STRENGTHS AND WEAKNESSES OF THE GQM APPROACH
IN DEVELOPING SOFTWARE SIZE METRICS

Dr. Peter Hantos
Xerox Corporation

701 South Aviation Boulevard, MS: ESAE-375
El Segundo CA 90245
Phone: (310) 333 - 9038
Internet: peter.hantos@usa.xerox.com

ABSTRACT
GQM (Goal-Question-Metric) is one of the most powerful approaches for metrics definition. Beside being elegant and simple, numerous examples are also available in the literature to demonstrate the successful application in large companies like Hewlett-Packard and Motorola. In this paper we apply the GQM concept to the size metrics used as main input to one of the most popular software cost estimation tool, COCOMO, and present a foundation for defining an appropriate O-O size metric. Applying the correctness notion to Metrics, we determine that GQM is necessary but not sufficient for defining a size metric for COCOMO. To alleviate the identified problems, we propose an extension to GQM, called GQM-Rx.

Next, we present an analysis of some potential O-O metrics, and explore the usability of a Weighted Class Metric for size estimation. We conclude, that class-based metric is not necessary to develop size metric for O-O schedule and effort estimation. Building on this conclusion, we present guidelines that we feel necessary and helpful for the definition of an appropriate O-O size metrics for COCOMO.

KEYWORDS:
GQM, Software Metrics, O-O Metrics, COCOMO

BIOGRAPHY
Dr. Peter Hantos is Principal Scientist at the Xerox Corporate Engineering Center. In this capacity his tasks include the coaching and mentoring of software development teams across Xerox on software technology related issues, identifying and sharing best practices, and the internalization and institutionalization of a systematic software risk management approach for Xerox. His current, main accomplishment was the definition of the Xerox software development process standard, called the Time To Market Software Subprocess. He is also the Xerox representative in the USC/CSE COCOMO II Industrial Affiliates Program. In his previous position, he managed a software engineering organization providing methodology, tools, software metrics and infrastructure support for members of the Xerox Corporate Research and Technology Division located in El Segundo, California. Dr. Hantos is a Senior Member of the IEEE and Member, ACM.
Introduction

- On Structured Programming:
  - "GOTOs can be hazardous to your health."
    E. Dijkstra
- On Lines of Code and Function Points:
  - "SLOC is dead. Long live the FP."
    Paraphrased for Capers Jones
- On Function Points:
  - "Backfire can backfire!"
    A. Lubatovsky
- On the Goal-Question-Metrics approach:
  - "Over-the-counter GQM can be hazardous to your health."
    P. Hantos
Objectives

✓ Ultimate goal:
  - O-O Size Metric for COCOMO
✓ Presentation Objectives:
  - Explore the use of GQM
  - Suggest extensions to GQM
  - Use the extended GQM to define O-O Size Metric
  - Propose an approach to O-O Size Metric

Agenda

- Introduction
- Objectives
- GQM
- A Little Metrics Theory
- The GQM-Ř Approach
- O-O: What could we measure?
- Class and SLOC Life-Cycles
- Exploring Weighted Classes
- Iterative Project Life-Cycle and Activities
- Conclusions
- References
Goal Based Measurement*

- Each metric supports multiple goals
- Questions focus on metric selection and in-process analysis

* Source: Vic Basili [1]

Goal/Question/Metric Approach*

- Develop your set of corporate, division, and project goals for productivity and quality. Translate these goals into measurement goals.
- Generate questions that quantifiably define your goals as completely as possible.
- Specify the metrics needed to be collected to answer your questions and track process and product conformance to the goals.
- Develop mechanisms for data collection.
- Collect, validate, and analyze the data in real-time to provide feedback to projects for corrective action.
- Analyze the data in a post-mortem fashion to assess conformance to the goals and make recommendations for future improvements.

* Source: Vic Basili
A Little Metrics Theory

Fenton [2] presents 4 questions:

1. How much do we need to know about some attribute before we can consider measuring it?
2. How do we know if we have really measured the attribute we wanted to measure?
3. Using measurement, what kind of statements can we make about an attribute and the entities which possess it which are meaningful?
4. What kind of operations can we perform on measures which are meaningful?

“Correctness” in Metrics*

Correctness in metrics is a relative notion: Consistency of Use vis-à-vis Specification.

Basic notation:
(M, P: assertions, U: use)

\{M\} \cup \{P\}

What this means (total correctness):
Any use of Metric M will result in a state satisfying P

* These ideas were inspired by B. Meyer's work [3]
"Correctness" in Metrics (cont.)

In our case M=M1, P=G1, G2, G3.

\{M1\} \cup \{G1, G2, G3\}

What this means:

- Any use of Metric M1 will satisfy G1, G2, and G3 goals.
- M1 seems to be a "Correct" Metric from the Metrics Designer's point of view.

Questions a Metrics Promoter would have to ask:
1. How do we know what the Goals were?
2. What about other Goals?

The GQM-R\textsubscript{X} Approach*

Simplified GQM view: \hspace{1cm} The GQM-R\textsubscript{X} view:

\begin{align*}
G & \downarrow \\
Q & \downarrow \\
M & \downarrow \\
\text{(P)} & \rightarrow M \leftarrow \text{R} \leftarrow \text{C}
\end{align*}

Similarly to a medication prescription, we need:

C: Contra-indications (For example ~G4, ~G5, ~G6)
R: Rules of use
P: A "nice to have" feature: A plausible description.

* Paraphrasing B. Meyer, it could be also called as "Metrics by Contract"
COCOMO-related examples

✓ G1 goal for size metrics: Applicable to COCOMO
✓ M1 metric: KLOC $\rightarrow M1$ is "correct"
✓ What about G2? Use M1 as a normalizer in defining metric M2, Defect Density (Example: Defects per KLOC.) $\rightarrow M1$ is "correct"
✓ What about G3? Use M1 as a normalizer for defining metric for people's productivity (Example: LOC/day/Peter.) $\rightarrow M1$ is "incorrect"

O-O: What could we measure?

- Number of classes in the particular application
- Number of classes in the total system
- Number of clients of this class
- Number of parents of this class
- Number of suppliers to a class
- Number of features of this class
- Number of exported features of this class
- Depth of inheritance hierarchy in this class

✓ Any of these seems to be appropriate for:
  - use as a size metric?
  - use as a size metric at schedule and effort estimation?
More requirements for “size”

✓ Metrics framework:
  - has to be generally applicable to O-O methods and
    process models
  - has to be applicable to any O-O language environment
  - has to evolve as understanding grows
  - has to include a transparent data-collection method

✓ Observation:
  “CLASS” seems to be the single central artifact or basic unit in O-O development.

Iterative Life-Cycle of a Class

1. Named
2. Specified or Designed
3. Major features and constraints are identified
4. Implemented
5. All features are coded
6. Class not yet completely tested
7. Verified/validated
8. Unit tested
9. Integration/System test includes this class
10. ("Eliminated" - optional, done in the class context)
11. Generalized
12. Specialized
SLOC Life-cycle

- Created
- Verified
- Compiled
- Validated
- Run-time
- (Eliminated - optional)

Remarks:
- This is basically a waterfall model
- It is the life-cycle of one single instruction

SLOC

✔ A Plausible Explanation

- Code is the most accepted natural building block or "brick" for software development
- It is (still...) viewed as the only "tangible" artifact
- On SLOC level the effort is considered uniform
- This notion is supported by the simple life-cycle
- All "creative", and consequently variable effort has to be considered and modeled above the SLOC level
Class-related Metrics

Details of using Class-related Metrics for Software Size (and Cost...) Estimation:
- "Class" is viewed as the natural building block of an O-O system
- Phases in the evolution of a class are reflecting on the completion status of the class only
- The state of the class is changing during the overall development life-cycle
- The notion above is supported by the iterative nature of the class life-cycle discussed earlier

Class-related Metrics (cont.)

More details of using Class-related Metrics for Software Size (and Cost...) Estimation:
- All class-related metrics will be most likely indirect metrics, meaning that they will involve the measurement of more than one attribute
- These measurements will always be implemented differently depending on the context in which they are required
- Before use, indirect measures always have to be validated within the context
- Completion status of classes does not capture any notion of effort

In view of these concerns, choosing class as the "brick" is not obvious...
Exploring Weighted Classes

† Determining the size of a Weighted Class:
- Number of features in the class
  - Simple count, available early
- "Size" of the class / feature interface
  - Available at the end of the Specification Phase
- SLOC per feature
  - Available after the Implementation Phase
  - But changes during Validation and Verification
  - Could dramatically change during Generalization

\[ \begin{align*}
G & \downarrow \\
Q & \downarrow \\
(P) & \rightarrow M \leftarrow R \leftrightarrow C \\
\text{Weighted Class Size Metric definition - } GQM-Rx \text{ analysis} \\
G: & \text{ Define O-O Size Metric} \\
Q: & \text{ What can be measured at all in O-O?} \\
M: & \text{ Follow steps outlined on the previous slide} \\
C: & \text{ It might be overly complex or inappropriate for COCOMO} \\
R: & \text{ Missing, no context is given (or even considered ...) for use} \\
P: & \text{ Overly complex as it is, but no plausible explanation can be given for using it in COCOMO for schedule or effort estimation.}
\end{align*} \]
Weighted Class Analysis

✓ Note the "Number of Features" and "SLOC per feature" steps:
  - Eventually we have to transition to SLOC anyway
  - This is how the difference between the life-cycle model of a "class" and a "SLOC" is resolved.

✓ The relevance of Interface Size for Classes:
  - Dependencies
  - Class coupling

Stepping back...

✓ Is O-O development really so different?
  - Class Diagram is like Work Breakdown Structure
  - Number of Classes is like Number of Functions or Subroutines in a program
  - etc.

✓ What is different:
  - Mind-set (or "paradigm")

✓ The key issue is:
  - Not the "O-O Paradigm", like the use of polymorphism, inheritance and such...
  - but the Iterative Development Life-cycle.
Iterative Life-Cycle

Effort data needs to be collected for the Iterations

Project Activity Distribution

✓ Determining Project Activity Distribution in the Iterations is not different from how it was determined for non O-O systems

✓ New Issues:
  - Determine the correlation between DSLOC and Number of Iterations in the Product Development Life-cycle Phases
Conclusions

✓ GQM-related Conclusions:
  - During the attempt of developing an O-O size metrics for COCOMO, the GQM approach was used
  - Applying the CORRECTNESS notion to Metrics, it was determined that GQM is necessary but not sufficient for defining such metrics
  - GQM-R Extension was proposed and used

Conclusions (Cont.)

✓ COCOMO-Related Conclusions:
  - Our analysis indicates that CLASS-based metrics is not necessary for sizing in the effort and schedule estimation context
  - It is necessary though to calibrate COCOMO for the new, iterative life-cycle model.
References

