SOFTWARE METRICS

Quality Assurance for Information Technology

Dávid Gégény - KHIWFS

„You can't manage what you can't control, and you can't control what you don't measure.”

- Tom DeMarco

Software metrics

- Metrics quantify the characteristics of the product or process
- Improvability is important
- Objectivity
- Comparison
- Analysing metrics is part of quality control and quality assurance

Types of software metrics

- Process metrics
  - e.g. mean time to repair
- Product metrics
  - e.g. lines of code, cyclomatic complexity

Process metrics

- Efficiency of development
  - new lines of codes per month
    - should not measure productivity (copy-paste)
  - new function points per month
  - etc.
- Quality of process
  - number of bugs per line of code
  - Mean Time To Repair (MTTR)
  - Mean Time To Failure (MTTF)
  - Mean Time Between Failures (MTBF)
  - Probability of Failure On Demand (POFOD)

Role of metrics

- Risk analysis
- Risk reduction
- Resource and cost estimations
- Software quality determination

Product metrics

- Size metrics
- Inheritance metrics
- Complexity
- Cohesion
- Coupling
- Bad Smell and Cloning
- Quality of Source (e.g. number of fouls)

Size metrics

- Lines of Code (LOC)
  - a connection was assumed with the number of system defects
  - but it is dependent on
    - programming language
    - programming style
    - system type
- Efficient Lines of Code (ELOC)
  - not empty and not comment lines
- Number of Classes (NCL)
- Number of Attributes/Methods (NA/NM)

OO metrics

- these metrics are for object oriented programming
- Depth of Inheritance Tree (DIT)
  - max. length from the node to the root
- Response For a Class (RFC)
  - Number of methods that can be invoked in response to another class
- Specialization Index (SIX)

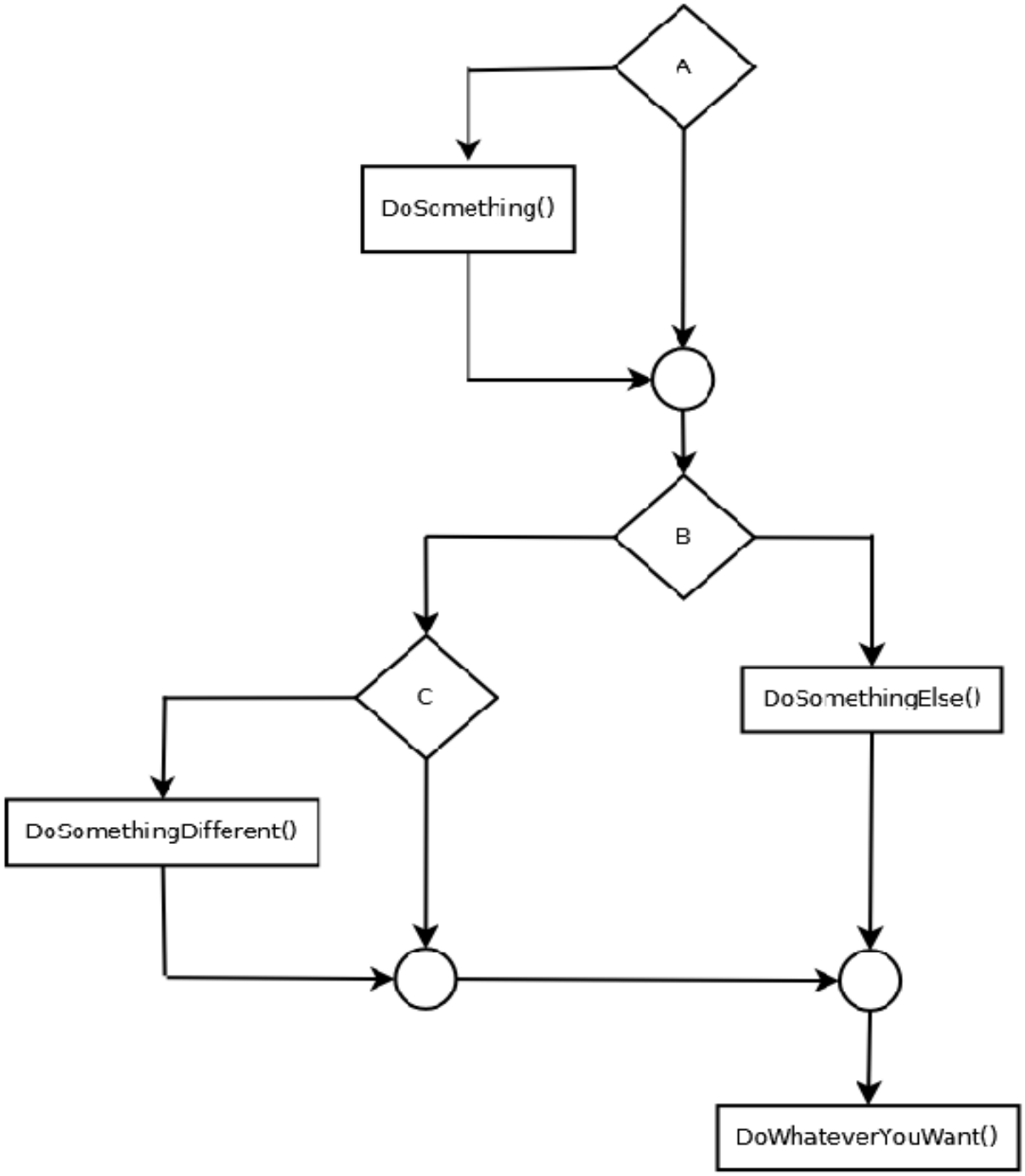
Complexity (McCabe)

- Cyclomatic complexity
- Graph metric
- Used on control flow graph
- E – N + 2P
  - E – number of edges
  - N – number of nodes
  - P – number of graph components (usually one)
- The higher the complexity the harder it is to test, change or understand the code
- Time and space complexity are also important („O” notation)

```
if (A)
{
    DoSomething();
}

if (B)
{
    if (C)
    {
        DoSomethingDifferent() { }
    }
}
else
{
    DoSomethingElse();
}

DoWhateverYouWant();
```



No. of Nodes: 10
No. of Edges: 12
No. of components: 1

Cyclomatic complexity:
$12 - 10 + 2 \cdot 1 = 4$

WMC

- Weighted Methods per Class
- Sum of the McCabe complexity of encapsulated methods
- A high value might point to a design flaw

LCOM

- Lack of Cohesion on Methods
- Measures how much the methods of a class are connected to each other
- A high value means better encapsulation
- Low value might suggest design flaws and high complexity

CBO

- Coupling Between Object classes
- How much classes are connected (e.g. through methods)
- Number of classes the class uses (through methods, attributes or inheritance)
- High value means
  - poor encapsulation
  - high number of defects
  - poor testability
  - sensitivity to change
- Really high correlation with the number of defects

Fan-in, fan-out

- Fan-in – number of calling modules
- Fan-out – number of called modules
- Henry and Kafura complexity
  - Complexity = Length • (fan-in • fan-out)^2
- Card and Glass complexity
  - structural: S(i) = fan-out^2(i)
  - data complexity: D(i) = v(i) / (fan-out(i) + 1)
    - v(i) – number of I/O parameters
  - system complexity: C(i) = S(i) + D(i)

Clone metrics

- „Copy-paste usage”
- The code is harder to change or understand
- Useless code parts might come up
- CCL – Clone Classes
- CI – Clone Instances
- CC – Clone Coverage
  - percentage of source code identified as clone
  - high CC means more errors, lower understandability

```
static void Main(string[] args)
{
    //declare array f as double array and
    //assign initial values to double arrays a, b, c, d, e

    for (int i = 0; i < a.Length; i++)
    {
        f[i] = 2 * a[i] * a[i] + 3;
    }
    DoSomething(f);

    for (int i = 0; i < b.Length; i++)
    {
        f[i] = 2 * b[i] * b[i] + 3;
    }
    DoSomething(f);

    for (int i = 0; i < c.Length; i++)
    {
        f[i] = 2 * c[i] * c[i] + 3;
    }
    DoSomething(f);

    for (int i = 0; i < d.Length; i++)
    {
        f[i] = 2 * d[i] * d[i] + 3;
    }
    DoSomething(f);

    for (int i = 0; i < e.Length; i++)
    {
        f[i] = 2 * e[i] * e[i] + 3;
    }
    DoSomething(f);
}
```

```
static double[] CalculateFunction(double[] array)
{
    double[] result = new double[array.Length];

    for (int i = 0; i < array.Length; i++)
    {
        result[i] = 2 * array[i] * array[i] + 3;
    }

    return result;
}

static void Main(string[] args)
{
    //declare array f as double array and
    //assign initial values to double arrays a, b, c, d, e

    f = CalculateFunction(a);
    DoSomething(f);

    f = CalculateFunction(b);
    DoSomething(f);

    f = CalculateFunction(c);
    DoSomething(f);

    f = CalculateFunction(d);
    DoSomething(f);

    f = CalculateFunction(e);
    DoSomething(f);
}
```

Aggregated metrics (ISO/IEC 9126)

- Functionality
- Reliability
- Usability
- Efficiency
- Managability
- Portability

Analysis

- Evaluated metrics should be analysed
- Baseline values are needed
  - Average of metrics of a large number of samples
- May require an expert

Bad Smell

- Data class (only members, maybe getters, setters)
- Feature envy (focus is on the members of another class)
- Large class
- Lazy class (parents, children or callers do all the work)
- Long method
- Long parameter list

Thank you for your attention!