



# Project case study

Ricca Filippo & Mariano Ceccato

ITC-Irst

Centro per la ricerca Scientifica e  
Tecnologica

{ricca, ceccato}@itc.it



# The project steps

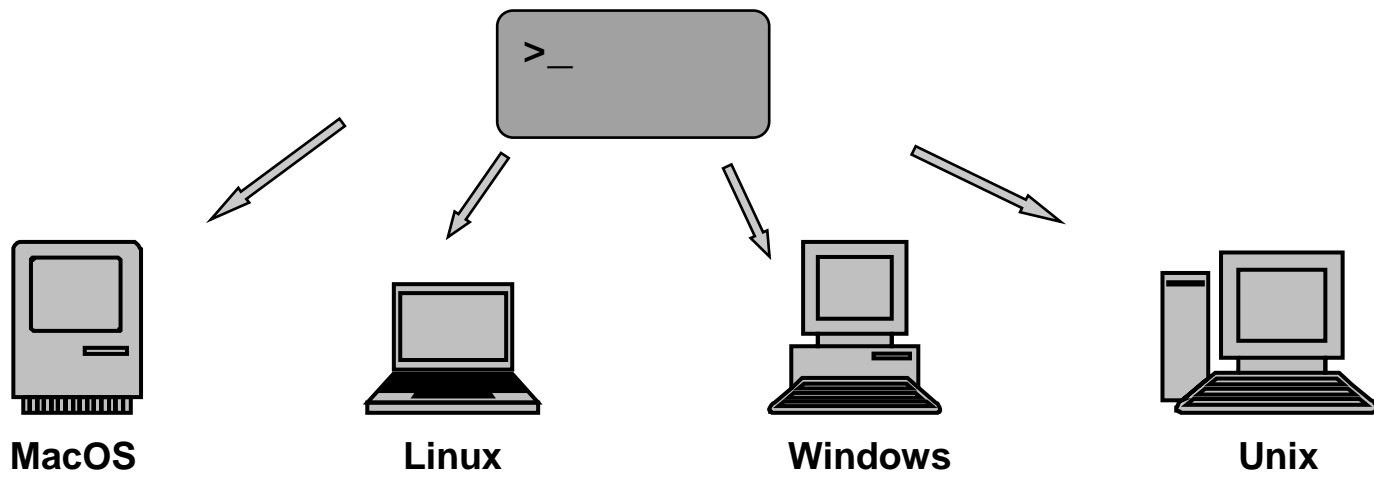
---

---

- ◆ **Reverse engineering. Collect some views from the case study code.**
- ◆ Instrumentation of the JAVA code using TXL and writing test cases with statement/branch coverage.
- ◆ Aspect identification.
- ◆ Extend the JAVA grammar for the ASPECTJ language.
- ◆ Aspects code generation using TXL
- ◆ Regression testing.

# Case study: Jconsole

- ◆ Open source cross-platform Java application “<http://javaconsole.sourceforge.net/>”
- ◆ 2 packages, 27 classes, 1600 LOC.
- ◆ It behaves as an operative system console, but in an operative system independent way.
- ◆ It can be plugged into various other applications.
- ◆ Lightweight and fully functional with a basic set of commands.
- ◆ More commands can be coded and the added in minutes.

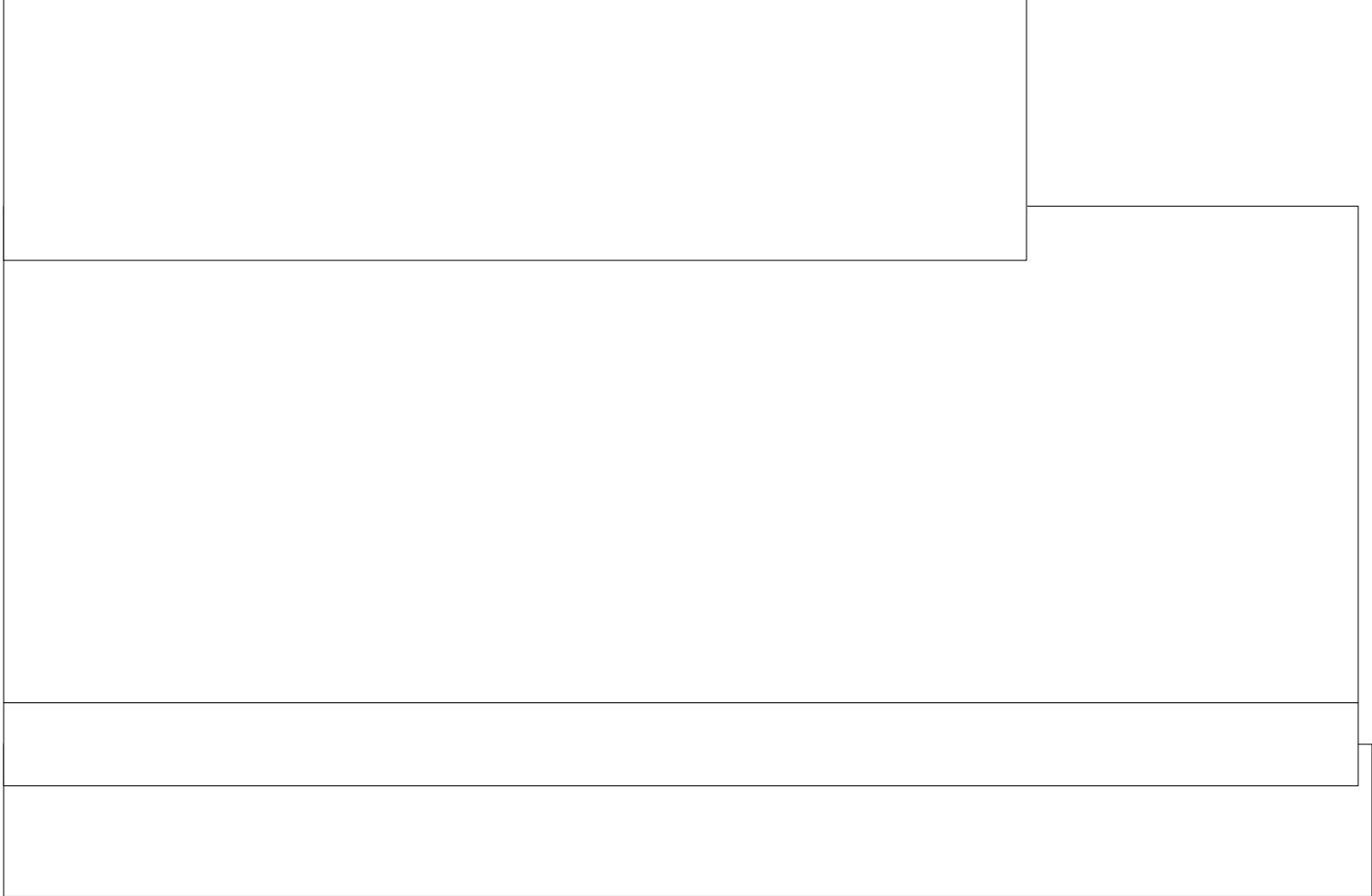




---

# Example

```
[linux_prompt]> java org.jconsole.Jconsole
```





# Jconsole: commands

---

---

help : provide help information.  
exit : exit the console.  
quit : exit the console, same as exit.  
hist : display the command history.  
pwd : print the current directory.  
set : change properties that affect behaviour of other  
      commands or list properties with the pattern.  
rmdir : remove a directory.  
ls : display the directory listing.  
cd : change the console directory.  
cp : copy a file.  
cls : clear the screen.  
ptrace: print the last Error Stact trace.  
alias : display the command aliases.  
home : cd to the console home dir.



# Jconsole: execution modes

---

---

- ◆ Interactive: typing commands directly on the console. You see the result of each command (useful to understand and practice).
- ◆ Batch: running a script on the console. You see the result of the whole script (useful when develop test cases).

# Class Organization

## `org.jconsole`

- ◆ Application business logic
- ◆ Core commands

```
org.jconsole.CDCommand  
org.jconsole.CLSCCommand  
org.jconsole.CPCommand  
org.jconsole.ExitCommand  
org.jconsole.HelpCommand  
org.jconsole.HistoryCommand  
org.jconsole.LSCCommand  
org.jconsole.MKDIRCommand  
org.jconsole.PWDCommand  
org.jconsole.QuitCommand  
org.jconsole.RMDIRCommand  
org.jconsole.SetCommand  
org.jconsole.ShowCommand
```

```
org.jconsole.StringUtil  
org.jconsole.SystemProperties  
org.jconsole.Utilities  
org.jconsole.PTrace  
org.jconsole.PushTrace
```

```
org.jconsole.JConsole  
org.jconsole.Console  
org.jconsole.ConsoleCommand  
org.jconsole.HistoryEntry  
org.jconsole.SetClassPath
```

```
org.jconsole.CommandFailedException  
org.jconsole.CommandResultException  
org.jconsole.ConsoleSystemException
```

## `org.jconsole.commands`

- ◆ Additional commands, (for example user defined ones)

```
org.jconsole.commands.pingCommand
```



# The project for the course



- ◆ **Reverse engineering. Collect some views from the case study code**
- ◆ Instrumentation of the JAVA code using TXL and writing test cases with statement/branch coverage.
- ◆ Aspect identification
- ◆ Extend the JAVA grammar for the ASPECTJ language
- ◆ AspectJ code generation using TXL
- ◆ Regression testing

# Reverse Engineering

Extract the UML class diagram for the *Jconsole* application using TXL and represent it in DOTTY:

- ◆ Classes involved, with attributes (fields)
- ◆ Generalization (“extends” relations)
- ◆ Associations (“use” relations)

**And deliver it!**

```
class A {  
  int x, y;  
}
```

```
class B {  
}
```

```
class C extends B {  
  A a;  
}
```

