



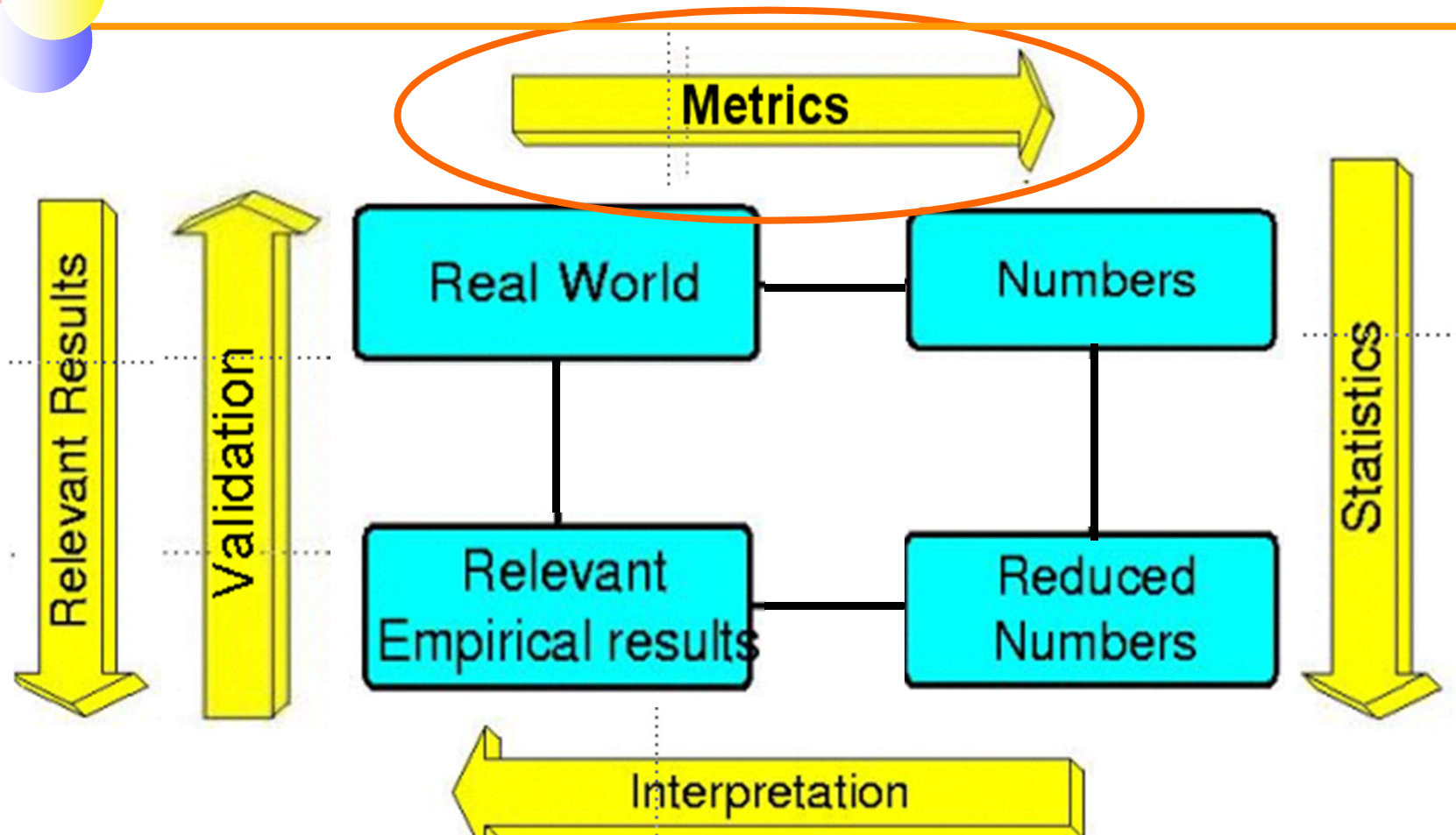
Software Metrics

Lecture 3

A Goal-based Framework for Software Measurement

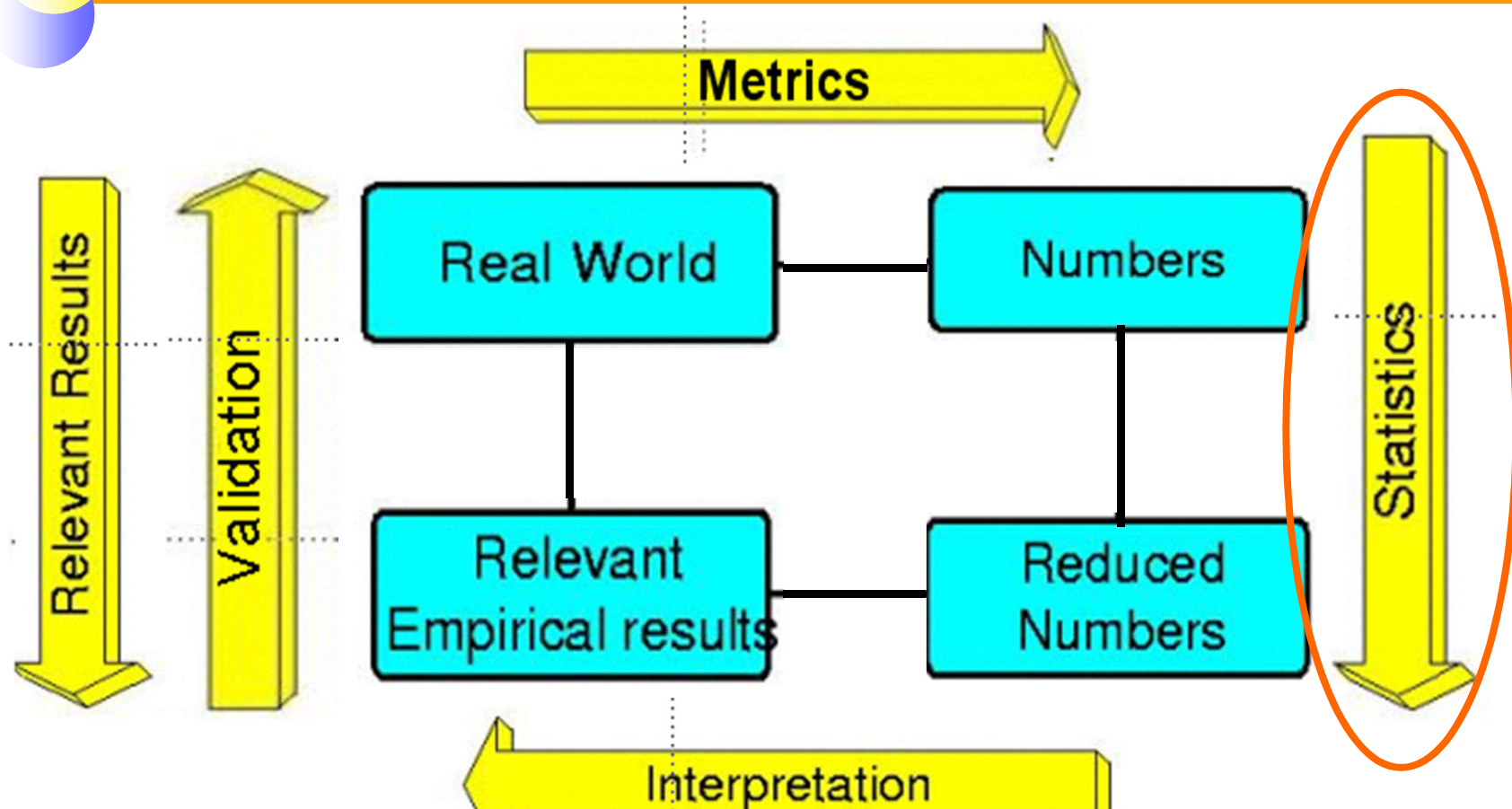
Yuming Zhou

Measurement Theory



- Any metric that satisfies the representation condition is a valid metric

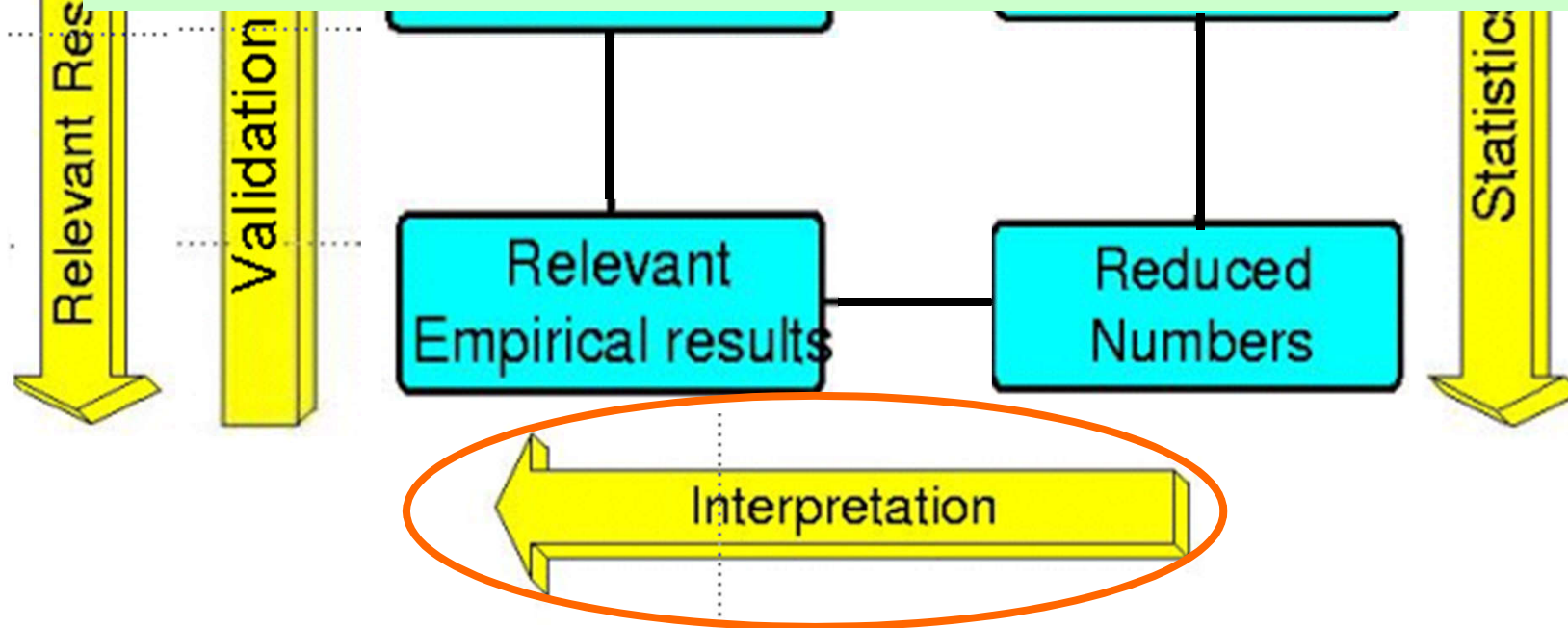
Measurement Theory



- Scale type affects the **types of operations and statistical analyses** that can be applied to the data

Measurement Theory

- A statement involving measurement is **meaningful** iff its truth value is invariant of transformations of allowable scales

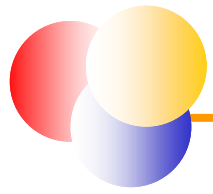




Contents

- ▶ Software Metrics Classification
- ▶ Goal-Based Framework
- ▶ Measurement Validation





Goal-Based Measurement (GBM)

- The primary question in goal-based measurement

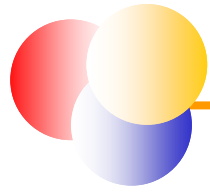
“What do we want to know or learn?”

instead of “What metrics should we use?”

Because the answers depend on your goals, no fixed set of metrics is universally appropriate

- Instead of attempting to develop general-purpose metrics, one has to describe **an adaptable process** that users can use to identify and define metrics that provide insights into their own development problem **授人以鱼，不如授人以渔**





GBM Process

- **Determining what to measure**
 - Identifying entities
 - Classifying entities to be examined
 - Determining relevant goals

- **Determining how to measure**
 - Inquire about metrics
 - Assign metrics



Section 1

Software Metrics Classification

- **Types of Entities**
- **Types of Attributes**





Types of Entities

■ Process

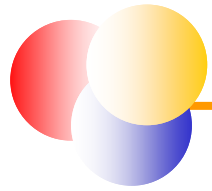
- A collection of software related activities usually associated with some timescale
- Different SE processes: development, maintenance, testing, reuse, configuration and management process, etc

■ Product

- Any artifacts, deliverables or documents that result from a process activity

■ Resource

- Entities required by a process activity



Types of Attributes

- **Internal attributes**

- Attributes that can be measured entirely in terms of the process, product or resource itself **separate from its behaviour**

- **External attributes**

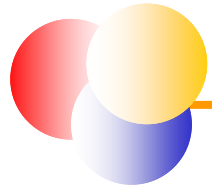
- Attributes that can be measured with respect to how the process, product or resource relates to its **environment through its behaviour**





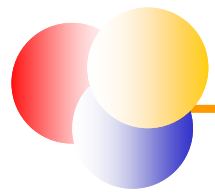
Sample Process Metrics /1

Process Entities	Attributes	Possible metrics
development process	elapsed time	Calendar days, working days
	milestones	Calendar dates
	development effort	Staff-hours, days, or months
test process	volume	Number of tests scheduled
	progress	Number of tests executed Number of tests passed



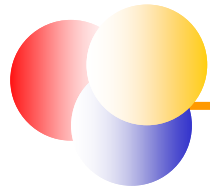
Sample Process Metrics /2

Process Entities	Attributes	Possible metrics
detailed design	elapsed time	calendar days, working days
	design quality	Defect density: number of design defects found in downstream activities divided by a metric of product size, such as function points or physical source lines of code.
maintenance	cost	dollars per year staff-hours per change request



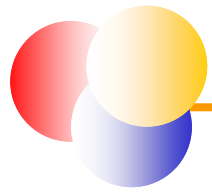
Sample Product Metrics /1

Product entities	Attributes	Possible metrics
System	Size	Number of modules Number of function points Number of physical source line of code
	defect density	Defects per KLOC Defects per function point
Module	Length	Physical source lines of code Logical source statements
	Percent reused	Ratio of unchanged physical lines to total physical lines, comments and blanks excluded



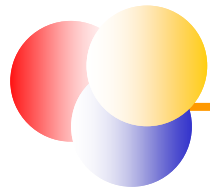
Sample Product Metrics /2

Product entities	Attributes	Possible metrics
Unit	Number of linearly independent flowpaths	McCabe's complexity
Document	Length	Number of pages
Line of code	Statement type	Type names
	Programming language	Language name



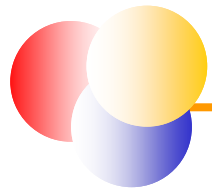
Sample Resource Metrics

Resource Entities	Attributes	Possible metrics
Assigned staff	Team size	Number of people assigned
	Experience	Years of domain experience Years of programming experience
CASE tools	Type	Name of type
	Is used?	Yes/no (a binary classification)
Time	Start date, due date	Calendar dates
	Execution time	CPU clocks



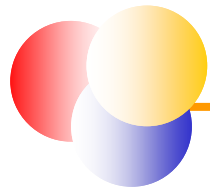
Sample Attributes (Process)

Processes	Internal attributes	External attributes
Construct Specifications	Time, effort, number of requirements changes, ...	Quality, cost, stability, ...
Detailed Designs	Time, effort, number of specification faults found ...	Cost-effectiveness, cost, ...
Testing	Time, effort, number of bugs found, ...	Cost-effectiveness, stability, cost, ...
...



Sample Attributes (Product)

Products	Internal attributes	External attributes
Specifications	Size, reuse, modularity, functionality, syntactic correctness...	Comprehensibility , maintainability, ...
Designs	Size, reuse, modularity, coupling, cohesiveness, functionality...	Quality, complexity, maintainability, ...
Code	Size, reuse, modularity, coupling, functionality, algorithm complexity, control-flow structuredness	Reliability , usability, maintainability, ...
Test Data	Size, coverage level,...	Quality, ...



Sample Attributes (Resources)

Resources	Internal attributes	External attributes
Personnel	Age, price, ...	Productivity, experience, intelligence, ...
Teams	Size, communication level, structuredness,	Productivity, quality
Software	Price, size,	Usability, reliability
Hardware	Price, speed, memory size, ...	Reliability, ...
Offices	Size, temperature, light	Comfort, quality

数据集示例1

COCOMO 81
(Promisedata.org)



1. Product

	<i>Description</i>	<i>Very Low</i>	<i>Low</i>	<i>Nominal</i>	<i>High</i>	<i>Very High</i>	<i>Extra High</i>
RELY	Required software reliability	0.75	0.88	1.00	1.15	1.40	-
DATA	Database size	-	0.94	1.00	1.08	1.16	-
CPLX	Product complexity	0.70	0.85	1.00	1.15	1.30	1.65



2. Computer

	<i>Description</i>	<i>Very Low</i>	<i>Low</i>	<i>Nominal</i>	<i>High</i>	<i>Very High</i>	<i>Extra High</i>
TIME	Execution time constraint	-	-	1.00	1.11	1.30	1.66
STOR	Main storage constraint	-	-	1.00	1.06	1.21	1.56
VIRT	Virtual machine volatility	-	0.87	1.00	1.15	1.30	-
TURN	Computer turnaround time	-	0.87	1.00	1.07	1.15	-



3. Personnel

	<i>Description</i>	<i>Very Low</i>	<i>Low</i>	<i>Nominal</i>	<i>High</i>	<i>Very High</i>	<i>Extra High</i>
ACAP	Analyst capability	1.46	1.19	1.00	0.86	0.71	-
AEXP	Applications experience	1.29	1.13	1.00	0.91	0.82	-
PCAP	Programmer capability	1.42	1.17	1.00	0.86	0.70	-
VEXP	Virtual machine experience	1.21	1.10	1.00	0.90	-	-
LEXP	Language experience	1.14	1.07	1.00	0.95	-	-

4. Project

	<i>Description</i>	<i>Very Low</i>	<i>Low</i>	<i>Nominal</i>	<i>High</i>	<i>Very High</i>	<i>Extra High</i>
MODP	Modern programming practices	1.24	1.10	1.00	0.91	0.82	-
TOOL	Software Tools	1.24	1.10	1.00	0.91	0.83	-
SCED	Development Schedule	1.23	1.08	1.00	1.04	1.10	-



数据集示例2

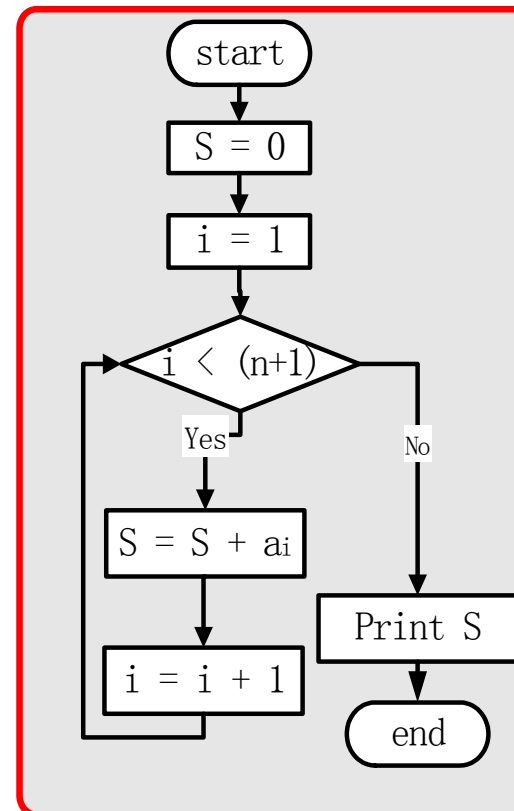
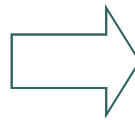
Ant 1.3
(Promisedata.org)



WMC

圈复杂性 $v(G) = e - n + 2p$
 $= 8 - 8 + 2 \times 1$
 $= 2$

```
void sum()
{
    S=0;
    i=1;
    while(i<n+1)
    {
        S=S+a[i];
        i=i+1;
    }
    printf("S
is %d", s);
}
```

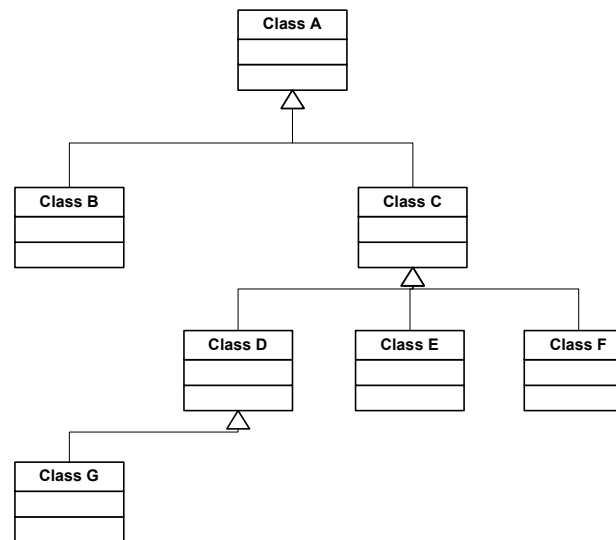


DIT和NOC

- $DIT(A)=0$
- $DIT(B,C)=1$
- $DIT(D,E,F)=2$
- $DIT(G)=3$

Another Example

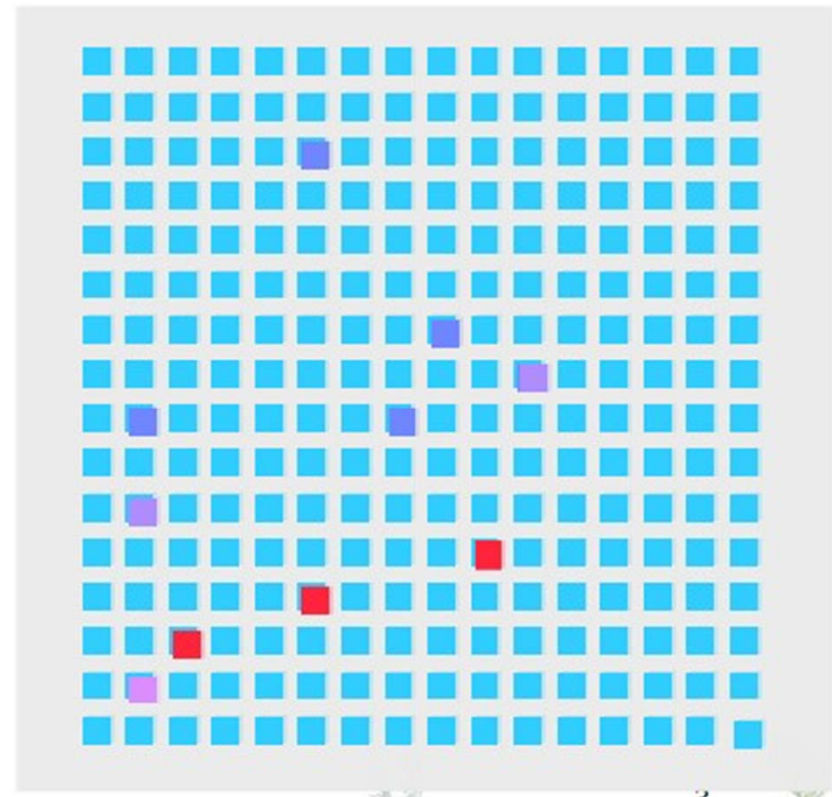
- $NOC(A)=2$
- $NOC(C)=3$
- $NOC(D)=1$
- $NOC(B,E,F,G)=0$



度量的一个重要应用场景

缺陷预测

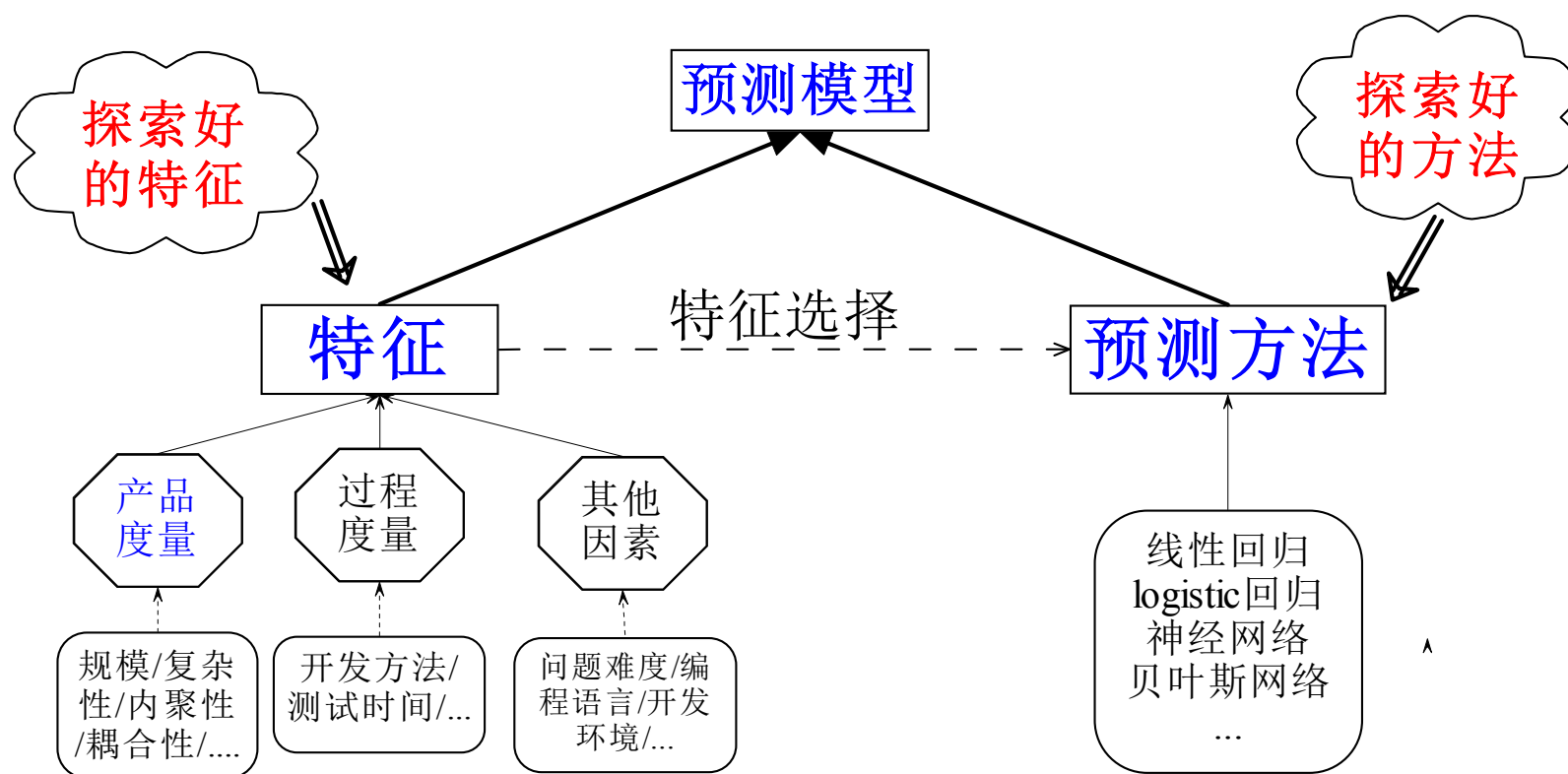
哪些模块有缺陷？



一个软件系统

度量的一个重要应用场景

缺陷预测



文献中常用的过程度量

Name	Description
NR	Number of revisions
NREF	Number of times a file has been refactored
NFIX	Number of times a file was involved in bug-fixing
NAUTH	Number of authors who committed the file
LINES	Lines added and removed (sum, max, average)
CHURN	Codechurn (sum, maximum and average) Codechurn is computed as $\sum_R (addedLOC - deletedLOC)$, where R is the set of all revisions
CHGSET	Change set size, <i>i.e.</i> , number of files committed together to the repository (maximum and average)
AGE	Age (in number of weeks) and weighted age computed as $\frac{\sum_{i=1}^N Age(i) \times addedLOC(i)}{\sum_{i=1}^N addedLOC(i)}$, where $Age(i)$ is the number of weeks starting from the release date for revision i , and $addedLOC(i)$ is the number of lines of code added at revision i



文献中常用的产品度量

Type	Name	Description
CK	WMC	Weighted Method Count
CK	DIT	Depth of Inheritance Tree
CK	RFC	Response For Class
CK	NOC	Number Of Children
CK	CBO	Coupling Between Objects
CK	LCOM	Lack of Cohesion in Methods
OO	FanIn	Number of other classes that reference the class
OO	FanOut	Number of other classes referenced by the class
OO	NOA	Number of attributes
OO	NOPA	Number of public attributes
OO	NOPRA	Number of private attributes
OO	NOAI	Number of attributes inherited
OO	LOC	Number of lines of code
OO	NOM	Number of methods
OO	NOPM	Number of public methods
OO	NOPRM	Number of private methods
OO	NOMI	Number of methods inherited





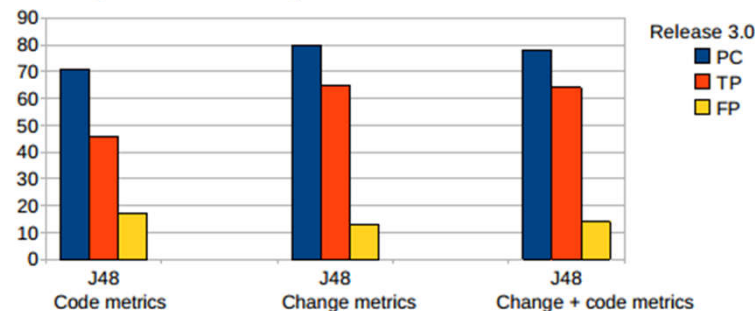
产品度量 **vs.** 过程度量
哪种预测缺陷更有效？



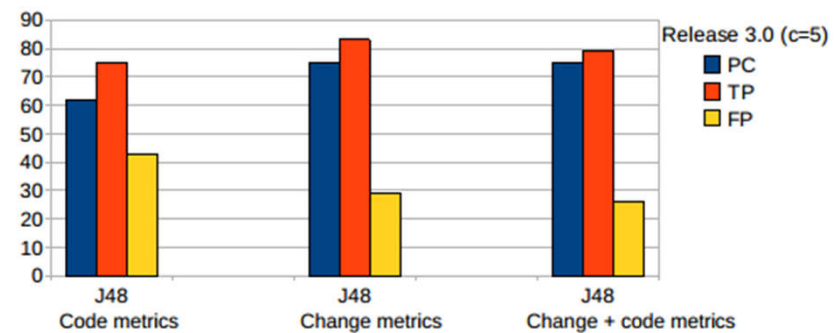
论文1:

Raimund Moser, Witold Pedrycz, and Giancarlo Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Proceedings of ICSE 2008*, pages 181–190, 2008.

Results (cost insensitive)



Results (cost sensitive)



预测

实际

	True Negative (TN)	False Positive (FP)
	False Negative (FN)	True Positive (TP)

$$\text{True positive rate (recall): } \frac{TP}{TP + FN}$$

$$\text{False positive rate: } \frac{FP}{FP + TN}$$

$$\text{Percentage of correctly predicted files: } \frac{TP + TN}{TP + TN + FP + FN}$$

论文2:

Evaluating defect prediction approaches: a benchmark and an extensive comparison
Marco D'Ambros, Michele Lanza and Romain Robbes, EMSE, 2012

分类

Predictor	Eclipse	Mylyn	Equinox	PDE	Lucene	AR
MOSER	0.921	0.864	0.876	0.853	0.881	<u>6</u>
NFIX-ONLY	0.795	0.643	0.742	0.616	0.754	24.2
NR	0.848	0.698	0.854	0.802	0.77	19.4
NFIX+NR	0.848	0.705	0.856	0.805	0.761	19.4
BUG-CAT	0.893	0.747	0.876	0.868	0.814	12.2
BUG-FIX	0.885	0.716	0.875	0.854	0.814	13.8
CK+OO	0.907	0.84	0.92	0.854	0.764	<u>7.2</u>
CK	0.905	0.803	0.909	0.798	0.721	13.8
OO	0.903	0.836	0.889	0.854	0.751	10.8
LOC	0.899	0.823	0.839	0.802	0.636	17.2

排序

Predictor	Eclipse	Mylyn	Equinox	PDE	Lucene	AR
<i>Popt</i>						
MOSER	0.871	0.832	0.898	0.776	0.843	<u>9.4</u>
NFIX-ONLY	0.783	0.642	0.831	0.636	0.707	23.8
NR	0.835	0.647	0.895	0.765	0.798	19
NFIX+NR	0.835	0.667	0.888	0.767	0.787	19.8
BUG-CAT	0.865	0.733	0.887	0.8	0.857	11.4
BUG-FIX	0.865	0.661	0.886	0.79	0.857	13.4
CK+OO	0.863	0.808	0.904	0.794	0.801	<u>10</u>
CK	0.857	0.763	0.894	0.769	0.731	17.8
OO	0.852	0.808	0.897	0.79	0.78	13.8
LOC	0.847	0.753	0.881	0.758	0.692	21.4



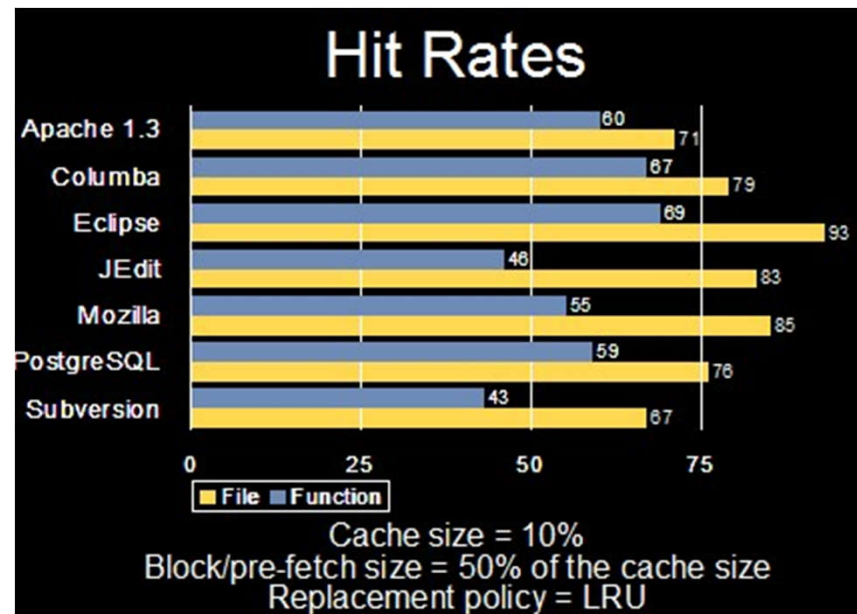
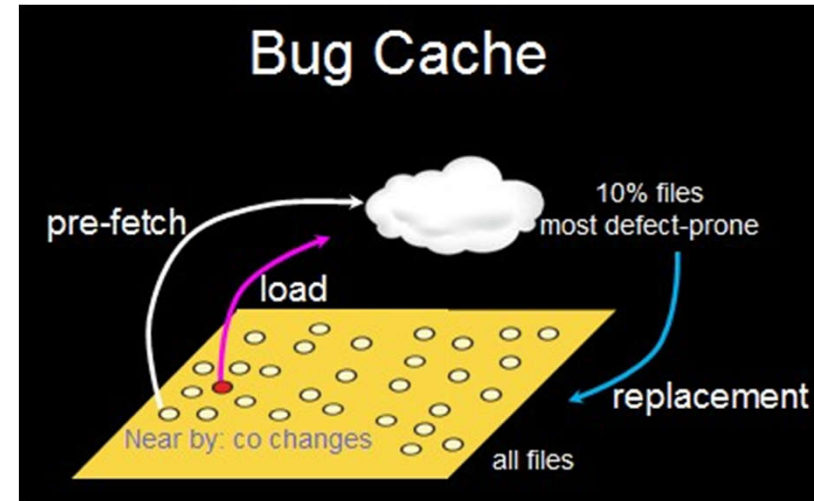
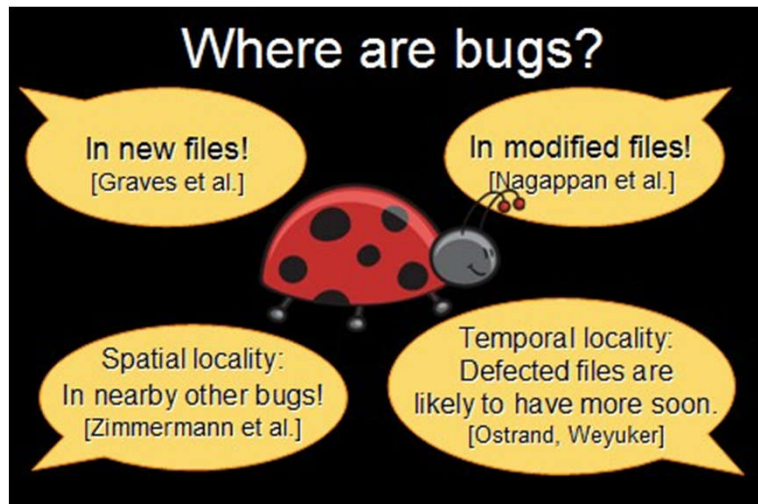
如何有效合并 产品度量和过程度量？



论文3:

S. Kim, et al. Predicting Faults from Cached History. ICSE 2007

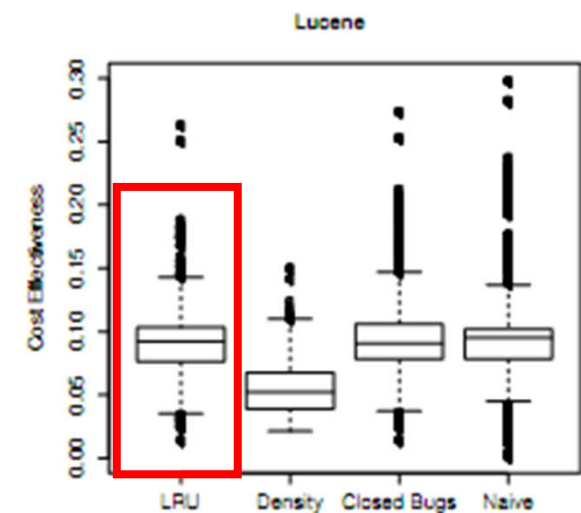
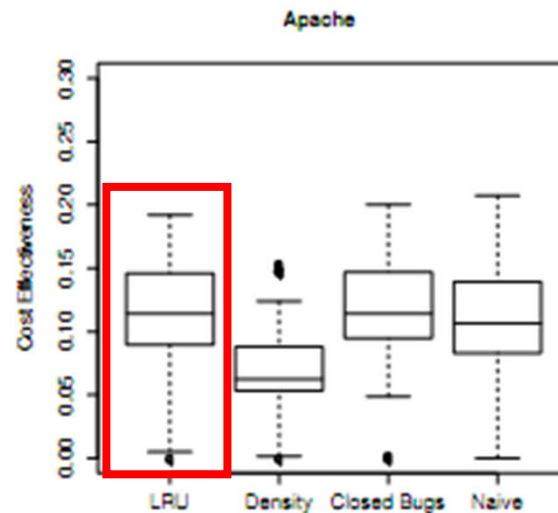
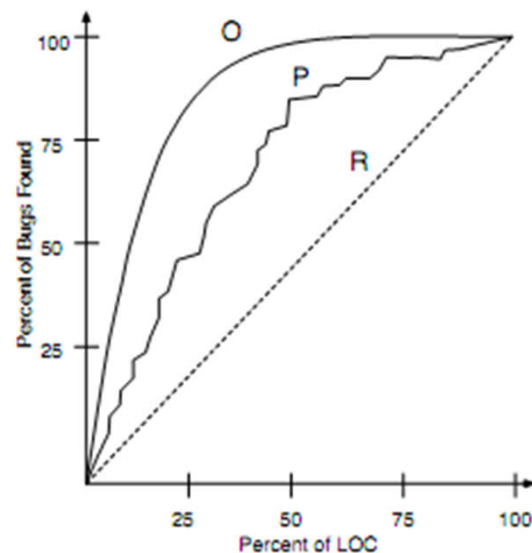
ACM SIGSOFT Distinguished Paper Award at ICSE 2007



bugCache真的有那么好吗？

论文4:

F. Rahman, et al. BugCache for inspections: hit or miss? FSE 2011.



The naive model is actually about the same utility for inspections, under the AUCEC₂₀ measure, as the best possible setting for FIXCACHE.

bugCache真的有那么好吗？

论文4:

F. Rahman, et al. BugCache for inspections: hit or miss? FSE 2011.
Nominee, ACM Distinguished Paper Award

如果考虑审查工作量，BugCache
方法与最简单的按历史缺陷数目排
序模型差不多！





既然考虑审查工作量时bugCache
的效果不好，如何改进？



Section 2

Goal-Based Framework

- **GQM Approach**

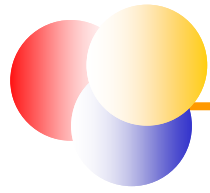




GQM Approach /1

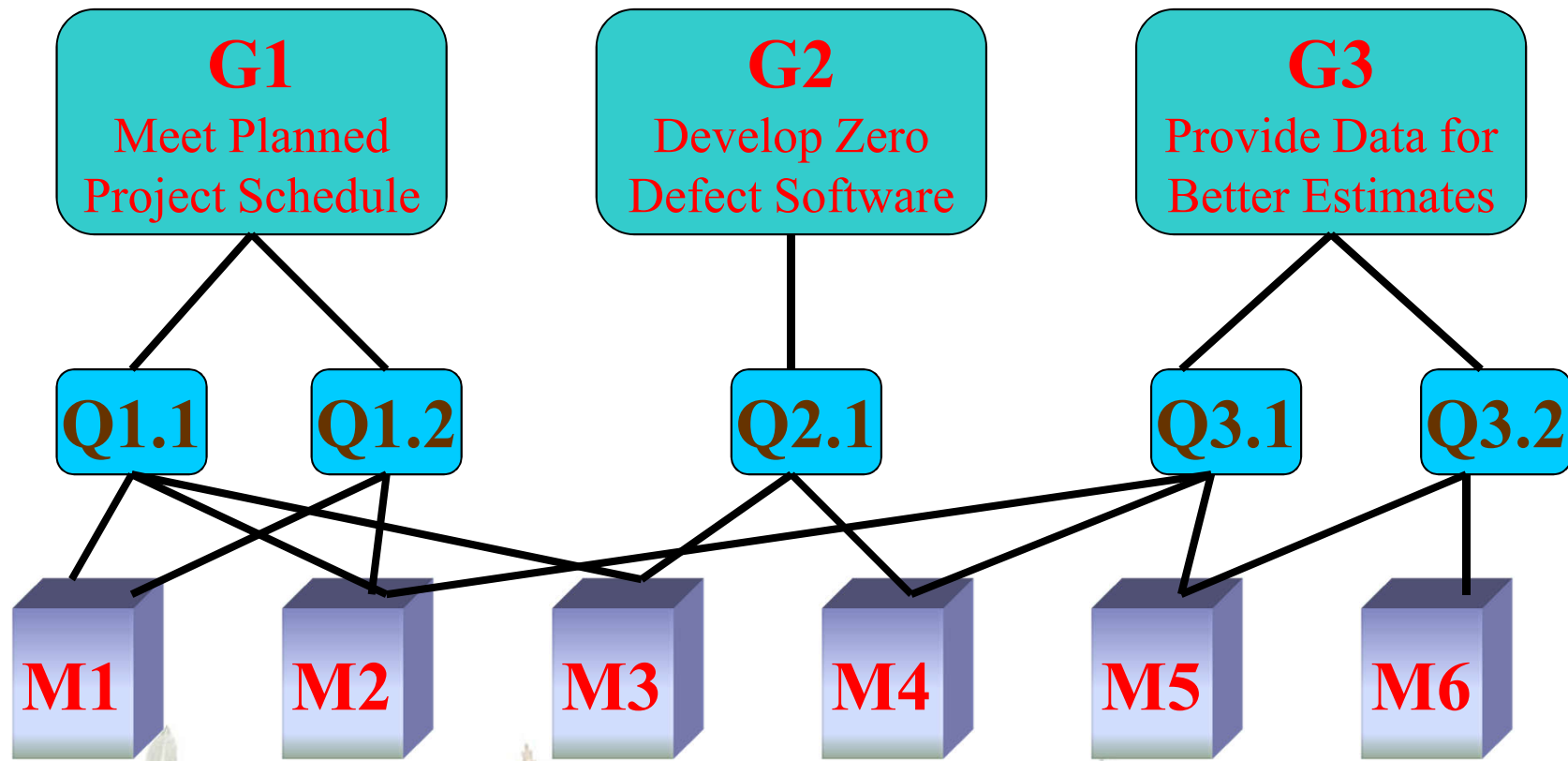
- Goal-Question-Metric (GQM) approach provides a framework for deriving metrics from organization or business goals
- References
 - V.R. Basili and D. Weiss (1984), *A Methodology for Collecting Valid Software Engineering Data*, IEEE Trans. Software Engineering, vol. 10, pp.728-738.
 - V.R. Basili and H.D. Rombach (1988), *The Tame Project: Towards Improvement-Oriented Software Environments*, IEEE Trans. Software Engineering, vol.14, pp.758-773.





GQM Approach /2

GQM Process

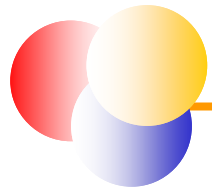




GQM Approach /3

- **Goal:** List major goals of development or maintenance project.
- **Question:** Derive from each goal the questions that must be answered to determine if the goals are being met.
- **Metrics:** Decide what must be measured in order to be able to answer the questions adequately.

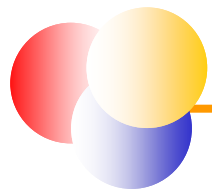




Template for Goal Definition (1)

- **Purpose:** To (characterize, evaluate, predict, motivate, etc.) the (process, product, model, metric, etc.) in order to (understand, assess, manage, engineer, learn, improve, etc.) it.
- **Example:** Evaluating maintenance process in order to improve it.

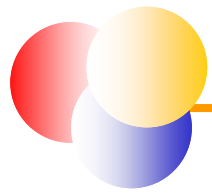




Template for Goal Definition (2)

- **Perspective:** Examine the (cost, effectiveness, correctness, defects, changes, product metrics, etc.) from the viewpoint of the (developer, manager, customer, etc.)
- **Example:** Examine the cost from the viewpoint of the manager.





Template for Goal Definition(3)

- **Environment:** The environment consists of the following: process factors, people factors, problem factors, methods, tools, constraints, etc.
- **Example:** The maintenance staff are poorly motivated programmers who have limited access to tools.





Examples (1)


- **Purpose:**

- Evaluate working environment **in order to** identify opportunities to improve the productivity of our development team

- **Perspective:**

- Examine the ratio of work time to break time of our employees, the accommodations, incentives, extra-curricular activities offered, workspace (room and desk size, ventilation) where our employees work **from the point of view of** the employees themselves

- **Environment and constraints:**

- Payroll applications programming in C++
 - 100 software developers with 5 or more years experience in C++
 - Customers are businesses
 - Do not maintain a reusable module database
 - Examine new projects completed and sold from 1/1/1998 to 31/12/2002
- 



Examples (2)

- **Purpose:**

- Evaluate the impact of various CASE tools on the productivity of the development team **in order to** help in the development of our product

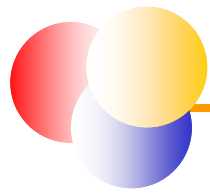
- **Perspective:**

- Examine the effectiveness of using various CASE tools **from the point of view** of the developers and testers

- **Environment and constraints:**

- Payroll applications programming in C++
- 100 software developers with 5 or more years experience in C++
- Customers are businesses
- Do not maintain a reusable module database
- Examine new projects completed and sold from 1/1/1998 to 31/12/2002

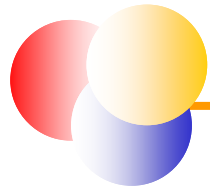




GQM, Entities and Attributes

- We can relate the GQM template to the Entity&Attribute framework.
- A Goal or question can be associated with at least one pair of Entities and Attributes.
- Once a measure is obtained, it can be related back to questions and goals.





Example: AT&T's GQM

- Goal: Better code inspection
- Subgoals:
 - Better inspection **planning**
 - Better **monitor and control** of the code
 - **Improving** code inspection

AT&T. Manage code inspection information. IEEE Software, 1994.





AT&T's GQM /1

GA: Code inspection: Plan

- **Q1:** How much does the inspection process cost?
 - **M1.1:** Average effort per KLOC
 - **M1.2:** Percentage of re-inspection

- **Q2:** How much calendar time does the inspection process take?
 - **M2.1:** Average effort per KLOC
 - **M2.2:** Total KLOC inspected





AT&T's GQM /2

GB: Code inspection: Monitor & control

- **Q1: What is the quality of the inspected software?**
 - **M1.1:** Average fault detected per KLOC
 - **M1.2:** Average inspection rate
 - **M1.3:** Average preparation rate
- **Q2: What degree did the staff conform to procedure?**
 - **M2.1:** Average inspection rate
 - **M2.2:** Average preparation rate
 - **M2.3:** Average lines of code inspected
 - **M2.4:** Percentage of re-inspection
- **Q3: What is the status of the inspection process?**
 - **M3.1:** Total KLOC inspected



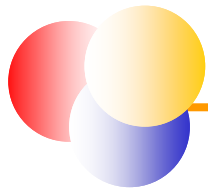


AT&T's GQM /3

GC: Code inspection: Improve

- **Q1: How effective is the inspection process?**
 - **M1.1: Defect removal efficiency**
 - **M1.2: Average fault detected per KLOC**
 - **M1.3: Average inspection rate**
 - **M1.4: Average preparation rate**
 - **M1.5: Average lines of code inspected**
- **Q2: What is the productivity of the inspection process?**
 - **M2.1: Average effort per fault detected**
 - **M2.2: Average inspection rate**
 - **M2.3: Average preparation rate**
 - **M2.4: Average lines of code inspected**





AT&T's Nine Metrics (1)

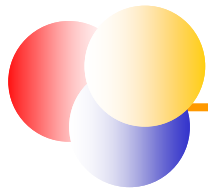
Total KLOC inspected $\Rightarrow \frac{\sum_{i=1}^N LOC\ inspect_i}{1000}$

Average LOC inspected $\Rightarrow \frac{Total\ KLOC\ inspected \times 1000}{N}$

Average preparation rate $\Rightarrow \frac{Total\ KLOC\ inspected \times 1000}{\sum_{i=1}^N \frac{preparation\ time_i}{Number\ of\ inspectors_i}}$

Average inspection rate $\Rightarrow \frac{Total\ KLOC\ inspected \times 1000}{\sum_{i=1}^N Inspection\ duration_i}$





AT&T's Nine Metrics (2)

Average effort per KLOC



$$\frac{\sum_{i=1}^N \text{Inspection effort}_i}{\text{Total KLOC inspected}}$$

Inspection effort_i = Preparation time_i + (Number of participants_i × Inspection duration_i) + Rework time_i

Average effort per fault detected



$$\frac{\sum_{i=1}^N \text{Inspection effort}_i}{\sum_{i=1}^N \text{Total fault found}_i}$$

Average faults detected per KLOC



$$\frac{\sum_{i=1}^N \text{Total faults detected}_i}{\text{Total KLOC inspected}}$$

Percentage of reinspections



$$\frac{\text{Number of reinspection dispositions}}{\text{Number of inspections}} \times 100$$

Defect removal efficiency



$$\frac{\sum_{i=1}^N \text{Total faults detected}_i}{\text{Total Coding faults detected}} \times 100$$

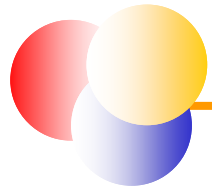
AT&T's Nine Metrics

TABLE 2
CODE-INSPECTION STATISTICS FOR FIRST SAMPLE PROJECT

Number of inspections in sample	27
Total KLOC inspected	9.3
Average LOC inspected (module size)	343
Average preparation rate (LOC/hour)	194
Average inspection rate (LOC/hour)	172
Total faults detected (observable and nonobservable) per KLOC	106
Percentage of reinspections	11

TABLE 3
CODE-INSPECTION STATISTICS FOR SECOND SAMPLE PROJECT

Number of inspections in sample	55
Total KLOC inspected	22.5
Average LOC inspected (module size)	409
Average preparation rate (LOC/hour)	121.9
Average inspection rate (LOC/hour)	154.8
Total faults detected (observable and nonobservable) per KLOC	89.7
Average effort per fault detected (hours/fault)	0.5

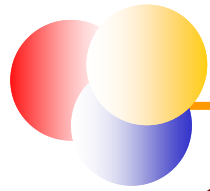


Example: HP's GQM

Three goals

- **A:** Maximize customer satisfaction
- **B:** Minimize engineering effort and schedule
- **C:** Minimize defects



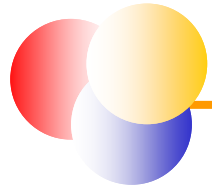


Example: HP's GQM-A /1

GA: Maximize customer satisfaction

- **QA1:** What are the attributes of customer satisfaction?
 - **MA1:** Functionality, usability, reliability, performance, supportability.
- **QA2:** What are the key indicators of customer satisfaction?
 - **MA2:** Survey, quality function deployment (QFD).
- **QA3:** What aspects result in customer satisfaction?
 - **MA3:** Survey, QFD.
- **QA4:** How satisfied are customers?
 - **MA4:** Survey, interview record, number of customers severely affected by defects.
- **QA5:** How many customers are affected by a problem?
 - **MA5:** Number of duplicate defects by severity



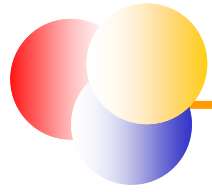


Example: HP's GQM-A /2

GA: Maximize customer satisfaction

- **QA6:** How many problems are affecting the customer?
 - **MA6-1:** Incoming defect rate
 - **MA6-2:** Open critical and serious defects
 - **MA6-3:** Defect report/fix ratio
 - **MA6-4:** Post-release defect density
- **QA7:** How long does it take to fix a problem?
 - **MA7-1:** Mean time to acknowledge problem
 - **MA7-2:** Mean time to deliver solution
 - **MA7-3:** Scheduled vs. actual delivery
 - **MA7-4:** Customer expectation of time to fix



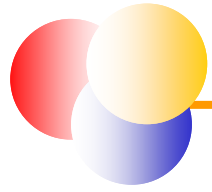


Example: HP's GQM-A /2

GA: Maximize customer satisfaction

- **QA8:** How does installing a fix affect the customer?
 - **MA8-1:** Time customers operation is down
 - **MA8-2:** Customers effort required during installation
- **QA9:** Where are the bottlenecks?
 - **MA9:** Backlog status, time spent doing different activities



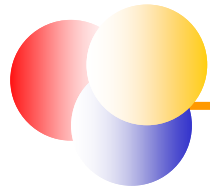


Example: HP's GQM-B /1

GB: Minimize engineering effort & schedule

- **QB1:** Where are the worst rework loops in the process?
 - **MB1:** Person-months by product-component-activity.
- **QB2:** What are the total life-cycle maintenance and support costs for the product?
 - **MB2.1:** Person-months by product-component-activity.
 - **MB2.2:** Person-months by corrective, adaptive, perfective maintenance.
- **QB3:** What development methods affect maintenance costs?
 - **MB3:** Pre-release records of methods and post-release costs.



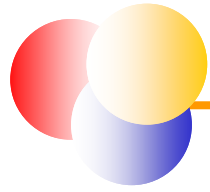


Example: HP's GQM-B /2

GB: Minimize engineering effort & schedule

- **QB4:** How maintainable is the product as changes occur?
 - **MB4.1:** Incoming problem rate
 - **MB4.2:** detect density
 - **MB4.3:** Code stability
 - **MB4.4:** Complexity
 - **MB4.5:** Number of modules changed to fix one detect
- **QB5:** What will process monitoring cost and where are the costs distributed?
 - **MB5:** Person-months and costs
- **QB6:** What will maintenance requirements be?
 - **MB6.1:** Code stability, complexity, size
 - **MB6.2:** Pre-release defect density



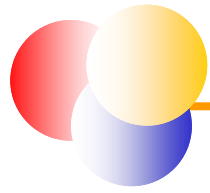


Example: HP's GQM-B /3

GB: Minimize engineering effort & schedule

- **QB7:** How can we predict cycle time, reliability, and effort?
 - **MB7.1:** Calendar time
 - **MB7.2:** Person-month
 - **MB7.3:** Defect density
 - **MB7.4:** Number of detects to fix
 - **MB7.5:** Defect report/fix ratio
 - **MB7.6:** Code stability
 - **MB7.7:** Complexity
 - **MB7.8:** Number of lines to change



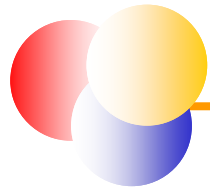


Example: HP's GQM-B /4

GB: Minimize engineering effort & schedule

- **QB8:** What practices yield best results?
 - **MB8:** Correlations between pre-release practices and customer satisfaction data
- **QB9:** How much do the maintenance phase activities cost?
 - **MB9:** Personal-months and cost
- **QB10:** What are major cost components?
 - **MB10:** Person-months by product-component-activity
- **QB11:** How do costs change over time?
 - **MB11:** Track cost components over entity maintenance life-cycle.



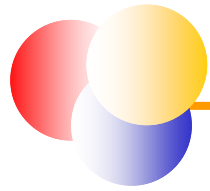


Example: HP's GQM-C /1

GC: Minimize defects

- **QC1:** What are key Indicators of process health and how are we doing?
 - **MC1:** Release schedule met, trends of defect density, serious and critical detects.
- **QC2:** What are high-leverage opportunities for preventive maintenance?
 - **MC2.1:** Defect categorization
 - **MC2.2:** Code stability
- **QC3:** Are fixes effective with less side effects?
 - **MC3:** Defect report/fix ratio



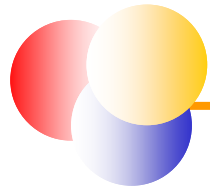


Example: HP's GQM-C /2

GC: Minimize defects

- **QC4:** What is the post-release quality of each module?
 - **MC4:** Defect density, critical and serious detects.
- **QC5:** What are we doing right?
 - **MC5.1:** Defect removal efficiency
 - **MC5.2:** Defect report/fix ratio
- **QC6:** How do we know when to release?
 - **MC6.1:** Predicted defect detection and remaining defects.
 - **MC6.2:** Test coverage.





Example: HP's GQM-C /3

GC: Minimize defects

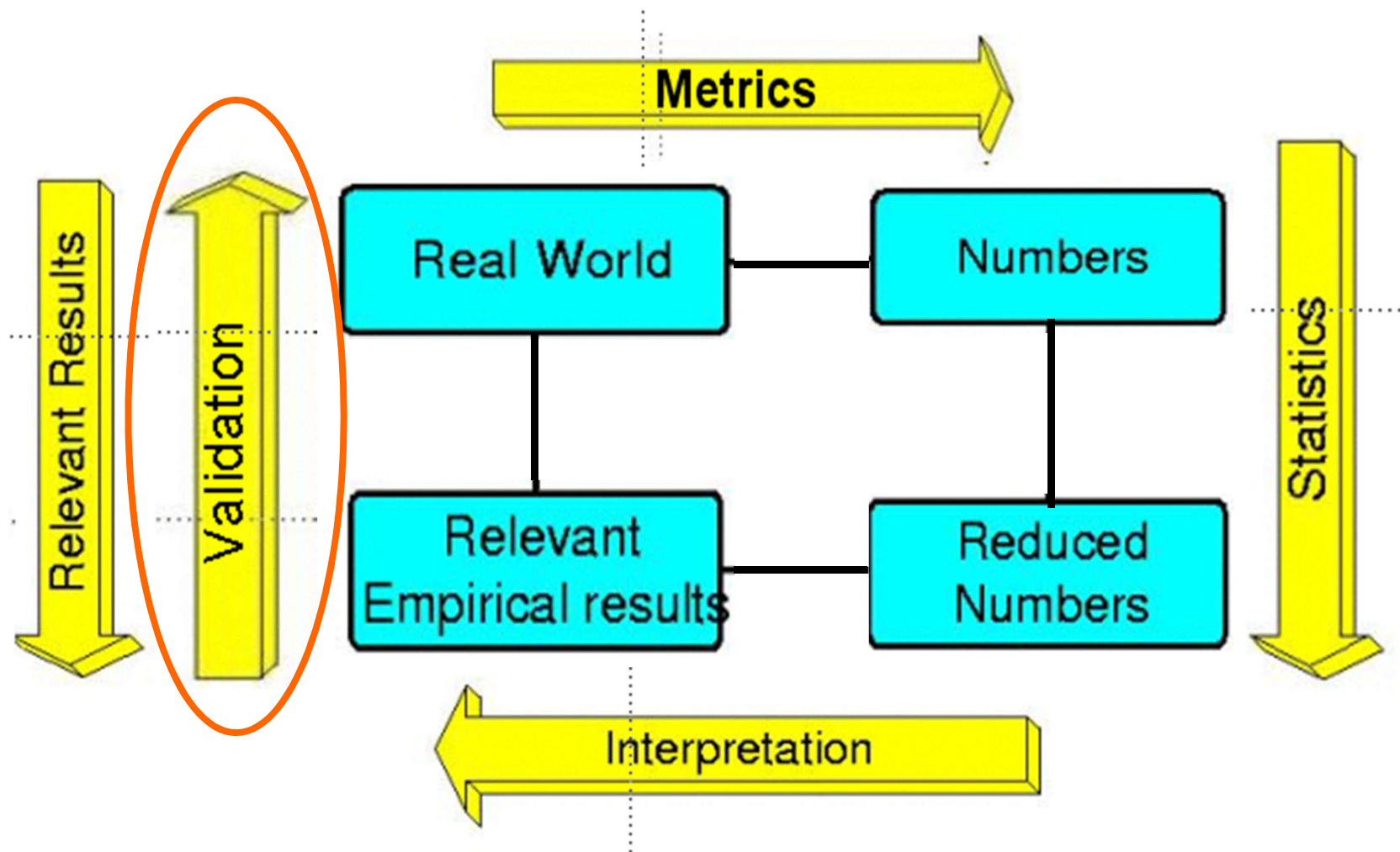
- **QC7:** How effective is the development process in preventing defects?
 - **MC7:** Post-release detect density
- **QC8:** What can we predict will happen post-release based on pre-release data?
 - **MC8:** Corrections between pre-release complexity, defect density, stability, and customer survey data.
- **QC9:** What defects are getting through and there causes?
 - **MC9:** Detect categorization

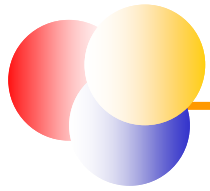


Section 3

Measurement Validation



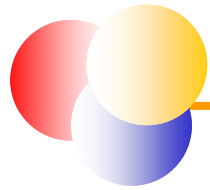




Review

- **Measurement systems** are used to assess existing entities by numerically characterizing one or more of its attributes
- **Prediction systems** are used to predict some attributes of a future entity, involving a mathematical model with associated prediction procedures
 - **Deterministic prediction system:** The same output will be generated for a given input
 - **Stochastic prediction system:** The output for a given input will vary probabilistically



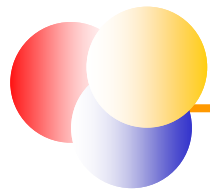


Measurement Validation

Does the metric used capture the information that it was intended for?

- **A metric is valid** if it accurately characterizes the attribute it claims to metric
- **A prediction system is valid** if it makes accurate predictions

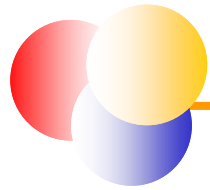




Validating Metrics

- **Definition:** The process of ensuring that the metric is a proper numerical characterization of the claimed attribute by showing that the representation condition is satisfied
- **Example: Measuring program length**
 - Any metric of length should satisfy conditions such as:
 - Length of joint programs should be
$$m(p1 ; p2) = m(p1) + m(p2)$$
 - If length of p1 is greater than p2, any metric of length should satisfy $m(p1) > m(p2)$

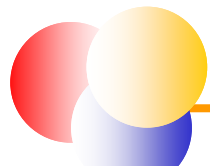




Validating Prediction Systems

- **Definition:** The process of establishing the accuracy of the prediction system by empirical means, i.e., by comparing model performance with known data in the given environment

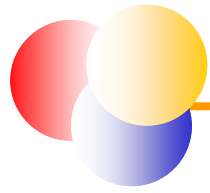




Classes of Prediction Systems

- **Class 1:** Using **internal attribute** metrics of early life-cycle **products** to predict metrics of **internal attributes** of later life-cycle **products**
 - **Example:** metrics of **size, modularity, and reuse of a specification** are used to predict **size of the final code**
- **Class 2:** Using early life-cycle **process attribute** metrics and **resource-attribute** metrics to predict metrics of attributes of later life-cycle **processes and resources**
 - **Example:** **the number of faults** found during **formal design review** is used to predict the **cost of implementation**

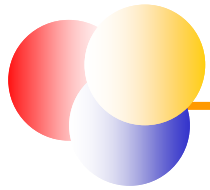




Classes of Prediction Systems

- **Class 3:** Using internal **product-attribute** metrics to predict **process attributes**
 - **Example:** metrics of **structuredness** are used to predict **time to perform some maintenance task**, or **number of faults found during unit testing**
- **Class 4:** Using **process metrics** to predict later **product metrics**
 - **Example:** External product attribute (**reliability**) is effectively defined in terms of process attributes (**operational failures**)





Validating Metrics: Wide Sense

- A metric is valid in wide sense if :
 - It is internally valid, i.e., can be specified entirely in terms of the process, product or resource itself separate from its behaviour
 - It is a component of a valid prediction system





Summary

- **Software metric classification?**
- **Goal-based framework?**





Further reading

1. R. E. Park, W. B. Goethert, W. A. Florac. [Goal-Driven Software Measurement-A Guidebook](#). CMU/SEI-96-HB-002, (189 pages)
2. J. Barnard, A. Price. Managing Code Inspection Information. IEEE Software, 1994. (11 pages)



**Thanks for your time and
attention!**



怎样过滤掉空值的函数 (通常是没有源代码的库函数)?

函数名	相对路径	Cyclomatic	CountLine	CountPath	MaxNest	irKnots	CountInput	CountOutput
atoi								
atof								
strtod		27	97	9408	3	37	6	7
malloc		1	6	1	0	0	41	2
realloc		1	7	1	0	0	16	2
free		1	6	1	0	0	285	1
abort								
exit								
getenv								
qsort								



怎样过滤掉空值的函数 (通常是没有源代码的库函数)?

方法一

```
$db->ents("function ~unknown ~unresolved")
```

方法二

```
next if ($func->library() =~ m/Standard/i);
```

方法三

```
$val = $func->metric("Cyclomatic");  
if (defined($val)) {...}
```



练习：用Perl收集缺陷数据

度量数据

A	B	C	D	E	F	G	H	I
funcName	funcPath	SLOC	FANIN	FANOUT	NPATH	CyclomatiKnots		bugs
update_line	bash-3.2/lib/readline/display.c	292	27	26	1E+09	61	32	5
rl_redisplay	bash-3.2/lib/readline/display.c	429	43	48	1E+09	122	34	3
parse_and_execute	bash-3.2/builtins/evalstring.c	151	42	30	57856	27	16	2
execute_command_internal	bash-3.2/execute_cmd.c	250	46	75	11534449	66	110	2
execute_cond_node	bash-3.2/execute_cmd.c	79	11	10	156	21	6	2
_rl_move_cursor_relative	bash-3.2/lib/readline/display.c	57	16	7	99	14	6	2


缺陷数据

A	B	C	D	E	F	G	H	I
funcName	funcPath	SLOC	FANIN	FANOUT	NPATH	CyclomatiKnots		bugs
update_line	bash-3.2/lib/readline/display.c	292	27	26	1E+09	61	32	5
rl_redisplay	bash-3.2/lib/readline/display.c	429	43	48	1E+09	122	34	3
parse_and_execute	bash-3.2/builtins/evalstring.c	151	42	30	57856	27	16	2
execute_command_internal	bash-3.2/execute_cmd.c	250	46	75	11534449	66	110	2
execute_cond_node	bash-3.2/execute_cmd.c	79	11	10	156	21	6	2
_rl_move_cursor_relative	bash-3.2/lib/readline/display.c	57	16	7	99	14	6	2

练习：用Perl收集缺陷数据

(1) 下载Bash 3.2的补丁（51个patch）

<http://ftp.gnu.org/gnu/bash/bash-3.2-patches/>



文件(F) 编辑(E) 查看(V) 历史(S) 书签(B) 工具(T) 帮助(H)

Index of /gnu/bash/bash-3.2-patches/

← ftp.gnu.org/gnu/bash/bash-3.2-patches/

火狐官方网站 火狐主页 最常访问 火狐官方网站 新手

Index of /gnu/bash/bash

	Name	Last modified	Size	Description
📁	Parent Directory	-		
📄	bash32-001	2006-10-17 11:56	1.5K	
📄	bash32-001.sig	2006-10-17 11:56	65	
📄	bash32-002	2006-10-30 17:41	1.5K	
📄	bash32-002.sig	2006-10-30 17:41	65	
📄	bash32-003	2006-10-30 17:42	4.5K	
📄	bash32-003.sig	2006-10-30 17:42	65	
📄	bash32-004	2006-11-09 10:04	2.5K	
📄	bash32-004.sig	2006-11-09 10:04	65	
📄	bash32-005	2006-11-09 10:04	5.8K	
📄	bash32-005.sig	2006-11-09 10:04	65	
📄	bash32-006	2006-12-12 16:38	1.3K	
📄	bash32-006.sig	2006-12-12 16:38	65	

练习：用Perl收集缺陷数据

每个patch的格式

BASH PATCH REPORT

名称

=====

```
1
2
3
4 Bash-Release: 3.2
5 Patch-ID: bash32-004
6
7 Bug-Reported-by: Stuart Shelton <srcshelton@gmail.com>
8 Bug-Reference-ID: <619141e40610261203y6cda5aa6i23cb24c7aeba996e@mail.gm
9 Bug-Reference-URL:
10
11 Bug-Description:
12
13 A bug in the parameter pattern substitution implementation treated a pa
14 whose first character was `/' (after expansion) as specifying global
15 replacement.
16
17 Patch:
18
19 *** ../bash-3.2/subst.c Tue Sep 19 08:35:09 2006
20 --- subst.c Thu Oct 26 09:17:50 2006
21 *****
22 *** 5707,5712 ****
23 --- 5707,5717 ----
```

练习：用Perl收集缺陷数据

每个patch的格式

```
1      BASH PATCH REPORT
2      =====
3
4  Bash-Release: 3.2
5  Patch-ID: bash32-004
6
7  Bug-Reported-by:  Stuart Shelton <srcshelton@gmail.com>
8  Bug-Reference-ID: <619141e40610261203y6cda5aa6i23cb24c7aeba996e@mail.gm
9  Bug-Reference-URL:
10
11 Bug-Description:
12
13 A bug in the parameter pattern substitution implementation treated a pa
14 whose first character was `/' (after expansion) as specifying global
15 replacement.
16
17 Patch:
18
19 *** ../bash-3.2/subst.c Tue Sep 19 08:35:09 2006
20 --- subst.c Thu Oct 26 09:17:50 2006
21 *****
22 *** 5707,5712 ****
23 --- 5707,5717 ----
```

被打补丁程序的版本号以及补丁的编号

练习：用Perl收集缺陷数据

每个patch的格式

```
1      BASH PATCH REPORT
2      =====
3
4  Bash-Release: 3.2
5  Patch-ID: bash32-004
6
7  Bug-Reported-by: Stuart Shelton <srcshelton@gmail.com>
8  Bug-Reference-ID: <619141e40610261203y6cda5aa6i23cb24c7aeba996e@mail.gm
9  Bug-Reference-URL:
10
11 Bug-Description:
12
13 A bug in the parameter pattern substitution implementation treated a pa
14 whose first character was `/' (after expansion) as specifying global
15 replacement.
16
17 Patch:
18
19 *** ../bash-3.2/subst.c Tue Sep 19 08:35:09 2006
20 --- subst.c Thu Oct 26 09:17:50 2006
21 *****
22 *** 5707,5712 ****
23 --- 5707,5717 ----
```

Bug报告者、确认者以及确认URL

练习：用Perl收集缺陷数据

每个patch的格式

```
1      BASH PATCH REPORT
2      =====
3
4  Bash-Release: 3.2
5  Patch-ID: bash32-004
6
7  Bug-Reported-by: Stuart Shelton <srcshelton@gmail.com>
8  Bug-Reference-ID: <619141e40610261203y6cda5aa6i23cb24c7aeba996e@mail.gm
9  Bug-Reference-URL:
10
11 Bug-Description:
12
13 A bug in the parameter pattern substitution implementation treated a pa
14 whose first character was `/' (after expansion) as specifying global
15 replacement.
16
17 Patch:
18
19 *** ../bash-3.2/subst.c Tue Sep 19 08:35:09 2006
20 --- subst.c Thu Oct 26 09:17:50 2006
21 *****
22 *** 5707,5712 ****
23 --- 5707,5717 ----
```

Bug描述

练习：用Perl收集缺陷数据

每个patch的格式

```
1      BASH PATCH REPORT
2      =====
3
4  Bash-Release: 3.2
5  Patch-ID: bash32-004
6
7  Bug-Reported-by: Stuart Shelton <srcshelton@gmail.com>
8  Bug-Reference-ID: <619141e40610261203y6cda5aa6i23cb24c7aeba996e@mail.gm
9  Bug-Reference-URL:
10
11 Bug-Description:
12
13 A bug in the parameter pattern substitution implementation treated a pa
14 whose first character was `/' (after expansion) as specifying global
15 replacement.
16
17 Patch:
18
19 *** ../bash-3.2/subst.c Tue Sep 19 08:35:09 2006
20 --- subst.c Thu Oct 26 09:17:50 2006
21 ****
22 *** 5707,5712 ****
23 --- 5707,5717 ----
```

补丁代码

练习：用Perl收集缺陷数据

Patch举例

bash32-004

怎样读懂patch信息

http://www.ruanyifeng.com/blog/2012/08/how_to_read_diff.html



练习：用Perl收集缺陷数据

(2) 编写一个Perl脚本，分析每个patch文件，统计每一个函数上的缺陷数目

规则：

- a) 每个函数的初始缺陷数目为0
- b) 分析每个patch，如果一个函数的代码在这个patch中被修改了，则它的缺陷数目增加1

在10月30日前提交：Perl脚本以及数据集

