LOG8430: Architecture logicielle et conception avancée

Software Frameworks, Plugins and Metaprogramming

Automne 2017

Fabio Petrillo

Chargé de Cours

1

Page 2

Software measures (metrics)

- Size
  - **Source lines of code (SLOC)**
  - Statements, classes, methods, etc
- **Cyclomatic complexity**
- Coupling
- Code coverage
- Depth of Inheritance
- Maintainability – complexity/size

2

Page 3

Source lines of code (SLOC)

- Windows
  - 1993 – NT 3.1 – 4-5 MLOC
  - 2003 – Server 2003 – 20 MLOC (x5)
- Debian
  - 2000 – V 2.2 – 55-59 MLOC
  - 2009 – V 7.0 – 419 MLOC (~ x10)

https://en.wikipedia.org/wiki/Source_lines_of_code

Page 4

Source lines of code (SLOC)

- Counting the number of lines of code
- Intuitive

- “Simple” to calculate
- It depends language and coding standards
- Warning: **NEVER** use SLOC as a **productivity** metric!

Page 5

Cyclomatic complexity

- Thomas J. McCabe (1976)
- Number of independent paths
  - Methods, classes, or modules
- Control flow graph
- Cmc = Edges – Nodes + 2*Connections

Page 6

$$Mc = 9 - 8 + 2*1 = 3 \qquad Mc = 10 - 8 + 2*1 = 4$$

http://www.tutorialspoint.com/software_engineering/software_design_complexity.htm
https://en.wikipedia.org/wiki/Cyclomatic_complexity

6

Page 7

Page 8

Software Frameworks

Page 9

Software Frameworks

- Development from scratch?

Page 10

Software Framework

- Development from scratch?
- Rare -> practically no
- Framework oriented
- Framework -> as a language

Page 11

Software Framework

- Set of engineering **decisions**/choices
- Reusable platform
- **Template** project, libraries, tools
- Facilitate software development
- Previous experiences
- **Technological** decisions
- Each context/problem -> a framework
- “Vendor lock-in” antipattern

Kinds of Framework

- Web frameworks
- Enterprise frameworks
- Content Management Systems
- Game engine
- Mobile
- REST/Microservices

- Data processing (Hadoop, Spark)

Page 14

Web frameworks (Javascript)

Page 15

Enterprise frameworks

Example on Ruby on Rails

- Install ruby
- Install rails
  - gem install rails
- Create a new application
  - rails new blog
- Create a controller

- bin/rails generate controller Welcome index
- Run the application
  - cd blog
  - bin/rails server
- http://localhost:3000/welcome/index.html

Content Management System (CMS)

Page 18

Game Engine

Page 19

Page 20

How to choose a framework

- Popularity and community size
  - Number of developers
- Philosophy -> meet your needs
- Scaling
- Deployment - Hosting
- Sustainability

- Documentation
  - Support
  - Training

License -> GPL, MIT Licenses

How to choose a framework

- Popularity and community size
  - Number of developers
- Philosophy -> meet your needs
- Scaling
- Deployment - Hosting
- Sustainability
- Documentation
  - Support
  - Training

License -> GPL, MIT Licenses

System Stack

- Combination of frameworks to create a system

http://svsg.co/how-to-choose-your-tech-stack/

System Stack – Stackshare

- http://stackshare.io/
- Tools to share stacks
- Searching tools

- Popularity
- Trending
- Discover new tools and services

Plugins (Plugiciel)

Page 26

Plugiciel

- Les programmes nécessitent donc
  - Une plateforme de programmation favorisant l'indépendance des composants
  - Un format de livraison « standardisé »
  - Une plateforme d'exécution permettant le remplacement à chaud
- Les programmes doivent donc être formés de composants réutilisables et interchangeables en cours d'exécution

Page 27

Plugiciel

- « Un [plugiciel] est un programme qui interagit avec un logiciel principal, appelé programme hôte, pour lui apporter de nouvelles fonctionnalités » [Wikipedia]
  - ils ne peuvent fonctionner seuls car ils sont uniquement destinés à apporter une fonctionnalité à un ou plusieurs logiciels ;
  - ils sont mis au point par des personnes n'ayant pas nécessairement de relation avec les auteurs du logiciel principal.

Plugiciel

- L'objectif de concevoir un logiciel sous forme de plugiciels est de permettre:
  - L'ajout des fonctionnalités sans avoir à tout reprogrammer
  - Permettre aux utilisateurs d'ajouter leurs propres

fonctionnalités de manière indépendante

– Cette indépendance inclut la possibilité pour le logiciel principal d'évoluer tout en restant compatible avec les plugiciels existants ; **cette condition est cependant loin d'être toujours remplie.**

Plugiciels - exemples

- Firefox
- Chrome
- Wordpress
- Eclipse
- ….

Eclipse Plugin Architecture

Page 32

Page 33

Points d’extension

- Extensions
  - Points d’extension
    - point d'ancrage dans plugins
    - le “provide” des composants
    - ressemble à la déclaration d'une interface, le plugiciel informe qu’il est ouvert à l’ajout de nouvelles fonctionnalités d’une certaine façon
    - mais description en XML précisant la grammaire que les meta-data des extensions doivent suivre
  - Extension
    - greffon attaché à un point d'extension (déclaration de la nouvelle fonctionnalité d’extension)
    - le “require” des composants

Répertoires composants un plugiciel

35

Page 36

Metaprogramming

Interconnections

- Clients–Libraries/Frameworks
  - Linking
  - Forking
  - Inter-process communication
  - Subclassing
  - Dynamic loading/invoking

Page 38

Interconnections

- Linking (Contrast with virtual machines)
  - Typically C/C++
  - Several object files (.o)
  - One executable (.exe)

Page 39

Interconnections

- Forking
  - Typical in most languages
  - Process duplication
    - Is a real duplication
    - Creates a new OS process

```
final StringBuffer commandLine = new StringBuffer();
commandLine.append("..\\DOT\\bin\\dotty ");
commandLine.append(aFilePath);
final Process process =
        Runtime.getRuntime().exec(commandLine.toString());
final OutputMonitor errorStreamMonitor =
        new OutputMonitor(...,process.getErrorStream(),...);
errorStreamMonitor.start();
final OutputMonitor inputStreamMonitor =
        new OutputMonitor(...,process.getInputStream(),...);
inputStreamMonitor.start();

try {
        process.waitFor();
}
catch (final InterruptedException ie) {
        ie.printStackTrace(
        Output.getInstance().errorOutput());
}

if (process.exitValue() != 0) {
        ...
}
```

Page 40

Interconnections

- IPC
  - Typical in most languages
  - Use remote procedure calls

– Use well-defined protocols
  - COM
  - CORBA
  - XML-RPC
– Web services
– REST

Interconnections

```
package kr.ac.yonsei.cse3009.rpc;

import org.apache.xmlrpc.server.PropertyHandlerMapping;
import org.apache.xmlrpc.server.XmlRpcServer;
import org.apache.xmlrpc.webserver.WebServer;

public class Server {
        public static void main(final String[] args)
                throws Exception {

                final WebServer webServer = new WebServer(8080);
                final XmlRpcServer xmlRpcServer =
                        webServer.getXmlRpcServer();
                final PropertyHandlerMapping phm =
                        new PropertyHandlerMapping();
                phm.addHandler("Calculator", Calculator.class);
                xmlRpcServer.setHandlerMapping(phm);

                webServer.start();
        }
}

package kr.ac.yonsei.cse3009.rpc;

public class Calculator {
        public int add(final int i1, final int i2) {
```

```
package kr.ac.yonsei.cse3009.rpc;

import java.net.URL;
import org.apache.xmlrpc.client.XmlRpcClient;
import org.apache.xmlrpc.client.XmlRpcClientConfigImpl;

public class Client {
        public static void main(final String[] args)
                throws Exception {

                final XmlRpcClientConfigImpl config =
                        new XmlRpcClientConfigImpl();
                config.setServerURL(
                        new URL("http://127.0.0.1:8080/xmlrpc"));

                final XmlRpcClient client = new XmlRpcClient();
                client.setConfig(config);

                final Object[] params = new Object[] {
                        new Integer(33), new Integer(9) };
                final Integer result = (Integer)
                        client.execute("Calculator.add", params);
                System.out.println(result);
        }
```

```
                return i1 + i2;
        }
        public int sub(final int i1, final int i2) {
                return i1 - i2;
        }
}                                                                                    }
```

41

Page 42

Interconnections

- Subclassing
  - Hooks and templates
    - Hot spots = hooks
    - Frozen spots = templates
  - Hooks are typically abstract methods
  - Templates typically use hooks

42

Page 43

Interconnections

- Subclassing
  - Hooks and templates
    - JUnit

Interconnections

```
public abstract class TestCase
        extends Assert implements Test {

        public void runBare() throws Throwable {
```

- Template

- Hooks

```
                setUp();
                try {
                        runTest();
                }
                finally {
                        tearDown();
                }
        }

        protected void setUp() throws Exception {
        }

        protected void tearDown() throws Exception {
        }
        ...
}
```

44

Page 45

Subclassing

- Heavily used in object-oriented programs
  - Heavily used in design patterns

    (Only *Singleton* does not explicitly use subclassing)

    - *Abstract Factory*
    - *Composite*
    - *Decorator*
    - *Observer*

- *Visitor*
- ….

45

Page 46

Dynamic loading

- In different programming languages (but not all), it is the possibility to **load**, **use**, and **unload** a piece of code at runtime
- In Java, it is the possibility to load and unload **a class** and to choose and invoke **its methods** (and to access its fields…) at runtime

46

Page 47

Chargement dynamique

- Charger une classe à partir de son nom
  - Classe enveloppante (*wrapper*)
- Appeler une méthode de la classe
- Charger une classe vs. Chargeur de classe

```
public final class WrapperMain {
	public static void main(String[] args) {
		try {
			Class toBeRun = Class.forName(args[0]);
			Method mainMethod =
				toBeRun.getMethod("main",
					new Class[] { String[].class });
			final long startTime =
				System.currentTimeMillis();
			mainMethod.invoke(null,
				new Object[] { new String[0] });
			final long endTime =
				System.currentTimeMillis();
			System.out.println();
			System.out.println(endTime - startTime);
		}
		catch (final Exception e) {
			e.printStackTrace(
				Output.getInstance().errorOutput());
		}
	}
}
```

Class Loading

- Virtual machines
  - Interpreters

  - Closed world
- Must
  - Access resources
  - Load classes

Class Loading

http://map.sdsu.edu/geog583/images/week8.3.gif 49

Page 50

•Class Loading

http://www.onjava.com/2005/01/26/graphics/Figure01_MultipleClassLoaders.JPG 50

Page 51

.Class Loading

```
public class ClassLoader extends java.lang.ClassLoader {
        private final String directory;

        public ClassLoader(
                final java.lang.ClassLoader parent,
                final String directory) {

                super(parent);
                this.directory = directory;
        }
```

```
protected Class findClass(final String name) {
        Class newClass = null;
        try {
                newClass = super.findClass(name);
        }
        catch (final ClassNotFoundException cnfe) {
                final String osName =
                        this.directory + name.replace('.', '/') + ".class";

                try {
                        final FileInputStream fis = new FileInputStream(osName);
```

```
                    newClass = this.defineClasses(name, fis);
                }
                catch (final ClassFormatError cfe) {
                    cfe.printStackTrace(Output.getInstance().errorOutput());
                }
                catch (final FileNotFoundException fnfe) {
                    // fnfe.printStackTrace();
                }
            }

            return newClass;
        }
```

52

Page 53

```
    private Class defineClasses(final String name, final InputStream inputStream) {
        try {
            int b;
            int length = 0;
            byte[] bytes = new byte[4];
            while ((b = inputStream.read()) != -1) {
                if (length == bytes.length) {
                    final byte[] temp = new byte[length + 4];
                    System.arraycopy(bytes, 0, temp, 0, length);
                    bytes = temp;
                }
                bytes[length] = (byte) b;
                length++;
            }

            System.out.println(name);
            final Class newClass = this.defineClass(name, bytes, 0, length);
            return newClass;
        }
        catch (final IOException ioe) {
            return null;
        }
        catch (final NoClassDefFoundError ncdfe) {
            ncdfe.printStackTrace(Output.getInstance().errorOutput());
            return null;
        }
    }
```

Page 54

Metaclass

- Class
  - A programming construct that encapsulates some state (fields) and behaviours (methods)
  - A construct whose instances are **objects**
- Metaclass
  - A programming construct that encapsulates some state (fields) and behaviours (methods)
  - A construct whose instances are **classes**

Page 55

Metaclass

- An object is an instance of a class
  - A class generates an object
- A class is an instance of a metaclass
  - A metaclass generates a class
- A metaclass is an instance of…

  (a) A meta-metaclass?

  (b) Itself?

Metaclass

- An object is an instance of a class
  - A class generates an object

- A class is an instance of a metaclass
  - A metaclass generates a class
- A metaclass is an instance of…
  (a) A meta-metaclass?
  (b) **Itself**

Metaclass

- The class Object is the parent of all classes, including metaclasses
- The class Class is the generator of all classes, including Object

57

Page 58

Des langages de programmation à objets
à la représentation des connaissances à
travers le MOP : vers une intégration
par Gabriel Pavillet. Thèse de doctorat
en Informatique sous la direction de
Roland Ducournau, soutenue en 2000
à Montpellier 2

58

Page 59

Metaclass

- In Java, the metaclass is anonymous and cannot be modified
- In Smalltalk, “[t]here is only one instance of a particular Metaclass, namely the class which is being described”

Reflection

- Reflection is the ability of a computer

program to **examine and modify** the
structure and behaviour of an object at
runtime.

Page 61

Scenario

– Given a class C
– Given an object o, instance of C
– Identify all the methods available on o
– Invoke a method using its name foo

```
public class C {
        private int i;
        public C(final int anInt) {
                this.i = anInt;
        }
        public void foo(final String s)
{
```

```
                System.out.print(s);
                System.out.println(this.i);
            }
        }
```

61

Page 62

Scenario

– Given a class C
– Given an object o, instance of C
– Identify all the methods available on o
– Invoke a method using its name foo

```
final C o = new C(42);

System.out.println("Identify all the methods available on o" );
final Class<?> classOfO = o.getClass();
final Method[] methodsOfC = classOfO.getMethods();
for (int i = 0; i < methodsOfC. length; i++) {
        final Method method = methodsOfC[i];
        System.out.print('\t');
        System.out.println(method.getName());
}

System.out.println("Invoke a method using its name foo" );
final Method fooMethod = classOfO.getMethod( "foo",
         new Class[] { String. class });
fooMethod.invoke(o, new Object[] { "\tThis is foo: " });
```

62

Page 63

Scenario

– Given a class C
– Given an object o, instance of C
– Identify all the methods available on o
– Invoke a method using its name foo

```
Identify all the methods available on
o
        foo
        getClass
        hashCode
        equals
        toString
        notify
        notifyAll
        wait
        wait
        wait
Invoke a method using its name foo
        This is foo: 42
```

63