Software Quality and Software evolution

introduction

Alexandre Bergel
Romain Robbes
{abergel,rrobbes}@dcc.uchile.cl

Software evolve, pretty much as
any artifact created by humans

A software get born one day, fathered by a team comprising developers and managers. Similarly to human being, it grows up to reach a mature stage. Each new requirement asked by customers contributes to this grow. The grow goes smoothly when it has been foreseen. In case of non anticipation, evolution results in a degradation in quality and maintainability.

software |ˈsôftˌwe(ə)r|
noun

the programs and other operating information used by a computer

quality |ˈkwäləte͞|
noun

the degree of excellence of something

evolution |ˌevəˈloō sh ən|
noun

the gradual development of something, esp. from a simple to a more complex form

evolution |ˌevəˈloō sh ən|
noun

the gradual development of something, esp. from a simple to a more *complex* form

The important point to keep in mind is that “evolution” is related to “complexity”, by definition.

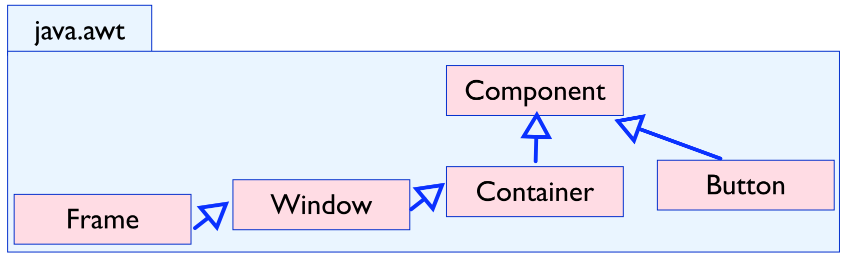
The goal of this lecture is to:

- introduce some problems related to software evolution
- introduce mechanisms and approaches to cope with this complexity
- learn how to use some tools and techniques to cope with software complexity

Let’s pick an example, the AWT and Swing libraries

A more complete description of this example may be found in
Alexandre Bergel, Stéphane Ducasse, and Oscar Nierstrasz, Classbox/J: Controlling the Scope of Change in Java, In Proceedings of Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'05), New York, NY, USA, ACM Press, pp. 177-189, 2005
http://www.iam.unibe.ch/~scg/Archive/Papers/Berg05bclassboxjOOPSLA.pdf

Presentation of AWT

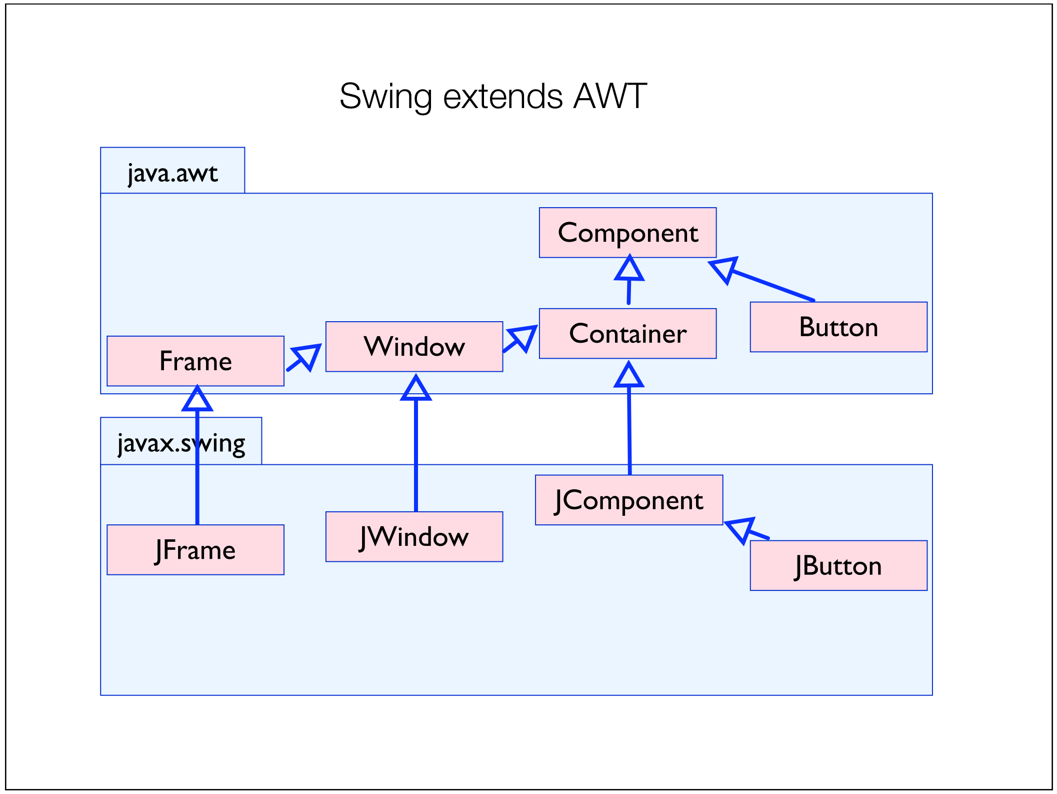


Widgets are components (i.e., inherit from Component)

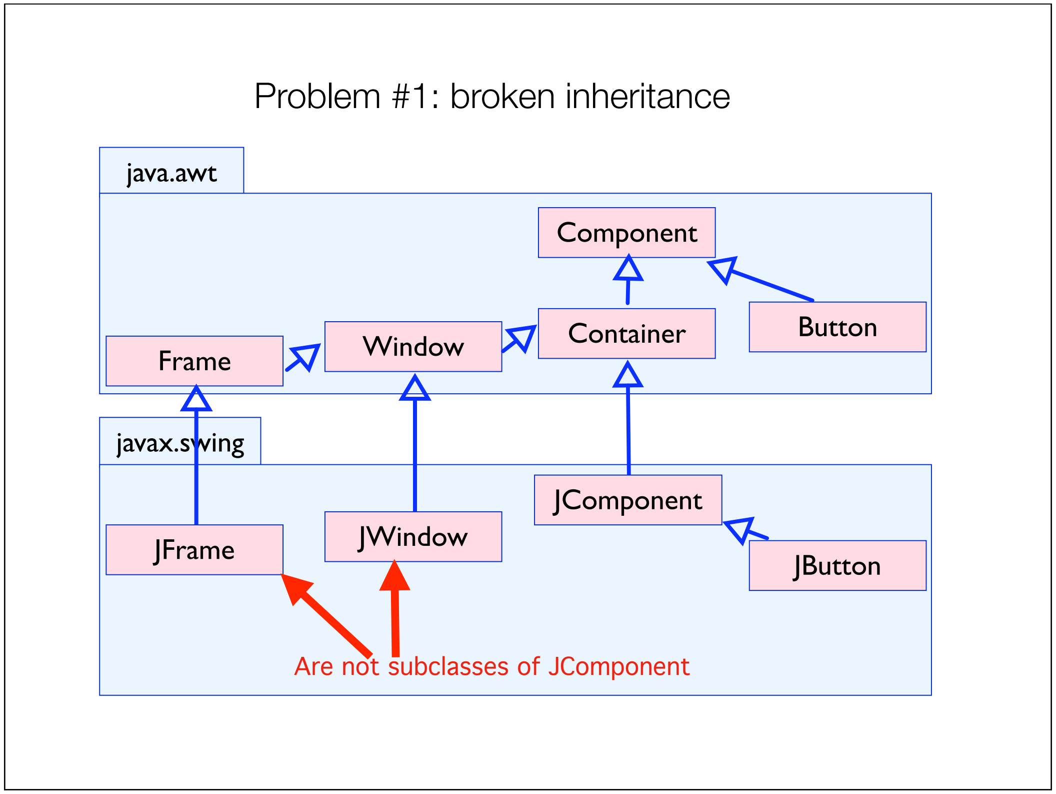
A frame is a window (Frame is a subclass of Window)

The slide shows the 5 more representative of the AWT library. Just from the class hierarchy organization, two statements at least can be formulated:

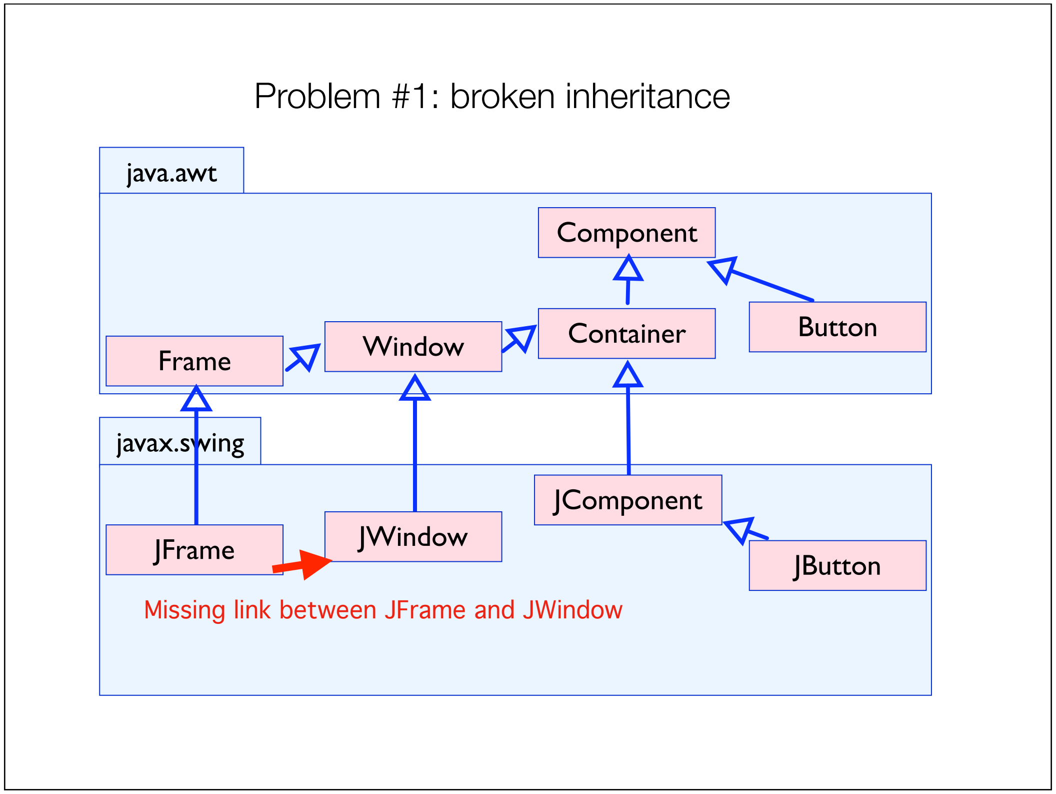
- All AWT widgets are components
- A frame is a window since the class Frame inherits from the class Window.



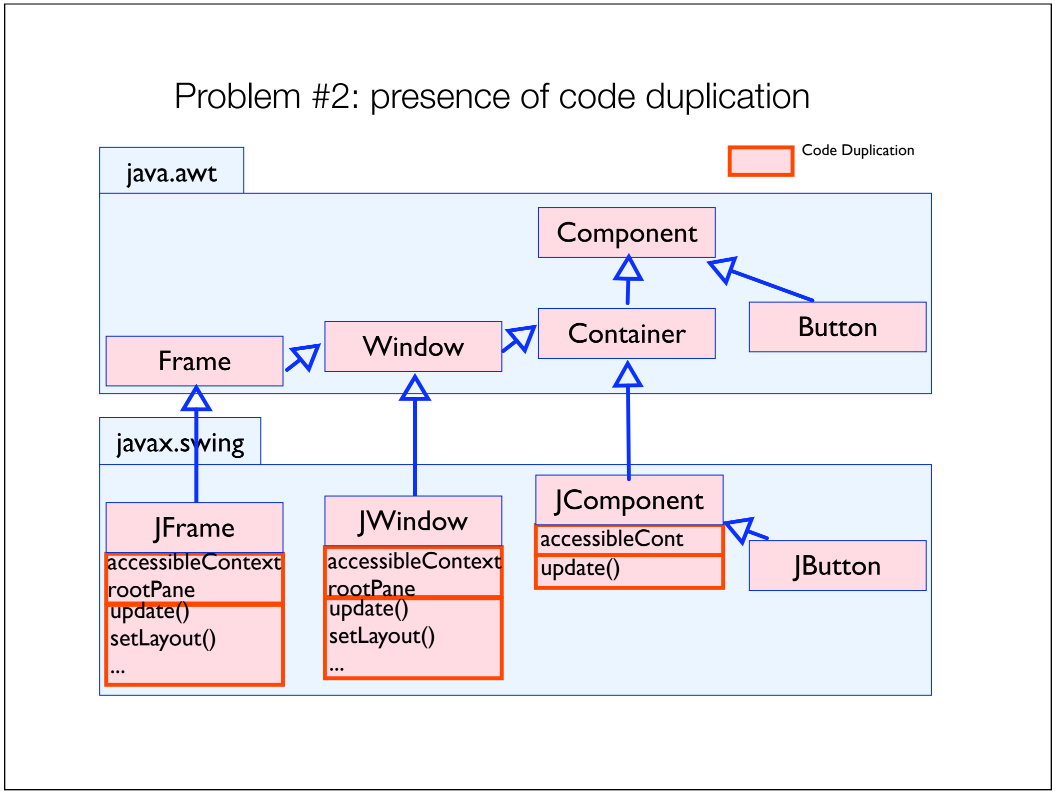
The package javax.swing models the core of Swing. The 4 most important classes are represented on the slide. Inheritance is used to define Swing as an extension of AWT.



As we saw, in AWT all widgets are AWT components and a frame is a window. In Swing, this does not hold anymore. Whereas a JButton is a JComponent, a JFrame and a JWindow are not a JComponent.



Moreover, a frame is not a window in Swing.



This kind of “missing inheritance link” shown before needs to be simulated somehow. This missing link is “emulated” by duplicating the code from JComponent to JWindow and JFrame and from JWindow to JFrame. 50% of JWindow is duplicated in 30% of JFrame. This corresponds to hundreds of lines of code.

Presence of code duplication is well know to complicate maintenance since duplicated part needs to be synchronized in case of a modification.

Problem #3: explicit type checks and casts

```
public class Container extends Component {
  Component components[] = new Component [0];
  public Component add (Component comp) {...}
}

public class JComponent extends Container {
  public void paintChildren (Graphics g) {
    for (; i>=0 ; i--) {
     Component comp = getComponent (i);
     isJComponent = (comp instanceof JComponent);
     ...
     ((JComponent) comp).getBounds();
    }
  }}
```

The third problem identified in Swing is the excessive presence of explicit type checks and casts. As an example, consider the class Container. The class Container belongs to AWT. As its name will testify, a container contains other components. The variable ‘components’ is an array of Component

JComponent is a subclass of Container that defines a central notion in Swing. A Swing component may contain other components therefore. When displayed on the screen, the method paintChildren is invoked and iterates over all contained components. Each component needs to know whether it is a AWT or Swing one, since an AWT and a Swing component are not displayed following the same manner. Distinction between an AWT and a Swing component is made using meta-programming contastruct, such as 'instanceof'. Downcasts are then subsequentially used. Downcasts have the tendency to throw runtime exception in case of failed runtime check. This way of programming decrease the readibility of the code and may hamper the robustness.

let’s step back

AWT couldn’t be enhanced without risk of breaking existing code

Swing is built on top of AWT using subclassing

As a result, the quality of the Java GUI framework decreased with its evolution

AWT was released with Java 1.0 in January 1996. AWT was meant to be a small library to design graphical user interfaces. It only has 5 main widgets. The increasing popularity of Java forced Sun to deliver a better framework to design GUI. Modifying AWT in such a drastic way (look and feel, more widget, double buffering where the most wanted features) couldn't be achieved without impacting numerous already existing clients.

Instead of modifying Swing, Sun adopted the resolution to create Swing, a new library at the top of AWT. Swing was not made from scratch, AWT code has been reused by being subclassed.

Because Java does not provide better way to extend the code then subclassing, quality of Swing is clearly suboptimal. The issues presented before are not the only one. For example, the component layouting is another part of the library that are poorly designed.

Why should we care?

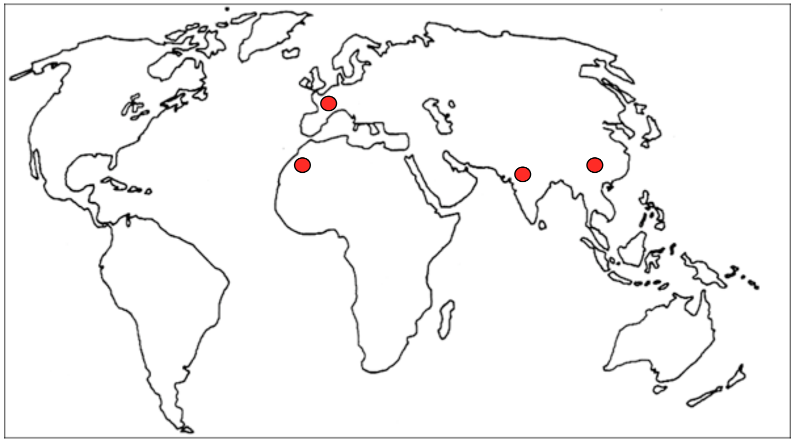
- Swing appeared in 1998, and has almost not evolved since!
- Swing is too heavy to be ported on PDA and cellphones
- SWT is becoming a new standard
- Either a system evolves, or it dies

Yeah, Swing is poorly designed. What shall I care?
You care because Swing hasn't significantly evolved for a long time and its pachydermatous content can hardly be ported to embedded and light consumption devices. Moreover, supporting native widgets is clearly the way that should be adopted by GUI frameworks in order to benefit from advanced OS capabilities. Other libraries such as SWT are slowly becoming a new standard.

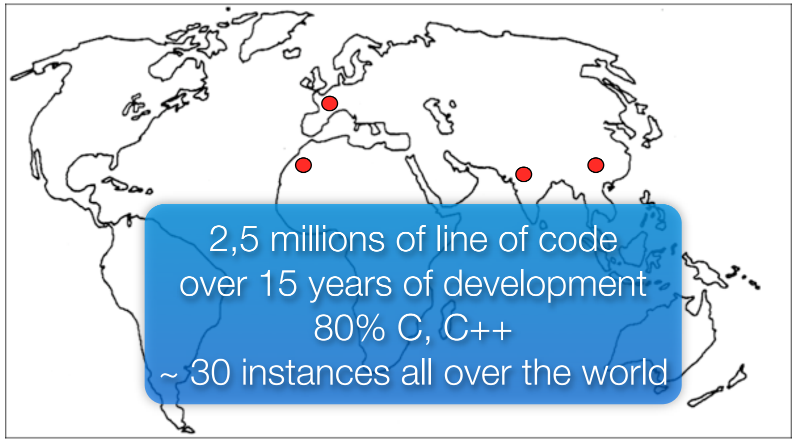
Either a system evolves, or it dies [Lehman94]

Other examples

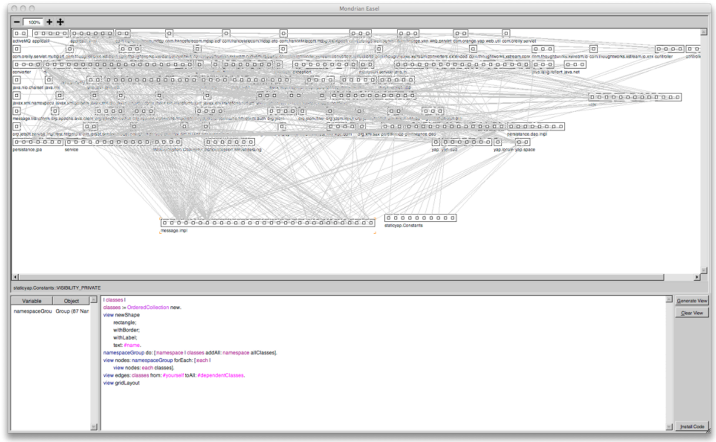
Construction sites for an European truck maker



Construction sites for an European truck maker



Large software in a French telecom company



~100 packages
~ 500 classes

Paris, 2008

Typical large scale long living systems

- Large
  - thousands of classes
  - 2s to read a line of 1 Million LOC system => 3 months
- Undocumented - knowledge loss
- Lack of structure overview (layers, cycles, core)
- Multi developers
- Multi years development

“As soon as the competent crew leaves the organization, the maintenance iceberg becomes visible”

Chris Verhoef

System evolution is like... SimCity

Course content

Programming language

- Pharo

Programming environment

- Moose, software visualization

Software engineering

- Source code quality, testing, analyzing code repository

Course content

- International experts and researchers
  - Participation of widely recognized researchers in the field
- Summer school in November 9-14
  - The rôle of programming languages in software evolution

Class evaluation

4 activities ("tareas")

quality assessment

take a software and tell me its quality using tools and visualization offered by the Moose platform

reengineering

improve the quality of a software written in Pharo. A report has to come with your improvement suggestions. The author of the software will then be contacted

Class evaluation

- paper presentation
  - present some papers
- tool development in Pharo and Moose

Tool development

Stored on SqueakSource

Ask for user feedback

Conducted in Pharo and Moose

Producing a short report

Lecture agenda

09/08/10 class begin, today!

1,5 weeks: intro + small exercise

- pick a java project and assess its quality

3 weeks: tarea 1

- Moose, visualization (system complexity, class blueprint)

1,5 week: learning Pharo

3 weeks: tarea 2

- Pick a Pharo application, use DSM, Package blueprint, SUnit and test coverage tool

Lecture agenda

1 week: Mining Software Repositories

5 weeks: tool coding

27/11/10 class ends

“The best way to predict the future is to invent it”

Alan Kay