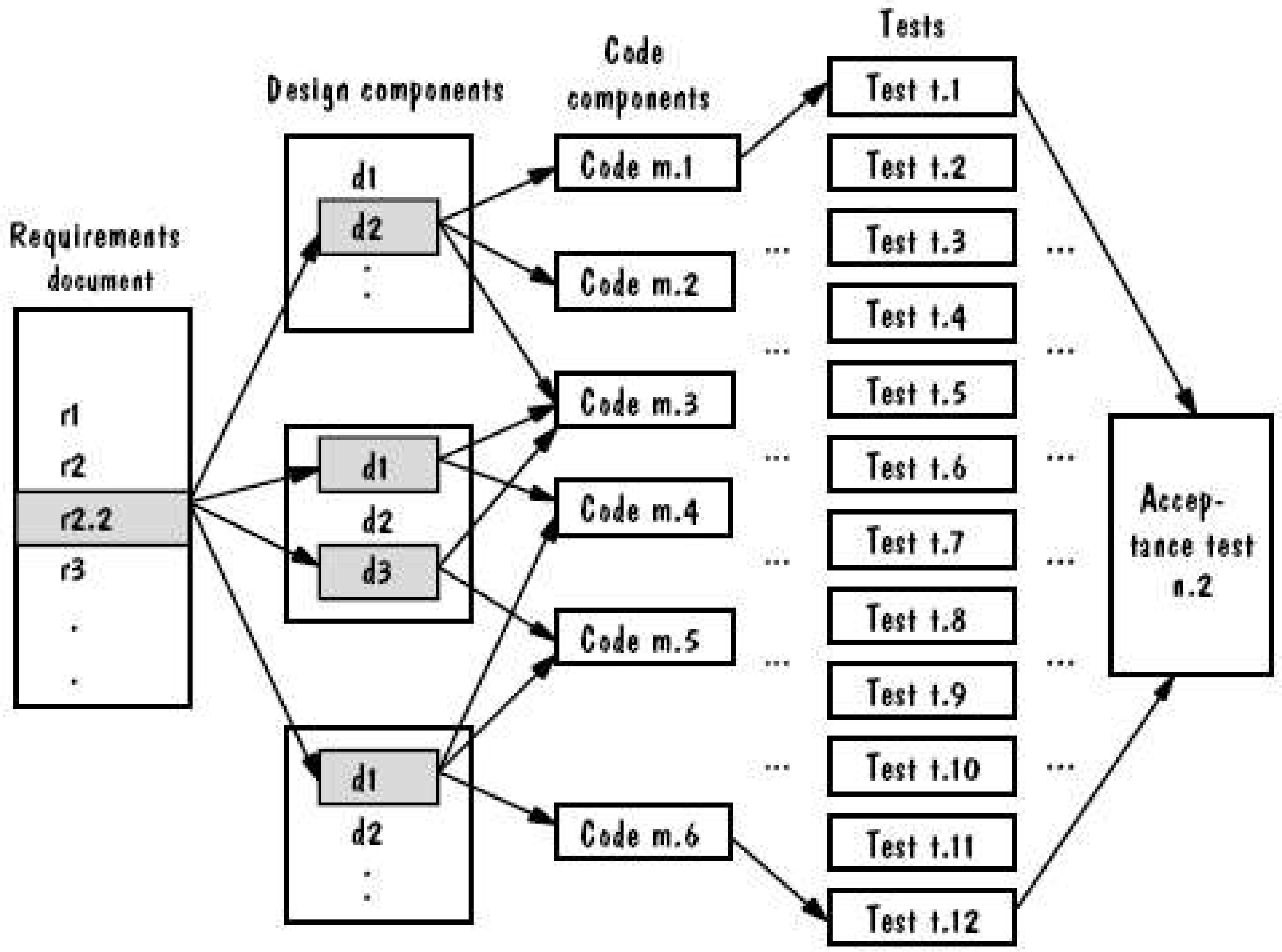
Impact analysis后果分析

- Workproduct工作产品：any development artifact whose change is significant改动起来有重要影响的开发工件
- Horizontal traceability水平可追踪性：relationships of components across collections of workproducts工作产品集之间组件的关系
- Vertical traceability垂直可追踪性：relationships among parts of a workproduct工作产品中各部分之间的关系

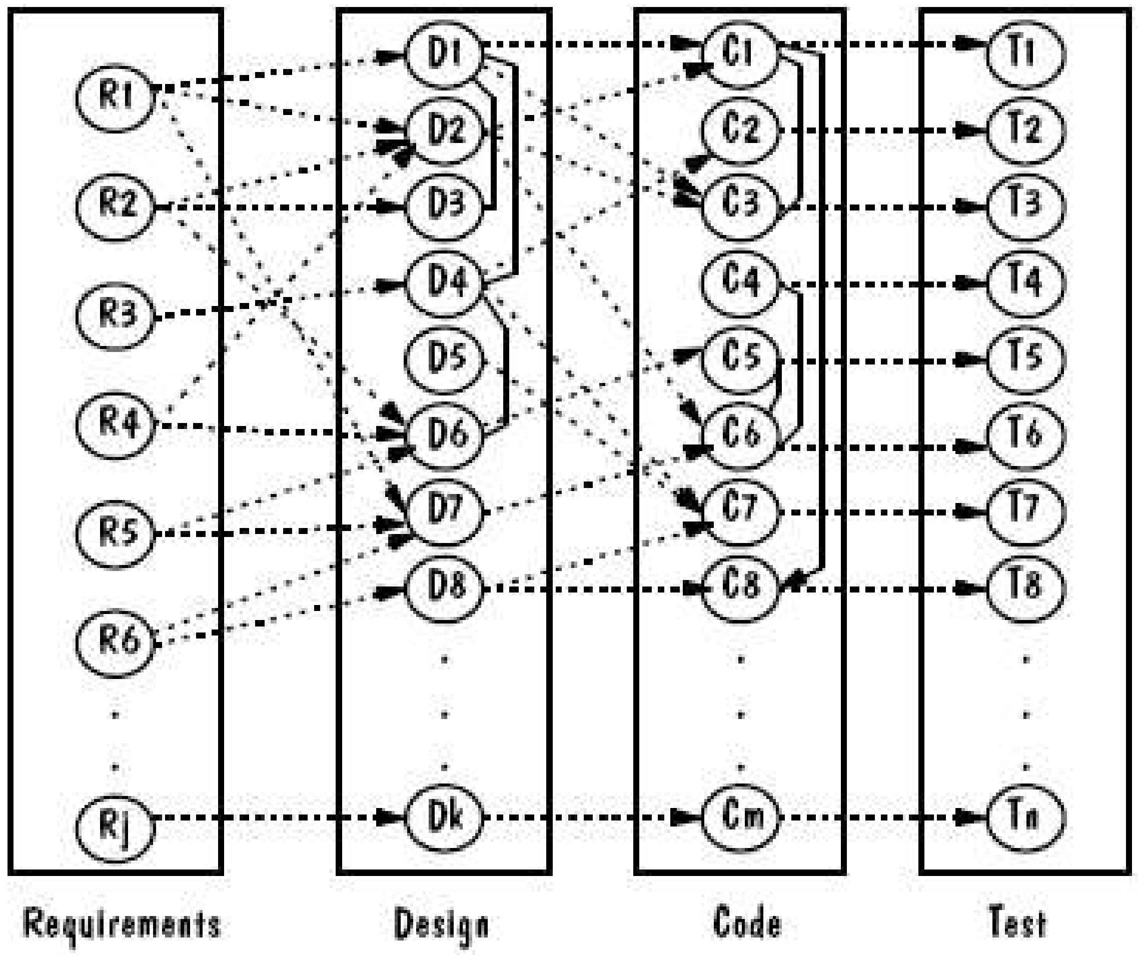
软件维护活动


Preventive
Adaptive
Corrective
Perfective
Manage software maintenance
NEW SYSTEM
Change Request
Analyze software change impact
Understand software under change
Implement maintenance change
Account for ripple effect
(Re)test affected software
Existing system
Impact/scope
Traceability
roadmap
Complexity
Modularity
Documentation
Self-descriptiveness
Adaptability
Stability
Consistency
Testability
Verifiability
Completeness

Horizontal traceability水平可追踪性



Underlying graph for maintenance维护的基础图



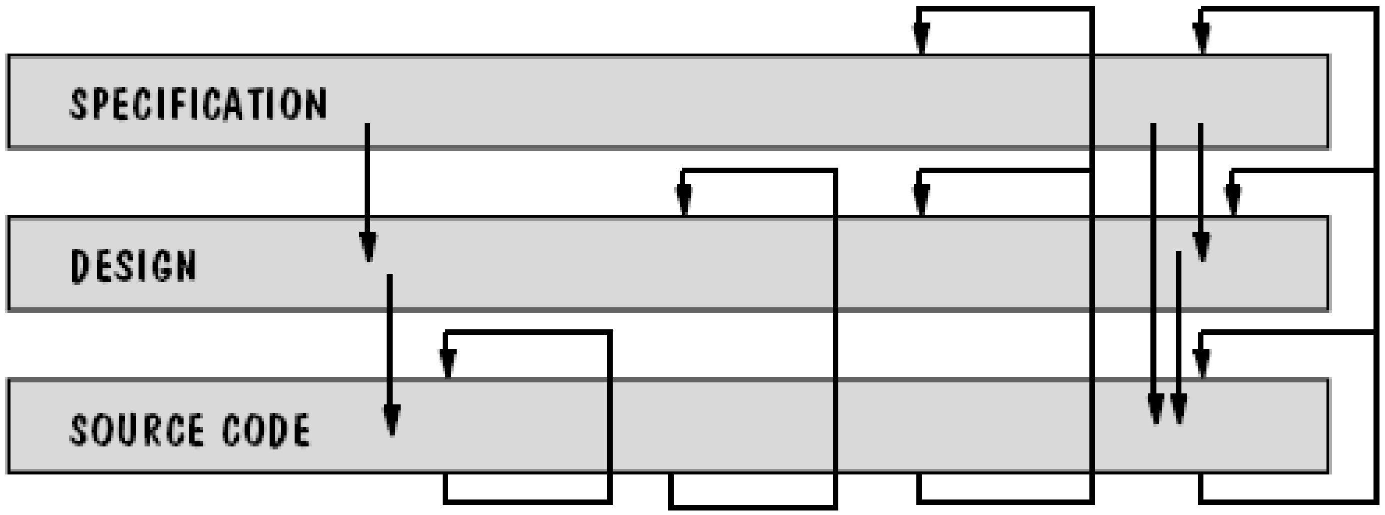
Automated maintenance tools自动维护工具

- Text editors文本编辑器
- File comparators文件比较器
- Compilers and linkers编译器和连接器
- Debugging tools调试工具
- Cross-reference generators交叉引用生成器
- Static code analyzers静态代码分析器
- Configuration management repositories配置管理库

Software rejuvenation软件再生

- Redocumentation文档重构：static analysis adds more information静态分析，得出更多的信息
- Restructuring结构重组：transform to improve code structure使代码结构变好
- Reverse engineering逆向工程：recreate design and specification information from the code从代码中重构出设计和说明信息
- Reengineering再工程:

软件再生分类



Forward Engineering
- forward progression through process

Restructuring
- from code
- internally represent
- iteratively simplify structure and eliminate dead code
- regenerate code

Redocumenting
- from code
- static analysis reports on structure, complexity, volume, data, etc.
- not based on software methods

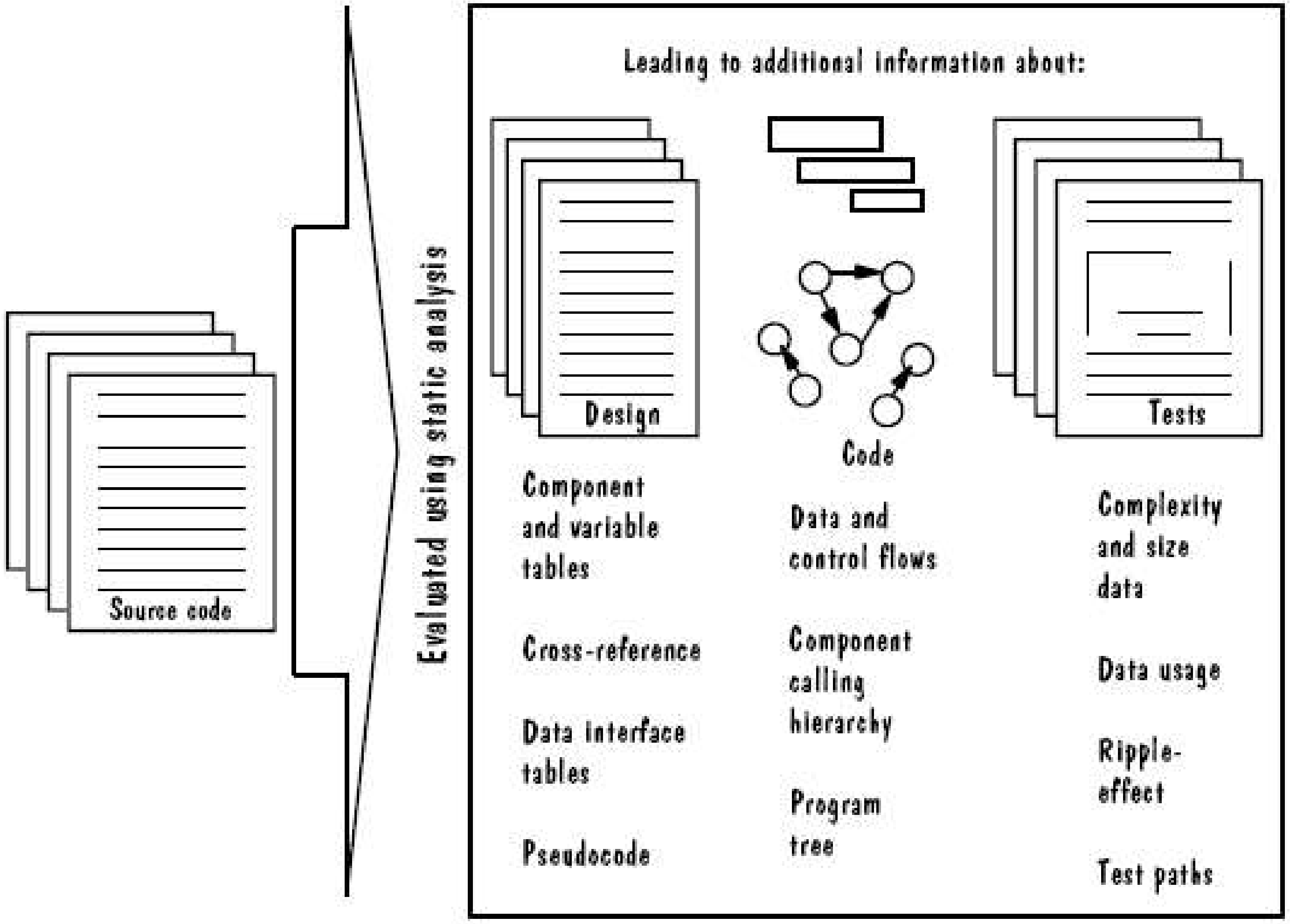
Reverse Engineering
- from code
- produces design and specification based on accepted software methods
- manages representation

Reengineering
- from code
- reverse-engineer code
- forward-engineer: complete and modify representation regenerate code

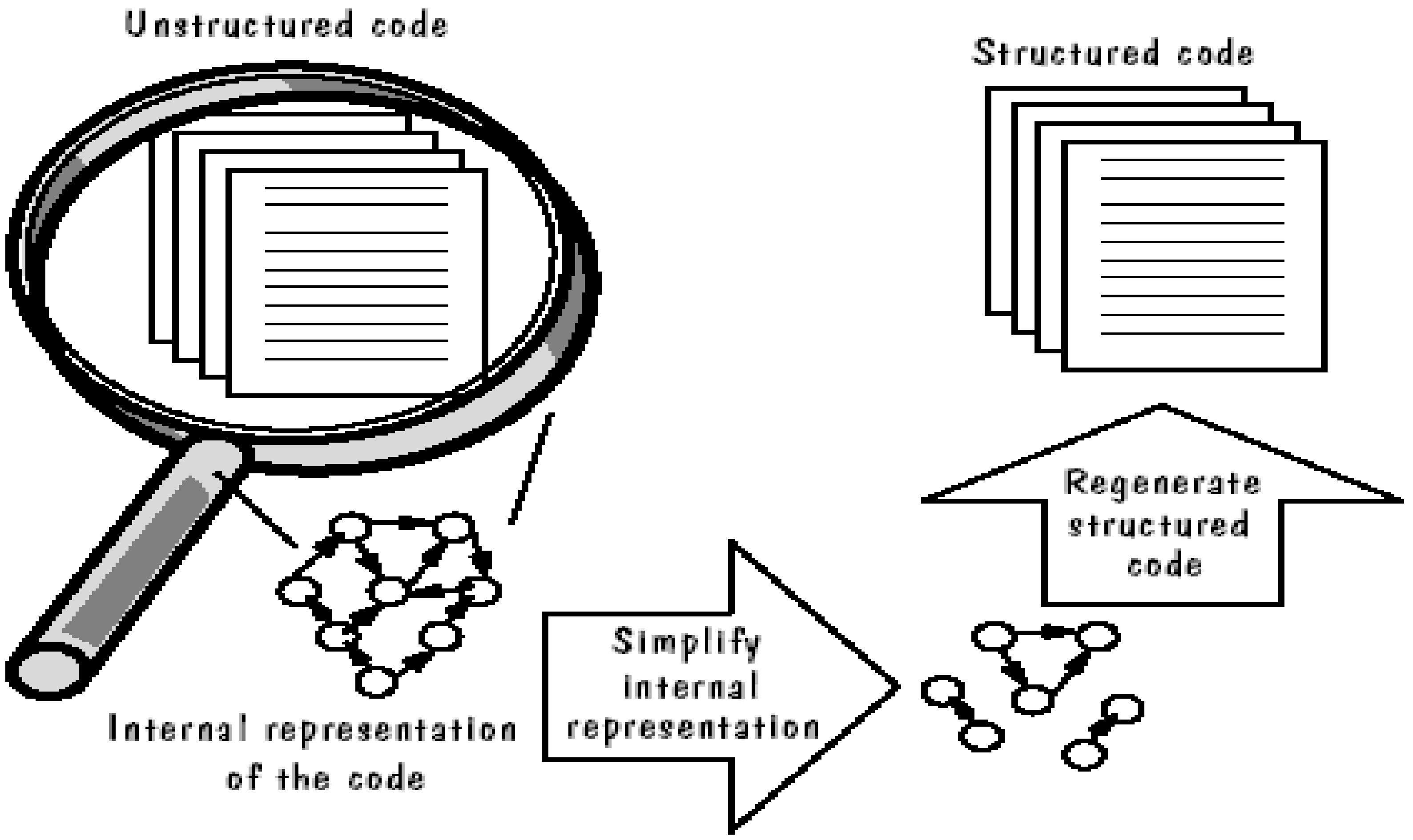
Redocumentation文档重构

- Output may include输出可能包括:
  - component calling relationships组件调用关系
  - data-interface tables数据接口表
  - data-dictionary information数据字典信息
  - data flow tables or diagrams数据流表或图
  - control flow tables or diagrams控制流表或图
  - Pseudocode伪代码
  - test paths测试路径
  - component and variable cross-references组件和变量的交叉引用

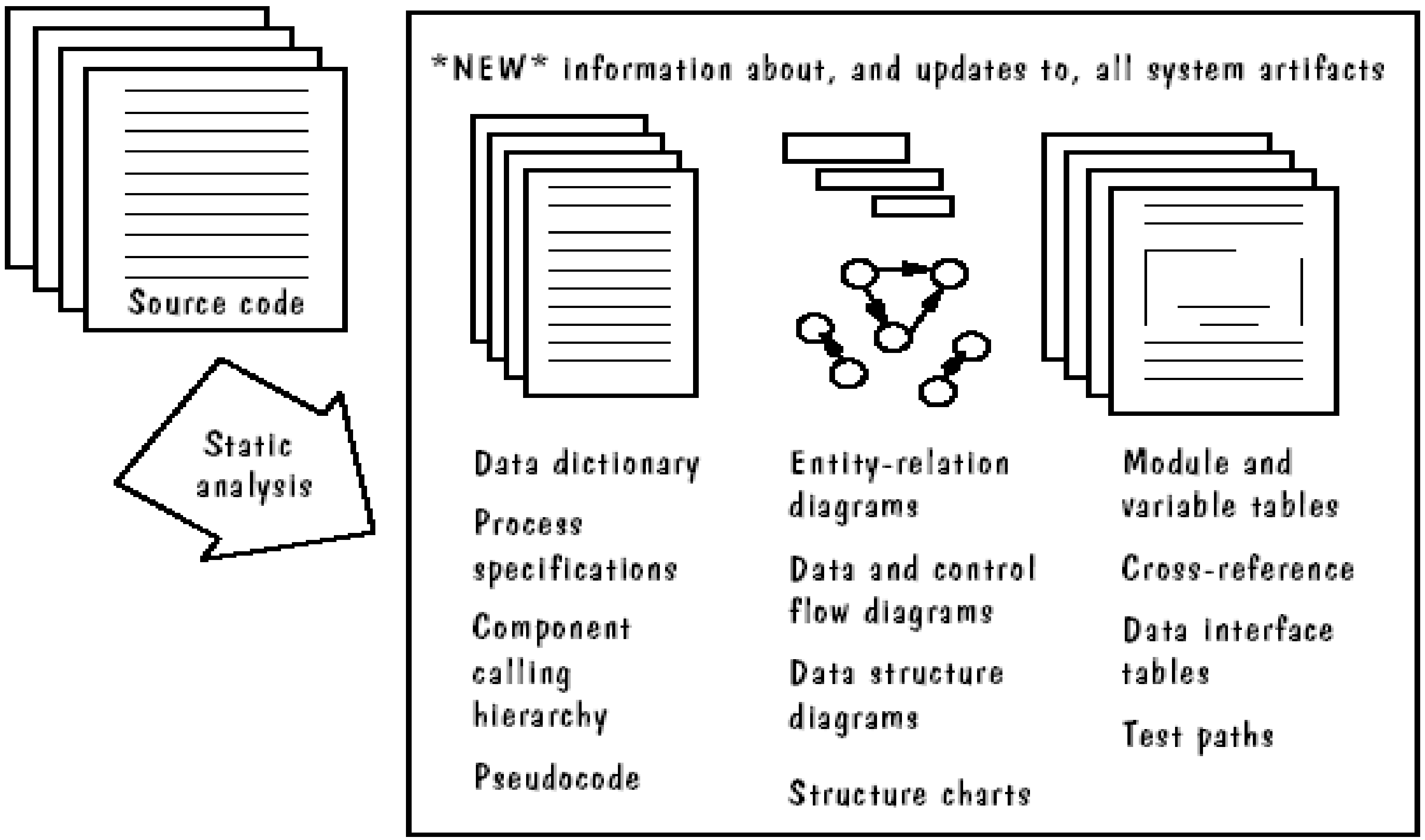
Redocumentation



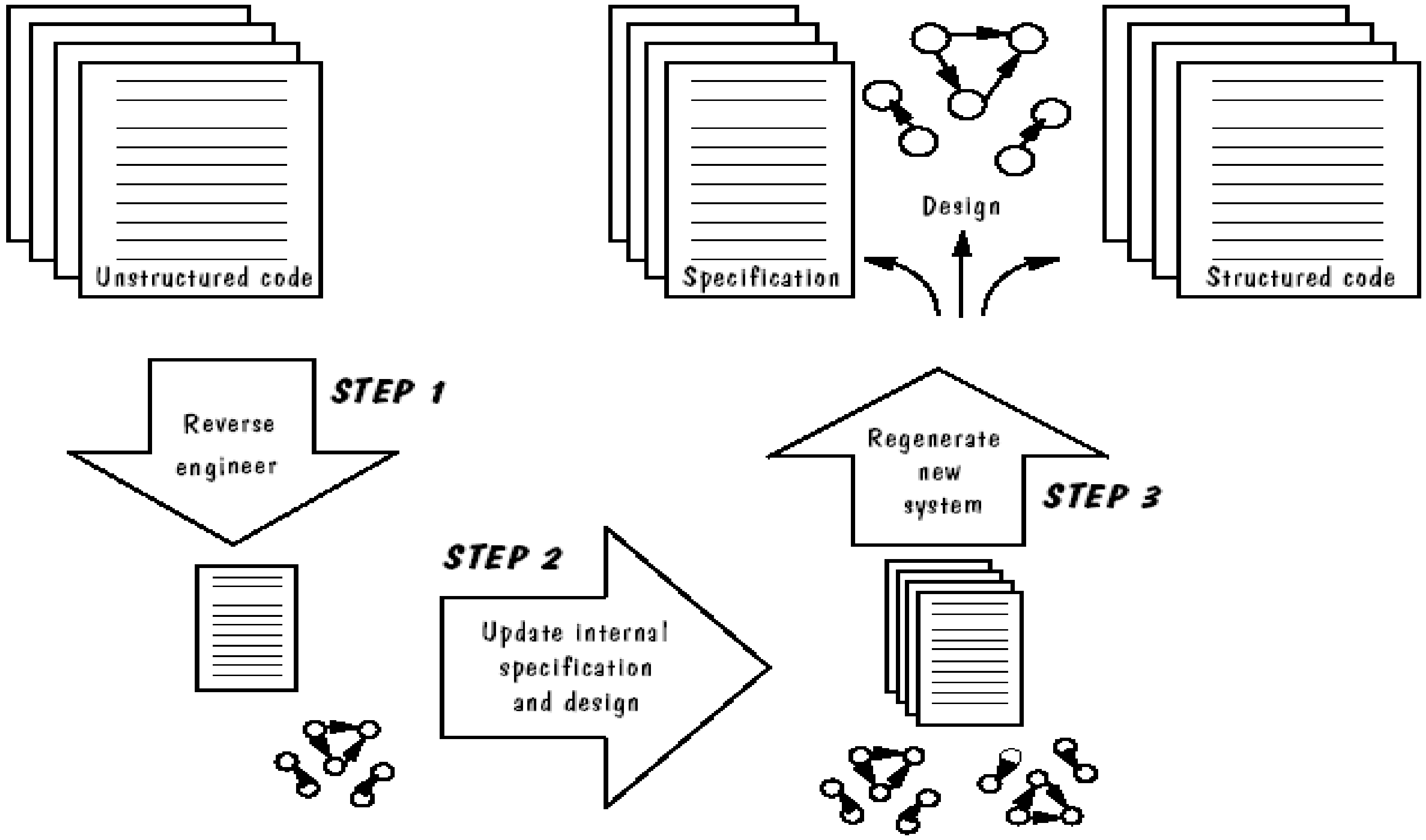
Restructuring结构重组


Unstructured code
Structured code
Regenerate structured code
Simplify internal representation
Internal representation of the code

Reverse Engineering逆向工程



Reengineering再工程


Unstructured code
Specification
Design
Structured code
STEP 1
Reverse engineer
STEP 2
Update internal specification and design
STEP 3
Regenerate new system