HMS Analytical Software

Software Design Patterns in R

R User Conference UseR! 2011 - 17 August 2011

Friedrich Schuster

Company

HMS Analytical Software

- HMS Analytical Software is a specialist for Information Technology in the field of Data Analysis and Business Intelligence Systems
- Profile
  - 40 employees in Heidelberg, Germany
  - SAS Institute Silver Consulting Partner for 14 years
  - Doing data oriented software projects for more than 20 years
- Technologies
  - Analytics and Data Management: SAS, JMP, R, Microsoft SQL Server
  - Application Development: Microsoft .NET, Java

Our IT Services for the Life Science Industry (SAS, JMP and R)

HMS Analytical Software

Independent Consulting

Programming

Data Management

Training and Individual Coaching

Application Development and Integration

Software Validation

Friedrich Schuster

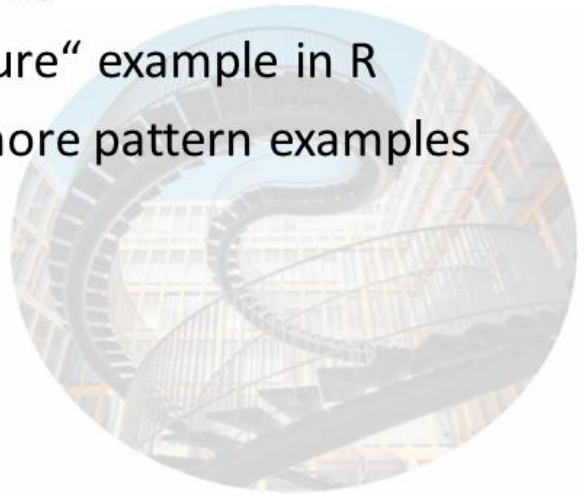
Introduction

- My name is Friedrich Schuster.
- I am a software developer, working mostly with object-oriented languages …
- … in projects for the pharmaceutical industry.
- Analytical Software in Heidelberg (the company I work for) was established in 1989 and specializes in information technology for data analysis and business intelligence.

Agenda

HMS Analytical Software

- What is a design pattern?
- What are patterns good for?
- R language and design patterns
- „Factory Method“ and „Closure“ example in R
- Pattern catalogs and some more pattern examples
- Conclusions



What was the idea behind this presentation?

Well, in Java, software development patterns are everywhere.

So the basic question was: how about patterns in R? Do they exist? Are they important?

What is a design pattern?

First of all: what is a design pattern?

Patterns are everywhere. This is a staircase.

What is a design pattern?

This is quite a different looking staircase.

The basic principle is the same: „a flight of stairs, a supporting framework, and a handrail".

You all know the staircase pattern.

What is a design pattern?

Friedrich Schuster

That's another staircase. With additional features.

The last example shows that it is easy to identify a "nonstandard implementation" of the staircase pattern.

Do you need to describe a staircase in every detail? It depends. Not if you say „Go downstairs, use the staircase over there". Now for building a new staircase, knowing the pattern alone is not sufficient. You will first have to carefully design every part of it.

That means that the pattern not only helps building actual staircases, but also simplifies communication.

What is a design pattern?

HMS Analytical Software

- A **generalized, reusable and time-tested solution.**
- Every pattern has a **name**.
- Every pattern has a **description of its general principle.**
- A pattern may also contain additional information:
  - About the reasons why to use a pattern („**Motivation**“),
  - where to use it („**Applicability**“),
  - what elements it consists of („**Participants**“),
  - how the elements interact („**Collaborations**“),
  - the **consequences** of its use,
  - and **code examples**.
- Patterns are organized into **catalogs**.

Software design patterns are similar.

- They are **generalized, reusable and time-tested templates of solutions**.
- Every pattern has a **name, ...**
- … a **description of the general principle** of the solution and ...
- … also includes **reasons why** to use the pattern, **where** to use it, what **elements it consists of**, h**ow the elements interact**, the **consequences** of its use, and **code examples**.
- **Multiple patterns are organized into catalogues**.

In short: patterns are templates and describe design alternatives.

Benefits

HMS Analytical Software

- **Easy to understand**.
- Pattern catalogs **simplify comparing and selecting** appropriate solutions for a specific problem.
- Consequences of using a pattern are well known, so patterns **reduce the risk** of unforeseen negative effects.
- Re-use of proven concepts **increases productivity**.
- **Simplify communication** and documentation.
- Make **code more readable** and **reduce maintenance costs.**

Why are design patterns useful? Just a few reasons:

- They are easy to understand.
- Patterns reduce the risk of unforeseen negative consequences of a design.
- Re-use of patterns increases productivity.

The overall effect is that patterns help to improve the quality and value of the software.

R language and design patterns

HMS Analytical Software

- R follows the **functional** paradigm, but not strictly.
- R is **a dynamic** language.
- In R, **everything is an object**, including functions.
- R is a **domain specific language** („DSL“)
- Modification of objects at runtime is possible.

Definition problems in R:

- What are the basic „entities“ that patterns consist of?
- Where does plain code end and where do patterns begin?

Friedrich Schuster

As programming languages, R and Java are quite different.

For example R is a functional language.

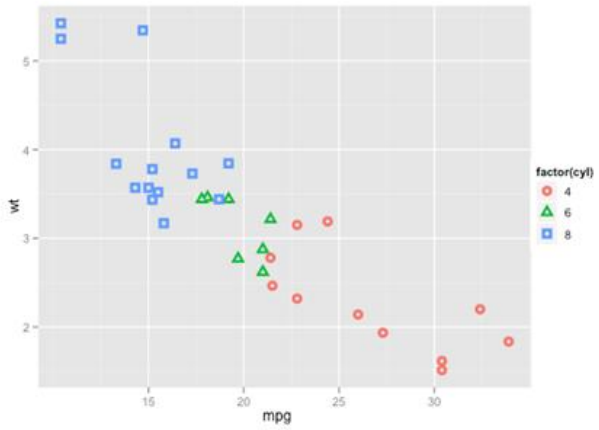
That means that …
… patterns in R cannot be defined on the class level alone. In a functional language the basic unit of code is a function. But R also has object-oriented features.

So in R patterns consist of **functions**, of **classes** and **methods**, of **packages** and **environments** (plus **all combinations**).

Factory Method Pattern

HMS Analytical Software

- Factory method pattern



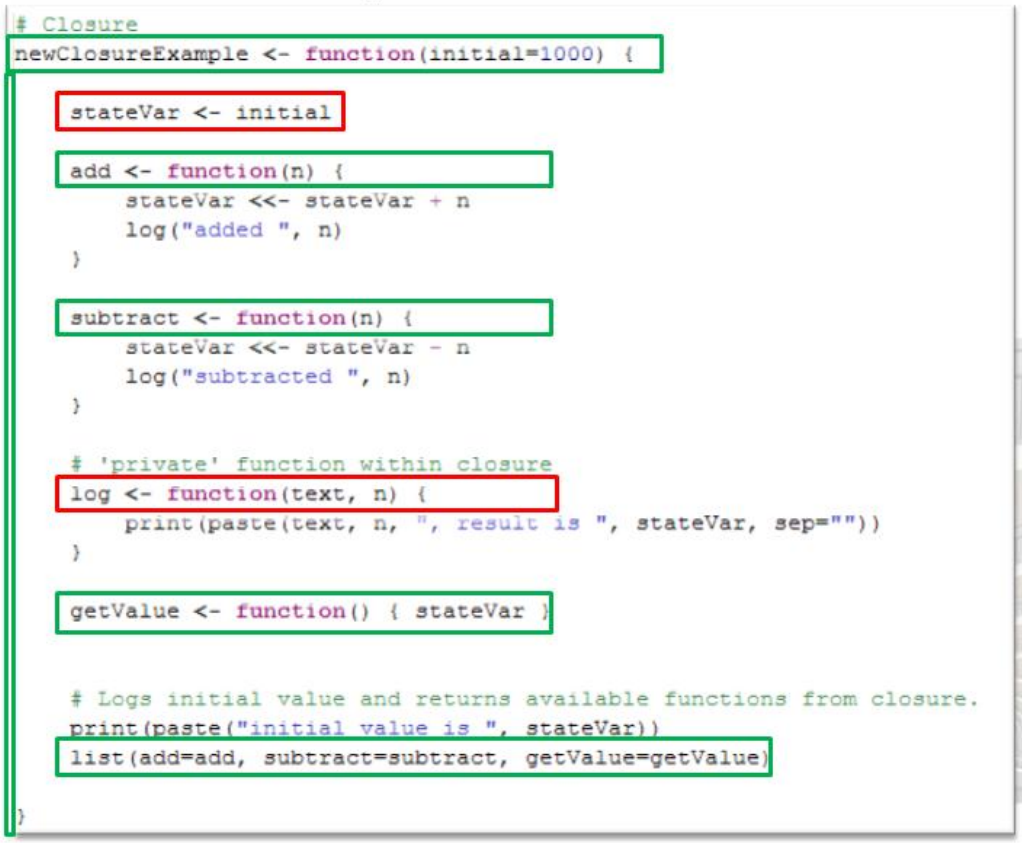
Friedrich Schuster

First pattern example: the "**factory method**" from object oriented programming.

Let's assume that in this scatterplot every symbol is a separate object. The plotting program does not create the symbols, but calls a separate function for this which returns the required symbol object. This function is called the "factory method".

So the plotting program does not have to know what exact type of symbol it has to create. It just has to know how to fetch symbols from the "factory method". Exchanging sets of symbols becomes easy: just supply a different factory method. So a factory makes a program independent of how objects are created.

Closure example in R



```
# Closure
newClosureExample <- function(initial=1000) {

    stateVar <- initial

    add <- function(n) {
        stateVar <<- stateVar + n
        log("added ", n)
    }

    subtract <- function(n) {
        stateVar <<- stateVar - n
        log("subtracted ", n)
    }

    # 'private' function within closure
    log <- function(text, n) {
        print(paste(text, n, ", result is ", stateVar, sep=""))
    }

    getValue <- function() { stateVar }


    # Logs initial value and returns available functions from closure.
    print(paste("initial value is ", stateVar))
    list(add=add, subtract=subtract, getValue=getValue)

}
```


Another more abstract example: this is the **code** illustrating the **Closure pattern**.
I will not go into every detail.

- The **intent** of this pattern is to create local values in their own environment along with the functions that use and update them.
- In this example: the (outer) function „newClosureExample()" creates the closure.
- Within the closure inner functions and a state variable are defined and then returned as a list object to the caller.
- The idea of a closure is quite similar to the class concept in object-oriented programming: it is possible to maintain the state of a variable plus the surrounding functionality as one object (in this case as a list object). (**Motivation**)
- The **participants** of the pattern are the outer function, the enclosed environment, the inner functions sand the state variable.
- Where and when can it be used? For example: for computing a summary value by continuously updating the state variable.
- Calling the Closure function returns a list object with the "publicly" available functions (the green functions). All other inner functions remain private (the red function log()).

Closure example in R

```r
# Closure
newClosureExample <- function(initial=1000) {

    stateVar <- initial

    add <- function(n) {
        stateVar <<- stateVar + n
        log("added ", n)
    }

    subtract <- function(n) {
        stateVar <<- stateVar - n
        log("subtracted ", n)
    }

    # 'private' function within closure
    log <- function(text, n) {
        print(paste(text, n, ", result is ", stateVar, sep=""))
    }

    getValue <- function() { stateVar }


    # Logs initial value and returns available functions from closure.
    print(paste("initial value is ", stateVar))
    list(add=add, subtract=subtract, getValue=getValue)

}
```

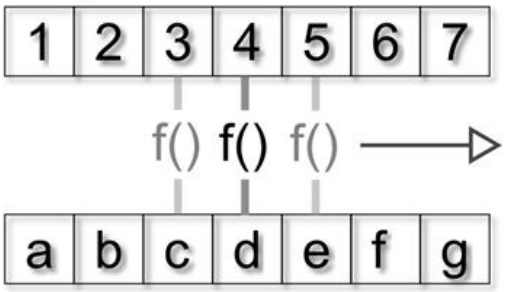
```r
# Closure usage
a <- newClosureExample(1)   # a: intial value 1
b <- newClosureExample()    # b: intial value 1000
a$add(10)                   # a + 10 = 11
a$subtract(3)               # a - 3 = 8
a$getValue()                # a: final value 8
b$add(1)                    # b + 1 = 1001
b$getValue()                # b: final value 1001
```


Using the closure ...

- The closure object is created by calling the closure function ("newClosureExample()")
- For updating the state variable the public functions are called with the „$" operator.

More Examples

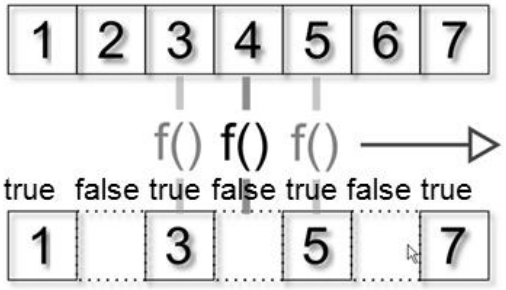
- Map



Friedrich Schuster

More Examples

- Filter



Friedrich Schuster
© 2011 HMS Analytical Software GmbH

Some examples of „Pattern-like" concepts from **functional programming**:

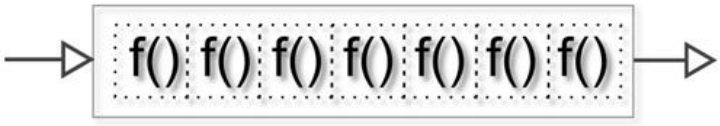
- **(Map**:) You all know the the Map concept: a function is applied to all elements of a vector. It returns a vector of the same size with the results of each function application. This is identical to the „apply()"-family of function in R.
- **(Filter:)** In the Filter concept a function is also applied to all elements of a vector. But it returns only a subset of the original collection. In R, there is a „Filter()" function in the base package „funprog".

Please note: Map() and Filter() are not patterns in a strict sense: because all the functionality is already in the language, and no additional coding is required by the user.

More Examples

HMS Analytical Software

- Compose



Friedrich Schuster

More Examples

HMS Analytical Software

- Chain of responsibility pattern

(same idea, but with classes and methods)

Friedrich Schuster

One more example:

Let's assume that data from different experiments has to be processed regularly. Experiments change often, so the processing has to be flexible and extensible.
Common processing steps exist for all experiments. Other steps are only valid for some of the experiments.
The idea is to pass every data object (no matter what experiment it is from) to an chain of processing steps. The first step receives the data and handles, then forwards it to the next step, which does likewise. Internally every step inspects the data, and only processes it if it's from the right kind of experiment.

Interestingly there are two patterns for this kind of task

- **Compose()** from functional programming: Computes a result by chaining multiple functions together and passing the result of each one to the next. The last result is the final one. This function is implemented in the „roxygen" package.

And the

- „**Chain of responsibility**" pattern from object/oriented programming. It works with classes and methods, not with functions as objects.

So the question is which one to use? Just apply the „KISS" principle: "**keep it short and simple**". (and use Compose(), because it is much simpler to implement in R).

The patterns presented were just a few examples of a larger number of known patterns. Unfortunately there is no pattern catalogue for R (that I am aware of). … If you find one please don't forget to let me know.

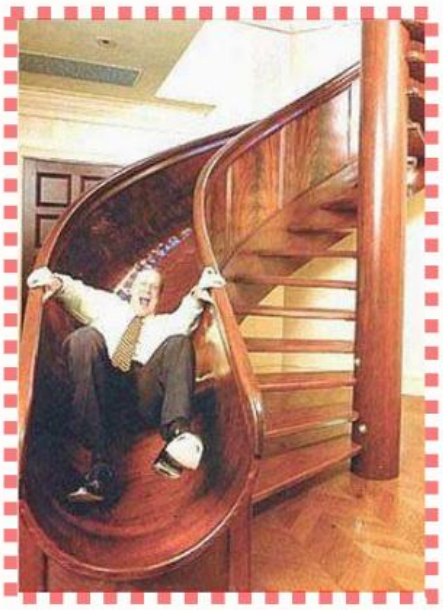
Conclusions

HMS Analytical Software

Please remember:

- **Pattern catalogs** are a great **source of ideas** of how to approach software development problems without re-inventing the wheel in every single case.
- **Pattern names simplify communication** among developers and domain experts.

Thank you!



So back to the initial questions:

Do patterns exist in R? Yes.

Are they important? To my surprise mostly not. At least not for simple programming tasks. With package development the situation changes: package development itself follows a pattern.

In the end using a domain specific language with powerful functional features means that you don't need a lot of patterns for programming (at least not most of the time).

Now if you forget most of what you heard and remember only a two things, let it be this:

1. Patterns and pattern catalogues are a great source of ideas.
2. Using pattern names simplifies communication.

Hope you liked the presentation, and enjoy the conference,

Thank you!

Thank you

HMS Analytical Software

Friedrich Schuster
Dipl. Soz.
Senior Software Engineer

HMS Analytical Software GmbH
Rohrbacher Str. 26 • 69115 Heidelberg
Telephone +49 6221 6051-42
Friedrich.Schuster@analytical-software.de
www.analytical-software.de

Bibliography

HMS Analytical Software

- John M. Chambers (2008): Software for Data Analysis. Programming with R. Springer.
- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (1996): Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software. Addison-Wesley, 1. Auflage 1996, korrigierter Nachdruck
- Peter Norvig (1996): Design Patterns in Dynamic Programming. http://norvig.com/design-patterns/ppframe.htm
- R Development Core Team (2011). R Language Definition. Version 2.13.0 R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-13-5, URL http://www.R-project.org.
- Gregory T. Sullivan (2002): Advanced Programming Lanugage Features for Executable Design Patterns “Better Patterns Through Reflection”. Massachusetts Institiute of Technology, http://sullwood.org/gregs/AIM-2002-005.pdf