# Metrics

Taken from material by

Michael L. Collard

# Definitions

- A **measure** is quantitative indication of extent, amount, dimension, capacity, or size of some attribute of a product or process.
  - E.g., number of faults

- A **metric** is quantitative measure of degree to which a system, component or process possesses a given attribute
  - E.g., number of faults the system experiences in a year of operation

# Why Measure Software?

- Determine the quality of the current product or process
- Predict qualities of a product/process
- Improve quality of a product/process

# Examples

- Estimate the cost & schedule of future projects
- Evaluate the productivity impacts of new tools and techniques
- Establish productivity trends over time
- Improve software quality
- Forecast future staffing needs
- Anticipate and reduce future maintenance needs

# Metric Classification

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

# Product vs. Process

- **Process metrics**
  - Insights of process paradigm, software engineering tasks, work product, or milestones
  - Lead to long term process improvement
- **Product metrics**
  - Assesses the state of the project
  - Track potential risks
  - Uncover problem areas
  - Adjust workflow or tasks
  - Evaluate teams ability to control quality

# 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 and Measures

- **Measures**:
  - Size of the software produced
  - LOC - Lines Of Code
  - KLOC - 1000 Lines Of Code
  - SLOC – Statement Lines of Code
    - Ignore whitespaces, …
- Typical **metrics**:
  - Errors/KLOC, defects/KLOC, cost/LOC, documentation pages/KLOC

# Metrics of Software Quality

- **Correctness**: degree to which a program operates according to specification
  - Defects/KLOC
    - Defect is a verified lack of conformance to requirements
  - Failures/hours of operation
- **Maintainability**: degree to which a program is open to change
  - Mean time to change
  - Cost to correct
- **Integrity:** degree to which a program is resilient to attacks
  - Fault tolerance, security & threats
- **Usability**: easiness to use
  - Training time, skill level necessary to use, increase in productivity, obtained through subjective questionnaire or controlled experiment

# McCabe's Complexity Metrics

- 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
    - That is, one or more code statements
  - Edges represent control flow between nodes
- **Cyclomatic complexity**: set of independent paths through the graph

# Chidamber and Kemerer Metrics

- Weighted methods per class (WMC)
- Depth of inheritance tree (DIT)
- Number of children (NOC)
- Coupling between object classes (CBO)
- Response for class (RFC)
- Lack of cohesion metric (LCOM)

# WMC: Weighted Methods per Class

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

- ..where $c_i$ is the complexity (e.g., volume, cyclomatic complexity, etc.) of each method, n is the number of methods in a class

- The number of methods and complexity of methods is an indicator of how much time and effort is required to develop and maintain a class
- The larger the number of methods in a class, the greater the potential impact on the subclasses
- Classes with a large number of methods are likely to be more application specific, limiting the possible reuse

# DIT: Depth of Inheritance Tree

- Maximum length from a node to the root
  - In Java, we only typically consider the first root developed within the project
- Lower level subclasses inherit a number of methods making behavior harder to predict
- Deeper trees indicate greater design complexity

# NOC: Number of Children

- Number of subclasses immediately subordinate to a class

- 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

# CBO: Coupling between Objects/Classes

- Number of collaborations between two classes
  - That is, the fan-out of a class C, which is the number of other classes that are referenced in C
  - A reference to another class A, is a reference to a method or a data member of class A

- 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

# RFC: Response for a Class

- Number of methods that could be called in response to a method invocation to a class
  - …local and remote

- As RFC increases
  - Testing effort increases
  - The greater the complexity of an object, the harder it is to understand what's going on

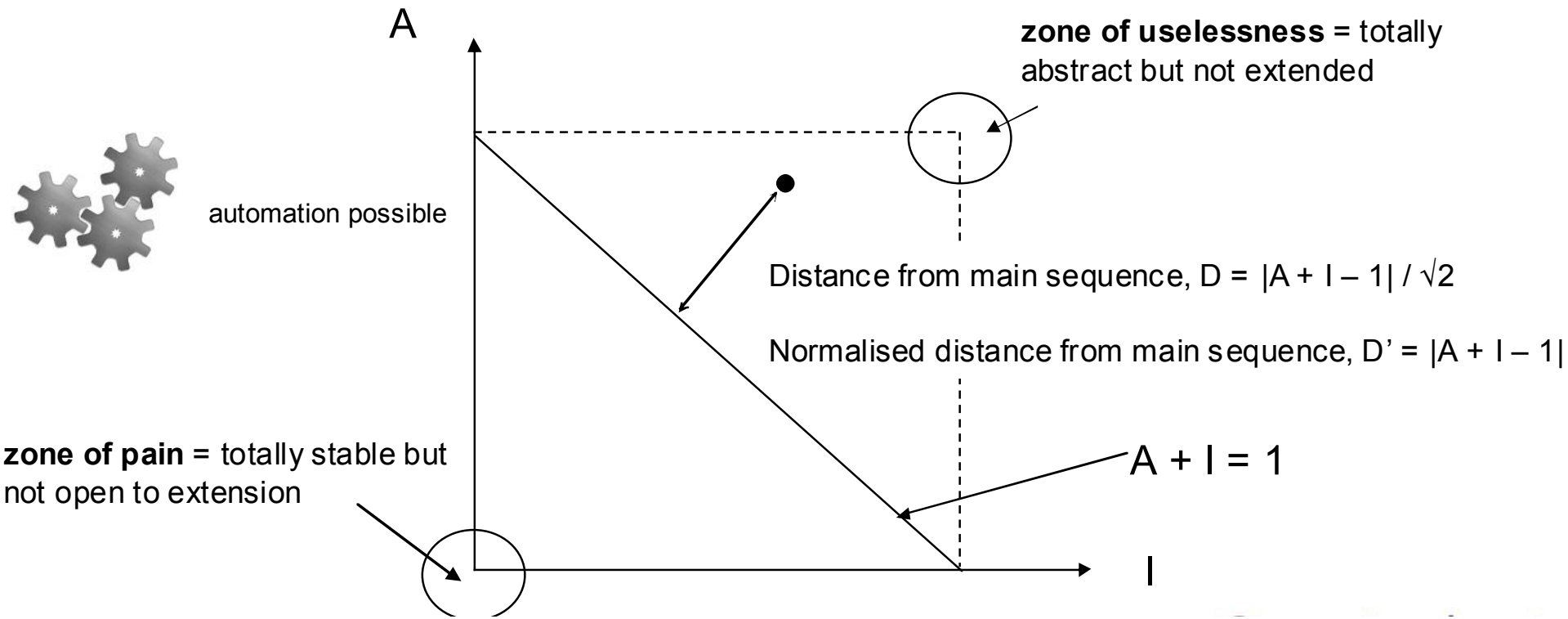
# LCOM: Lack of Cohesion in Methods

- Number of pairs of methods in a class that don't have at least one field in common minus the number of pairs of methods in the class that do share at least one field
- When this value is negative, the metric value is set to 0
- Lower metric values represent “better” situations

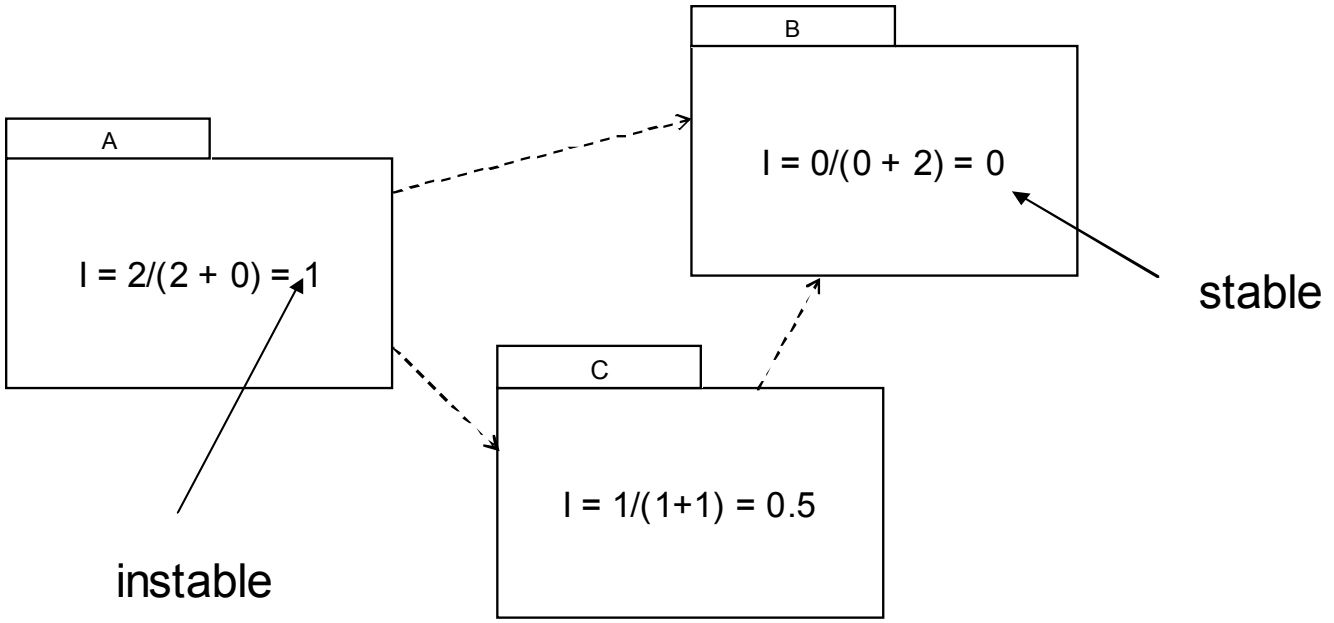
# Misure e metriche di stabilità

- CC: numero di classi concrete in un package P
- AC: numero di classi astratte o di interfacce in un package P
- Ca (afferent coupling): numero di package che dipendono da classi del package P
- Ce (efferent coupling): numero di package da cui dipendono le classi del package P
- A (Astrattezza): percentuale di classi astratte del package P, definita come il rapporto AC / (AC + CC)
- I (Instabilità): rapporto fra l'accoppiamento efferente e l'accoppiamento totale Ce / (Ce + Ca)
  - È un indicatore della difficoltà di modificare il package P
  - Se infatti è alto, ciò significa che molti package sarebbero interessati
- D (Distanza dalla sequenza principale): distanza del package P nel piano A,I dalla retta di equazione A + I = 1
  - E' un indicatore del compromesso raggiunto dal package fra astrattezza (A = 1, I = 0) e instabilità (A = 0, I = 1)

# Diversi casi



# Cosa fare?




- Giochiamo nel piano:
  - Spostare delle classi concrete in package instabili
  - Rendere astratte classi in package stabili
  - …