Software Metris and Quality Engineering

CSE 8314 — Fall 2013

Prof. Jeff Tian, tian@lyle.smu.edu
CSE, SMU, Dallas, TX 75275
(214) 768-2861; Fax: (214) 768-3085
www.lyle.smu.edu/~tian/class/8314.13f

Module IV: Formal Models for Metrics Evaluation

- Formal Models/Axioms for Metrics Evaluation
- Tian-Zelkowitz Approach
- Application and Validation;

Overview: Measurement

To achieve the goal of controlled software development, we need to:

- Develop an *engineering* discipline;
- Measure and evaluate the working product;
- Construct a *scientific* model for program measurement:
  - ▷ Techniques from other disciplines;
  - ▷ Develop new techniques if necessary;
  - ▷ Basic questions:
    - – What to measure: goal & environ.
    - – How to measure it: metrics & tools
    - – Selection and validation

Overview: Our Solution Strategy

Need a *scientific* model of program complexity:

1. Develop a *theory* of program complexity to organize empirical knowledge;
2. Develop a *technique* for measure evaluation and selection to extrapolate measurement activities to new applications;
3. *Validate* the model using data from NASA Software Engineering Laboratory.

Comments: The theory is a systematic extension to earlier studies by Prather, Fenton and Whitty, and Weyuker.

Previous Work: Prather

- Prather's axioms:
  - ▷ $m(S_1; S_2; \ldots; S_n) \geq \sum_i m(S_i)$
  - ▷ $2(m(S_1) + m(S_2))$
    $\geq m(if\ P\ then\ S_1\ else\ S_2)$
    $> m(S_1) + m(S_2)$
  - ▷ $2m(S) \geq m(while\ P\ do\ S) > m(S)$

- Observations/discussions:
  - ▷ earliest attempt on axiomatic model
  - ▷ some intuition captured:
    – e.g., interactions
  - ▷ limited scope
  - ▷ justification for some axioms?

Previous Work: Fenton

- Fenton's hierarchical complexity:
  - ▷ $m(seq(F_1, \ldots, F_n))$
    $= g_n(m(F_1), \ldots, m(F_n))$
  - ▷ $m(F(F_1 \; on \; x_1, \ldots, F_n \; on \; x_n))$
    $= h_F(m(F), m(F_1), \ldots, m(F_n))$

- Observations/discussions:
  - ▷ general framework
    – too general?
  - ▷ contrast with Prather's work
  - ▷ relation to later work
    – add specifics
    – measurement theory based work

Previous Work: Weyuker

- Weyuker's Desirable Properties:

1. $(\exists P, Q)\ (\mathcal{V}(P) \neq \mathcal{V}(Q)\ )$
2. $\{P,\ \mathcal{V}(P) = c\ \}$ is finite
3. $(\exists P, Q)\ (P \neq Q) \wedge (\mathcal{V}(P) = \mathcal{V}(Q)\ )$
4. $(\exists P, Q)\ (\boxed{\text{P}} = \boxed{\text{Q}}) \wedge (\mathcal{V}(P) \neq \mathcal{V}(Q)\ )$
5. $(\forall P, Q)$
   $(\mathcal{V}(P) \leq \mathcal{V}(P;Q) \wedge \mathcal{V}(Q) \leq \mathcal{V}(P;Q)\ )$
6. $(\exists P, Q, R)$
   $(\mathcal{V}(P) = \mathcal{V}(Q) \wedge \mathcal{V}(P;R) \neq \mathcal{V}(Q;R)\ )$
7. $(\exists P, Q)\ (P = perm(Q) \wedge \mathcal{V}(P) \neq \mathcal{V}(Q)\ )$
8. $(\forall P)\ (\forall x, y)\ \mathcal{V}(P) = \mathcal{V}(P_y^x)$
9. $(\exists P, Q)\ (\mathcal{V}(P) + \mathcal{V}(Q) < \mathcal{V}(P;Q)\ )$

Previous Work: Weyuker

- About Weyuker's properties:
  - ▷ more systematic treatment
  - ▷ inspired/lead to many followup work
    - – positive: refinement
    - – negative: counter examples
    - – other: development & alternatives

- Tian/Zelkowitz as followup:
  - ▷ merit of Weyuker's properties
  - ▷ some universally satisfied
    - – basis for universal axioms
  - ▷ some for certain types of metrics
    - – classification?
  - ▷ a theory based on the above
  - ▷ an evaluation/selection process

Overview: Tian/Zelkowitz

- Tian/Zelkowith Theory/Framework

- **Axioms:** Define program complexity and state common properties.

- **Dimensionality Analysis:** provide the basis for metrics classification

  - ▷ Aspects or dimensions: presentation, control, data;

  - ▷ Levels: lexical, syntactic, semantic.

- **Classification Scheme:** Define mutually exclusive and collectively exhaustive classes.

Theory: Axiom Overview

Complexity: Relationship between program-pairs;

Comparability: Programs with identical functionality are comparable (**A1**);
Composite programs are comparable to their components (**A2**).

Monotonicity: Sufficiently large programs will become more complex (**A3**).

Measurability: Measures on programs must agree with underlying complexity (**A4**).

Diversity: Distribution of measured complexity must not form a single cluster (**A5**).

Theory: Defining Complexity

Definition: A *complexity ranking* $\mathcal{R}$ is a binary relation on programs. Given programs P and Q, we interpret $\mathcal{R}(P,Q)$ as P being no more complex than Q.

$\mathcal{C}(P,Q)$ iff $\mathcal{R}(P,Q) \vee \mathcal{R}(Q,P)$.

Comments:

- ▷ It is *internal* to the programs;
- ▷ Related empirical to external properties;
- ▷ Very broad definition, need further qualification and quantification.

Theory: Comparability Axioms

Axiom A1: $(\forall P, Q)$ ($\boxed{\text{P}} = \boxed{\text{Q}} \Rightarrow \mathcal{C}(P, Q)$)
i.e., functionally equivalent programs are comparable.

Axiom A2: $(\forall P, Q)$ $(IN(P, Q) \Rightarrow \mathcal{C}(P, Q)$)
i.e., a composite program is comparable with its components.

Comments:

- ▷ Hard problem due to undecidability;
- ▷ $\mathcal{R}$ is self-reflexive;
- ▷ $\mathcal{R}$ is not transitive.

Theory: Monotonicity Axiom

- **Axiom A3:** $(\exists K \in \mathcal{N})(\forall P, Q)$
  $(\ (IN(P,Q) \wedge (dist(P,Q) > K)) \Rightarrow R(P,Q)\ )$
  i.e., sufficiently large programs will not be ranked lower in complexity.

- **Comments:**
  - ▷ General trend must be followed;
  - ▷ Local deviations allowed.

Theory: Measure Definition Axiom

Definition: A *complexity measure* $\mathcal{V}$ is a quantification of complexity ranking $\mathcal{R}$. $\mathcal{V}$ maps programs into real numbers:

$$\mathcal{V} : \mathbf{U} \Rightarrow \Re$$

Axiom A4: $(\forall P, Q)\ (\mathcal{R}(P,Q) \Rightarrow \mathcal{V}(P) \leq \mathcal{V}(Q)\ )$ i.e., a measure must agree with the ranking it is approximating.

Non-Axiom: Commonly assumed by other complexity models:

$$(\forall P, Q)\ (\mathcal{V}(P) \leq \mathcal{V}(Q) \Rightarrow \mathcal{R}(P,Q))$$

- ▷ Incomparable cases;
- ▷ Non-transitive cases;

Theory: Distribution Axiom

Requirement: Measure values must not cluster around one single dominating point.

Axiom A5: $(\forall k \in \Re)(\exists \delta > 0)$

$$(|\mathbf{U} - \{P : \mathcal{V}(P) \in [k - \delta, k + \delta]\}| = |\mathbf{U}|)$$

Axiom A5': $(\forall k \in \Re)(\exists \delta > 0)$

$$\sum_{P, \mathcal{V}(P) \in [k-\delta, k+\delta]} prob(P) \quad < \quad 1$$

Rationale: A single dominating cluster is disallowed because it fails to achieve the goal of providing comparison for programs.

Theory: Complexity Dimensions

Presentation: Physical presentation for readers that has no effect on functionality.

Control: Instructions, control structures, and control dependencies.

Data: Data items, data structures, and data dependencies.

Comments:

- ▷ Control + Data = Abstract;
- ▷ Orthogonal dimensions.

Theory: Measurement Levels

Lexical: Token based measure computation;

Syntactic: Directly syntax based measure computation;

Semantic: Semantic analysis needed for measure computation.

Comments:

- 27 possible points in a 3-D space;
- Space proximity $\approx$ Measure similarity;
- Dividing control and data dimensions: 1. count, 2. structure, 3. dependency.

Theory: Vertical Classification

Classification based on computational models used:

- ▷ Depend only on syntax trees of programs?
  Yes, Abstract. No, non-abstract.
- ▷ Invariant to renaming?
  Yes, Functional. No, non-functional.

$$\text{All} \begin{cases} \text{Abstract} \begin{cases} \text{Functional} \\ \\ \text{Non Functional} \end{cases} \\ \\ \text{Non Abstract} \end{cases}$$

Theory: Vertical Classification Example

- All
  - Abstract $\left\{ \begin{array}{c} scan, stmt, ss, fp, \\ cyc, knot, du, hac, ac \end{array} \right\}$
    - Functional $\left\{ \begin{array}{c} stmt, ss, fp, cyc, \\ knot, du, hac, ac \end{array} \right\}$
    - Non Functional $\{scan\}$
  - Non Abstract $\{lc\}$

Theory: Hierarchical Classification

Classification based on complexity relations of component-composite programs:

- ▷ Sensitive to context?
  Yes, interactional. No, context free.
- ▷ Depend only on building element but not organization?
  Yes, Primitive. No, non-primitive.
- ▷ Capture both interface and internal?
  Yes, Overall. No, non-overall.

$$\text{All}\begin{cases}\text{Context Free}\begin{cases}\text{Primitive}\\ \text{Non Primitive}\end{cases}\\ \text{Interactional}\begin{cases}\text{Overall}\\ \text{Non Overall}\end{cases}\end{cases}$$

Theory: Hier. Classification Example

- All
  - Context Free $\left\{ \begin{array}{c} scan, stmt, \\ ss, cyc, knot \end{array} \right\}$
    - Primitive $\left\{ \begin{array}{c} scan, stmt, \\ ss, cyc, knot \end{array} \right\}$
    - Non Primitive $\{\}$
  - Interactional $\{fp, du, hac, ar\}$
    - Overall $\{du, hac, ar\}$
    - Non Overall $\{fp\}$

Evaluation: Problems and Solutions

Problem: Evaluation of complexity measures;

Assumption: Measures satisfy Axiom A4;

View: Measures as points in a measure space;

Solution Strategy:

- ▷ Define feasible region by using axioms and classification as boundary conditions;
- ▷ Derive scales for measures within the feasible region;
- ▷ Aggregate evaluations and select the optimal measure.

Evaluation: Boundary Conditions

Axioms as testable predicates:

BC$_1$. Axiom A1: ($\mathcal{D}(\mathcal{V})$— domain of $\mathcal{V}$)

$$(\forall P, Q)(\ \boxed{\text{P}} = \boxed{\text{Q}} \wedge P \in \mathcal{D}(\mathcal{V})\ ) \Rightarrow Q \in \mathcal{D}(\mathcal{V})$$

BC$_2$. Axiom A2:

$$(\forall P, Q)(\ (\text{IN}(P, Q) \vee \text{IN}(Q, P)) \wedge P \in \mathcal{D}(\mathcal{V})\ )$$

BC$_3$. Modified Axiom A3:

$$(\exists K)(\forall P, Q)\ (\ dist(P, Q) > K\ ) \Rightarrow \mathcal{V}(P) \leq \mathcal{V}(Q)$$

BC$_4$. Assumed true.

BC$_5$. Modified Axiom A5:

$$(\forall k \in \Re)(\exists \delta > 0)\ prob(\mathcal{V}(P) \in [k{-}\delta, k{+}\delta])\ <\ 1$$

Evaluation: Screening Using Axioms

Example Measure: $\mathcal{V}(P) = 1 - \frac{1}{s(P)}$, where $s(P)$ is the statement count of P.

Screening:

- ▷ $\mathbf{BC}_1$ is satisfied because $\mathcal{D}(\mathcal{V}) = \mathbf{U}$;
- ▷ $\mathbf{BC}_2$ same as above;
- ▷ $\mathbf{BC}_3$ is satisfied because:

$$(\forall P, Q) \quad (0 < s(P) < s(Q)) \;\Rightarrow\; \left(1 - \frac{1}{s(P)} \;<\; 1 - \frac{1}{s(Q)} \right)$$

- ▷ $\mathbf{BC}_5$ is not satisfied because:

$$prob(\mathcal{V}(P) \in [1 - \delta, 1 + \delta]) \quad = \quad 1$$

Result: Reject $\mathcal{V}$ due to violation of $\mathbf{BC}_5$.

Evaluation: Screening Using Classes

BC_6: Appropriate class depends on goals.

Goal 1. Documentation vs. comprehension.
Target: Non-abstract class.
Reject: Abstract class.

Goal 2. Object code size assessment.
Target: Abstract class.

- ▷ Total line count might be acceptable;
- ▷ Blank line count is rejected.

Goal 3. Programming effort prediction.
Target: Both abstract & non-abstract classes.
Reason: Both contribute to effort.

Evaluation: Monotonicity Scale

Assumption: Prefer measures that better approximates monotonicity;

Need to capture the *extent* and *frequency* of non-monotonic deviations;

Scale S$_1$: The monotonicity scale is $\langle T,\ p_m \rangle$, where T is the period of monotonicity:

$$T = \min_K(\ dist(P, Q) > K\ \Rightarrow\ \mathcal{V}(P) \leq \mathcal{V}(Q)\ )$$

and p_m is the conditional probability of non-monotonic component-composite pairs:

$$p_m = prob(\mathcal{V}(P) > \mathcal{V}(Q)\ |\ \text{IN}(P, Q))$$

Evaluation: Distribution Scale

Assumption: Uniform distribution desirable.

Need to capture:

- ▷ Significant points on the scale;
- ▷ Uniformity of these points.

Uniformity Scale S$_3$: For measure $\mathcal{V}$, $\epsilon > 0$, $\delta > 0$, and $p_k = prob(k\delta \leq \mathcal{V}(P) < (k+1)\delta)$, **S**$_3 = \langle n,\ d \rangle$, where n and d are the cardinality and the normalized $s.d.$ of $\{p_k \mid p_k > \epsilon\}$.

$$d = \begin{cases} 0 & \text{if } n = 0 \\ \sqrt{\frac{\sum_k (1-np_k)^2}{n}} & \text{otherwise} \end{cases}$$

Evaluation: Scale & Dominance Relation

Global Scaling Vector $\mathcal{G}$ is defined on relevant scales $\{\mathbf{S}_j\}$ with $\mathcal{G}(\mathcal{V})[i]$ defined successively as:

$$\mathcal{G}(\mathcal{V})[i] = \begin{cases} \mathbf{S}_j(\mathcal{V})[k] & \text{if opt} = \max \\ -\mathbf{S}_j(\mathcal{V})[k] & \text{if opt} = \min \end{cases}$$

until all individual scaling dimensions $\mathbf{S}_j(\mathcal{V})[k]$ are exhausted.

Dominance Relation: A measure $\mathcal{V}_i$ is said to dominate another measure $\mathcal{V}_j$ if

$$(\ \mathcal{G}(\mathcal{V}_i) \geq \mathcal{G}(\mathcal{V}_j)\ ) \wedge (\exists k)(\ \mathcal{G}(\mathcal{V}_i)[k] > \mathcal{G}(\mathcal{V}_j)[k]\ )$$

Elimination: All dominated measures are eliminated.

Evaluation: Objective Function

Assess the importance and trade-off among $\mathbf{S}_j$ to form weight vector $\mathcal{W}$.

$$(\forall i, j, k)\ \mathcal{W}(\mathcal{V}_i)[k] = \mathcal{W}(\mathcal{V}_j)[k] = \mathcal{W}[k]$$

Example: the weight for $\mathbf{S}_1$ could be $d \times p_c$.

The selection problem reduces to the constrained optimization problem:

$$\max_i \left(f_i = \sum_j \mathcal{G}(\mathcal{V}_i)[j] * \mathcal{W}[j]) \right)$$

such that:

$\mathcal{V}_i$ satisfies all boundary conditions.

Application and Model Validation

1. Application domain: risk identification for projects in NASA/SEL;

2. Pilot experiment: apply the scientific model to select complexity measures;

3. Data collection: run multiple applications and collect results;

4. Analyze resulting data-points to validate the scientific model.

Application: Risk Identification

Risk in software decisions:

- ▷ Multiple alternatives;
- ▷ Uncertainty about future development;
- ▷ Large investment;
- ▷ Significant consequences.

Risk Identification via CTA (Selby&Porter):

- ▷ Risk: likelihood of high cost or effort;
- ▷ High cost: highest quartile (80:20 rule);
- ▷ Basis: historical data;
- ▷ Methodology: classification trees.

Application: CTA Prediction Example

Predictions are made based on:

- ▷ Classification tree;
- ▷ Sample module measurement data:

	Modules				
	m_1	m_2	m_3	m_4	m_5
cyclomatic complexity	3	8	13	30	45
function plus module call	8	40	7	3	12
operators count	30	18	10	33	58
module calls	3	4	3	0	5
prediction	–	?	–	–	+
actual	–	–	+	–	+

Application: CTA Cost & Performance

Cost factors:

- ▷ Tree generation: measure pool size **S**;
- ▷ Tree usage: tree-complexity/node-count.

Performance Measures:

- ▷ Coverage: Predictions made;
- ▷ Accuracy: Correct predictions;
- ▷ Completeness: Correct predictions of actual high cost modules;
- ▷ Consistency: Correct high cost predictions.

Application: CTA Performance

Compare predicted and actual data:

$$\text{Coverage} = \frac{P}{P + N}$$

$$\text{Accuracy} = \frac{M_{11} + M_{22}}{P}$$

$$\text{Completeness} = \frac{M_{11}}{A_+}$$

$$\text{Consistency} = \frac{M_{11}}{P_+}$$

		Actual		
		+	−	+/−
Predicted	+	M_{11}	M_{12}	P_+
	−	M_{21}	M_{22}	P_-
	+/−	A_+	A_-	P
not identified				N

Pilot: Problem & Screening

Environment: NASA/SEL;

Goal: Identify high cost modules using complexity measures.
Consequence: Eliminate non-complexity measures, reducing **S** from 74 to 40 .

Screening of measures:

- ▷ $\mathbf{BC}_1$ and $\mathbf{BC}_2$ true because $\mathcal{D}(\mathcal{V}) = \mathbf{U}$;
- ▷ $\mathbf{BC}_3$ eliminates right half of Table 2;
- ▷ $\mathbf{BC}_5$ true from observing data;
- ▷ $\mathbf{BC}_6$ true because all aspects contribute to total effort;
- ▷ **Result:** Measure pool **S** reduced from 40 to 18.

Pilot: Measure Selection

Criteria: Conformance between $\mathcal{V}$ and *total-effort* distribution. No need for $\mathbf{S}_1$.

Derivation: Mark a quartile "+" if $p_i(\mathcal{V}) \geq 0.75$ and "−" if $n_i(\mathcal{V}) \geq 0.75$, where:

- ▷ $m_i(\mathcal{V}) = \#$modules in quartile i;
- ▷ $p_i(\mathcal{V}) = prob(m_4(\text{effort})|m_i(\mathcal{V}))$;
- ▷ $n_i(\mathcal{V}) = 1 - p_i(\mathcal{V})$.

$$\max_{\mathcal{V}, \mathcal{V} \in \mathbf{S}} \left\{ \sum_{\substack{i=1, \\ p_i(\mathcal{V}) \geq 0.75 \\ \vee n_i(\mathcal{V}) \geq 0.75}}^{4} \{m_i(\mathcal{V})p_i(\mathcal{V}) + m_i(\mathcal{V})n_i(\mathcal{V})\} \right\}$$

Pilot: Prediction Result

ACTA:

		Actual		
		+	–	total
Predicted	+	7	17	24
	–	4	143	147
	total	11	160	171

OCTA:

		Actual		
		+	–	total
Predicted	+	7	32	39
	–	4	129	133
	total	11	161	172

performance measure	OCTA	ACTA
not identified	4	5
correctly identified	136	150
incorrectly identified	36	21
coverage	97%	97%
accuracy	79%	87%
completeness	63%	63%
consistency	17%	29%

Validation: Problems and Environment

Goal: Extrapolate pilot study result to validate our model.

Embedded Environment: NASA/SEL:

- ▷ 16 ground support projects ;
- ▷ SLOC: 3K to 112K of Fortran code;
- ▷ Staffing: 4-23 (5-25M / 5-140 MM);
- ▷ Modules: 83-531/proj, 4700+ total;
- ▷ Measures: 74 collected.

Direct Environment: CTA:

- ▷ Training set size: 1;
- ▷ Testing on immediate next project;
- ▷ 10 data points from 16 raw data;
- ▷ 5 data points from isolated data.

Validation: Result Comparison

Overall Comparison:

		OCTA	ACTA
cost	measure pool	40	18
	tree size (all)	12.5	9.1
	tree size (-1)	7.3	4.4
perform-ance	coverage	97.6%	97.0%
	accuracy	69.7%	74.5%
	consistency	38.4%	50.4%
	completeness	35.6%	36.0%

Validation: Validation Result

1. Comparing with original CTA, measure selection using our model is effective:
   - ▷ Cost: Measure pool size and classification tree complexity are reduced dramatically;
   - ▷ Performance: Coverage and completeness remain virtually the same; Accuracy and consistency are improved.

2. Comparing with random guessing, CTA based on either measure selection method made great improvement, well worth the cost.

3. The multiple data-points indicate the validity of our model.

Validation: Baselines

Baseline 1: Original CTA.

Baseline 2: Optimal Random Guessing:

$$\begin{aligned} \text{coverage} &= 100\% \\ \text{accuracy} &= 62.5\% \\ \text{completeness} &= 25\% \\ \text{consistency} &= 25\% \end{aligned}$$

Comment: Other random guessing:

- ▷ consistency $\equiv$ 25%;
- ▷ max(accuracy) = 75% with 0 completeness;
- ▷ max(completeness) = 100% with 25% accuracy.

Conclusion

- Our model provides a scientific model of program complexity to understand and improve software process;
- Our theory of program complexity embodies the empirical research and extends formal models in this area;
- Our technique of measure evaluation demonstrates the usability of our theory in solving software engineering problems;
- Our model appears valid and effective as demonstrated by the multiple applications.