

Using biology and ecology as inspiration for software maintenance

Philippe Grosjean <Philippe.Grosjean@umons.ac.be>

University of Mons, Complexys Institute
Numerical Ecology of Aquatic Systems Laboratory (EcoNum)

CSMR-WCRE 2014

complexys

INSTITUT DE RECHERCHE
SUR LES SYSTÈMES COMPLEXES
DE L'UMONS



UMONS

Disclaimer

I am a biologist, not a computer scientist.
My point of view on software maintenance and evolution
may be naive or even, silly...

However, I would be happy if it triggers discussion!



Overview

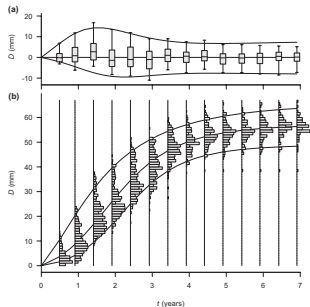
- 1 Who am I?
- 2 Evolution
 - Darwinian evolution of living organisms on earth
 - Alternate evolution mechanisms in living systems
- 3 Ecology
 - Food web
 - Collaboration versus competition
- 4 Life history
 - Case #1: an odd feature, really?
 - Case #2: 'life history' of an R script
- 5 Conclusions

A short presentation of the speaker



Marine biologist, biostatistician and software developer

- **Bioengineer** with a Ph.D. thesis in **marine biology** (growth model of sea urchins)
- Additional skills developed in **(bio)statistics** during post-docs and consultancy during 4 years all around Europe (France, Ireland, Spain, U.K.)
- **EcoNum lab** created in 2004 at UMONS
- Interested by **interdisciplinary work**: biology, chemistry, modelling, statistics, computing science
- Write **software for ecology** in Java, R, ...



Growth model of sea urchins

Ecophysiology of corals in mesocosms

Tropical coral reefs constitute beautiful and diverse ecosystems, but they are endangered by climate changes, overfishing and pollution.

At EcoNum lab, we study how the environment affects growth, reproduction and health (ecophysiology) of tropical corals in artificial reef mesocosms.



Automatic identification of plankton

Plankton (the organisms that drift in the middle of the column water) is a diverse community. There are easily thousands of species in 1L of seawater.

At EcoNum lab, we develop tools to automatically enumerate plankton using image analysis and supervised classification algorithms.

R Console

```
Type rfNews() to see new features/changes
A ZIClass object predicting for 5 classes
[1] "Chaetognatha" "Copepoda"
[5] "Salpida"

Algorithm used: randomForest
Mismatch in classification: 0%
k-fold cross validation error estimation (k = 10):
13.69%

Error per class:
              Error (%)
Copepoda      5.00
Crustacea other 15.28
Chaetognatha  15.79
Salpida       26.32
marine snow   26.47

classes      predicted
01 02 03 04 05
01 Chaetognatha  32  2  3  0  1
02 Copepoda      0 95  4  1  0
03 Crustacea other 0  9 61  2  0
04 marine snow   1  1  3 25  4
05 Salpida       2  0  0  3 14
```

ZoomImage1 assistant

Ready - C:/Zoo/PhytoImage Examples/Scan016-train&data

1.jpg MTLG 2004-10-20.H1+A1_137.jpg

7.jpg MTLG 2004-10-20.H1+A1_187.jpg



Software development

Most of the time (>99%) I am developing **solutions in R**

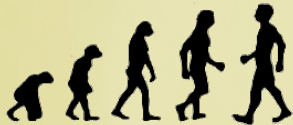
- Author and **maintainer** of 11 R packages
- Main **translator** of R in French
- **SciViews** (GUI), **tinn-r** (Code editor)
- **mlearning** and **zooimage** (machine learning)
- **aurelhy** (multivariate spatial interpolation), ...



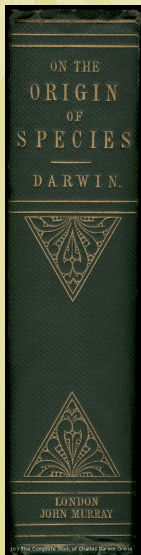
Evolution



Stop following me.



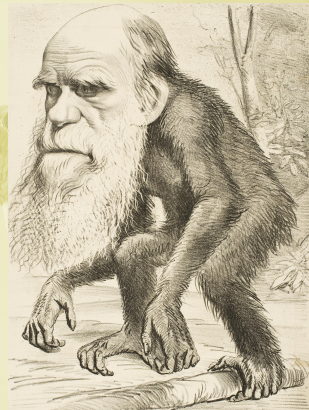
Darwinian evolution



Charles Darwin (1860). "On the origin of species by means of natural selection"

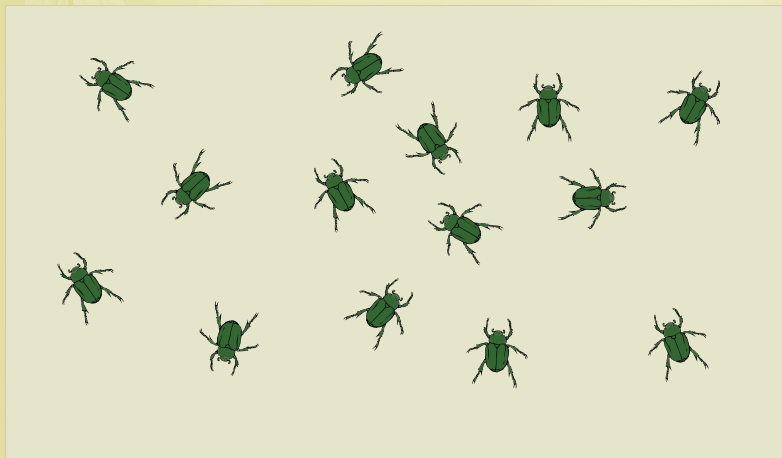
Natural selection is the **major process of evolution** of life on earth.

*(note the famous cartoon at right published in 1871 is **not** a realistic representation of the Master!)*



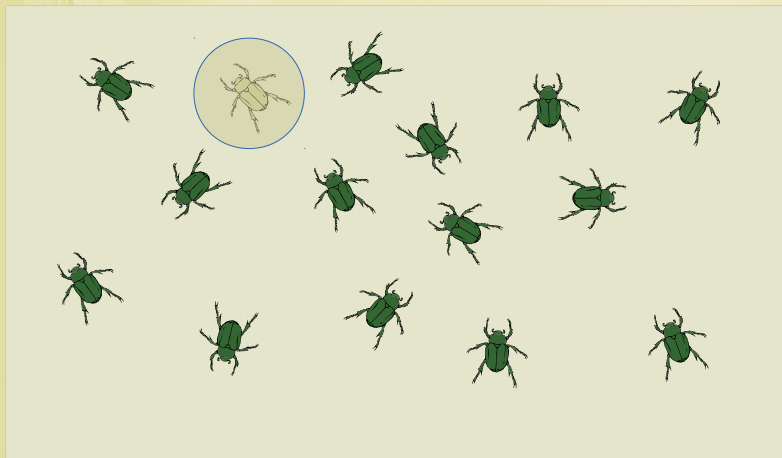
Darwinian evolution

A population of green bugs...



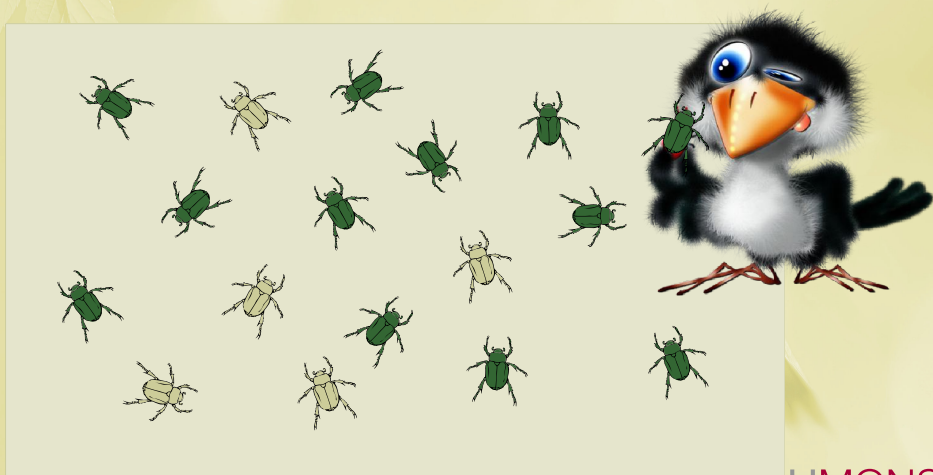
Darwinian evolution

Random mutations from generation to generation, e.g., a colour change.



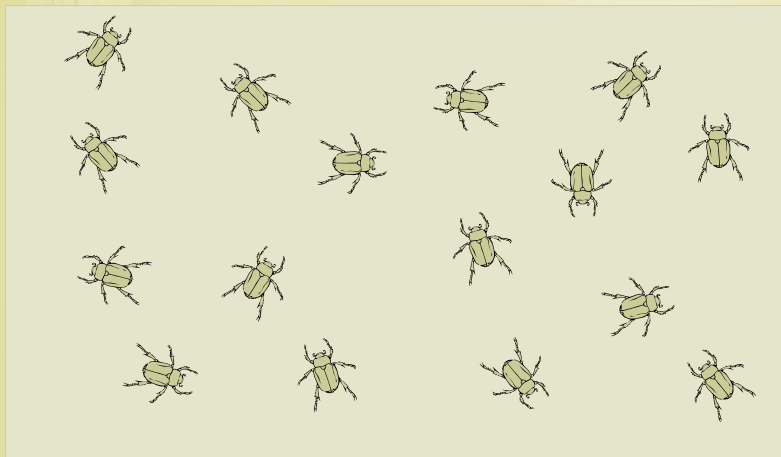
Darwinian evolution

Natural selection. Visual predators detect dark bugs more easily on light background.



Darwinian evolution

If **selective pressure** is strong enough, one variety **can outcompete** another one. It is said to have a better or higher **fitness**.



Self-organisation and self-maintenance

- Natural selection allows **keeping the fittest** from generation to generation.
- Many generations means evolution toward **self-organised complexity!**
- Automatic elimination of “wrong” branches: **self-maintenance!**
- **Recycling of the resources:** dead organisms are decomposed and reused by plants

Would Darwinism apply to software engineering and maintenance?

Providing one could simulate it in an “software evolution sandbox”:

- E.g., using a cluster of computers,
- Successive installs of software; random changes in a few code lines each time
- Selection of the instances that produce something interesting at each step, ...

Do you think this could be an interesting experiment?

Self-organisation and self-maintenance

- Natural selection allows **keeping the fittest** from generation to generation.
- Many generations means evolution toward **self-organised complexity!**
- Automatic elimination of “wrong” branches: **self-maintenance!**
- **Recycling of the resources:** dead organisms are decomposed and reused by plants

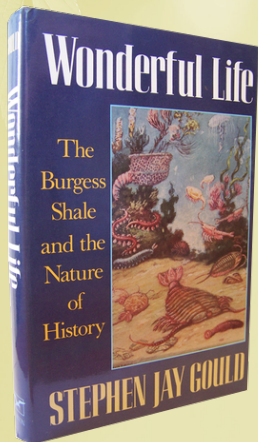
Would Darwinism apply to software engineering and maintenance?

Providing one could simulate it in an “software evolution sandbox”:

- E.g., using a cluster of computers,
- Successive installs of software; random changes in a few code lines each time
- Selection of the instances that produce something interesting at each step, ...

Do you think this could be an interesting experiment?

Self-organisation and self-maintenance applied to software?



Answer: **NO!**

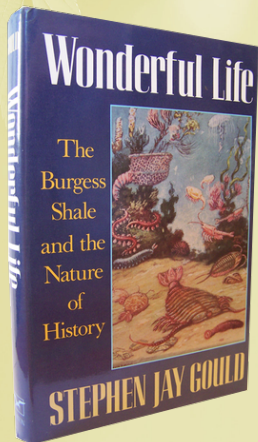
Because...

... you would not have enough computers to get something useful from it in a reasonable time frame!

- Look at these facts:

- Most of life's evolution done by bacteria... man is only "a wildly improbable evolutionary event" (Gould 1989, "Wonderful life", p. 291).
- There are at least 10^{30} individual bacteria on earth (Kallmeyer et al 2012, PNAS).
- You will need a similar number of computers to get enough random combinations, ...
This is simply impossible!

Self-organisation and self-maintenance applied to software?



Answer: **NO!**

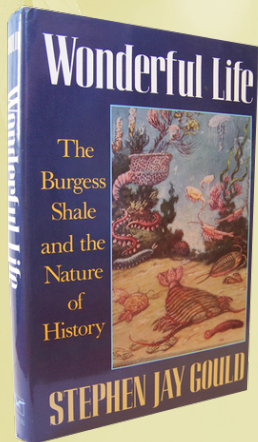
Because...

... you would not have enough computers to get something useful from it in a reasonable time frame!

■ Look at these facts:

- Most of life's evolution done by bacteria... man is only "a wildly improbable evolutionary event" (Gould 1989, "Wonderful life", p. 291).
- There are at least 10^{30} individual bacteria on earth (Kallmeyer et al 2012, PNAS).
- You will need a similar number of computers to get enough random combinations, ...
This is simply impossible!

Self-organisation and self-maintenance applied to software?



Answer: **NO!**

Because...

... you would not have enough computers to get something useful from it in a reasonable time frame!

■ Look at these facts:

- Most of life's evolution done by **bacteria**... man is only "a wildly improbable evolutionary event" (Gould 1989, "Wonderful life", p. 291).
- There are at least 10^{30} **individual bacteria** on earth (Kallmeyer et al 2012, PNAS).
- You will need a similar number of computers to get enough random combinations, ...
This is simply impossible!

How could this be useful for software development? (I)

Genetic algorithms apply this technique for optimisation, but usually with a much more limited set of parameters to vary.

The critical step here is the *random mutation*. So, it could work if it is:

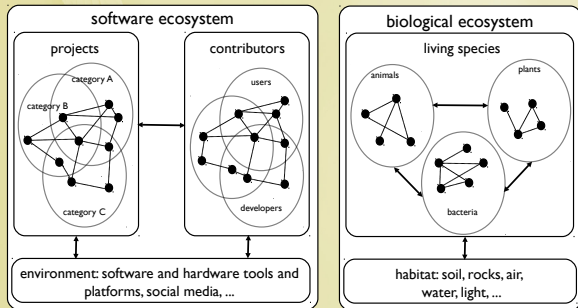
- limited to the portion of code that produces failure (self-adaptative software systems, Müller et Villega 2014),
- replaced by a set of schemes or patterns known to work in various circumstances,
- replaced by a mechanisms that generate diversification of software automatically (see the *diversify* project in the conference)
- ...

How could this be useful for software development? (II)

Comprehension of software ecosystems could also benefit from the study of biological systems considering:

- Many complex systems share some **common emergent properties**
- **Analogies** exist in software and biological ecosystems components

See presentations in the *Project Track* session, such *ECOS (Ecological Studies of Open Source Software Ecosystems)*



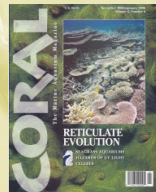
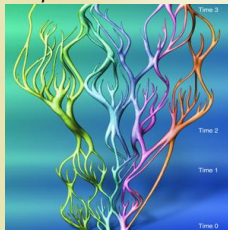
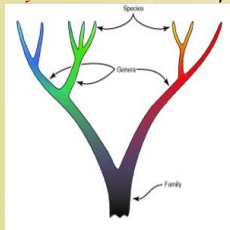
Reticulate evolution theory

- Evolution leads to **speciation** (individuals of different species cannot reproduce, or cannot produce fertile descendants) leading to the **tree of life**

Software: few forked projects merge subsequently (Robles et González-Barahona, OSS 2012): it also applies to software. *Difference:* forking is often considered a bad thing for software!

- Sometimes, **hybridisation becomes possible** again (e.g., geographical barrier that disappears), leading to the **merge** of some branches. This is demonstrated, for instance, for corals (Veron 1995)

Software: GITHUB repositories promote reticulate evolution through **fork and merge**



Darwinian evolution (left) versus reticulate evolution (right), from Veron 1995.

Holobionts and lateral gene transfert

- There are many other mechanisms allowing adaptation of living organisms:
 - An **holobiont** is constituted by two or more species that live together in close mutually beneficial interactions (so-called, **symbionts**). Ex.: lichen are made of algae and fungi. It gives them ecological advantages.

Software:

Assemblage of large building blocks is a common practice in software engineering

- Genes can be transferred from one individual (species) to another one by different mechanisms, including by means of viruses. This is called “**lateral (or horizontal) gene transfer**”. It is common in bacteria.

Software:

This matches reuse of smaller building blocks across software projects

Co-evolution

- Co-evolution of two biological systems occurs when there is a dependence between them. For instances, hummingbirds and flowers.



- Co-evolution occurs for software too, as you know: database schema *versus* code for the analysis (Goeminne & Mens 2013).
- Another example: servers and clients (see diversify talk).

Ecology



Definitions

Ecosystem (biological):

"... includes all living organisms in an area as well as its physical environment functioning together as a unit."

Value: fitness, i.e., potential to produce viable offspring. **Resources:** food.

- **Ecosystem (software, business-oriented):** *"... actors functioning as a unit and interacting with a shared market for software and services..."* (Jansen et al 2009).

Value: money (directly or indirectly). **Resources:** manpower + money.

- **Ecosystem (software, socio-technical):** *"a collection of software projects that are developed and evolve together in the same environment."* (Lungu 2008).

Value: code/software fitness (quality, maintainability, ...). **Resources:** developers.

Different definitions!

Different points-of-view lead to different definitions of ecosystem. We adhere to the socio-technical definition, closer to the biological definition.

Definitions

Resilience (biological):

“Capacity of an ecosystem or an organism to recover from a perturbation”.

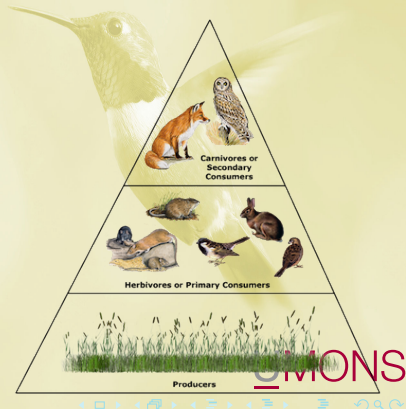
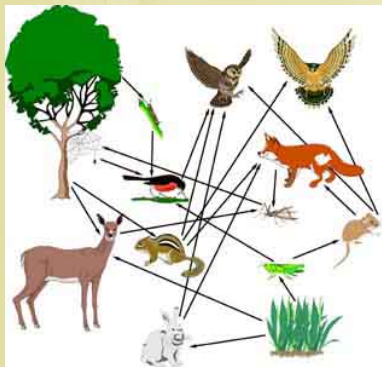
Biodiversity is generally associated with a higher resilience.

- **Resilience (software):** *“ability of a software to correctly handle unanticipated exceptions.”* (Corny et al 2013).

This definition is much narrower than the biological one!

Food web

- A **food web** (or trophic web) is a diagram linking **preys to their predators**
- At the bottom are the **producers**: plants transforming minerals into organic matter
- Above are the **consumers**, including **predators**. They form successive **trophic levels**
- The food web is summarised by stacking trophic levels inside a **pyramid**



Top-down, bottom-up and wasp-waist control

- Dynamics of the ecosystems are driven by trophic levels that **control others**
- If resources are limited, so are producers and we have a **bottom-up control**
- If predators are efficient, they limit lower trophic levels: this the **top-down control**
- **Wasp-waist controlled** ecosystems exhibit both controls simultaneously

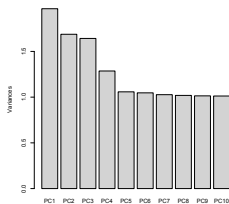
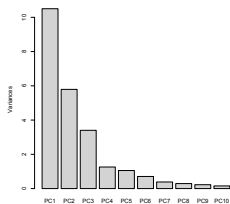
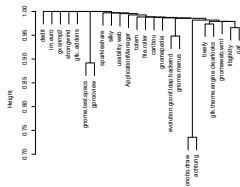
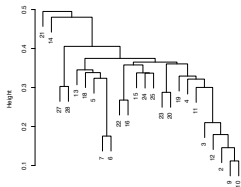
What about software ecosystems?

Producers are code **developers** and **consumers** are software **users**. Who controls how software evolves?

Developers produce the code, but what about the impact of users' feedback?

- Lot of data available about **code and developers of OSS**.
- Lack of information on the **role of users** in software development.
This requires to analyse bug tracking, mailing lists, forums, stackoverflow, etc.

Collaboration versus competition



- At left, 24 random locations described by their vegetal communities; at right, 24 random Gnome projects described by their developers communities
- Hypothesis: such differences occur because there is much more competition in biological system and collaboration in software system

More (sane) competition in software ecosystems?

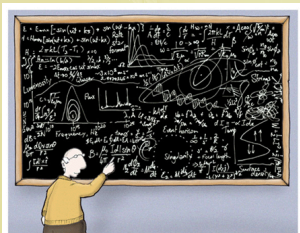
Competition is a key mechanism for the evolution of biological ecosystem (Darwinism)

Question:

Wouldn't it be nice to have a little bit more competition in software ecosystems?

- Competition should be “**sane**” (based on code/software quality), not on commercial or juridic strategies. Easier with OSS.
- **Machine learning used to model GUI according to user behaviour** (Mezhoudi & Vanderdonckt 2013). Couldn't this be adapted to other software components?
- **Suggestion:** meta-software that would connect users request to several possible implementations, with learning which one is best in such or such situation.
- **Feedback to developers:** kind of life contest.

Life history: the R case



Statistics made easy with R

UMONS

Life history / program history



- **Life history** is age, size, development, maturation, reproduction, survival and lifespan of an organisms
- An **adult living organism** does not tell everything about its fitness: all life stages **must be analysed too**

Adult mayflies

are fragile insects that cannot eat and die in a single day. *However, their immature stages are voracious aquatic animals that last for up to four years for certain species!*

- One could consider a '**program history**', including all stages from initial design to debugging or refactoring of a software
- Similarly, **analysing code out-of-context** may not tell everything about it

Life history / program history



- **Life history** is age, size, development, maturation, reproduction, survival and lifespan of an organisms
- An **adult living organism** does not tell everything about its fitness: all life stages **must be analysed too**

Adult mayflies

are fragile insects that cannot eat and die in a single day. *However, their immature stages are voracious aquatic animals that last for up to four years for certain species!*

- One could consider a '**program history**', including all stages from initial design to debugging or refactoring of a software
- Similarly, **analysing code out-of-context** may not tell everything about it

Life history / program history



- **Life history** is age, size, development, maturation, reproduction, survival and lifespan of an organisms
- An **adult living organism** does not tell everything about its fitness: all life stages **must be analysed too**

Adult mayflies

are fragile insects that cannot eat and die in a single day. *However, their immature stages are voracious aquatic animals that last for up to four years for certain species!*

- One could consider a '**program history**', including all stages from initial design to debugging or refactoring of a software
- Similarly, **analysing code out-of-context** may not tell everything about it

R, what a strange language?

R is...

“a language and environment for statistical computing and graphics”
 inspired by the S language (Chambers et al, 1976):
 it is designed **“to turn ideas into software, quickly and faithfully”**.



- R is a success story in Open Source communities (e.g. TIOBE) and the *lingua franca* for statisticians.
- There are more than 5000 contributed packages to CRAN (see Claes et al in ERA Track session 1).
- Yet, language design specialists (Morandat et al 2012) consider that: *“this rather unlikely linguistic cocktail would probably never have been prepared by computer scientists.”*
- It is largely inefficient in memory usage and in performances.
 So what makes it so successful?

R, what a strange language?

R is...

“a language and environment for statistical computing and graphics”
 inspired by the S language (Chambers et al, 1976):
 it is designed “**to turn ideas into software, quickly and faithfully**”.



- R is a **success story** in Open Source communities (e.g., TIOBE) and the *lingua franca* for statisticians.
- There are more than 5000 contributed packages to CRAN (see *Claes et al in ERA Track session 1*).
- Yet, language design specialists (Morandat et al 2012) consider that: “*this rather unlikely linguistic cocktail would probably never have been prepared by computer scientists.*”
- It is largely inefficient in memory usage and in performances.
 So what makes it so successful?


R, what a strange language?

R is...

“a language and environment for statistical computing and graphics”
 inspired by the S language (Chambers et al, 1976):
 it is designed “**to turn ideas into software, quickly and faithfully**”.



- R is a **success story** in Open Source communities (e.g., TIOBE) and the *lingua franca* for statisticians.
- There are more than 5000 contributed packages to CRAN (see Claes et al in *ERA Track session 1*).
- Yet, language design specialists (Morandat et al 2012) consider that: “*this rather unlikely linguistic cocktail would probably never have been prepared by computer scientists.*”
- It is **largely inefficient in memory usage and in performances.**
 So what makes it so successful?



R studied as a living organism
Case #1: an odd feature, really?

A simple, odd feature...

Considering **c** is the name of a function
that concatenates its arguments into a vector...

```
c = c(1, 4, 3, 2)
```

```
d = c(5, c)
```

What do you think about this code?

'c()' versus 'c'

In R language, `c` can be the name of a vector and of a function **at the same time!**

```
R> c <- c(1, 4, 3, 2)
```

```
R> c
```

```
[1] 1 4 3 2
```

```
R> d <- c(5, c)
```

```
R> d
```

```
[1] 5 1 4 3 2
```

```
R> get("c", mode = "function")
```

```
function (... , recursive = FALSE) .Primitive("c")
```

What do you think about it?

- Silly, unreadable, ambiguous, dangerous!
- This is, *at best*, error-prone!

A little bit of psychology...

THINKING,
FAST AND SLOW



DANIEL
KAHNEMAN

WINNER OF THE NOBEL PRIZE IN ECONOMICS

“Thinking fast and slow”, Daniel Kahneman (2011):
We have **two modes of thought**, so-called
System 1 and *System 2*.

System 1

is fast and instinctive and is **very efficient in contextual associations**.

System 2

is slow, requires considerably more energy but is **more logical**.

A little bit of psychology...

THINKING,
FAST AND SLOW



DANIEL
KAHNEMAN

WINNER OF THE NOBEL PRIZE IN ECONOMICS

“Thinking fast and slow”, Daniel Kahneman (2011):
We have **two modes of thought**, so-called
System 1 and *System 2*.

System 1

is fast and instinctive and is **very efficient in contextual associations**.

System 2

is slow, requires considerably more energy but is **more logical**.

A little bit of psychology...

THINKING,
FAST AND SLOW



DANIEL
KAHNEMAN

WINNER OF THE NOBEL PRIZE IN ECONOMICS

“Thinking fast and slow”, Daniel Kahneman (2011):
We have **two modes of thought**, so-called
System 1 and *System 2*.

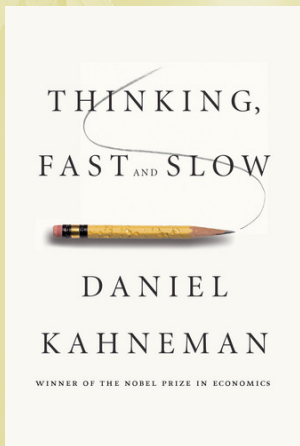
System 1

is fast and instinctive and is **very efficient in contextual associations**.

System 2

is slow, requires considerably more energy but is **more logical**.

A little bit of psychology...



First example

Imagine your cat and your best friend are both called 'Tom', and you say:

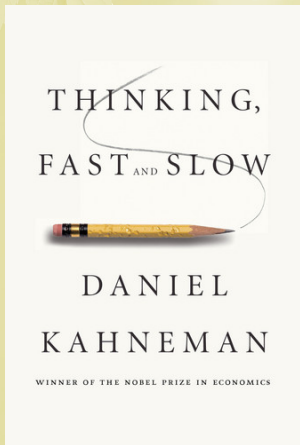
- I use to go for a beer with Tom on Saturday night.
 - Tom laps up its milk and purrs with pleasure.
 - Everybody knows who is who, effortlessly.
- Your System 1 does that for you.*

Second example

What makes:

- 24×36
 - It's a lot longer and painful to figure it out.
- Your System 2 is operating here.*

A little bit of psychology...



First example

Imagine your cat and your best friend are both called 'Tom', and you say:

- I use to go for a beer with Tom on Saturday night.
- Tom laps up its milk and purrs with pleasure.
- Everybody knows who is who, effortlessly.

Your System 1 does that for you.

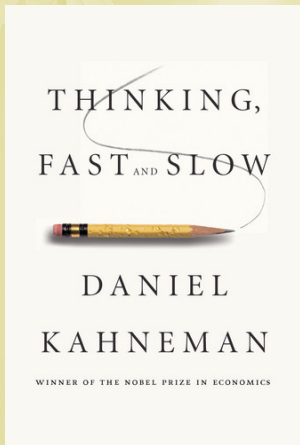
Second example

What makes:

- 24×36
- It's a lot longer and painful to figure it out.

Your System 2 is operating here.

A little bit of psychology...



First example

Imagine your cat and your best friend are both called 'Tom', and you say:

- I use to go for a beer with Tom on Saturday night.
- Tom laps up its milk and purrs with pleasure.
- Everybody knows who is who, effortlessly.

Your System 1 does that for you.

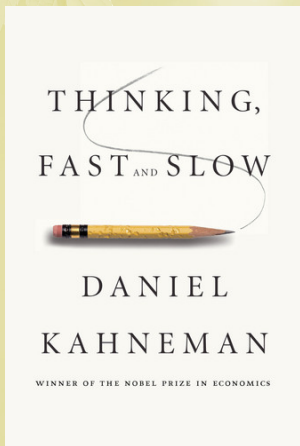
Second example

What makes:

- 24×36
- It's a lot longer and painful to figure it out.

Your System 2 is operating here.

A little bit of psychology...



First example

Imagine your cat and your best friend are both called 'Tom', and you say:

- I use to go for a beer with Tom on Saturday night.
- Tom laps up its milk and purrs with pleasure.
- Everybody knows who is who, effortlessly.

Your System 1 does that for you.

Second example

What makes:

- 24×36
- It's a lot longer and painful to figure it out.

Your System 2 is operating here.

A little bit of psychology...

Now, look at this (Kahneman 2011):

A B C

1 2 3 1 4

You read 'ABC' and '121314', isn't it?

A little bit of psychology...

Now, look at this (Kahneman 2011):

A B C

1 2 3 4

You read 'ABC' and '12314', isn't it?

A little bit of psychology...

Note that the central part is **identical**.

ABC

121314

Your *System 1* is really quick and excellent on contextual interpretation!

A little bit of psychology...

Note that the central part is **identical**.

ABC

121314

Your *System 1* is really quick and excellent on contextual interpretation!

'c()' versus 'c', revisited

- As **computer scientist or engineer**, you **strongly reject** possible use of the same name for different objects in the same scope.
- *On the other hand:*
 - **Interactive use** of a language needs **short names**, and these are **limited resources**.
 - The human brain **easily discriminates** between the function **c()** and the vector **c** according to the context.

```
R> c <- 1:5          # The 'c' vector
R> d <- c(1, 4, 3)  # The 'c()' function
R> e <- c(1, 4, c)  # Both of them
```

- **R does the same**; **c** is available for a variable, although already used for a function. *Considering the **whole context** (interactive use, the way human brain works, ...) can change perception of what's right and what's wrong!*

Evolution, ecology and 'c()' versus 'c'

Biology could lead to more ad hoc and pragmatic design

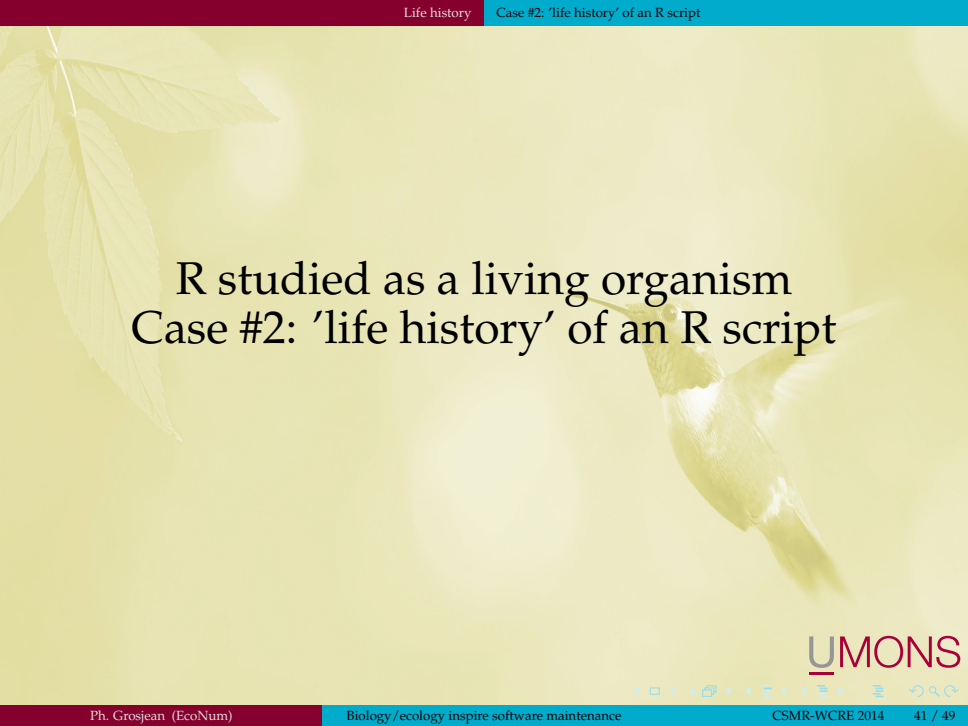
- Although clearly suboptimal, simultaneous use of `c` and `c()` works for both human reader and R parser: **it is a possible feature**
- It gives a **slight advantage** in a specific context: **interactive use**

That is enough to make it workable!

This is exactly what happens in evolution and ecology

Any globally workable solution, **even suboptimal from a given point of view**, can be an evolutionary/ecologically advantage.

Shouldn't language and software design consider a little bit more the **whole context**?
Not only **pure performances and functionalities**?



R studied as a living organism

Case #2: 'life history' of an R script

The software “wars”

There has been always software “wars”:

- OSes; Windows against Mac OS against Unix/Linux
- Web browsers
- Office suites
- Text editors
- ...

... and now, there is the **data analysis software war** with **Python** *versus* **R** *versus* **Julia** *versus* ...

Python versus R versus Julia

Current debate about which language is better for *data analysis* on the Internet.

- Most of the arguments are about **code performances**
(table from <http://julialang.org> with times relative to C code).

	Fortran gcc 4.8.1	Julia	Python	R	...
fib	0.26	0.91	30.37	411.36	
parse_int	5.05	1.60	13.95	59.40	
quicksort	1.11	1.14	31.98	524.29	
mantel	0.86	0.85	14.19	106.97	
pi_sum	0.80	1.00	16.33	15.42	
rand_mat_stat	0.64	1.66	13.52	10.84	
rand_mat_mult	0.96	1.01	3.41	3.98	

- Yes, Python and Julia can be must faster than R.
- However, R strength is in **speed for turning ideas into software**,
that includes time to develop the code too!

Write once use many times... or use once?

- Most software are **written (and compiled) once to be used many times**.

E.g., a web browser

- In that case, time and effort in developing and optimising code for **maximum performances** always pays.
- A language that **optimise speed and memory usage** is suited here (C/C++, Java, ...)

- Sometimes, code is **written once to be used once** (or reused a few times only).

E.g., script for a statistical analysis or graph

- Time and effort required to write code counts as much as its final performances
- A language optimised to "turn ideas into software, quickly and faithfully" (Chambers) should be used instead, like R

Analysis of R code speed

out-of-context. Trying to compare R with languages designed for a different usage is like comparing apples and oranges.

Write once use many times... or use once?

- Most software are **written (and compiled) once to be used many times**.
E.g., a web browser
 - In that case, time and effort in developing and optimising code for **maximum performances** always pays.
 - A language that **optimise speed and memory usage** is suited here (C/C++, Java, ...)
- Sometimes, code is **written once to be used once** (or reused a few times only).
E.g., script for a statistical analysis or graph
 - **Time and effort** required to write code counts as much as its **final performances**
 - A language **optimised to "turn ideas into software, quickly and faithfully"** (Chambers) should be used instead, like R

Analysis of R code speed

out-of-context. Trying to compare R with languages designed for a different usage is like comparing apples and oranges.

Write once use many times... or use once?

- Most software are **written (and compiled) once to be used many times**.

E.g., a web browser

- In that case, time and effort in developing and optimising code for **maximum performances** always pays.
- A language that **optimise speed and memory usage** is suited here (C/C++, Java, ...)

- Sometimes, code is **written once to be used once** (or reused a few times only).

E.g., script for a statistical analysis or graph

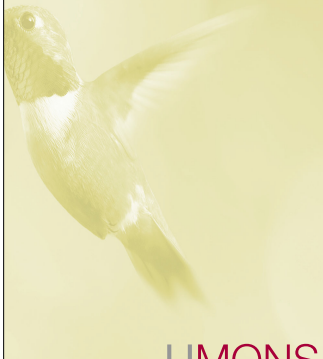
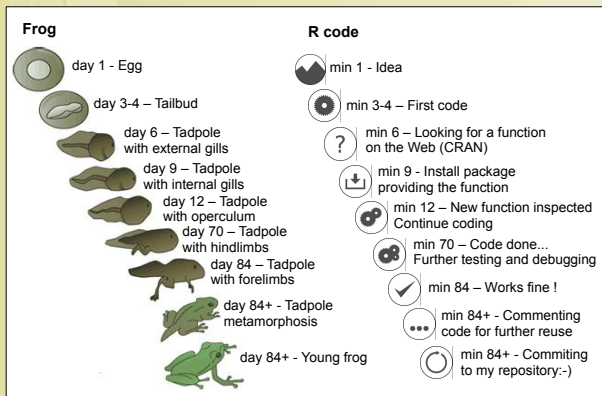
- **Time and effort** required to write code counts as much as its **final performances**
- A language **optimised to "turn ideas into software, quickly and faithfully"** (Chambers) should be used instead, like R

Analysis of R code speed

out-of-context. Trying to compare R with languages designed for a different usage is like comparing apples and oranges.

An R code history

- **Time and effort** required to write code counts as much as its **performances** in the *write once, use once* (or few) strategy
- It is like, say, analysing **life history of the tadpole** when looking at a **frog**



'Programming history' of a software

- Quantification of **code qualities** (speed, memory usage, modularity, good or bad patterns, etc.) is feasible
- **Quantification of tasks related** to the code (coding, debugging, maintenance effort) is perhaps more difficult

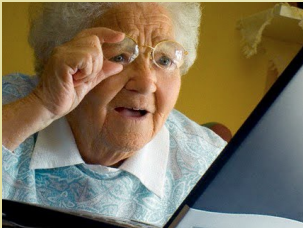
How to evaluate if language X allows to faster develop a working software than language Y, given this is strongly biased by the knowledge and programming habits of the programmer?

Code analysed in its development context

is especially important for *write once use once* strategy.
This is very close to the analysis of **life history** in biology.

Big question: *maintainability of code written with a language like R*
(see M. Claes presentation in **ERA Track** session 1)

Conclusions



Conclusions

From a naive look at software engineering through the eyes of a biologist, here are my questions:

- 1 Software engineering **cannot use Darwinian mechanisms *as is***. But there are many similarities between other biological and software evolution mechanisms. *Does it means potentials inspirations for getting more self-organising and/or self-maintained systems?*
- 2 There is a relative lack of data on users impact on OSS development, success and survival. It is hard to evaluate impact of top-down *versus* bottom-up control. *Works needs to be done there?*
- 3 *More competition?* Biological systems would suggest that, used wisely, it could perhaps help improve software
- 4 Multidisciplinary study of the **enlarged context** (social, economic, psychologic, etc.) of code or languages features may bring some interesting insights. *Kind of study of 'life history' of your code?*

Conclusions

From a naive look at software engineering through the eyes of a biologist, here are my questions:

- 1 Software engineering **cannot use Darwinian mechanisms** *as is*. But there are **many similarities** between other biological and software evolution mechanisms. *Does it mean potentials inspirations for getting more self-organising and/or self-maintained systems?*
- 2 There is a relative lack of data on users impact on OSS development, success and survival. It is hard to evaluate impact of top-down *versus* bottom-up control. *Works needs to be done there?*
- 3 *More competition?* Biological systems would suggest that, used wisely, it could perhaps help improve software
- 4 Multidisciplinary study of the **enlarged context** (social, economic, psychologic, etc.) of code or languages features may bring some interesting insights. *Kind of study of 'life history' of your code?*

Conclusions

From a naive look at software engineering through the eyes of a biologist, here are my questions:

- 1 Software engineering **cannot use Darwinian mechanisms** *as is*. But there are **many similarities** between other biological and software evolution mechanisms. *Does it means potentials inspirations for getting more self-organising and/or self-maintained systems?*
- 2 There is a relative **lack of data on users impact** on OSS development, success and survival. It is hard to evaluate impact of top-down *versus* bottom-up control. *Works needs to be done there?*
- 3 *More competition?* Biological systems would suggest that, used wisely, it could perhaps help improve software
- 4 *Multidisciplinary study of the enlarged context* (social, economic, psychologic, etc.) of code or languages features may bring some interesting insights. *Kind of study of 'life history' of your code?*

Conclusions

From a naive look at software engineering through the eyes of a biologist, here are my questions:

- 1 Software engineering **cannot use Darwinian mechanisms** *as is*. But there are **many similarities** between other biological and software evolution mechanisms. *Does it means potentials inspirations for getting more self-organising and/or self-maintained systems?*
- 2 There is a relative **lack of data on users impact** on OSS development, success and survival. It is hard to evaluate impact of top-down *versus* bottom-up control. *Works needs to be done there?*
- 3 *More competition?* Biological systems would suggest that, used wisely, it could perhaps help improve software
- 4 Multidisciplinary study of the **enlarged context** (social, economic, psychologic, etc.) of code or languages features may bring some interesting insights. *Kind of study of 'life history' of your code?*

Conclusions

From a naive look at software engineering through the eyes of a biologist, here are my questions:

- 1 Software engineering **cannot use Darwinian mechanisms** *as is*. But there are **many similarities** between other biological and software evolution mechanisms. *Does it mean potentials inspirations for getting more self-organising and/or self-maintained systems?*
- 2 There is a relative **lack of data on users impact** on OSS development, success and survival. It is hard to evaluate impact of top-down *versus* bottom-up control. *Works needs to be done there?*
- 3 *More competition?* Biological systems would suggest that, used wisely, it could perhaps help improve software
- 4 **Multidisciplinary study of the enlarged context** (social, economic, psychologic, etc.) of code or languages features may bring some interesting insights. *Kind of study of 'life history' of your code?*

Ideas developed in the **ECOS (Ecological Studies of Open Source Software Ecosystems)** project at UMONS.

- **Ph.D. student** position available immediately.
- **Post-doc** (6-12 months) and **visiting scientists** fundings available.

Thank you for your attention...
Any question?

