Software Quality and Metrics

IF3250 – Proyek Perangkat Lunak

Software Quality

Software Quality

- IEEE definition
  - Software quality is the degree to which system, component, or process meets
    - specified requirements.
    - customer or user needs or expectations.

Software Quality (2)

- Pressman's definition
  - Software quality is defined as
    - Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally development software.

ISO/IEC 25010:2011

- Systems and software Quality Requirements and Evaluation (SQuaRE)

System/Software Product Quality

Functional Suitability	Functional completeness, Functional Correctness, Functional Appropriateness
Performance Efficiency	Time Behaviour, Resource Utilization, Capacity
Compatibility *	Co-existence, Interoperability
Usability	Appropriateness Recognizability, Learnability, Operability, User Error Protection, User Interface Aesthetic, Accessibility
Reliability	Maturity, Availability, Fault Tolerance, Recoverability
Security *	Confidentiality, Integrity, Authenticity, Non-repudiation, Accountability
Maintainability	Modularity, Reusability, Analysability, Modifiability, Testability
Portability	Adaptability, Installability, Replaceability

Software Metrics

Why Metrics?

When you can measure what you are speaking about and express it in numbers, you know something about it; but when you cannot measure, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind: it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of science.

— Lord Kelvin

The importance of Measurement

- Measurement is *crucial* to the progress of all sciences, *even* Computer Science
- Scientific progress is made through
  - Observations and generalisations…
    - …based on data and measurements
  - Derivation of theories and…
    - …confirmation or refutation of these theories
- Measurement turns an art into a science

Some propositions

- Developers who drink coffee in the morning produce better code than those who do drink orange juice
- The more you test the system, the more reliable it will be in the field
- If you add more people to a project, it will be completed faster

How do you proof these propositions

Uses of Measurement

- Measurement helps us to understand
  - Makes the current activity visible
  - Measures establish guidelines
- Measurement allows us to control
  - Predict outcomes and change processes
- Measurement encourages us to improve
  - When we hold our product up to a measuring stick, we can establish quality targets and aim to improve

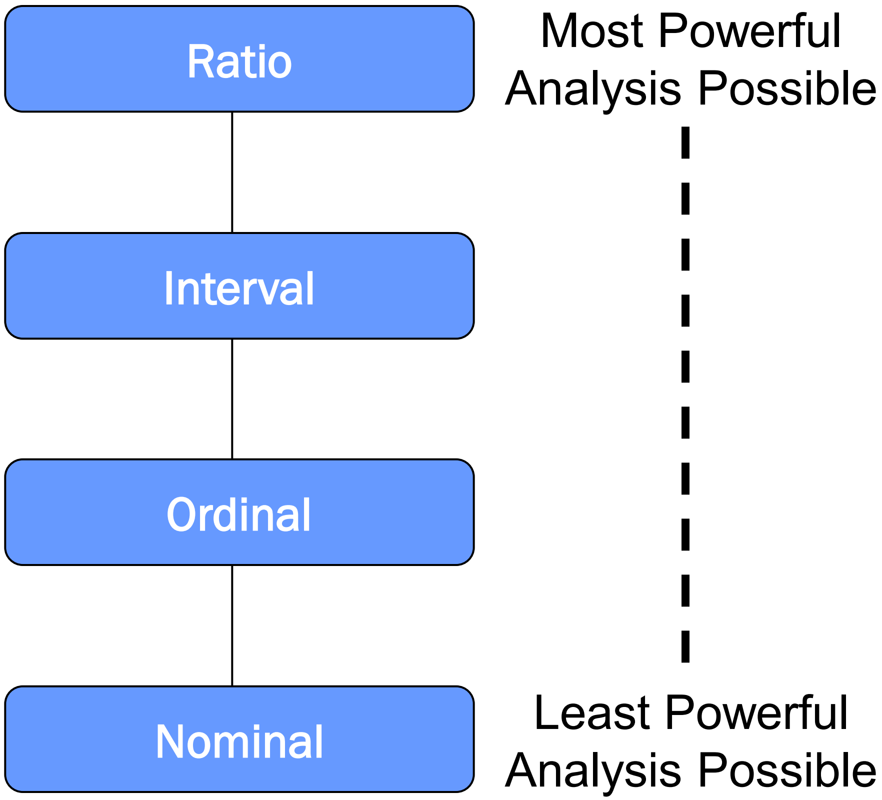
Levels of Scales

Various scales of measurements exist:

- Nominal Scale
  - E.g., religion, color, etc.
- Ordinal Scale
  - Degree classification, e.g., (S/M/L), (1^{st} class, 2^{nd} class, 3^{rd} class, etc)
- Interval Scale
  - Exact differences between measurement points, e.g., temperature
  - Addition/subtraction can be applied, but not multiplication/division
- Ratio Scale
  - Interval scale, where an absolute zero point can be located
  - Addition/subtraction and multiplication/division can be applied

Measurement Scales Hierarchy

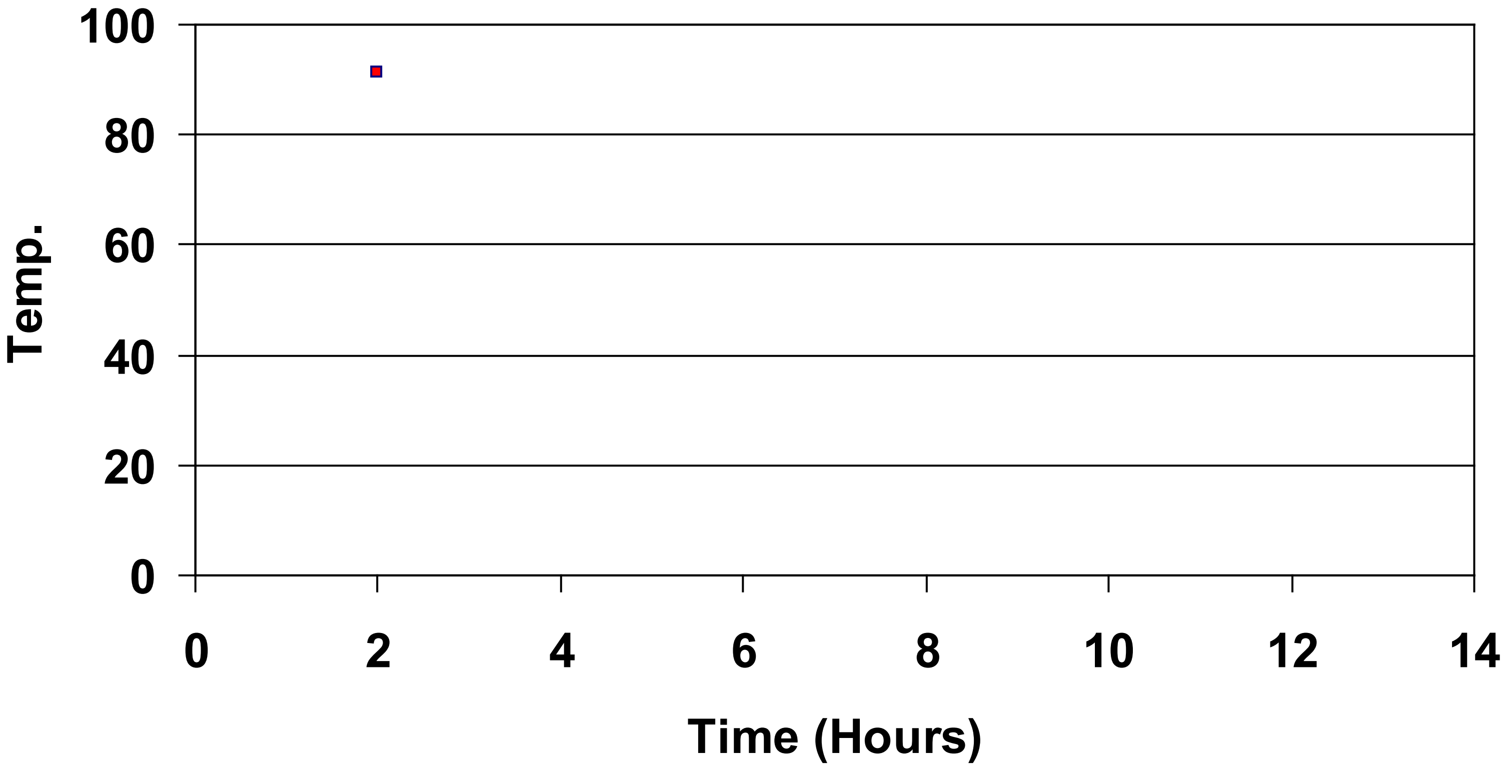
- Scales are hierarchical
- Each higher-level scale possesses all the properties of the lower ones
- A higher-level of measurement can be reduced to a lower one but not vice-versa



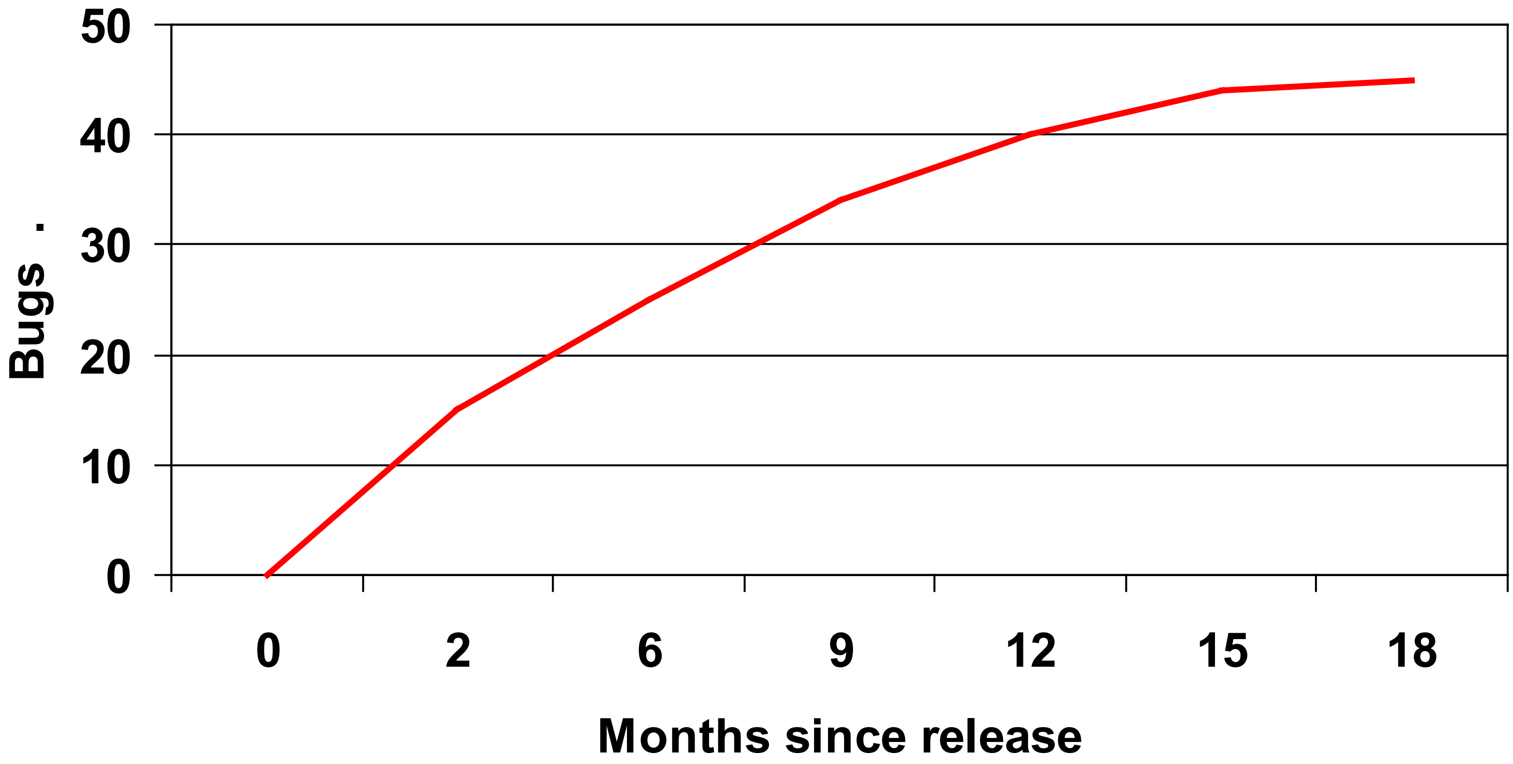
Measures, Metrics and Indicators

- **Measure** – An appraisal or ascertainment by comparing to a standard. E.g. Joe's body temperature is 99° fahrenheit
- **Metric** – A quantitative measure of the degree to which an element (e.g. software system) given attribute.
  - E.g. 2 errors were discovered by customers in 18 months (more meaningful than saying that 2 errors were found)
- **Indicator** – A device, variable or metric can indicate whether a particular state or goal has been achieved. Usually used to draw someone's attention to something.
  - E.g. A half-mast flag indicates that someone has died

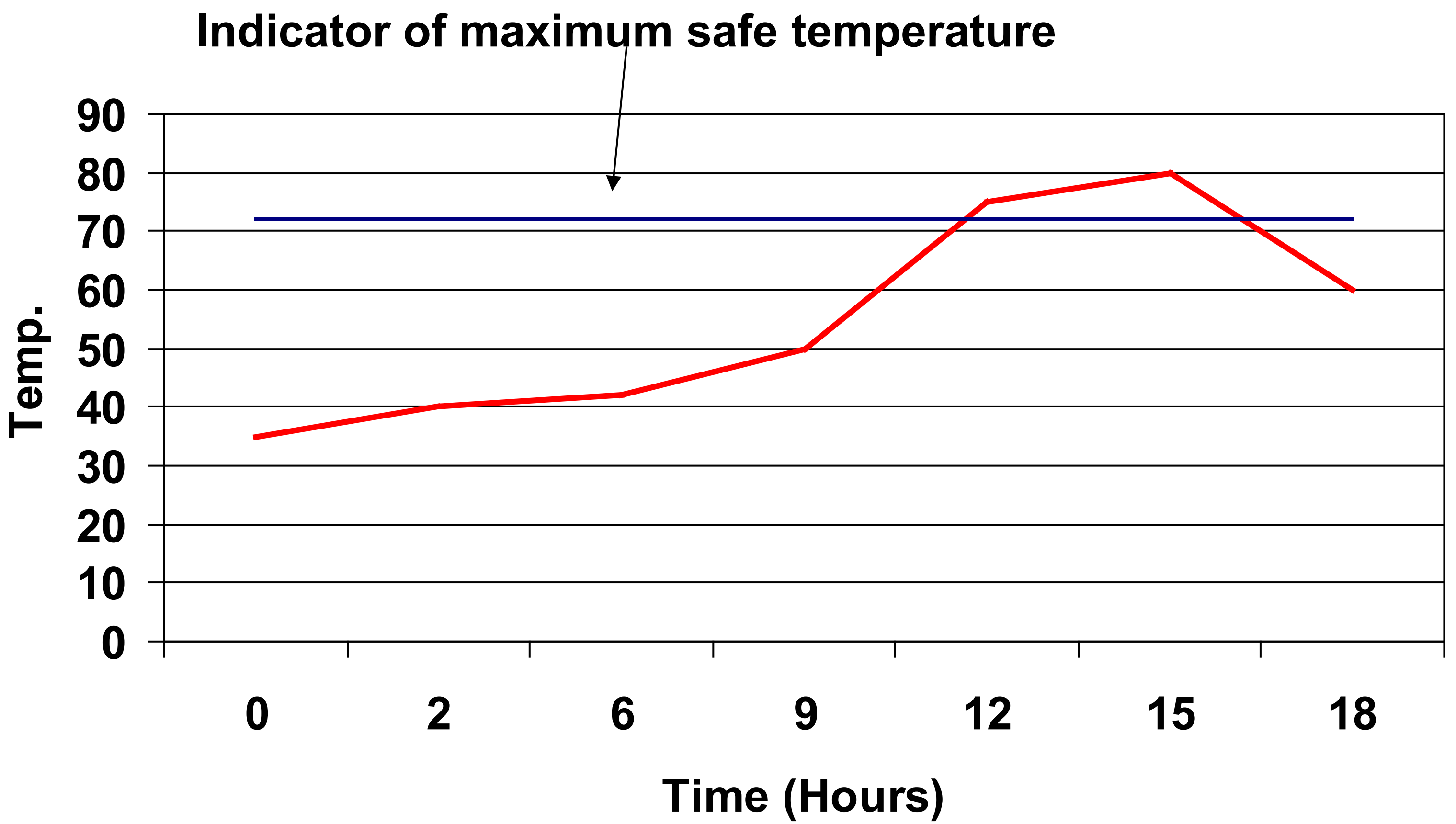
Example of a Measure



Example of a Metric



Example of a Indicator



Direct and Indirect Measures

- Direct Measures
  - *Measured* directly in terms of the observed attribute (usually by counting)
    - Length of source-code, Duration of process, Number of defects discovered
- Indirect Measures
  - *Calculated* from other direct and indirect measures
    - Module Defect Density = Number of defects discovered / Length of source
    - Temperature (usually derived from the length of a liquid column)

Some Desirable Properties of Metrics

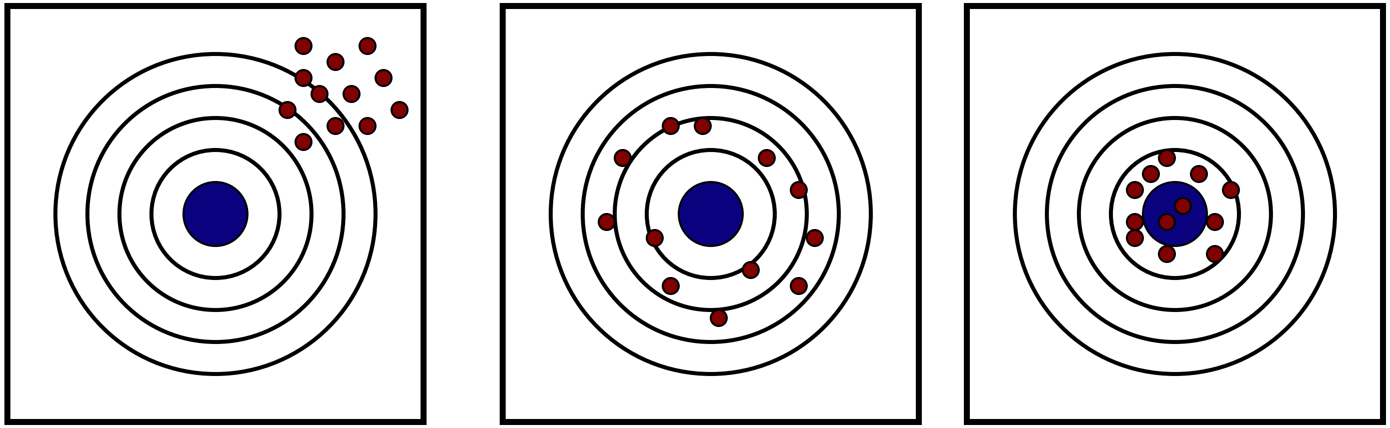
- Valid and reliable (consistent)
- Objective, precise
- Intuitive
- Robust (failure-tolerant)
- Automatable and economical (practical)
- …

Caveat: Attempts to define formally desirable properties have been heavily disputed …

See: 2.Brian Henderson-Sellers, *Object-Oriented Metrics: Measures of Complexity, Prentice-Hall, 1996, Ch. 2.6*

Validity and reliability

- A good metric is both valid *(measures what it is intended to measure)* and reliable *(yields consistent results)*



Reliable but not valid

Valid but not reliable

Valid and reliable

See: Stephen H. Kan, *Metrics and Models in Software Quality Engineering*, Addison Wesley, 2002. Ch. 3.4

Measuring Software

Why Measure Software?

Estimate cost and effort	measure correlation between specifications and final product
Improve productivity	measure value and cost of software
Improve software quality	measure usability, efficiency, maintainability ...
Improve reliability	measure mean time to failure, etc.
Evaluate methods and tools	measure productivity, quality, reliability ...

"You cannot control what you cannot measure" — De Marco, 1982
"What is not measurable, make measurable" — Galileo

What are Software Metrics?

Software metrics

- Any type of measurement which relates to a software system, process or related documentation
  - Lines of code in a program
  - the Fog index (calculates readability of a piece of documentation)

    $0.4 \times (\text{\# words} / \text{\# sentences}) + (\text{\% words} \geq 3 \text{ syllables})$
  - number of person-days required to implement a use-case

Possible Problems

Example: Compare productivity in lines of code per time unit.

Do we use the same units to compare?	What is a “line of code”? What is the “time unit”?
Is the context the same?	Were programmers familiar with the language?
Is “code size” really what we want to produce?	What about code quality?
How do we want to interpret results?	Average productivity of a programmer? Programmer X is twice as productive as Y?
What do we want to do with the results?	Do you reward “productive” programmers? Do you compare productivity of software processes?

Metric Classification

- Products
  - Explicit results of software development activities
  - Deliverables, documentation, by products
- Processes
  - Activities related to production of software
- Resources
  - Inputs into the software development activities
  - hardware, knowledge, people

Software Product Metrics

Types of Measures

- Direct Measures (internal attributes)
  - Cost, effort, LOC, speed, memory

- Indirect Measures (external attributes)
  - Functionality, quality, complexity, efficiency, reliability, maintainability

Size-Oriented Metrics

- Size of the software produced
- *LOC* - Lines Of Code
- *KLOC* - 1000 Lines Of Code
- *SLOC* – Statement Lines of Code (ignore whitespace)
- Typical Measures:
  - Errors/KLOC, Defects/KLOC, Cost/LOC, Documentation Pages/KLOC

LOC Metrics

- Easy to use
- Easy to compute
- Language & programmer dependent

Complexity Metrics

- LOC metrics represents functions of complexity
  - But, language and programmer dependent
  - We need more than LOC metrics to measure complexity
- Classical complexity metrics
  - Halstead's Software Science (entropy measures)
    - η_1 - number of distinct operators
    - η_2 - number of distinct operands
    - N_1 - total number of operators
    - N_2 - total number of operands

Example

```
if (k < 2)
{
  if (k > 3)
    x = x*k;
}
```

- Distinct operators: if ( ) { } > < = * ;
- Distinct operands: k 2 3 x
- η_1 = 10
- η_2 = 4
- N_1 = 13
- N_2 = 7

Halstead's Complexity Measures

- Program vocabulary: $\eta = \eta_1 + \eta_2$
- Program length: $N = N_1 + N_2$
- Calculated estimated program length: $\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$
- Purity ratio: $PR = \hat{N}/_N$
- Volume: $V = N \times \log_2 \eta$
- Difficulty: $D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2}$
  - The difficulty measure is related to the difficulty of the program to write or understand, e.g., when doing code review.
- Effort: $E = D \times V$
- Time required to program: $T = \frac{E}{18}$ seconds
- Number of delivered bugs: $B = \frac{E^{\frac{2}{3}}}{3000}$

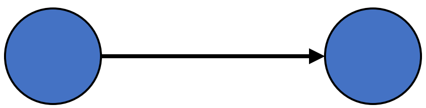
https://en.wikipedia.org/wiki/Halstead_complexity_measures

McCabe's Complexity Measures

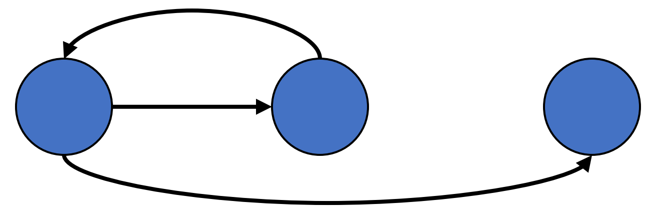
- McCabe's metrics are based on a control flow representation of the program.
- A program graph is used to depict control flow.
- Nodes represent processing tasks (one or more code statements)
- Edges represent control flow between nodes

Flow Graph Notation

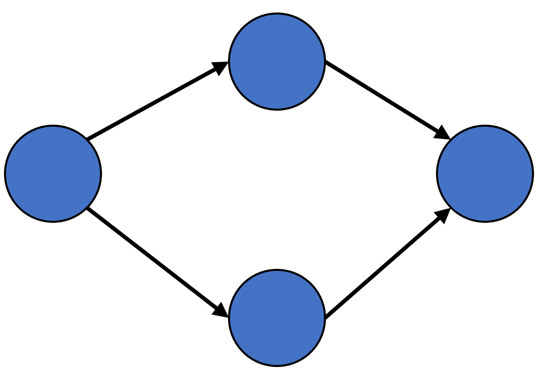
Sequence



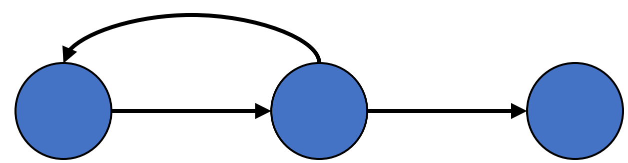
While



If-then-else



Until



Cyclomatic Complexity

- Set of independent paths through the graph (basis set)

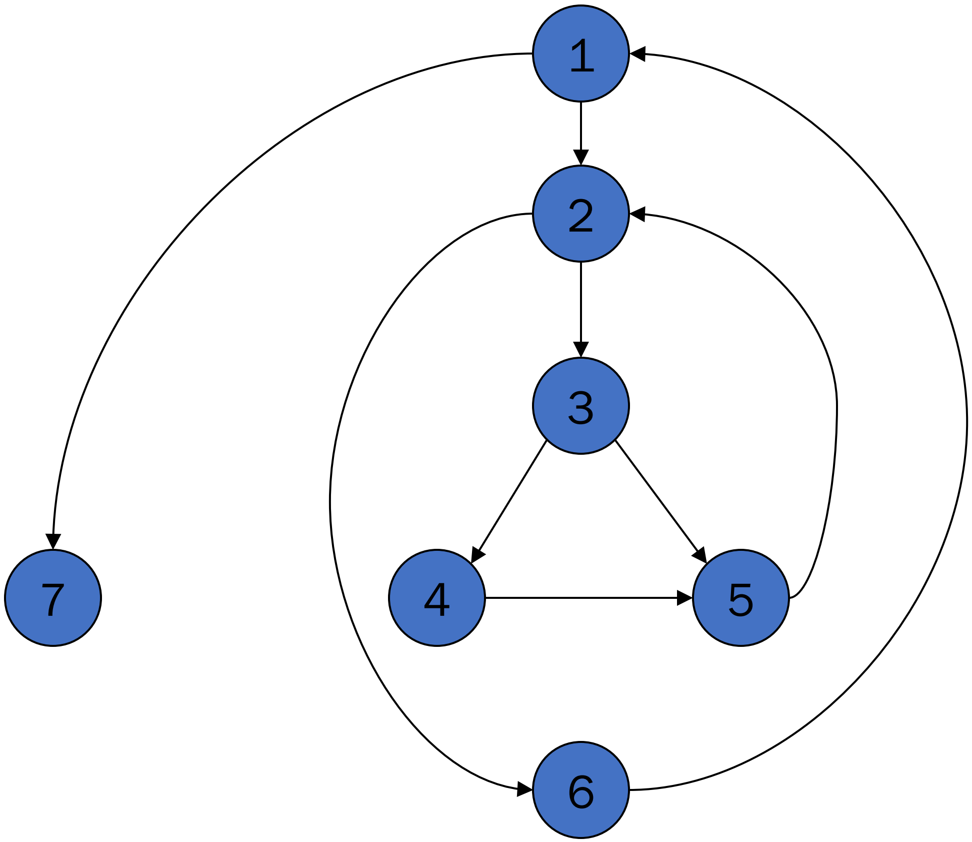
- $V(G) = E - N + 2$
  - E is the number of flow graph edges
  - N is the number of nodes

- $V(G) = P + 1$
  - P is the number of predicate nodes

Example

```
i = 0;
while (i<n-1) do
  j = i + 1;
  while (j<n) do
    if A[i]<A[j] then
      swap(A[i], A[j]);
  end do;
  i=i+1;
end do;
```

Flow Graph



Computing V(G)

- V(G) = 9 – 7 + 2 = 4
- V(G) = 3 + 1 = 4
- Basis Set
  - 1, 7
  - 1, 2, 6, 1, 7
  - 1, 2, 3, 4, 5, 2, 6, 1, 7
  - 1, 2, 3, 5, 2, 6, 1, 7

Meaning

- V(G) is the number of (enclosed) regions/areas of the planar graph
- Number of regions increases with the number of decision paths and loops
- A quantitative measure of testing difficulty and an indication of ultimate reliability
- Experimental data shows value of V(G) should be no more then 10 - testing is very difficulty above this value

McClure's Complexity Metric

- Complexity = C + V
  - C is the number of comparisons in a module
  - V is the number of control variables referenced in the module
  - decisional complexity
- Similar to McCabe's but with regard to control variables

High-level Design Metrics

- [Card & Glass]
- Structural Complexity S(i) of a module i.
  - $S(i) = f_{out}^2(i)$
  - Fan out is the number of modules immediately subordinate (directly invoked).
- Data Complexity D(i)
  - $D(i) = v(i)/[f_{out}(i)+1]$
  - v(i) is the number of inputs and outputs passed to and from i
- System Complexity C(i)
  - $C(i) = S(i) + D(i)$
  - As each increases the overall complexity of the architecture increases

Metrics for the Object Oriented Software

- WMC (Weighted Methods per Class)
- CS (Class Size)
- NoC (Number of Children)
- PF (Polymorphism Factor)
- DIT (Depth of Inheritance Tree)
- NOO (Number of Operations Overidden)
- NOA (Number of Operations Added)
- MIF (Method Inheritance Factor)
- Coupling Metrics:
  - RFC (Response for a class)
  - CF (Coupling Factor)
  - MPC (Message Passing Coupling)
  - CBO (Coupling between Objects)
- LCOM (Lack of cohesion of methods)
- ...

Weighted Methods per Class

$$\text{WMC} = \sum_{i=1}^{n} c_i$$

- c_i is the complexity (e.g., volume, cyclomatic complexity, etc.) of each method
- Viewpoints: (of Chidamber and Kemerer)
  - The number of methods and complexity of methods is an indicator of ***how much time and effort is required to develop and maintain*** the object
  - The ***larger the number of methods*** *in an object, the greater the potential* ***impact on the children***
  - Objects with ***large number of methods*** are likely to be more application specific, ***limiting the possible reuse***

Class Size

- CS
  - Total number of operations (inherited, private, public)
  - Number of attributes (inherited, private, public)

- May be an indication of too much responsibility for a class

Number of Children

- NOC is the number of subclasses immediately subordinate to a class
- Viewpoints:
  - As NOC grows, reuse increases - but the abstraction may be diluted
  - Depth is generally better than breadth in class hierarchy, since it promotes reuse of methods through inheritance
  - Classes higher up in the hierarchy should have more sub-classes then those lower down
  - NOC gives an idea of the potential influence a class has on the design: classes with large number of children may require more testing

Polymorphism Factor

$$PF = \frac{\sum_i M_o(C_i)}{\sum_i [M_n(C_i) * DC(C_i)]}$$

- $M_n()$ is the number of new methods
- $M_o()$ is the number of overriding methods
- DC() number of descendent classes of a base class
- The number of methods that redefines inherited methods, divided by maximum number of possible distinct polymorphic situations

Depth of Inheritance Tree

- DIT is the maximum length from a node to the root (base class)
- Viewpoints:
  - Lower level subclasses inherit a number of methods making behavior harder to predict
  - Deeper trees indicate greater design complexity

Number of Operations Overridden

- NOO

- A large number for NOO indicates possible problems with the design
- Poor abstraction in inheritance hierarchy

Number of Operations Added

- NOA

- The number of operations added by a subclass
- As operations are added it is farther away from super class
- As depth increases NOA should decrease

Method Inheritance Factor

$$\text{MIF} = \frac{\sum_{i=1}^{n} M_i(C_i)}{\sum_{i=1}^{n} M_a(C_i)}$$

- $M_i(C_i)$ is the number of methods inherited and not overridden in C_i
- $M_a(C_i)$ is the number of methods that can be invoked with C_i
- $M_d(C_i)$ is the number of methods declared in C_i

MIF

- $M_a(C_i) = M_d(C_i) + M_i(C_i)$
- All that can be invoked = new or overloaded + things inherited

- MIF is [0,1]
- MIF near 1 means little specialization
- MIF near 0 means large change

Response for a Class

- RFC is the number of methods that could be called in response to a message to a class (local + remote)

- Viewpoints:
  - As RFC increases
    - testing effort increases
    - greater the complexity of the object
    - harder it is to understand

Coupling Factor

$$CF = \frac{\sum_i \sum_j is\_client(C_i, C_j)}{(TC^2 - TC)}.$$

- is_client(x,y) = 1 iff a relationship exists between the client class and the server class. 0 otherwise
- (TC^2-TC) is the total number of relationships possible
- CF is [0,1] with 1 meaning high coupling

Coupling between Objects

- CBO is the number of collaborations between two classes (fan-out of a class C)
  - the number of other classes that are referenced in the class C (a reference to another class, A, is an reference to a method or a data member of class A)
- Viewpoints:
  - As collaboration increases reuse decreases
  - High fan-outs represent class coupling to other classes/objects and thus are undesirable
  - High fan-ins represent good object designs and high level of reuse
  - Not possible to maintain high fan-in and low fan outs across the entire system

Lack of Cohesion in Methods (LCOM)

- Class C_k with n methods $M_1,\ldots M_n$
- I_j is the set of instance variables used by M_j
- There are n such sets I_1 ,...., I_n
  - $P = \{(I_i, I_j) \mid (I_i \cap I_j) = \varnothing\}$
  - $Q = \{(I_i, I_j) \mid (I_i \cap I_j) \neq \varnothing\}$
- If all n sets I_i are $\varnothing$ then $P = \varnothing$
- LCOM = $|P| - |Q|$, if $|P| > |Q|$
- LCOM = 0 otherwise

Example LCOM

- Take class C with M_1, M_2, M_3
  - $I_1 = \{a, b, c, d, e\}$
  - $I_2 = \{a, b, e\}$
  - $I_3 = \{x, y, z\}$
  - $P = \{(I_1, I_3), (I_2, I_3)\}$
  - $Q = \{(I_1, I_2)\}$

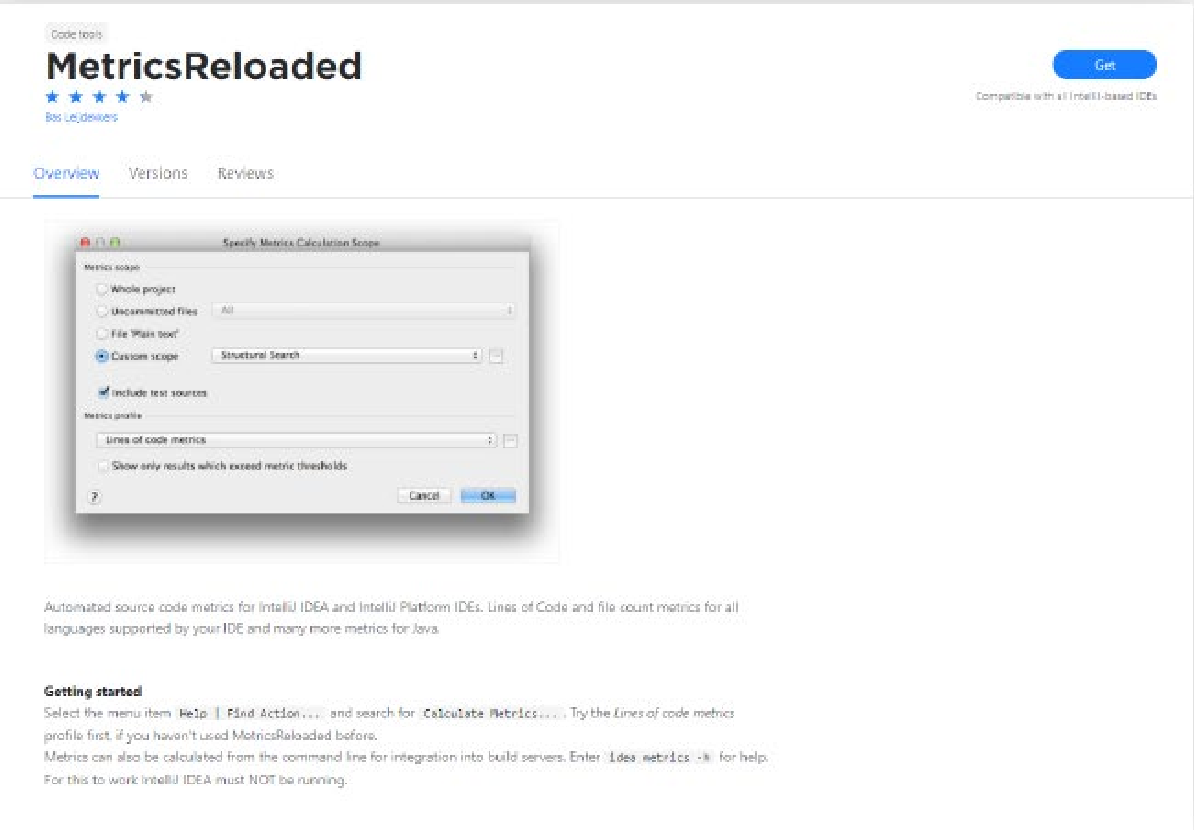
- Thus LCOM = 1

Explanation LCOM

- LCOM is the number of empty intersections minus the number of non-empty intersections
- This is a notion of degree of similarity of methods
- If two methods use common instance variables then they are similar
- LCOM of zero is not maximally cohesive
- $|P| = |Q|$ or $|P| < |Q|$

Software Measurement Tools

MetricsReloaded



IntelliJ IDE Plugin

- Supported Language: Java

Supported Metrics/Measurement

- Chidamber-Kemerer
- Class-count
- Complexity
- Dependency
- LOC
- Martin Packaging

Exploration and slides by Arrival Dwi Sentosa

MetricsReloaded – Sample Results

3. Chidamber-Kemerer Metrics (CBO, DIT, LCOM, RFC, WMC)

Class

class	CBO	DIT	LCOM	NOC	RFC	WMC
id.odt.simposiumasiaoceania2019.util.Util	2	1	2	0	16	3
id.odt.simposiumasiaoceania2019.util.DateUtils	1	1	5	0	15	10
id.odt.simposiumasiaoceania2019.model.UserModel	3	1	20	0	43	43
id.odt.simposiumasiaoceania2019.model.PanitiaModel	1	1	4	0	10	10
id.odt.simposiumasiaoceania2019.model.ActionModel	2	1	4	0	10	10
id.odt.simposiumasiaoceania2019.BaseApp	7	4	2	0	9	2
id.odt.simposiumasiaoceania2019.adapter.PendafterAdapter.ViewHolder	3	2	0	0	3	1
id.odt.simposiumasiaoceania2019.adapter.PendaftarAdapter	6	2	2	0	41	9
id.odt.simposiumasiaoceania2019.adapter.ActivityAdapter.ViewHolder	3	2	0	0	3	1
id.odt.simposiumasiaoceania2019.adapter.ActivityAdapter	6	2	2	0	16	4
id.odt.simposiumasiaoceania2019.activity.UserApproveActivity	7	8	2	0	28	7
id.odt.simposiumasiaoceania2019.activity.RecordsActivity	7	8	2	0	28	6
id.odt.simposiumasiaoceania2019.activity.QRScanActivity	6	8	1	0	60	25
id.odt.simposiumasiaoceania2019.activity.MainActivity	9	8	3	0	147	50
id.odt.simposiumasiaoceania2019.activity.LoginActivity	6	8	2	0	41	19
id.odt.simposiumasiaoceania2019.activity.Excel2FirestoreActivity	6	8	2	0	71	21
Total						222
Average	4.69	4.06	3.31	0.00	33.81	13.88

- Coupling Between Objects
- Depth of Inheritances Tree
- Lack of Cohesion Methods
- Response for Class
- Weighted Method Complexity

Exploration and slides by Arrival Dwi Sentosa

MetricsReloaded – Sample Results

Methods

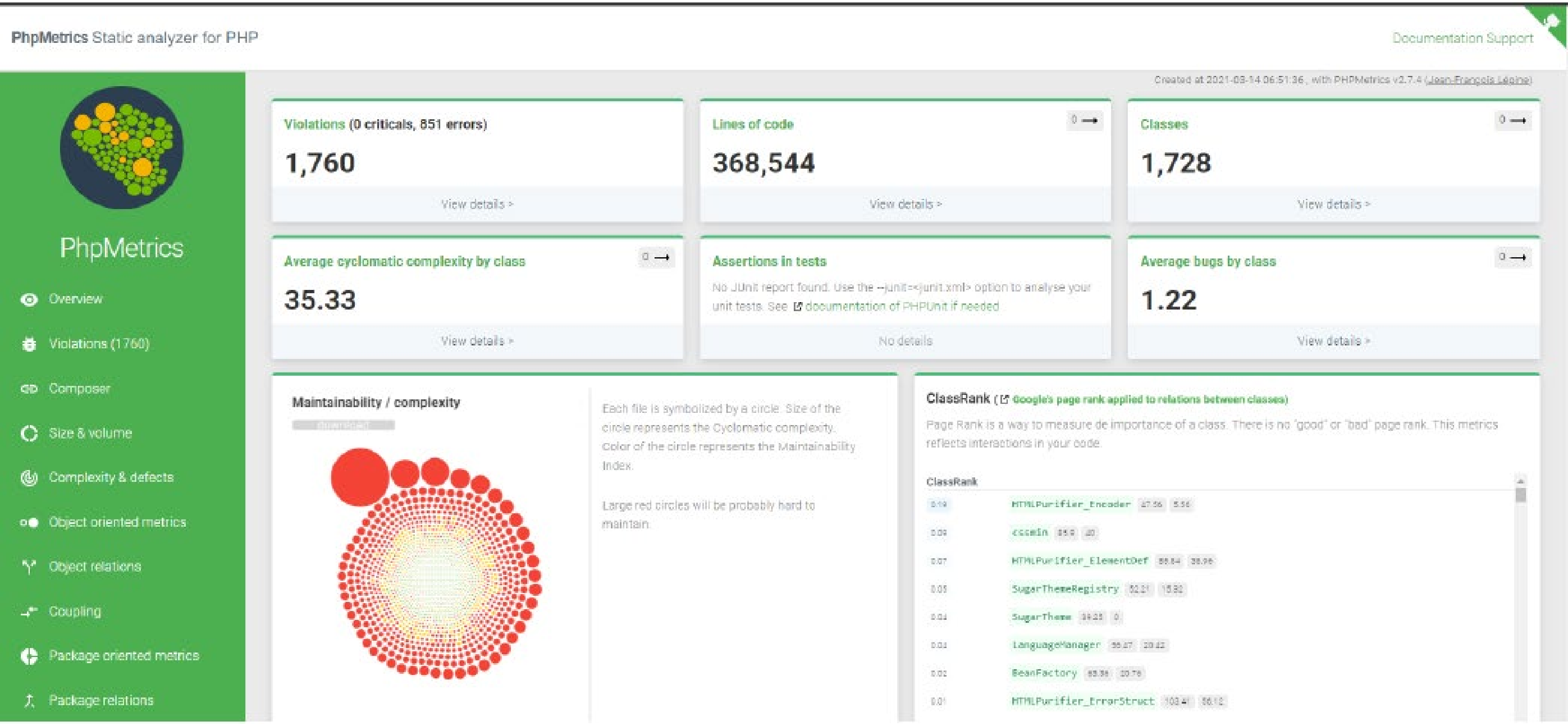
method	CogC	ev(G)	iv(G)	v(G)
id.odt.simposiumasiaoceania2019.model.UserModel.setNo_passport(String)	0	1	1	1
id.odt.simposiumasiaoceania2019.model.UserModel.setNohp(String)	0	1	1	1
id.odt.simposiumasiaoceania2019.model.UserModel.setPuasa(boolean)	0	1	1	1
id.odt.simposiumasiaoceania2019.model.UserModel.setStatus(ArrayList<String>)	0	1	1	1
id.odt.simposiumasiaoceania2019.model.UserModel.setUid(String)	0	1	1	1
id.odt.simposiumasiaoceania2019.model.UserModel.setUniversitas(String)	0	1	1	1
id.odt.simposiumasiaoceania2019.model.UserModel.setVege(boolean)	0	1	1	1
id.odt.simposiumasiaoceania2019.model.UserModel.setWechat(String)	0	1	1	1
id.odt.simposiumasiaoceania2019.model.UserModel.setWhatsapp(String)	0	1	1	1
id.odt.simposiumasiaoceania2019.model.UserModel.UserModel()	0	1	1	1
id.odt.simposiumasiaoceania2019.model.UserModel.UserModel(String,boolean,String,String,long,String,Strin	0	1	1	1
id.odt.simposiumasiaoceania2019.model.UserModel.UserModel(String,boolean,String,String,long,String,Strin	0	1	1	1
id.odt.simposiumasiaoceania2019.util.DateUtils.DateUtils()	0	1	1	1
id.odt.simposiumasiaoceania2019.util.DateUtils.formatDate(long)	0	1	1	1
id.odt.simposiumasiaoceania2019.util.DateUtils.formatDateTime(long)	2	2	2	2
id.odt.simposiumasiaoceania2019.util.DateUtils.formatTime(long)	0	1	1	1
id.odt.simposiumasiaoceania2019.util.DateUtils.formatTimeWithMarker(long)	0	1	1	1
id.odt.simposiumasiaoceania2019.util.DateUtils.getHourOfDay(long)	0	1	1	1
id.odt.simposiumasiaoceania2019.util.DateUtils.getMinute(long)	0	1	1	1
id.odt.simposiumasiaoceania2019.util.DateUtils.hasSameDate(long,long)	0	1	1	1
id.odt.simposiumasiaoceania2019.util.DateUtils.isToday(long)	0	1	1	1
id.odt.simposiumasiaoceania2019.util.Util.decodeFile(String)	3	1	2	4
id.odt.simposiumasiaoceania2019.util.Util.getDate(long,String)	0	1	1	1
null.onSuccess(DocumentReference)	0			
null.onSuccess(QuerySnapshot)	1			
Total	267	135	206	228
Average	2.07	1.06	1.62	1.80

3. Cyclomatic Complexity

- Cognitive Complexity
- Essential Cyclomatic Complexity
- Design Complexity
- Cyclomatic Complexity

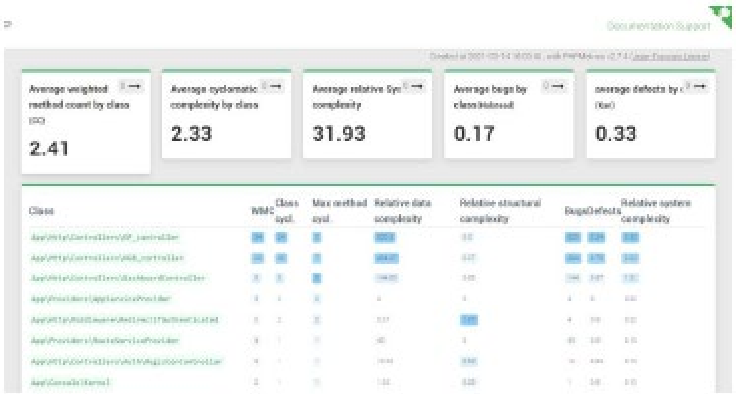
Exploration and slides by Arrival Dwi Sentosa

PHPMetrics – Sample Dashboard

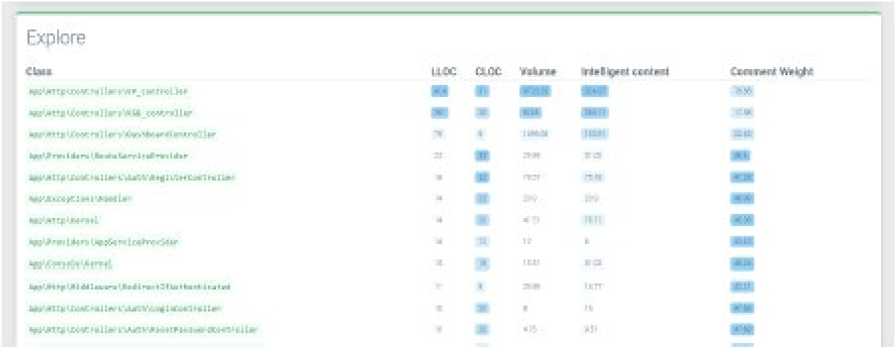


Exploration and slides by Labib Izzatur Rahman

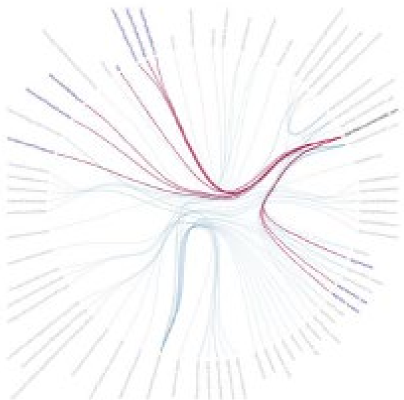
PHPMetrics – Sample Metrics



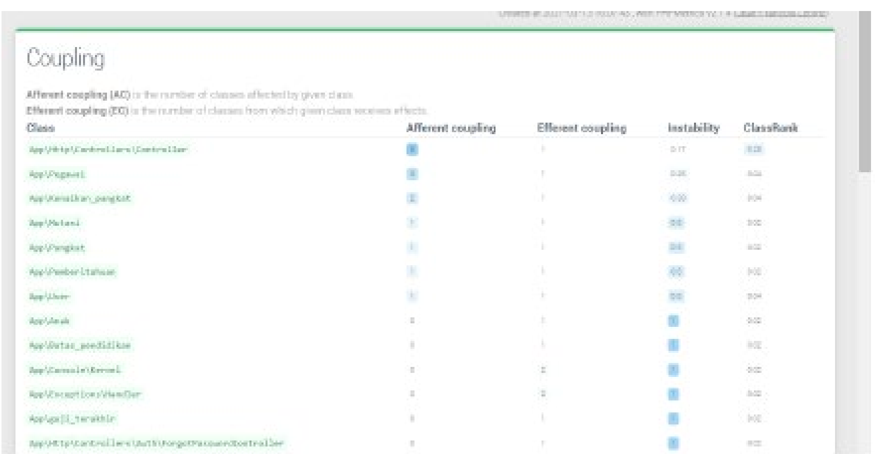
Complexity & defect



Size & Volume



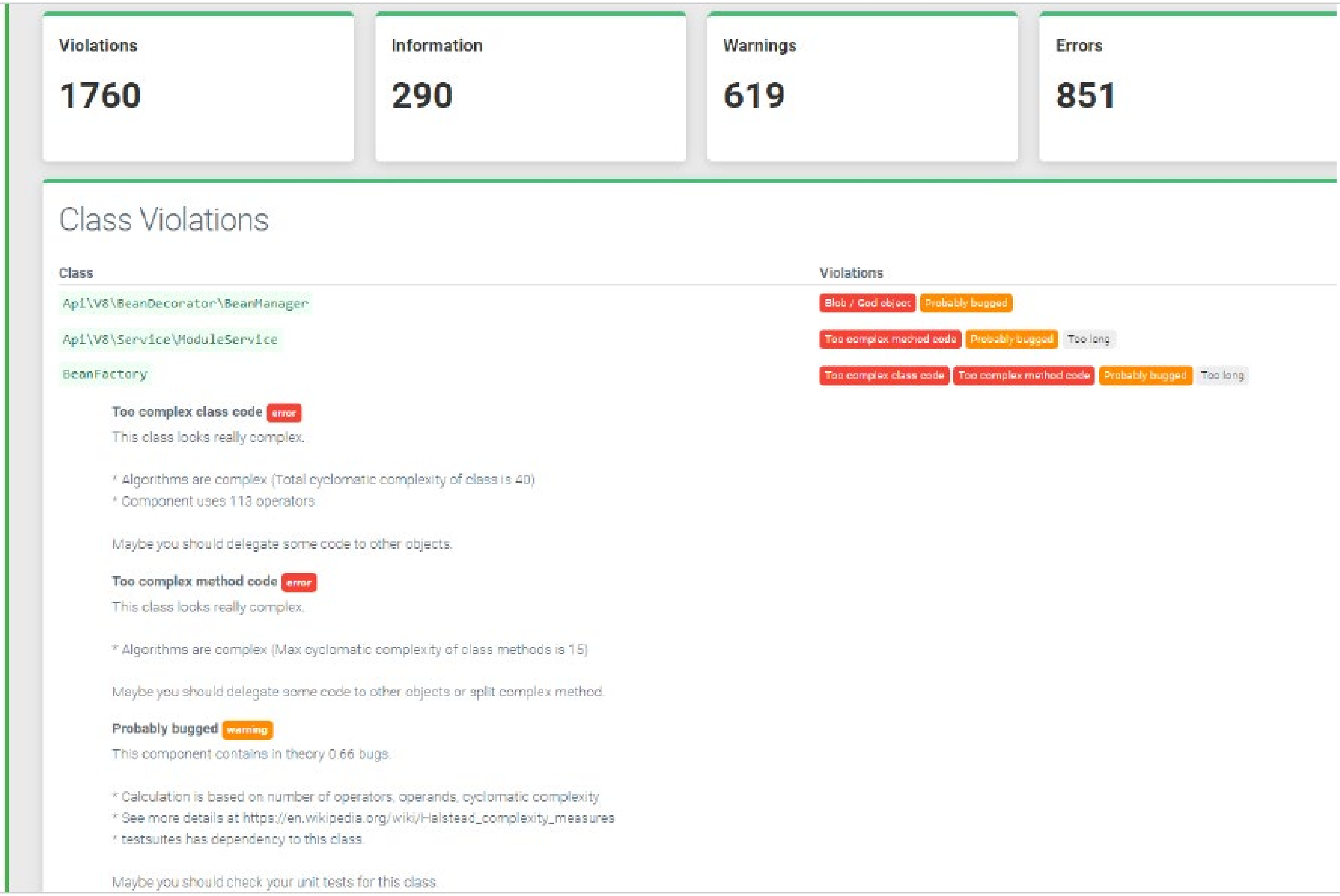
Object Relation



Coupling

Exploration and slides by Sigit Widodo

PHPMetrics – Sample Static Analysis

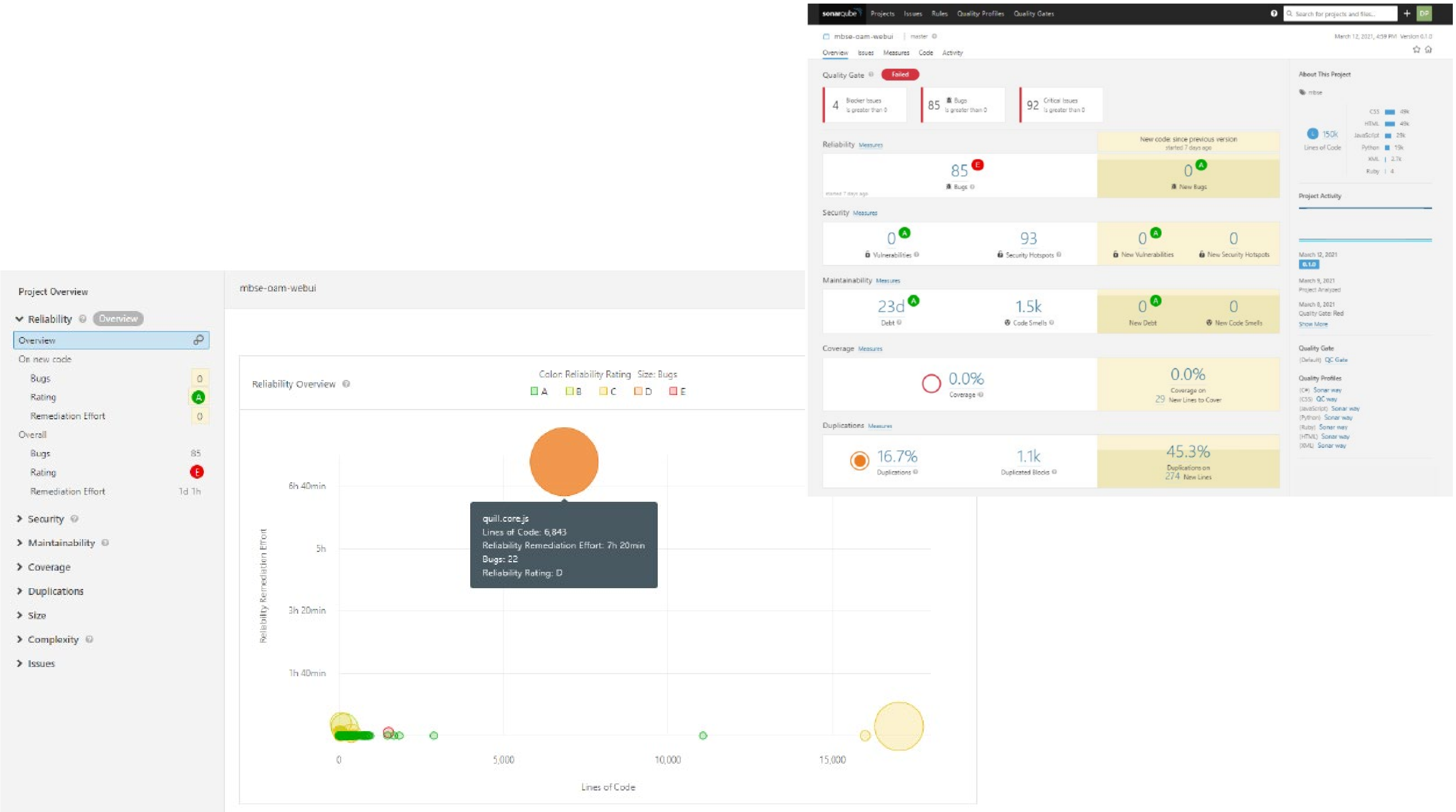


Exploration and slides by Arrival Dwi Sentosa

sonarqube

- A platform for continuous inspection for Code Quality
- For 27 programming languages:
  - Java (including Android), C#, C/C++, JavaScript, TypeScript, Python, Go, Swift, COBOL, Apex, PHP, Kotlin, Ruby, Scala, HTML, CSS, ABAP, Flex, Objective-C, PL/I, PL/SQL, RPG, T-SQL, VB.NET, VB6, XML
- Open-source with free and enterprise versions are available
- Can be intergrated with IDEs and CI/CD pipelines
- Static analysis (code smells, vulnerabilities, etc) + software metrics (LOC, Cyclomatic Complexity, etc)

SonarQube – Sample Reliability Dashboard



Exploration and slides by Deka Panca Gustiawan

SonarQube – Sample Quality Gates

Quality Gates Create

Sonar way DEFAULT BUILT-IN

eauction

QUALITY GATE STATUS

Failed

3 conditions failed

On Overall Code

319	Cognitive Complexity is greater than 10
321	Cyclomatic Complexity is greater than 15
106,357	Lines of Code is greater than 100

eauction Rename Copy Set as Default Delete

Conditions Add Condition

Conditions on New Code

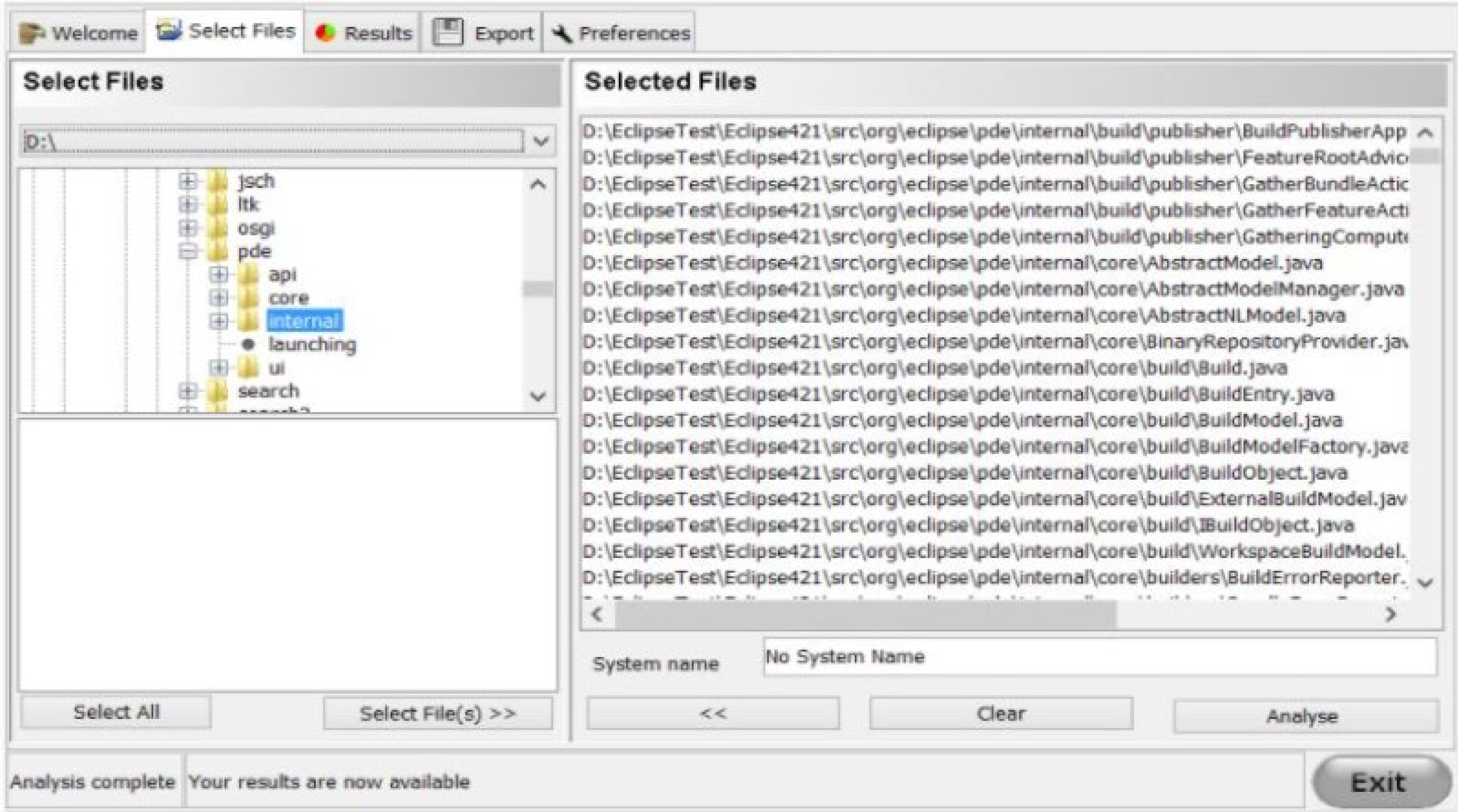
Metric	Operator	Value	Edit	Delete
Coverage	is less than	80.0%		
Duplicated Lines (%)	is greater than	3.0%		
Maintainability Rating	is worse than	A		
Reliability Rating	is worse than	A		
Security Hotspots Reviewed	is less than	100%		
Security Rating	is worse than	A		

Conditions on Overall Code

Metric	Operator	Value	Edit	Delete
Classes	is greater than	100		
Cognitive Complexity	is greater than	10		
Cyclomatic Complexity	is greater than	15		
Lines of Code	is greater than	100		

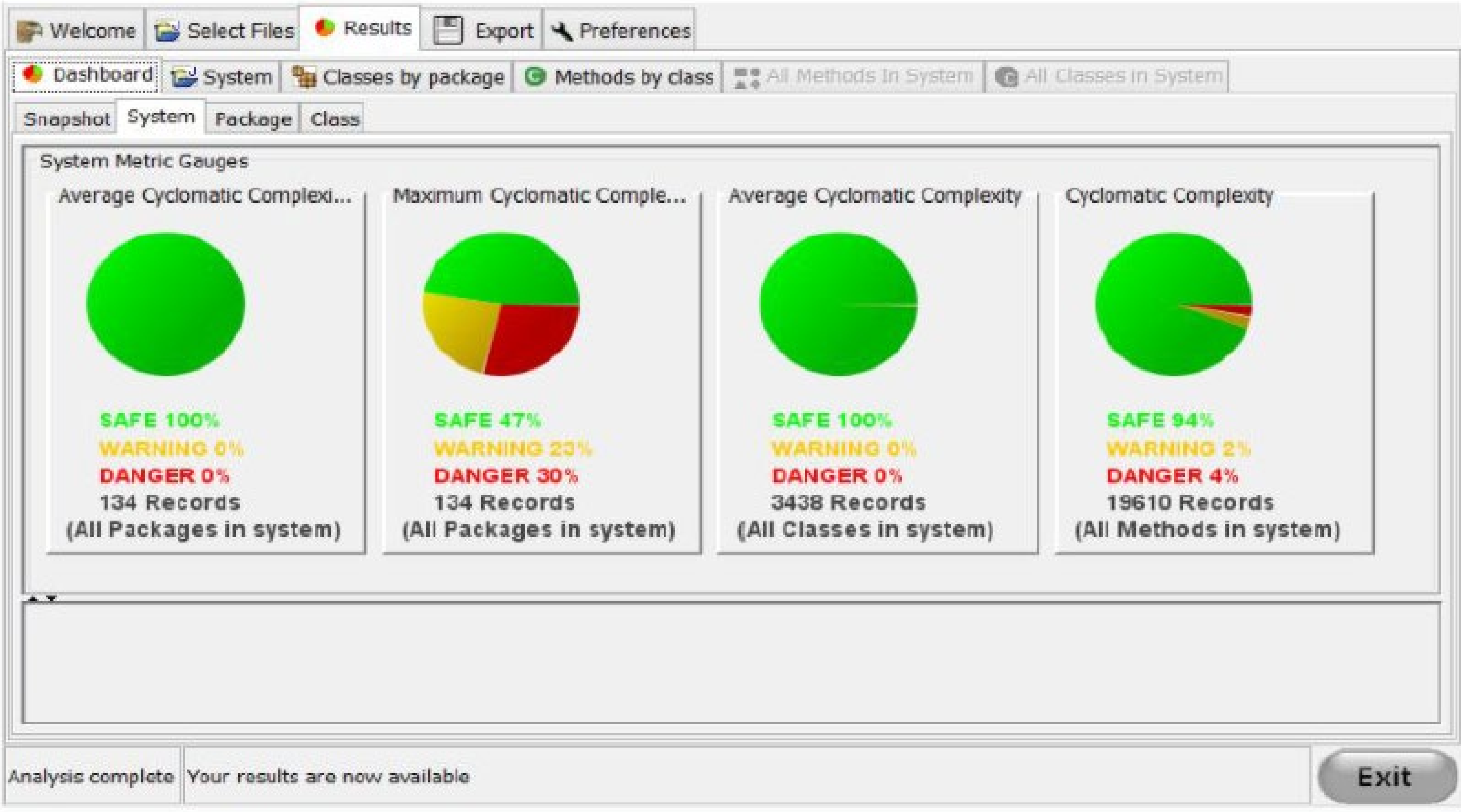
Exploration and slides by Bagus Enggar Tiasto

JHawk – Input Java Files



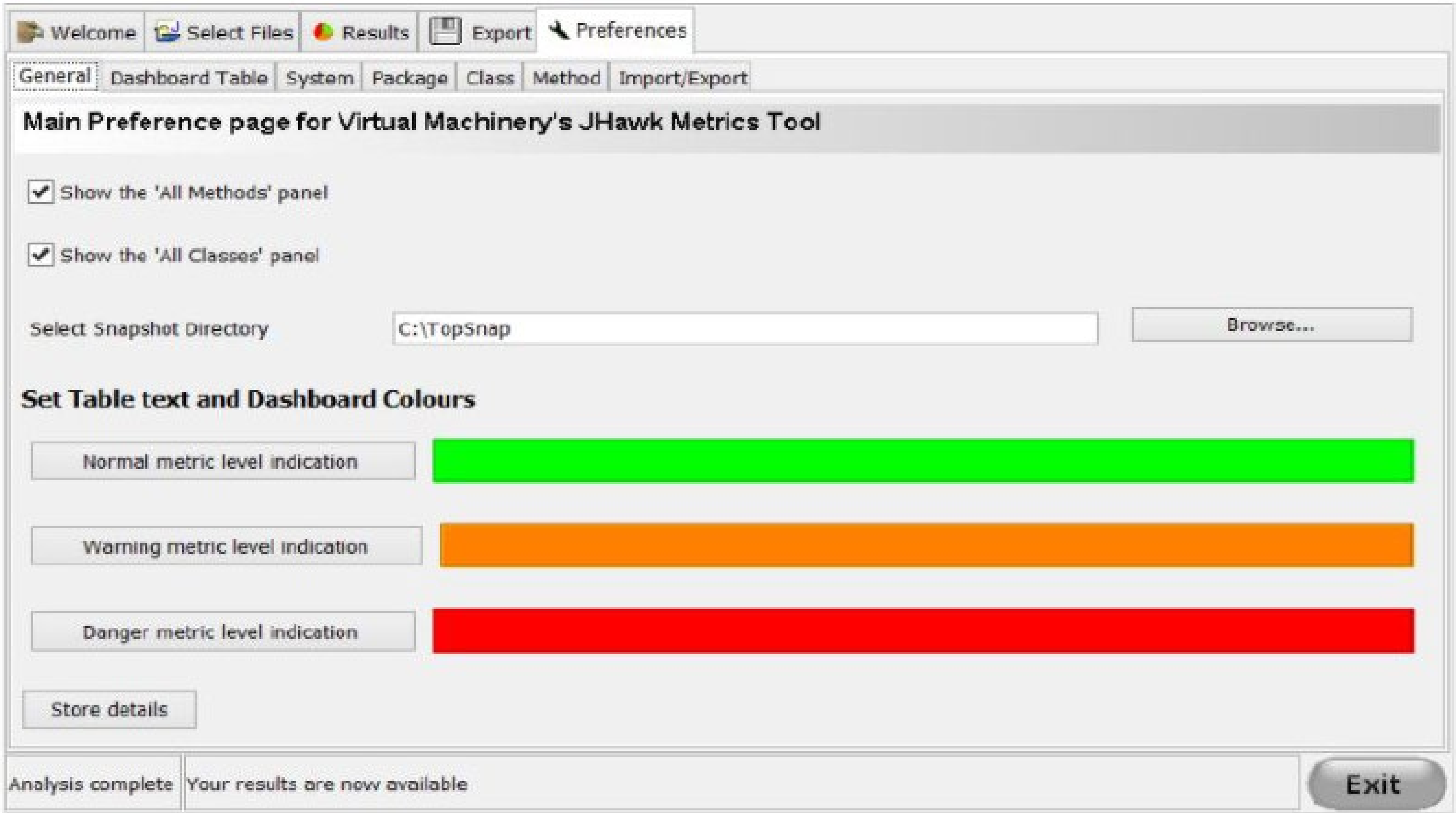
Exploration and slides by Dimas Aditia P

JHawk – Sample Results



Exploration and slides by Dimas Aditia P

JHawk – Sample Preferences



Exploration and slides by Dimas Aditia P

Radon

Welcome to Radon's documentation!

build passing | coverage 90% | pypi v4.5.0 | downloads 111k/month | format wheel | license MIT

Radon is a Python tool which computes various code metrics. Supported metrics are:

- raw metrics: SLOC, comment lines, blank lines, &c.
- Cyclomatic Complexity (i.e. McCabe's Complexity)
- Halstead metrics (all of them)
- the Maintainability Index (a Visual Studio metric)

Radon can be used either from the command line or programmatically through its API.

Exploration and slides by Ihsan Muhammad Asnadi

Radon – Sample Metrics

```
my_first_calculator.py/my_first_calculator.py
    LOC: 20822
    LLOC: 20816
    SLOC: 20816
    Comments: 4
    Single comments: 2
    Multi: 0
    Blank: 4
    - Comment Stats
        (C % L): 0%
        (C % S): 0%
        (C + M % L): 0%
```

Size and comments metrics

Halstead metrics

```
my_first_calculator.py/my_first_calculator.py:
    h1: 3
    h2: 31271
    N1: 41618
    N2: 93640
    vocabulary: 31274
    length: 135258
    calculated_length: 466960.1423920738
    volume: 2019763.9085143406
    difficulty: 4.491701576540565
    effort: 9072176.732113596
    time: 504009.8184507553
    bugs: 673.2546361714469
```

Exploration and slides by Ihsan Muhammad Asnadi