340CT
Software Quality and Process Management

Dr. Yih-Ling Hedley
Email: aa0817@coventry.ac.uk

Requirements Metrics 1

- **Function-based metrics**:
  - use the **function point** as a measure of the "size" of the specification

A **function point**: a unit of measurement to express the amount of business functionality a software system provides to a user, such as data input screen, a report etc.

Requirements Metrics 2

- **Specification Metrics**: used as an indication of quality by measuring **number of requirements** by type
- **Specificity =**

$$n_{ui} / n_r$$

- **Where n_{ui} : the number of functional requirements that are interpreted identically by all reviewers**

  n_r : the total number of requirements (both functional and non-functional) in the requirements document

Activity

- Q: What are the examples of non-functional requirements that can be used in specification metrics?

- Non-functional requirements are for example:
  - ✓**Completeness, Correctness, Understandability, Consistency, …**

Design Metrics

- Architectural design metrics
  - **Structural complexity = f(fan-out)**
  - **Data complexity = f(input & output variables, fan-out)**
  - **System complexity = f(structural & data complexity)**

Fan-out: number of calls by other methods within the object and calls by external methods

Fan-in: number of calls to other methods within the object and calls to external methods

Component-Level Design Metrics

- **Cohesion** (Bieman & Ott, 1994)
  data slices (i.e. a backward walk through a module that looks for data values affecting the module location from which the walk began) , **glue tokens** (variables lie on data slices), **stickiness** (i.e. binding of number of data slices by the glue token)

  - Q: "What % of tokens (i.e. variables) in the module are relevant to all data slices?" The higher the percentage in relevance, the more cohesive the module is.

Activity

- Q: What would be measured with the question, "How many input/output parameters with access to global data, or external calls to a module?"

- A: **Coupling** as this metric examines **data and control flow, global access or external calls to the program** (Dhama, 1994)

Activity

- Q: What would be measured with the question, "How many independent paths within a block of program?"

- A: **Complexity** (e.g. McCabe's Cyclomatic Complexity, a numerical measure of the complexity of the flow of control of a sequential module)

Size-Oriented Metrics 1

- **Size-Oriented Metrics:** measure the length of code using **Number of Lines of Code (NLOC or LOC)** or **Delivered Source Instructions (DSI)** (Lines of Code: the measure was first proposed when programs were typed on cards with one line per card)
  - Errors per KLOC
  - Defects per KLOC
  - £ per KLOC
  - £ per documentation page
  - documentation pages per KLOC
  - errors per person-month
  - LOC per person-month

DSI generally includes delivered support software but excludes non-delivered support software such as test drivers. However, if the latter is developed with their own reviews, test plans, documentation, etc., then they should be included. The "source instructions" include all program instructions but excludes comments and unmodified utility software.

Size-Oriented Metrics 1.1

- Lines of code are easily counted, but
  - LOC **not necessarily related to quality**
  - **Programming languages differ** widely in LOC per functional requirement
  - **dependent on the programming language and programmer** (e.g. may penalise well-designed but short programs)
  - Source code delivery only a small part of the total development effort.

Size-Oriented Metrics 1.2

- What LOC cannot measure: e.g.
  - **People** factors
    - ✓**Team size**
    - ✓**Skills**

Activity

- Q: What else LOC cannot measure?

- A: LOC also cannot measure the following:
  - ✓**Process factors (techniques, tools)**
  - ✓**Product factors (reliability, performance)**
  - ✓**Resource factors (people, tools)**

Function-Oriented Metrics: Function Points 1

- Function points are typically used to:
  - **Estimate the cost and effort** required to design, code and test a software system;
  - **Predict the number of errors**;
  - **Predict the number of components** in the software.

YL Hedley

Function-Oriented Metrics: Function Points 2

- Function points: Advantages
  - **Programming language independent**
  - Used readily **countable characteristics** early in the software process
  - Makes it easier to **measure the impact of reusable components**

YL Hedley

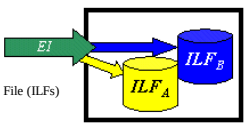
Function Points 1

- Based on the **amount of functionality**
- A **weighted estimate** of
  - **External inputs (EIs)**
  - **External outputs (EOs)**
  - **Internal logical files (ILFs)**
  - **External interface files (EIFs)**
  - **External inquiries (EIs)**

X Fei & YL Hedley

Function Points 1.1

- **External inputs (EIs)**
  - gives the user the capability to maintain the data in ILF's through adding, changing and deleting its contents.
  - Examples: **Data entry by users; Data or file feeds by external applications.**



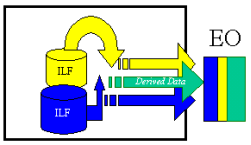
Note: Internal Logical File (ILFs)

X Fei & YL Hedley

Source: http://www.softwaremetrics.com/fpafund.htm

Function Points 1.2

- **External outputs (EOs))**
  - Gives the user the ability to produce outputs. The results displayed are derived using data in ILF's that is maintained and data that is referenced.
  - Examples: **Reports** created by the application being counted, where the reports include derived information.



X Fei & YL Hedley

Function Points 1.3

- **Internal logical files (ILFs)**
  - Logical groupings of data in a system, maintained by an end user
  - Example:
    - **Tables** in a relational database.
    - **Flat files**.
    - **Application control information**.

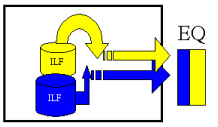
X Fei & YL Hedley

Function Points 1.4

- **External Interface Files (EIFs)**
  - Groupings of data from another system that are used only for reference purposes.
  - The data resides entirely outside the application and is maintained by another application.
  - The external interface file is an internal logical file for another application. The data that an application will **use/reference, but data that is not maintained by that application**

Function Points 1.5

- **External inquiries (EQs)**
  - A direct retrieval of information contained on the files. The input process does not update any Internal Logical Files, and the output does not contain derived data.
  - Examples: **Reports** created by the application being counted, where the report does not include any derived data.



Function Points 2

Function Point Table

Number of FPs	Complexity		
External user type	Low	Average	High
External input type	3	4	6
External output type	4	5	7
Logical internal file type	7	10	15
External interface file type	5	7	10
External inquiry type	3	4	6

Function Points 3

- Used to estimate LOC
  - **LOC = AVC * number of function points**
  - AVC (**Average LOC**):
    - language-dependent factor varying from 200-300 for assemble language to 2-40 for a 4GL
- Limitation: subjective

Object-Oriented Metrics 1

- Chidamber and Kemerer
  - **Weighted Methods Per Class** (WMC)**:** Size metric
  - **Response For A Class** (RFC)**:** Size metric
  - **Coupling Between Object Classes** (CBO)**:** Coupling metrics
  - **Lack Of Cohesion In Methods** (LCOM)**:** Cohesion metrics
  - **Depth Of The Inheritance Tree** (DIT)**:** Inheritance metrics
  - **Number Of Children** (NOC)**:** Inheritance metric

Source: Pressman, R.S., Software Engineering: a Practitioner's Approach

Object-Oriented Metrics 1.1

- **Weighted Methods Per Class (WMC)**
  - Measures the **number of methods** in each class, **weighted by the complexity of each method**.
  - A simple method may have a complexity of 1, and a large and complex method a much higher value. The larger the value for this metric, the more complex the object class.

Source: Pressman, R.S., Software Engineering: a Practitioner's Approach

Object-Oriented Metrics 1.2

- **Coupling Between Object Classes (CBO)**
  - Measures how much coupling exists between classes when methods in one class use methods or instance variables defined in a different class.
  - A **high value indicates that classes are highly dependent,** and therefore it is more likely that changing one class will affect other classes in the program. As CBO increases, the reusability of the class decreases. CBO should be kept as low as possible since it complicates modifications.

Source: Pressman, R.S., Software Engineering: a Practitioner's Approach

Object-Oriented Metrics 1.3

- **Depth Of The Inheritance Tree (DIT)**
  - Measures the **number of discrete levels in the inheritance tree** where subclasses inherit attributes and operations (methods) from superclasses.
  - The **maximum length from the node to the root of the tree**. The deeper the inheritance tree, the more complex the design.
  - As DIT grows, it becomes difficult to predict the behaviour of a class, but at the same time many methods may be reused.

Source: Pressman, R.S., Software Engineering: a Practitioner's Approach

Object-Oriented Metrics 1.4

- **Number Of Children (NOC)**
  - Measures the **number of immediate subclasses in a class**.
  - Measures the **breadth of a class hierarchy**, whereas DIT measures its depth.
  - A **high value for NOC may indicate greater reuse**. It may mean that more effort should be made in validating base classes because of the number of subclasses that depend on them.

Source: Pressman, R.S., Software Engineering: a Practitioner's Approach

Object-Oriented Metrics 1.5

- **Response For A Class (RFC)**
  - Measures the number of methods that could potentially be executed in response to a message received by an object of that class.
  - Related to complexity. The higher the value for RFC, the more complex a class and hence the more likely it is that it will include errors. As RFC increases testing effort increases.

Source: Pressman, R.S., Software Engineering: a Practitioner's Approach

Object-Oriented Metrics 1.6

- **Lack Of Cohesion In Methods (LCOM)**
  - Calculated based on the **pairs of methods in a class**.
  - Calculating the difference between the **number of method pairs without shared attributes** and the **number of method pairs with shared attributes**.
  - In case no methods access the same attributes, LCOM is equal to 0. **As LCOM increases, coupling between methods (via attributes) increases** and thus class complexity increases.

Examples: **http://www.literateprogramming.com/ooapply.pdf**

Source: Pressman, R.S., Software Engineering: a Practitioner's Approach