SOFTENG 701:
Advanced Software Engineering Development Methods
Part 2

Lecture 2a: Quality Attributes

Ewan Tempero
Department of Computer Science

Agenda

- Admin
  - change due dates for assignments?
- Part 1: Quality Attributes and how to “measure” them
- Part 2: Advice for designing reusable classes

  Ralph E. Johnson and Brian Foote *Designing Reusable Classes* Journal of Object-Oriented Programming June/July 1988, Volume 1, Number 2, pages 22-35

Previously in SOFTENG701

- A programming language is a tool that can be used to "write a program" — that is, produce "source code"
- Source code by itself is of no value to the client—to be of value the code must be executed
- From the client's point of view, the program's value is determined by the degree to which it meets the requirements the client has (and hopefully conveyed to the developer)
- $\Rightarrow$ the client does not care about the choice of language, or even the "quality" of the program
- So who cares about program quality?

$\Rightarrow$ developer (more generally, development team)

Why do developers care about quality?

- “quality” typically means “be more efficient”
- e.g. improving quality means
  - produce code cheaper
  - produce code faster
  - produce code with less effort
  - produce code with fewer defects
  - deploy products faster
  - respond to change requests faster
  - fix faults faster

What might “Quality” mean?

- Extensibility
- Performance
- Security
- Understandability
- Readability
- Comprehensibility
- Modifiability
- Maintainability
- Portability
- Buildability
- Scaleability
- Availability
- Reliability
- . . .

Quality Attributes

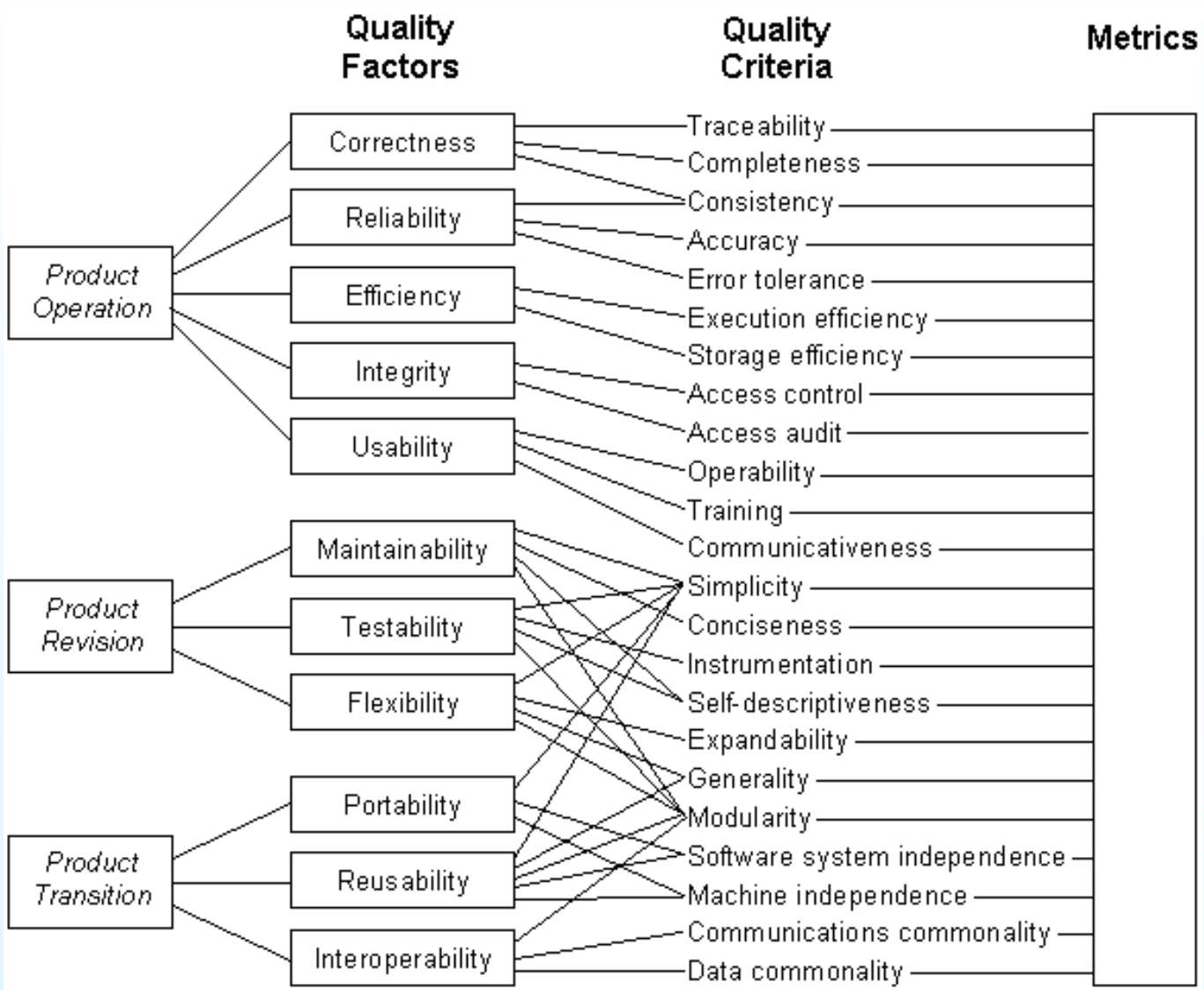
- quality — some notion of how "good" the product is $\Rightarrow$ "satisfies the needs of its stakeholders"
- A **quality attribute** is a [quantifiable] or testable property of a system that is used to indicate how well the system satisfies the needs of its stakeholders

> Bass, Clements, and Kazman (2013) "Software Architecture in Practice" (3rd ed) Addison-Wesley

Defining Quality

- There have been various attempts to define “quality” in general, and quality attributes in particular, over the years
  - McCall’s software quality model (1977)
  - Boehm’s software quality model (1978)
  - ISO/IEC 9126 (1991,2001)
  - ISO/IEC 25010 (2011)

McCall's Software Quality Model



Defining Quality Attributes: Maintainability (and related)

McCall the effort required to locate and fix a fault in the program within its operating environment.*

IEEE Standard Glossary of Software Engineering Terminology , 1990 the ease with which software can be maintained, enhanced, adapted, or corrected to satisfy specified requirements*

ISO/IEC 9126/25010 The degree to which the software product can be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications*

Boehm's Quality Model Maintainability is composed of Testability, Understandability, and Modifiability*

Random Web Source Modifiability encompasses two aspects: Maintainability and Flexibility*

Bass, Clements, Kazman (Modifiability) "a system is typically modifiable if changes involve the fewest possible number of distinct elements"

ISO/IEC 25010 (Modularity) The degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.*

(* quotes not necessarily dead accurate)

Measuring Quality Attributes

- Quality attributes are considered (in the measurement theory sense) external attributes — they cannot be measured independent of their environment
  - E.g. “Maintainability” — some modifications take more time than others ⇒ without knowing what modification is needed, cannot measure “degree to which the software product can be modified
- Much more of a problem, quality attributes are constructs
- A construct is a concept that is *quantifiable in principle* but *not directly measurable*

  ⇒ speaking of “measuring” maintainability (etc) makes not sense
- Instead, constructs are inferred from reflective indicators, direct measurements that generally correlate with our understanding of the construct
- A reasonable starting point is that each construct have three indicators

  ⇒ any claim that a quality attribute (construct) is measured by a single indicator is not to be trusted

Example: Maintainability and indicators

- A commonly-mentioned way to measure maintainability is “number of lines of code changed”
- But:
  - some lines are easier to change than others
  - “lines changed” usually treats “lines added”, “lines deleted”, and “lines modified” as being equal

  $\Rightarrow$ does not measure maintainability
- may be a reasonable indicator, but being only one, is not by itself sufficient to infer maintainability reliably

Example: Reusability

- Many supporters of “object-oriented design” offer reasons such as “improves reusability”

  B. Meyer, ”Reusability: The Case for Object-Oriented Design,” in IEEE Software, vol. 4, no. 2, pp. 50-64, March 1987.
  `http://dx.doi.org/10.1109/MS.1987.230097`

- Many discussions of reusability don’t actually give a definition of reusability. . .

  Meyer, again.

- . . . or give definitions that make no sense

  `https://en.wikipedia.org/wiki/Reusability`

- Definition 1: the ease with which a component can be used in a software system for which it was not originally developed for.

  Constructed by Ewan from hints and implications in various sources.

- Definition 2: “the degree to which a software module or other work product can be used in more than one computer program or software system”

  IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990

Measuring Reusability

- Which of these classes is more reusable?

```
// Version 1 (stores Object)
public class ArrayList { ... }
```

```
// Version 2 (generic)
public class ArrayList<T> { ... }
```

- “measurement” for reusability of component — “number of programs the component can be used” (indicating “degree”)

Measuring Reusability

- Which of these classes is more reusable?

```
// Version 1 (stores Object)
public class ArrayList { ... }
```

```
// Version 2 (generic)
public class ArrayList<T> { ... }
```

- "measurement" for reusability of component — "number of programs the component can be used" (indicating "degree")
- Version 1 can hold values of types that are unrelated, whereas Version 2 can only hold values of subtype of the instantiated type
  $\Rightarrow$ Version 1 can be use in more programs
  $\Rightarrow$ is more reusable?

Measuring Reusability

- "measurement" for reusability of component — effort taken to reuse the component
- (*how is effort measured — time taken until sure fault free? characters typed?*)
-

```
public void displayList(ArrayList list) {
  for (Object obj: list) {
    MyType value = (MyType)obj;
    value.display();
  }
}
```

```
public void displayList(ArrayList<Displayable> list) {
  for (Displayable value: list) {
    value.display();
  }
}
```

Measuring Reusability

- "measurement" for reusability of component — effort taken to reuse the component
- (*how is effort measured — time taken until sure fault free? characters typed?*)
-

```
public void displayList(ArrayList list) {
  for (Object obj: list) {
    MyType value = (MyType)obj;
    value.display();
  }
}
```

```
public void displayList(ArrayList<Displayable> list) {
  for (Displayable value: list) {
    value.display();
  }
}
```

Version 2 cannot have ClassCastException, so don't need to spend time checking for that possibility (or debugging when it actually arises) *and* requires fewer characters

$\Rightarrow$ Version 2 takes less effort

$\Rightarrow$ is more reusable?

Quality Tradeoffs

- Some quality attributes conflict. E.g
  - Rule of thumb: a component reusable is about 3 times more expensive (e.g. time to construct) than its non-reusable counterpart
  - A design with high “buildability” (time taken from beginning construction to deployment) is likely to have low reusability
  - Reusable components tend to have many parameters of different kinds and so seem to be harder to understand
- $\Rightarrow$ cannot satisfy all quality attributes all of the time.

Key Points

- There are many views of what quality means, and different kinds of quality.
- Whatever we think design quality may be, it will be difficult to measure.
- Yet to understand whether a design is “good”, or what exactly design advice is improving, requires some form of observation (i.e. measurement)
- You can’t measure what you don’t understand