SENG 301

David Ng

Fall 2016

Contents

1 September 13, 2016 **2**
1.1 Software Pedagogy 2
1.2 Activities in Development 2

2 September 15, 2016 **3**
2.1 Review of Object Oriented Programming 3

3 September 20, 2016 **4**
3.1 Modelling 4
3.2 Software Models 5
3.3 Model Considerations 5
3.4 Unified Modeling Language (UML) 5
3.5 Views and Use of Diagrams 6
3.6 Structural Modeling 6
3.7 Class Relationships 6

4 Paragraph **7**

5 Math **7**
5.1 Using alignment 7

6 Shortcuts **8**

7 Theorems and Proofs **8**

8 Referencing **8**

9 Numbered and Bulleted Lists **9**

1 September 13, 2016

1.1 Software Pedagogy

Software engineering is a disciplined approach to developing software in the real world. It is the application of a disciplined approach to the development, operation, and maintenance of software, and the study of these approaches.

In real software development, we are doing more than just implementing a data structure. Scale also matters (time, lines of code, developers, market). Change is a constant reality, and conflicting goals/ideas lead to messy situations. ”Software that is used is software that is changed.” Every system has bugs in it. Source code is a means to communicate with other people.

Traditionally, software engineering revolves around the following:

1. Models
2. Activities
3. Process (ordering and interweaving of activities)

1.2 Activities in Development

1. What should be done? One should read description and talk to people. Also, write down the needs in some manageable form. These are known respectively as requirements elicitation (or gathering) and requirements specification. We also need to consider whether anything has been missed. Is the project realistic or self-contradictory? This is referred to as requirements analysis.
2. How should it be done? This is a problem of design.
3. Do it. This is the implementation of the system.
4. Has it actually been done? This is the testing stage.

Additionally, the principles of deployment, maintenance and enhancement, management, configuration, and communication (including documentation) all need to be considered.

Software development is not algorithmic. This is due to the uncertain nature of the work, and well as the need to balance external concerns. For instance, time, money, people, technical issues,, business issues, and changing environments. However, software development can still be disciplined. For instance, the identification and application of best practices, and the decision making of when and how to apply these.

2 September 15, 2016

2.1 Review of Object Oriented Programming

Definition. Constructor has same name as class, and no return type. The constructor object is created with the new keyword.

(Two new keywords indicate two objects created). When out of scope, garbage collection cleans up memory. FileReader is created first, then Parser is created since it is an argument. Order of initiation or arguments not defined. Infinite objects are created. Each iteration, two objects are created.

Null is not an object, 2 objects have to exist.

There are no explicit objects. They are implied by variables (p).

1 class called script processor.

String, IVendingMachineFactory, FileNotFoundException, ParseException, Parser, FileReader.

Types are classes, interfaces, primitives, and arrays. That is, the number of classes with boolean. We note Java is a strongly typed programming language.

path is a local variable, path is an argument in FileReader, path, factory and debug are formal parameters.

p is used when methods are called.

p is null??????????. Calling methods may lead to crash.

We note that path, factory, and debug are variables. Java, reference types all pointer, not actual object. **Objects** are conceptual entities that act as metaphors for real life objects. They hold an individual identity, but this concept breaks down if you push too hard.

Objects are discrete, whereas the real world is largely continuous. Choosing objects to use in software is an important choice.

Remark. The source code does not mention specific objects. "new" when executed, indicates that an object should be created.

Definition. We can group objects into categories, thus allowing us to **classify** objects which have some common properties. Equivalently, a **class** consists of all objects that possess certain kinds of properties.

Example. *If class Student has property StudentID, then only objects with this property can be in this class.*

Example. *If an object has property StudentID, this may not suffice to be in the class Student.*

Remark. We note that an object can be in principle, an **instance** of a class. In fact, an object can be an instance of several classes at once. A class is **instantiated** to create an object. In Java, a class-oriented language, we are not allowed to change the class of an object. Additionally, an object may not be an instance of more than one class in Java.

Definition. **Reference types** are things that can be pointed to.

Example. *Classes, interfaces, enumerations, annotation types, and array are examples of reference types.*

Definition. **Primitive types** contain no references or pointers to the above.

Example. *char, int, and boolean are examples of primitive types.*

Definition. Class properties are realized as **fields** in Java. In C++, they are called **member variables**.

Each instance of a class typically contains its own copy of the (instance) fields defined by its class. Fields can also be shared across all instances. For instance, class, fields, and static fields. We note that the driver declares no fields, but this does not mean it has no fields (It may have inherited fields).

Formal Parameters of methods and constructors are variables with a scope only within that method or constructor. Local variables can also be defined, which have a scope only from the declaration until the end of the enclosing scope. We note that all variables have a type in Java.

Definition. The value of all fields in an object constitutes its current **state**.

A program's state includes the state of all its objects and the state of its variables. For some programs, external data is also relevant to the state. We note that state only exists at specific moments of runtime.

3 September 20, 2016

3.1 Modelling

Definition. Every **model** is an idea or set of ideas about something. A model often discards unimportant details, while emphasizing the important details. Models must be represented through notations, words, images, etc.

Example. *We can consider a commercial airliner. This is a huge and complex machine. We learn the basic principles upon which it is built (mechanics, electronics, software). We need to figure out what we want to know about it, what features affect maneuverability, and how to represent that information to inform others.*

We are also concerned with the level of abstraction with the amount of detail included. How much detail should be shown. What should the form of the model be? We must consider what our purpose is in creating a particular model. Is it a simple presentation for non-technical customers, is it a general presentation of ideas, is it a deep and thorough presentation of technical understanding?

Some Considerations include the target audience, the purpose for the development of the model, and the maturity of what has been done and what is to be done moving forward.

3.2 Software Models

Models of software are often represented as documentation. For instance, there is user documentation, and developer documentation.

Remark. We note that source code itself can be considered a model of run-time behaviour.

It is often a common mistake to confuse models with their representations. The means of representation will always bias an understanding of a model. This is due to a model's inability to capture every aspect of a real object.

3.3 Model Considerations

We strive to make good models. The following are some properties to avoid:

- Not attempting to serve any particular purpose.
- Too complex or simple for appropriate use.
- Misrepresents reality through incorrect details.
- Abstracts away the wrong details.
- Physically implausible.
- Language or communication that is unfamiliar to the target audience.

In considering whether a particular model is worthwhile, we need to consider their cost to produce, whether they permit an analysis of complex ideas, and whether they can be understood. Sometimes, developers may feel models are a waste of time, but it is important to produce jsut enough modeling as necessary.

3.4 Unified Modeling Language (UML)

UML is usually graphical and can be used to represent models about software. UML is descriptive rather than prescriptive. That is, there are different ways to explain things. Formal languages, such as mathematics, are self-consistent, unambiguous, and complete. It is therefore more difficult to describe fuzzier concepts. Natural languages, such as English, may be ambiguous, incomplete, and inconsistent. This makes natural languages difficult to explain crisp logical concepts in. UMLis an attempt to bridge this gap. Some details may be precise, while others are fuzzy.

We have three levels of incorrectness associated with all languages. We consider the kinds of errors of UML. **Syntactic** errors for instance ask whether one is representing classes with rectangles. **Semantic** errors for instance ask whether there are loops in the inheritance hierarchy. **Conceptual** errors for instance ask whether the objects communicate the information needed to perform the computations.

3.5 Views and Use of Diagrams

There are two main views regarding UML diagrams. There is the **structural view** and the **behavioural view**. Structural concerns how the software is divided into objects, classes, etc., and how these are related to each other. Behavioural concerns what happens at run-time to perform computations and how the objects interact.

Different diagrams are used to emphasize different aspects of a model. The same kind of diagram can be used to represent different kinds of models. For instance, the process of doing taxes manually and the process of implementing a tax program.

3.6 Structural Modeling

Structure diagrams are used to represent classes, relationships between types, objects, and packages. These diagrams are not used to represent run-time behaviour.

Objects are represented inside a rectangle and is underlined. Relationships between objects (called **links**) are lines between the rectangular boxes. The line may contain an arrow to clarify the relationship. A link always connects two objects exactly. Links can be unidirectional, bidirectional, or not specified. the direction indicate which object can send messages (the other object can only respond).

Objects can have types, which follow the object after a colon. The type we choose to denote for an object is a modeling choice. Objects may have properties that contain values (called **slots**).

Classes also use rectangles. they can be distinguished from objects since they are not followeed by a colon, and usually are not underlined. The **attributes** are properties of instances of the class and **operations** are actions that instances of the class can perform.

Remark. We note that attributes and operations can be supressed.

3.7 Class Relationships

There are six kinds of class relationships:

1. **Inheritance**
2. **Dependency**: A certain class depends on another if the other class is required for this class' implementation. Formal parameters, local variables, static operation invocations are all kinds of dependency relationships.
3. **Association**: Individual objects of one class are connected with individual objects of another class. An association generally indicates that two objects are loosely but specially related to each other. Typically, an association is indication of a field in the corresponding implementation.
4. **Aggregation**: Aggregation is a "Has-a" relationship that connects "whole" objects with their "parts".

5. **Composition**: Composition is a stronger relationship than aggregation. That is, the part has no independent existence. When the whole is destroyed, so are all the parts.

6. **Containment**: Nested classes is an indication of a containment relationship.

We note that some of the classes are stronger than others. Therefore, we use only the strongest applicable relationship. Usually, we do not use more than one relationship between two classes when one implies the other.

Remark. Aggregation and composition are relatively uncommon, and are thus unlikely to span one's entire diagram.

3.8 Class Generalization

Definition. Generalization is the inverse of specialization. A class that generalizes another means that the first is a superclass of the second, while the second class is a subclass of the first class. Specialization implies inheritance. That is, all the attributes and operations of the supertype appear within the subtype, unless marked as private (-).

4 Paragraph

In LaTeX, paragraphs are caused when two line breaks are used. Single line breaks are ignored. Hence this entire block is one paragraph.

Now this is a new paragraph. If you want to start a new line without a new paragraph, use two backslashes like this:
Now the next words will be on a new line. **As a general rule, use this as infrequently as possible.**

You can **bold** or *italicize* text. Try to not do so repeatedly for mechanical tasks by, e.g. using theorem environments (see Section 7).

5 Math

Inline math is created with dollar signs, like $e^{i\pi} = -1$ or $\frac{1}{2} \cdot 2 = 1$.

Display math is created as follows:

$$\sum_{k=1}^{n} k^3 = \left(\sum_{k=1}^{n} k\right)^2 .$$

This puts the math on a new line. Remember to properly add punctuation to the end of your sentences – display math is considered part of the sentence too!

Note that the use of `\left(` causes the parentheses to be the correct size. Without them, get something ugly like

$$\sum_{k=1}^{n} k^3 = (\sum_{k=1}^{n} k)^2.$$

5.1 Using alignment

Try this:

$$\begin{aligned}\prod_{k=1}^{4} (i - x_k)\,(i + x_k) &= P(i) \cdot P(-i) \\ &= (1 - b + d + i(c-a))\,(1 - b + d - i(c-a)) \\ &= (a-c)^2 + (b-d-1)^2\,.\end{aligned}$$

6 Shortcuts

In the beginning of the document we wrote

```
\newcommand{\half}{\frac{1}{2}}
\newcommand{\cbrt}[1]{\sqrt[3]{#1}}
```

Now we can use these shortcuts.

$$\frac{1}{2} + \frac{1}{2} = 1 \text{ and } \sqrt[3]{8} = 2.$$

7 Theorems and Proofs

Let us use the theorem environments we had in the beginning.

Definition. Let $\mathbb{R}$ denote the set of real numbers.

Notice how this makes the source code READABLE.

Theorem (Vasc's Inequality)**.** *For any a, b, c we have the inequality*

$$\left(a^2 + b^2 + c^2\right)^2 \geq 3\left(a^3 b + b^3 c + c^3 a\right).$$

For the proof of Theorem 7, we need the following lemma.

Lemma. *We have* $(x+y+z)^2 \geq 3(xy+yz+zx)$ *for any* $x, y, z \in \mathbb{R}$.

Proof. This can be rewritten as

$$\frac{1}{2}\left((x-y)^2 + (y-z)^2 + (z-x)^2\right) \geq 0$$

which is obvious. □

Proof of Theorem 7. In the lemma, put $x = a^2 - ab + bc$, $y = b^2 - bc + ca$, $z = c^2 - ca + ab$. $\square$

Remark. In section 7, equality holds if $a : b : c = \cos^2 \frac{2\pi}{7} : \cos^2 \frac{4\pi}{7} : \cos^2 \frac{6\pi}{7}$. This unusual equality case makes the theorem difficult to prove.

8 Referencing

The above examples are the simplest cases. You can get much fancier: check out the Wikibooks.

9 Numbered and Bulleted Lists

Here is a numbered list.

1. The environment name is "enumerate".
2. You can nest enumerates.
   - (a) Subitem
   - (b) Another subitem

$2\frac{1}{2}$. You can also customize any particular label.

3. But the labels continue onwards afterwards.

You can also create a bulleted list.

- The syntax is the same as "enumerate".
- However, we use "itemize" instead.