Chapter 5

Basics of Software Metrics

Contents

- Software crisis
- Software metrics
- Types of software metrics
- Object oriented metrics

Software Crisis

- According to American Programmer, 31.1% of computer software projects get canceled before they are completed,
- 52.7% will overrun their initial cost estimates by 189%.
- 94% of project start-ups are restarts of previously failed projects.

Solution

systematic approach to ***software development and measurement***

Software Metrics

- ***Software Metrics***
- Any type of measurement which relates to a software system, process or related documentation
  - Lines of code in a program
  - number of person-days required to implement a use-case
- It refers to a broad range of quantitative measurements for computer software that enable to:
  - improve the software process continuously
  - assist in quality control and productivity
  - assess the quality of technical products
  - assist in tactical decision-making

Measure, Metrics, Indicators

- **Measure**
  - provides a quantitative indication of the extent, amount, dimension, capacity, or size of some attributes of a product or process.
- **Metrics**
  - relates the individual measures in some way.
- **Indicator**
  - a combination of metrics that provide insight into the software process or project or product itself.

Types of Metrics

- **Basic Metrics**
  - **Defects-** A problem found during the review or other than the review of the phase where it was introduced.
  - **Effort-** the time invested (hours/days) in doing some work.
  - **Size-** the size of software in terms of line of code.
- **Derived Metrics**
  - By measuring the three basic metrics we can compute a variety of metrics that we can use to manage the quality.

Eg.

- **Defect injection rate for Project = Total no. of defects injected/size of software produced(KLOC)**

Contd.

E.g.

One programmer written 1000 LOC(Lines of Code) in 8 days and every day he is working for 10 hours. For every 1000 line he introduces 25 defects.

Effort=10*8=80 person hour

Productivity=1000/80= 12.5 LOC/person-hour

OO Design Metrics

Contd..

- **Class Size**-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 operations overridden by a subclass (NOO).**
  - ✓ There are instances when a subclass replaces an operation inherited from its super-class with a specialized version for its own use, this is called ***overriding***.
  - ✓ Large values for NOO generally indicate a design problem.
  - ✓ Since a subclass should be a specialization of its super-classes, it should primarily extend the services [operations] of the super-classes.
  - ✓ This should result in unique new method names.
  - ✓ If NOO is large, the designer has violated the abstraction implied by the superclass.
  - ✓ This results in a weak class hierarchy and OO software that can be difficult to test and modify.

Contd..

- **Specialization Index (SI).** The specialization index provides a rough indication of the degree of specialization for each of the subclasses in an OO system.
- Specialization can be achieved by adding or deleting operations.

$$SI = [NOO * level]/Mtotal$$

- where level *is the level in the class hierarchy at which the class resides*
- *Mtotal is* the total number of methods for the class
- The **higher is the value of SI**, the more likely the **class hierarchy has classes that do not conform to the super-class abstraction**

Contd..

- **Method inheritance factor (MIF).-**The degree to which the class architecture of an OO system makes use of inheritance for both methods (operations) and attributes is defined as,

$$\mathbf{MIF} = \sum Mi(Ci) / \sum Ma(Ci)$$

- where the summation occurs over *i = 1 to TC.*
- *TC is defined as the total number of* classes in the architecture, *Ci is a class within the architecture, and*

$$Ma(Ci) = Md(Ci) + Mi(Ci)$$

- *Ma(Ci) = the number of methods that can be invoked in association with Ci.*
- *Md(Ci)) = the number of methods declared in the class Ci.*
- *Mi(Ci) = the number of methods inherited (and not overridden) in Ci.*
- MIF is [0,1]
- MIF=0 means inheritance mechanism is not used.
- Measure of reuse via inheritance.

Contd..

- **Lack of cohesion in methods (LCOM)-** Each method within a class, **C,** accesses one or more attributes (also called *instance variables*).
- LCOM is the number of methods that access one or more of the same attributes.
- **If no methods access the same attributes**, then **LCOM = 0.**
- **If LCOM is high**, methods may be **coupled to one another** via attributes.
- This increases the complexity of the class design.
- In general, high values for LCOM imply that the class might be better designed by breaking it into two or more separate classes.
- It is desirable to keep cohesion high; that is, keep LCOM low.

Contd..

- **Number of children (NOC)-** The subclasses that are immediately subordinate to a class in the class hierarchy are termed its ***children***.
- As the number of children grows, reuse increases but also, as NOC increases, the abstraction represented by the parent class can be diluted.
- As NOC increases, the amount of testing (required to exercise each child in its operational context) will also increase.

Depth of inheritance tree

- Depth of a class in a class hierarchy determines potential for re-use.
- Deeper classes have higher potential for re-use.
- Maximum height of inheritance tree or level of a particular class in a tree
- Inheritance increases coupling... changing classes becomes harder.
- As DIT grows, it is likely that classes on lower level inherits lots of methods and overrides some.
- Thus predicting behavior for an object of a class becomes difficult.
- Depth of Inheritance (DIT) of class C is the length of the **shortest path** from the root of the inheritance tree to C.
- In case of multiple inheritance DIT is the **maximum length** of the path from the root to C.

Metrics for Maintenance

- **Software Maturity Index (SMI)**

  M_T = number of modules in the current release

  F_c = number of modules in the current release that have been changed

  F_a = number of modules in the current release that have been added

  F_d = number of modules from the preceding release that were deleted in the current release

  $$SMI = [M_T - (F_c + F_a + F_d)] / M_T$$

- **CFD total(Customer-Found Defects)**

  CFD=Number of customer-found defects/Assembly-equivalent total source size

Inspection metrics

- **Primitive Measurements:**
  - Lines of Code Inspected = loc
  - Hours Spent Preparing = prep_hrs
  - Hours Spent Inspecting = in_hrs
  - Discovered Defects = defects
- **Other Measurements:**
  - Preparation Rate = loc / prep_hrs
  - Inspection Rate = loc / in_hrs
  - Defect Detection Rate = defects / (prep_hrs + in_hrs)

Thank You