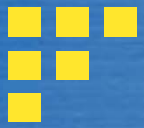
Chapter 15
Product Metrics for Software

Software Engineering: A Practitioner's Approach, 6th edition
by Roger S. Pressman

本章要点

- 软件质量
- 产品度量
  - 分析度量
  - 设计度量
  - 代码度量
  - 测试度量



概述

- 产品度量有助于软件工程师对其设计和构造的软件获得深层次的了解。
- 产品度量为分析、设计、编码和测试能更客观地执行和更定量的评估提供基础。
- 与项目度量和过程度量不同【第22章】。

15.1 软件质量

- 一般来讲，软件质量是对明确陈述的功能和性能需求、明确记录的开发标准以及对所有专业化软件开发应具备的隐含特征的符合度。
  - 软件需求是质量测量的基础，不符合需求就是没有质量。
  - 若未能遵守开发准则，则肯定质量有问题。
  - 若软件符合显示需求，但未能满足其隐式需求，则软件质量仍然值得怀疑。

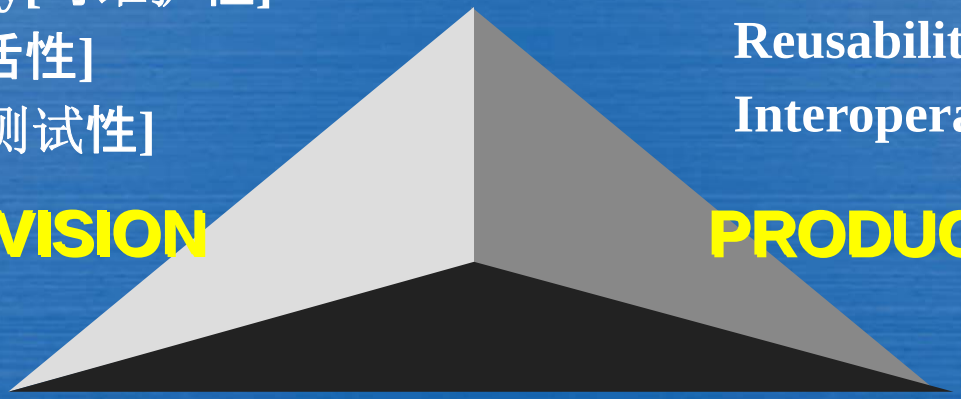
McCall's Triangle of Quality

Maintainability[可维护性]
Flexibility[灵活性]
Testability[可测试性]

PRODUCT REVISION

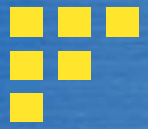
Portability[可移植性]
Reusability[可复用性]
Interoperability[互操作性]

PRODUCT TRANSITION



PRODUCT OPERATION

Correctness Usability Efficiency
Reliability Integrity



A Comment

McCall's quality factors were proposed in the early 1970s. They are as valid today as they were in that time. It's likely that software built to conform to these factors will exhibit high quality well into the 21st century, even if there are dramatic changes in technology.

ISO 9126质量因素

- 功能性: Functionality
- 可靠性: Reliability
- 易用性: Usability
- 效率: Efficiency
- 可维护性: Maintainability
- 可移植性: Portability

15.2 产品度量框架

- 软件产品：文档、代码、软件等
- 度量手段:
  - 测度（measures）
  - 度量（metrics）
  - 指标（Indicators）
- 度量原则
- 度量过程

度量原则

- 设定度量目标：The objectives of measurement should be established before data collection begins;
- 定义要明确：Each technical metric should be defined in an unambiguous manner;
- 有效理论支持：Metrics should be derived based on a theory that is valid for the domain of application (e.g., metrics for design should draw upon [利用] basic design concepts and principles and attempt to provide an indication of the presence of an attribute that is deemed desirable);
- 度量指标的选择要是最合适的：Metrics should be tailored to best accommodate specific products and processes.

典型度量过程

- Formulation【公式化】. The derivation of software measures and metrics appropriate for the representation of the software that is being considered.
- Collection【收集数据】. The mechanism used to accumulate data required to derive the formulated metrics.
- Analysis【分析结果】. The computation of metrics and the application of mathematical tools.
- Interpretation【解释评估】. The evaluation of metrics results in an effort to gain insight into the quality of the representation.
- Feedback【反馈】 Recommendations derived from the interpretation of product metrics transmitted to the software team.

面向目标的软件度量

- The Goal/Question/Metric（GQM）Paradigm
  - establish an explicit measurement goal that is specific to the process activity or product characteristic that is to be assessed
  - define a set of questions that must be answered in order to achieve the goal, and
  - identify well-formulated metrics that help to answer these questions.
- Goal definition template

  Analyze {the name of activity or attribute to be measured}
  for the purpose of {the overall objective of the analysis}
  with respect to {the aspect of the activity or attribute that is considered}
  from the viewpoint of {the people who have an interest in the measurement}
  in the context of {the environment in which the measurement takes place}.

度量属性的有效性

- **简单的和可计算的**. It should be relatively easy to learn how to derive the metric, and its computation should not demand inordinate effort or time
- **在经验上和直观上有说服力**. The metric should satisfy the engineer's intuitive notions about the product attribute under consideration
- **一致的和客观的**. The metric should always yield results that are unambiguous.
- **单位和量纲的使用是一致的**. The mathematical computation of the metric should use measures that do not lead to bizarre combinations of unit.
- **编程语言的独立性**. Metrics should be based on the analysis model, the design model, or the structure of the program itself.
- **高质量反馈的有效机制**. That is, the metric should provide a software engineer with information that can lead to a higher quality end product

Collection and Analysis Principles

- Whenever possible, data collection and analysis should be automated;
- Valid statistical techniques should be applied to establish relationship between internal product attributes and external quality characteristics
- Interpretative guidelines and recommendations should be established for each metric

软件产品度量全景

- 分析度量
- 设计度量
- 代码度量
- 测试度量

15.3 Analysis Metrics

- Function-based metrics: use the function point as a normalizing factor or as a measure of *the "size" of the specification*
- Specification metrics: used as an indication of *quality of analysis model and requirements specification* by measuring number of characteristics, such as un-ambiguity, integrity, correctness, understandability, etc.

Function-Based Metrics

- The function point metric (FP), first proposed by Albrecht [ALB79], can be used effectively as a means for measuring the functionality delivered by a system.
- Function points are derived using an empirical relationship based on countable (direct) measures of software's information domain and assessments of software complexity
- Information domain values are defined in the following manner:
  - number of external inputs (EIs)
  - number of external outputs (EOs)
  - number of external inquiries (EQs)
  - number of internal logical files (ILFs)
  - Number of external interface files (EIFs)

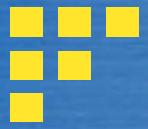
Function Points

Information Domain Value	Count		Weighting factor simple	average	complex		
External Inputs (EIs)	3	x	3	4	6	=	9
External Outputs (EOs)	2	x	4	5	7	=	8
External Inquiries (EQs)	2	x	3	4	6	=	6
Internal Logical Files (ILFs)	1	x	7	10	15	=	7
External Interface Files (EIFs)	4	x	5	7	10	=	20
Count total							50

FP=总计（count total）×【0.65+0.01× Σ(F_i)】

15.4 设计度量

- 体系结构设计度量
- OO设计度量
- 面向类的度量
- 构件级设计度量
- 面向操作度量
- 用户界面设计度量



Architectural Design Metrics

- Architectural design metrics
  - Structural complexity = g(fan-out)
  - Data complexity = f(input & output variables, fan-out)
  - System complexity = h(structural & data complexity)
- HK metric: architectural complexity as a function of fan-in and fan-out
- Morphology metrics[形态度量学]: a function of the number of modules and the number of interfaces between modules

Metrics for OO Design-I

- Whitmire [WHI97] describes nine distinct and measurable characteristics of an OO design:
  - **Size**
    - Size is defined in terms of four views: population, volume, length, and functionality
  - **Complexity**
    - How classes of an OO design are interrelated to one another
  - **Coupling**
    - The physical connections between elements of the OO design
  - **Sufficiency**
    - "the degree to which an abstraction possesses the features required of it, or the degree to which a design component possesses features in its abstraction, from the point of view of the current application."
  - **Completeness**
    - An indirect implication about the degree to which the abstraction or design component can be reused

Metrics for OO Design-II

- **Cohesion**
  - The degree to which all operations working together to achieve a single, well-defined purpose
- **Primitiveness**
  - Applied to both operations and classes, the degree to which an operation is atomic
- **Similarity**
  - The degree to which two or more classes are similar in terms of their structure, function, behavior, or purpose
- **Volatility**
  - Measures the likelihood that a change will occur

Class-Oriented Metrics (1)

- weighted methods per class (WMC)
- depth of the inheritance tree (DIT)
- number of children (NOC)
- coupling between object classes (CBO)
- response for a class (RFC)
- lack of cohesion in methods (LCOM)

Proposed by Chidamber and Kemerer

Class-Oriented Metrics (2)

- class size (CS)
- number of operations overridden by a subclass (NOO)
- number of operations added by a subclass (NOA)
- specialization index (SI)

Proposed by Lorenz and Kidd [LOR94]:

Class-Oriented Metrics (3)

- Method inheritance factor (MIF)
- Coupling factor (CF)
- Polymorphism factor (PF)

The MOOD Metrics Suite

Operation-Oriented Metrics

- average operation size (OS_{avg})
- operation complexity (OC)
- average number of parameters per operation (NP_{avg})

Proposed by Lorenz and Kidd [LOR94]:

Component-Level Design Metrics

- Cohesion metrics: a function of data objects and the locus of their definition
- Coupling metrics: a function of input and output parameters, global variables, and modules called
- Complexity metrics: hundreds have been proposed (e.g., cyclomatic complexity)

Interface Design Metrics

- Layout appropriateness[布局恰当性]: a function of layout entities, the geographic position and the "cost" of making transitions among entities

15.5 Code Metrics

- **Halstead's Software Science**: a comprehensive collection of metrics all predicated on[以...为基础] the number (count and occurrence) of operators and operands within a component or program
  - An "operator" is a fixed symbol or reserved word in a language, and "operand" is everything else: variable names, function names, numeric constants, etc. Comments don't count.

    $\mathbf{Long=n_1 log n_1 + n_2 log n_2}$

    $\mathbf{Volume=Long \times log_2(n_1+n_2)}$

15.6 Metrics for Testing

- Testing effort[测试工作量] can also be estimated using metrics derived from Halstead measures[参见15-7a, 15-7b]
- Binder [BIN94] suggests a broad array of design metrics that have a direct influence on the "testability" of an OO system.
  - Lack of cohesion in methods (LCOM).
  - Percent public and protected (PAP).
  - Public access to data members (PAD).
  - Number of root classes (NOR).
  - Fan-in (FIN).
  - Number of children (NOC) and depth of the inheritance tree (DIT).