Iterative Process to Improve GQM Models with Metrics Thresholds to Detect High-risk Files

Naohiko Tsuda, Masaki Takada, Hironori Washizaki,
and Yoshiaki Fukazawa
Dept. Computer Science and Engg.
Waseda Univ., 3-4-1 Ohkubo
Shinjuku-ku, Tokyo, 169-8555, Japan
Email: 821821@toki.waseda.jp, {washizaki, fukazawa}@waseda.jp

Shunsuke Sugimura, Yuichiro Yasuda,
and Masanao Futakami
Komatsu Ltd. Development Division
ICT Development Center
3-25-1, Shinomiya, Hiratsuka-shi
Kanagawa, 254-8555, Japan

***Abstract*—We propose an iterative process to improve GQM models with metrics thresholds to detect high-risk files.**

***Keywords*—Software Maintenance, Software Reusability, Software Measurement, Threshold, GQM.**

I. Proposal: Our Iterative Process

The presence of low maintainable (e.g., not understandable or changeable) files prevents reusing systems and future extensions. Although automatic detection of such high-risk files reduces the burden on inspectors, it is unclear how to define the optimal evaluation models for the detection. Fig. 1 overviews our iterative process to define and improve GQM models with metrics thresholds to detect high-risk files.

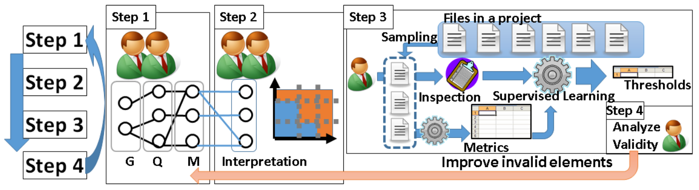

Fig. 1. An Overview of our iterative process

A. Step 1: Create a GQM Model

A team of developers, inspectors, etc. must create a Goal-Question-Metrics (GQM) model [1] to clarify what to measure. The most important aspect is that the team is comprised of experienced team members with respect to the target domain or project. First, set goals for the quality characteristic of the source code. Next, subdivide experts' tacit knowledge for code inspection into multiple viewpoints (questions (Q_i)). Finally, clarify the metrics related to each Q_i.

B. Step 2: Define Conditional Expressions (CEs)

The team must define how to interpret the metrics values corresponding to Q_i as CE_i, e.g. $(M_1 \leq T_1 || M_2 \leq T_2)$. Although thresholds-based CEs are easy to interpret, deciding the general and concrete thresholds is impractical because the context of the domain and project affects the distribution of the metrics [2]. In our process, Step 2 defines thresholds-based CEs without concrete values, while Step 3 determines concrete values for a specific project. Thus, our process provides cross-project reusable CEs and projects-specific valid thresholds.

C. Step 3: Optimize the Thresholds for CEs

1) Step 3.1: Obtain Labeled Training Data for Supervised Learning: Experts conduct a code inspection of sample files (training data) using a check-list (questions) defined by GQM. When the experts consider a file to be low (high) risk, they assign it an OK (NG) label to the file.

2) Step 3.2: Search the Optimal Thresholds with the Highest Degree of Expert-approximation: In this paper, the degree of expert-approximation means the degree of similarity between an automatic evaluation by a CE_i and the experts' inspection results for a Q_i. Metaheuristic algorithms (e.g., simulated annealing (SA)) can search for optimal thresholds. Then the problem becomes how to quantify the degree of expert-approximation. Here the F1 score or Cohen's kappa between training labels and predicted (automatically evaluated) labels indicates the degree.

D. Step 4: Analyze the Results in Step 3 for Improvement

A low degree of expert-approximation indicates that the CE and thresholds must be redefined. We classify the possible causes into four categories and describe how to improve them.

1) Inappropriate Training Data: When the ratio of OK to NG in the training labels of Q_i is biased, an expert-approximation will fail. Recollecting training data for Q_i should resolve this issue.

2) Mismatch of Experts' Viewpoints and Metrics: If experts and measurement tools handle different information, the definitions of the metrics and questions should be redefined.

3) Inappropriate GQM Model: An ambiguous question causes an inappropriate code inspection. Such questions should be subdivided into more specific questions. However, too many questions increase the cost of a check-list-based code inspection. Thus, similar questions should be integrated. After that, add or reduce the metrics as necessary.

References

[1] V.R. Basili, et al., "Goal Question Metric Approach," Encyclopedia of Software Engineering, John Wiley & Sons, Inc., , pp. 528-532, 1994.

[2] F. Zhang, et al., "How Does Context Affect the Distribution of Software Maintainability Metrics? ", ICSM'13, 2013.