BLG 475E: Software Quality and Testing
Fall 2014-15

Software quality metrics

Assist. Prof. Ayşe Tosun Mısırlı
tosunmisirli@itu.edu.tr

İstanbul Teknik Üniversitesi

Software engineering

- 3Ps in software development lifecycle
  - Product (Project)
  - Process
  - People

Product metrics

- Characteristics of the product
  - Size
  - Complexity
  - Design features
  - Performance
  - Quality level

Process metrics

- Improve software processes (development and maintenance)
  - Effectiveness of defect removal
  - The pattern of testing defect arrival
  - The response time of the fix process

Project metrics

- Project characteristics and execution
  - Number of developers
  - The staffing pattern over the software lifecycle
  - Cost, schedule and productivity
  - In-process quality metrics are both project and process.

Software quality metrics

- Subset of software metrics
- More closely associated with process and product aspects than with project.
- End product quality metrics and in-process quality metrics
  - Examples from product, in-process, maintenance quality

Product Quality Metrics

- Mean time to failure (MTTF)
- Defect density
- Customer problems
- Customer satisfaction

Product Quality Metrics

- Measured by
  - the number of “bugs” (functional defects)
  - How long the software can run before it “crashes”
- MTTF is used in safety critical systems
  - Measures the time between failures
  - Examples:
    - US air traffic control system can not be unavailable for more than 3 sec. per year.
    - Civilian airlines, probability of catastrophic failures must be no worse than 10^{-9} per hour.
- Defect density is used in commercial apps
  - Defects relative to the software size (LOC or function points)

Defect Density Metric

- Number of defects / (size of software)
- Size of software
- KLOC – Thousand lines of code
- FP – Function Points
- Time frame – L.O.P – Life of Product
  - Experience with operating systems show
    - 95% of software defects are found within four years (?)
  - Experience with application software
    - Two years
- Extreme caution is necessary when comparing defect densities of two products
  - If operational definitions of LOC, defect and time frames are not identical!

Lines of Code

- Count only executable lines
- Count executable lines plus data definitions
- Count executable lines, data definitions, and comments
- Count executable lines, data definitions, comments, and job control language
- Count lines as physical lines on an input screen
- Count lines as terminated by logical delimiters

```c
for (i = 0; i < 100; i += 1) printf("hello"); /* How many lines of code is this? */
```

```c
/* Now how many lines of code is this? */
for (i = 0; i < 100; i += 1)
{
    printf("hello");
}
```

C

```c
#include <stdio.h>

int main() {
    printf("\nHello world\n");
}
```

COBOL

```cobol
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. HELLOWORLD.
000300
000400*
000500 ENVIRONMENT DIVISION.
000600 CONFIGURATION SECTION.
000700 SOURCE-COMPUTER. RM-COBOL.
000800 OBJECT-COMPUTER. RM-COBOL.
000900
001000 DATA DIVISION.
001100 FILE SECTION.
001200
100000 PROCEDURE DIVISION.
100100
100200 MAIN-LOGIC SECTION.
100300 BEGIN.
100400     DISPLAY " " LINE 1 POSITION 1 ERASE EOS.
100500     DISPLAY "Hello world!" LINE 15 POSITION 10.
100600     STOP RUN.
100700 MAIN-LOGIC-EXIT.
100800     EXIT.
```

C	COBOL
Lines of code: 4 (excluding whitespace)	Lines of code: 17 (excluding whitespace)

Lines of Code

- In the context of defect rate calculation
- Productivity studies
  - The amount of LOC is negatively correlated with design efficiency
- Enhancements and new versions
  - LOC count for the entire product and the changed code
  - Defect tracking

Example: Lines of code defect rates

- LOC: source code instructions
- Two size metrics are
  1. Shipped source instructions (SSI)
  2. New and changed source instructions (CSI)
- Defect rate metrics
  - Total defects per KSSI (total release code quality)
  - Field defects per KSSI
  - Release-origin defects per KCSI (quality of new and changed code)
  - Release-origin field defects per KCSI

SSI (current release) = SSI (previous release)
+ CSI (new and changed code instructions for current release)
– deleted code (usually very small)
– changed code (to avoid double count in both SSI and CSI)

From Kan 2002

Customer's perspective

- Example
  - *Initial release of a product*
    - KCSI = KSSI = 50 KLOC
    - Defects / KCSI = 2.0
    - Total number of defects = 2 * 50 = **100**
  - *Second release*
    - KCSI = 20
    - KSSI = 50 + 20 (new and changed loc) – 4 (assuming 20% are changed loc) = 66
    - Defects / KCSI = 1.8 (assuming 10% improvement over the first release)
    - Total number of additional defects = 1.8 * 20 = **36**
  - *Third release*
    - KCSI = 30
    - KSSI = 66 + 30 – 6 (same assumption) = 90
    - Targeted number of additional defects = 36
    - Defects /KCSI = 36/30 = 1.2 or lower (at least 1/3 improvement)

100-36% reduction from customer's perspective

Customer problems metric

- How is it related to defects?

Defect Rate and Customer Problems Metrics

	Defect Rate	Problems per User-Month (PUM)
Numerator	Valid and unique product defects	All customer problems (defects and nondefects, first time and repeated)
Denominator	Size of product (KLOC or function point)	Customer usage of the product (user-months)
Measurement perspective	Producer—software development organization	Customer
Scope	Intrinsic product quality	Intrinsic product quality plus other factors

From Kan 2002

Function Points

- Size measure instead of LOC
- A collection of executable statements that performs a certain task together with declarations of the formal parameters and local variables manipulated by those statements
- Originated by Albrecht at IBM in mid 70s

Function Counts (Step 1)

- Weighted total of 5 major components:
  - Number of external inputs (transaction types)X4
  - Number of external outputs (report types)X5
  - Number of logical internal files (the ones that user may concieve, not the physical files)X10
  - Number of external interface files (files accessed by the apps but not maintained by it)X7
  - Number of external inquiries (types of online inquiries supported) X4
- These are average weighting factors

$$FC = \sum_{i=1}^{5} \sum_{j=1}^{3} w_{ij} \times x_{ij}$$

Function Counts

- There are low and high weighting factors depending on the complexity assessment of the app.:
  - External input: low complexity,3; high complexity, 6
  - External output: low complexity, 4; high complexity, 7
  - Logical internal file:low complexity, 7; high complexity, 15
  - External interface file: low complexity, 5; high complexity, 10
  - External inquiry: low complexity, 3; high complexity, 6

System characteristics (Step 2)

- Scale from 0 to 5
  1. Data communications
  2. Distributed functions
  3. Performance
  4. Heavily used configuration
  5. Transaction rate
  6. Online data entry
  7. End-user efficiency
  8. Online update
  9. Complex processing
  10. Reusability
  11. Installation ease
  12. Operational ease
  13. Multiple sites
  14. Facilitation of change

$$\mathrm{VAF} = 0.65 + 0.01 \sum_{i=1}^{14} c_i$$

FUNCTION POINT FORMULA

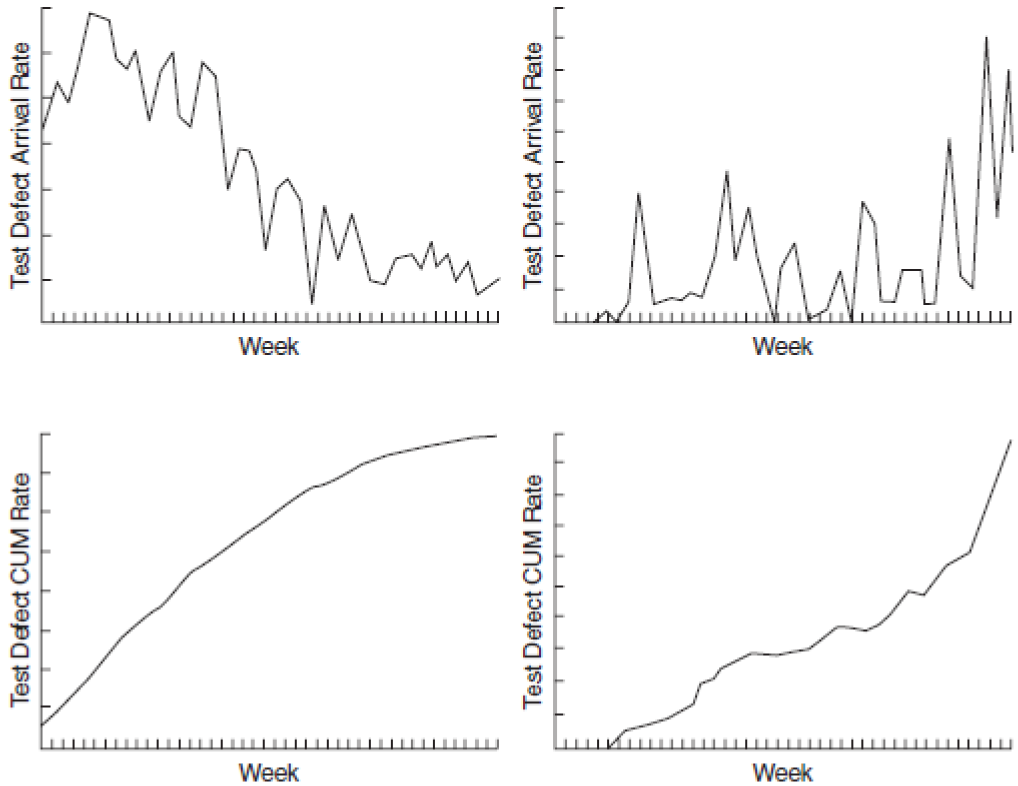
$$\mathrm{FP} = \mathrm{FC} \times \mathrm{VAF}$$

Suggested FP Defect Rates by CMMI

SEI CMM Level 1: 0.75
SEI CMM Level 2: 0.44
SEI CMM Level 3: 0.27
SEI CMM Level 4: 0.14
SEI CMM Level 5: 0.05

In-process Quality Metrics

- Defect arrival rate



From Kan 2002

- Which one is better/expected?
- Figures on left side or right side?
- Why?

In-process quality metrics

- Defect removal efficiency
  - DRE = E/(E+D)
  - E: number of errors found and removed before delivery of software
  - D: number of defects found after delivery
  - Question: Can D be calculated precisely?
  - Ideally, DRE=1.

Other Quality Metrics

- Defects are often monitored to evaluate software quality.

- Reducing defects are closely connected with their reasons.
  - Is software product problematic?
    - Too complex code, not easy to read, not following coding guidelines?
  - Is development process problematic?
    - Too many/few people working on the code, too many/few edits done?
  - Is design process problematic?
  - Is requirements problematic?
  - Are there problems in development team (communication)?

- Software product and process metrics are defined to find unique characteristics of defects.

Common metrics used to predict software quality

- Process
  - Requirements metrics
  - Object oriented design metrics
  - Churn/historical metrics
- Product
  - Object oriented design metrics
  - McCabe complexity metrics
  - Halstead metrics
  - Network metrics
- People
  - Organizational metrics
  - Developer related metrics

Code Metrics

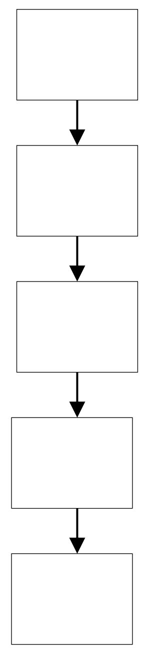
Object-oriented design metrics

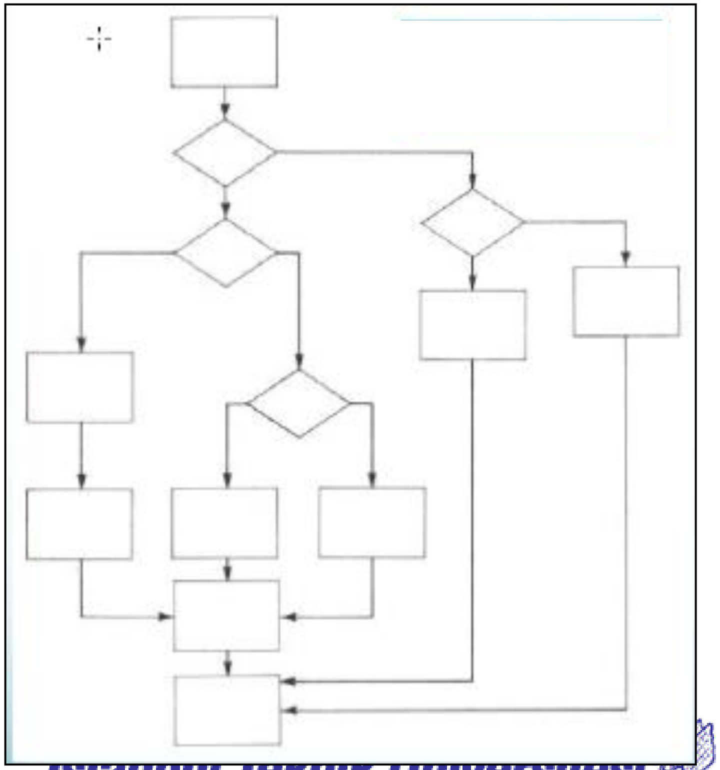
- By Chidamber and Kemerer 1994
- Weighted methods per class (WMC)
- Depth of inheritance tree (DIT)
  - Maximum length from the node to the root of the tree.
- Number of children (NOC)
- Coupling between object classes(CBO)
- Response for a class (RFC)
  - Set of methods that can potentially be executed in response to a message received by an object of that class
- Lack of cohesion in methods (LCOM)
  - If number of methods accessing one or more of the same attributes, LCOM is not 0.

Static code attributes

- McCabe's cyclomatic complexity
  - Max. number of linearly independent cycles in a strongly connected graph
  - E: edges, N: nodes
  - $v(G) = e - n + 2$

What is the complexity for each of these examples?





Static code attributes

- Halstead measures
  - A program can be considered as tokens (operators and operands)
  - Base measures:
    - Number of unique operators
    - Number of unique operands
    - Total number of operators
    - Total number of operands
  - Derived measures

N_1	num_operators
N_2	num_operands
μ_1	num_unique_operators
μ_2	num_unique_operands
N	length: $N = N_1 + N_2$
V	volume: $V = N * log_2 \mu$
L	level: $L = V^* / V$ where $V^* = (2 + \mu_2{}^*) log_2 (2 + \mu_2{}^*)$
D	difficulty: $D = 1/L$
I	content: $I = \hat{L} * V$ where $\hat{L} = \frac{2}{\mu_1} * \frac{\mu_2}{N_2}$
E	effort: $E = V/\hat{L}$
B	error_est
T	prog_time: $T = E/18$ seconds

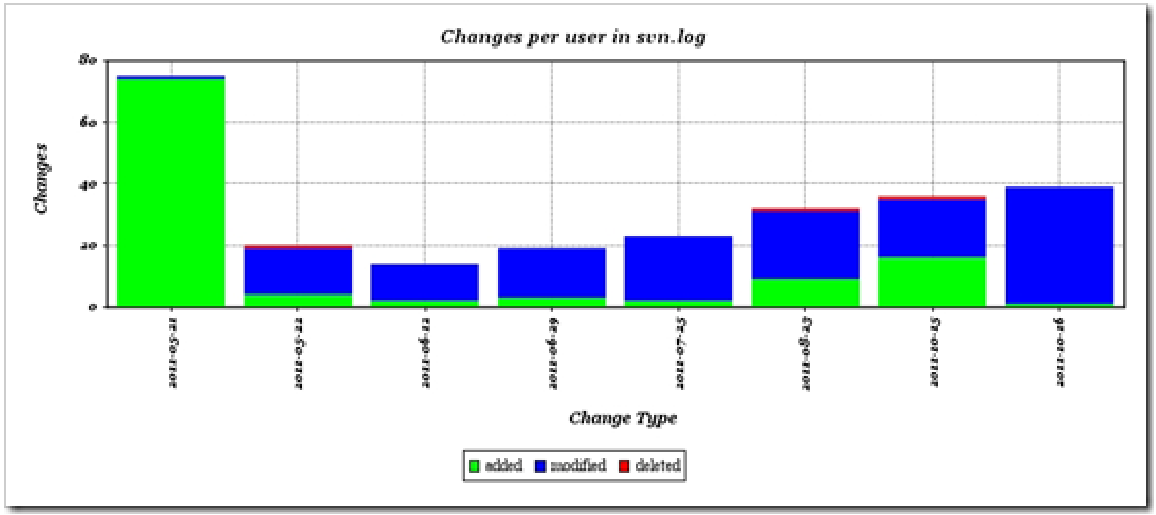
Standards for static code attributes

- Defined by NASA IV&V Metrics Program
- Not applicable to all programming languages

Metrik	Min	Max
Intelligent Content		50
Maximum Nesting Depth		3
Volume	30	1000
Total Operators	50	125
Time		5000
Difficulty		35
Vocabulary	25	75
Effort		100000
Unique Operands	10	40
Unique Operators	15	40
Total Operands	25	70
Architectural Complexity		60
Level	0.02	1
Ratio Of Comment To Code	0.15	
Length		300
Cyclomatic Complexity		10
Structural Complexity		5
Total Lines Of Code		

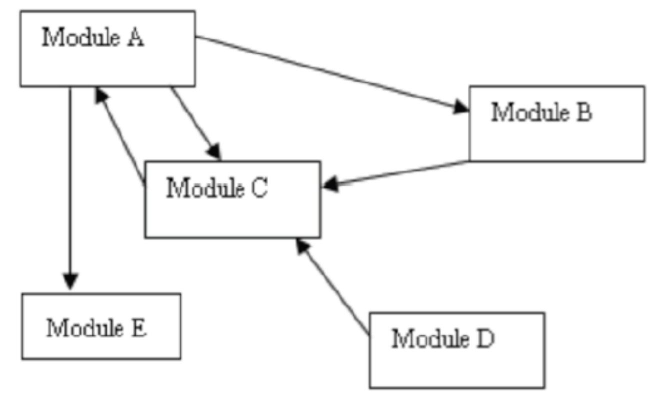
Code Churn Metrics

- Software change history
- Metrics
  - Total number of developers who made the edits
  - Total added, deleted and modified LOC
  - Number of times a file/binary/class was edited
  - Total number of consecutive edits
- Automated tools to collect these metrics
  - e.g. StatSVN



Dependency / Network Metrics

- Caller – callee dependencies



Ego-network metrics (computed each for incoming, outgoing, and undirected dependencies; descriptions adapted from Z & N [1]):

Size	Size	# nodes connected to the ego network
Ties	Ties	# directed ties corresponds to the number of edges
Pairs	Pairs	# ordered pairs is the maximal number of directed ties, i.e., Size × (Size - 1)
Density	Density	% of possible ties that are actually present, i.e., Ties/Pairs
WeakComp	WeakComp	# weak components in neighborhood
nWeakComp	nWeakComp	# weak components normalized by size, i.e., WeakComp/Size
TwoStepReach	TwoStepReach	% nodes that are two steps away

Centrality metrics (computed each for incoming, outgoing, and undirected dependencies; descriptions adapted from Z & N [1]):

Degree	Degree	# dependencies for an entity
nDegree	(none)	# dependencies for an entity normalized by number of entities
Closeness	Closeness	Sum of the lengths of the shortest paths from an entity (or to an entity) from all other entities
Reachability	dwReach	# entities that can be reached from a entity (or which can reach an entity)
Eigenvector	Eigenvector	assigns relative scores to all entities in the dependency graphs.
nEigenvector	(none)	assigns relative scores to all entities in the dependency graphs normalized by number of entities
Information	Information	Harmonic mean of the length of paths ending at an entity.
Betweenness	Betweenness	Measure for a entity on how many shortest paths between other entities it occurs
nBetweenness	(none)	Betweenness normalized by the number of entities

References

- Kan, S. 2002. Metrics and Models in Software Engineering, Addison Wesley.
- Fenton, N., Pfleger, S.L. 1996. Software Metrics: A Rigorous and Practical Approach, 2nd edition, International Thomson Computer Press, London.
- Galin, D. 2004. Software Quality Assurance: From theory to implementation, Addison Wesley.
- Menzies, T., Greenwald, T. and A. Frank. 2007. Data Mining Static Code Attributes to Learn Defect Predictors, IEEE Transactions on Software Engineering.
- Tosun, A., Turhan, B. and A. Bener. 2009. Validation of Network Measures as Indicators of Defective Modules in Software Systems, in Proc. Of the 5th International Conference on Predictor Models (PROMISE).