

Good Practices in (Statistical) Software Development

Alex Sánchez-Pla and Pol Castellano-Escuder

Genetics, Microbiology and Statistic Department. UB
Statistics and Bioinformatics Unit. VHIR
Grup de Recerca en Bioestadística i Bioinformàtica (GRBio)

Jan 26, 2019 - BIOSTATNET Meeting

http://github.com/pcastellanoescuder/Good_Practices_in_Software_Development

Readme

- License: Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License <http://creativecommons.org/licenses/by-nc-sa/4.0/>
- You are free to:
 - **Share** : copy and redistribute the material
 - **Adapt** : rebuild and transform the material
- Under the following conditions:
 - **Attribution** : You must give appropriate credit, provide a link to the license, and indicate if changes were made.
 - **NonCommercial** : You may not use this work for commercial purposes.
 - **Share Alike** : If you remix, transform, or build upon this work, you must distribute your contributions under the same license to this one.

Outline

- Introduction to the session.
- Statistics and Programming
 - The role of coding in statistical work
 - Teaching programming to statisticians and data scientists
- Can we build better software (software better?)
 - When do we call a piece of software “good”?
 - Which aspects should we account for
- Good practices for statistical software development


Introduction

- Presenter: Alex Sánchez Pla
- Job 1: Associate Professor. Statistics department, UB.
“Traditional approach”
 - Teach statistics (SI, ED), make research (GRBio)
 - Mostly programmed for myself. Some R packages.
- Job 2: Head of Statistics and Bioinformatics Unit. “Real world”
 - Coordinate (more than do) applied biostatistics and bioinformatics.
 - Lots of data, lots of programming
 - Discovered new concepts: Validation, Regulation, SOPs, Best Practices, ...

What is software development

- *Software development is a process by which standalone or individual software is created using a specific programming language. It involves writing a series of interrelated programming code, which provides the functionality of the developed software*
- Simple, yet functional. It embraces most programming activities we may be involved in,
 - From writing simple scripts, automating functions, running simulations.
 - Creating web programs, mobile apps, etc.
 - But also writing code for performing any type of statistical analyses.



How do we do software development?



Encuesta sobre desarrollo de software

1 Resultados Encuesta

2 Otras consideraciones

 Miriam Mota-Foix & Alex Sánchez
Unitat d'Estadística i Bioinformàtica - Vall d'Hebron Institut de Recerca (VHIR)
ueb@vhir.org
 UEB
25 enero, 2019



UNITAT
D'ESTADÍSTICA I
BIOINFORMÀTICA



Vall d'Hebron
Institut de Recerca



Encuesta sobre desarrollo de software

Miriam Mota-Foix & Alex Sánchez
Unitat d'Estadística i Bioinformàtica - Vall d'Hebron Institut de Recerca (VHIR)

25 enero, 2019

VHIR-UEB-FOR-013v.01

1 Resultados Encuesta

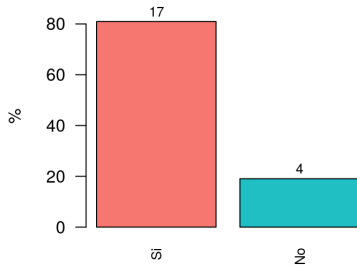
Se han obtenido un total de 21 respuestas.

¿Programas regularmente?

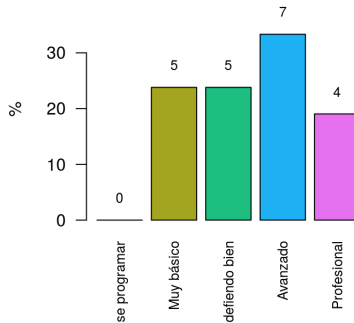
¿Como definirías tu nivel de programación?

Survey answers (1)

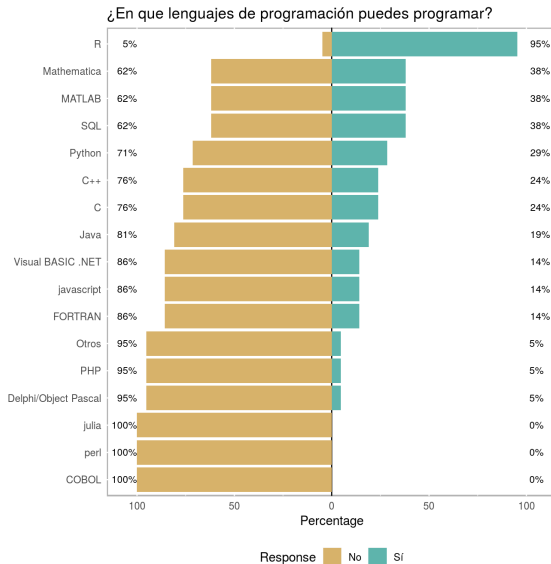
¿Programas regularmente?



¿Como definirías tu nivel de programación?

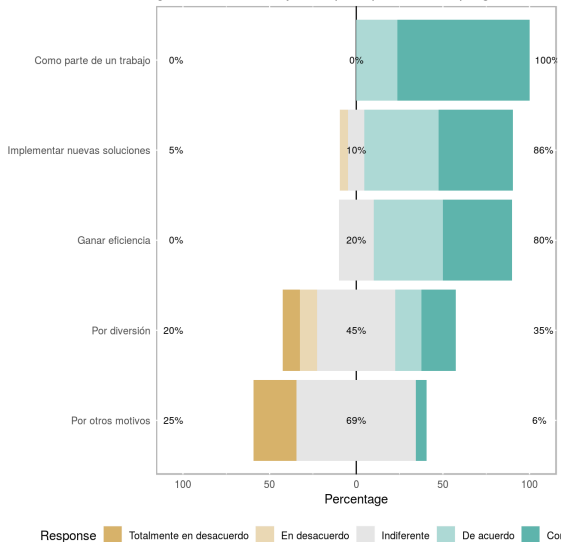


Survey answers (2)

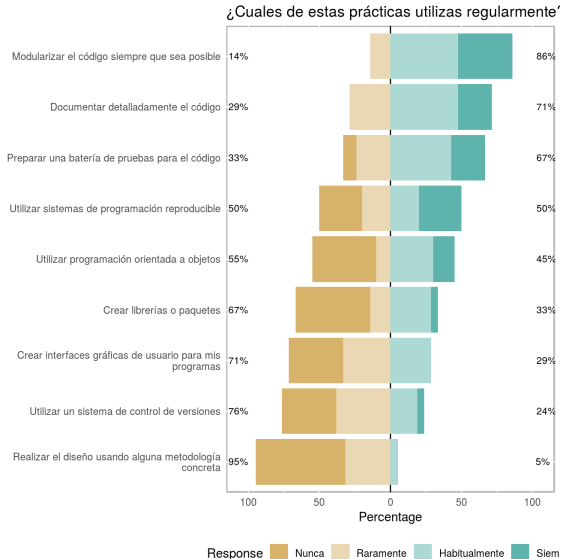


Survey answers (3)

¿Cuáles son los objetivos principales de tus programas?



Survey answers (4)



How important is programming in a mathematician's training?

1r Curs - Fase Inicial			
Càlcul en una variable 	Àlgebra lineal 	Informàtica 	Fonaments de la matemàtica 
Càlcul diferencial 	Geometria afi i euclidiana 	Àlgebra lineal numèrica 	Matemàtica discreta 
2n Curs			
Càlcul integral 	Àlgebra multilinear i geometria 	Algorísmia 	Programació matemàtica 
Funcions de variable complexa 	Topologia 	Física 	Anàlisi real 
3r Curs			
Equacions diferencials ordinàries 	Estructures algebraïques 	Càlcul numèric 	Teoria de la probabilitat 
Equacions en derivades parcials 	Geometria diferencial 	Models matemàtics de la física 	Estadística 
4t Curs			
Models matemàtics de la tecnologia 	Optativa 1	Optativa 2	Optativa 3
Treball de fi de grau	Optativa 4	Optativa 4	Optativa 6

Figure 1: Compulsory courses. Mathematics degree. UPC. (One course)

How important is programming in a statistician's training?

1r curs (UAB)	2n curs (UAB)
<ul style="list-style-type: none">- Càlcul- Estadística Descriptiva (1) (4)- Mètodes Algebraics per a l'Estadística- Introducció a la Programació (2)- Eines Informàtiques per a l'Estadística (1) (6)- Càlcul de Probabilitats- Estructures de Dades i Algorismes (2)- Programació Lineal- Modelització Matemàtica- Temes de Biociència i Ciència (anual) (1) (3) (4) (5) (6)	<ul style="list-style-type: none">- Distribucions Multidimensionals- Optimització i Processos Estocàstics- Bases de Dades (2) (3)- Inferència Estadística I- Estadística Oficial i Disseny d'Enquestes- Disseny d'Experiments- Models Lineals- Mostreig Estadístic- Inferència Estadística II- Anàlisi de la Supervivència i Fiabilitat
3r curs (UAB)	4t curs (UAB)
<ul style="list-style-type: none">- Anàlisi de Dades Categòriques- Anàlisi Multivariant- Sèries Temporals i Predicció- Aplicacions de l'Estadística a la Bioinformàtica- Aplicacions de l'Estadística a les Ciències de la Salut- Minería de Dades- Modelització Avançada- Simulació, Remostreig i Aplicacions- Introducció a l'Econometria- Control de Qualitat i Estadística Industrial	<ul style="list-style-type: none">- Treball de Final de Grau

Figure 2: Compulsory courses. Applied Statistics degree at UAB. (Three courses)

How important is programming in a data scientist's training?

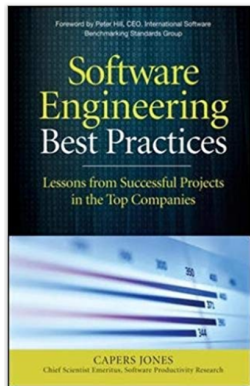
Primer cuatrimestre <ul style="list-style-type: none">ÁlgebraAlgoritmia y Programación 1CálculoLógica y Matemática Discreta	Tercer cuatrimestre <ul style="list-style-type: none">Algoritmia y Programación 3Bases de DatosProbabilidad y Estadística 2Señales y SistemasTeoría de la Información
Segundo cuatrimestre <ul style="list-style-type: none">Álgebra y Cálculo AvanzadosAlgoritmia y Programación 2ComputadoresProbabilidad y Estadística 1	Cuarto cuatrimestre <ul style="list-style-type: none">Análisis de DatosAprendizaje Automático 1Introducción al Procesado AudiovisualOptimización MatemáticaParalelismo y Sistemas Distribuidos
	Quinto cuatrimestre <ul style="list-style-type: none">Aprendizaje Automático 2Bases de Datos AvanzadasBúsqueda y Análisis de la InformaciónEmprendimiento e InnovaciónVisualización de la Información

Figure 3: Compulsory courses. Data Engineering degree. UPC. (Five courses)

Good practices in (statistical) software development

- *A good practice is not only a practice that is good, but a practice that has been proven to work well and produce good results, and is therefore recommended as a model.*
- There is no universally accepted list of “good practices” for software development but we can easily agree (can we?) on some practices that (maybe) can help our programs to be better.
- When we talk about software, we must bear in mind that there are many different types of software and that, sometimes, “good practices” can be different between software types. Some types of software could be: CLI (Command Line Interface), GUI (Graphical User Interface), Scripts, Packages or Web Apps.

Best practices in Software Engineering



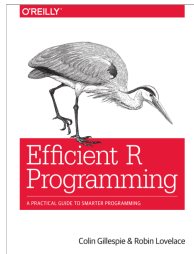
Contents at a Glance

Chapter 1. Introduction and Definitions of Software Best Practices	1
Chapter 2. Overview of 50 Software Best Practices	39
Chapter 3. A Preview of Software Development and Maintenance in 2049	177
Chapter 4. How Software Personnel Learn New Skills	227
Chapter 5. Software Team Organization and Specialization	275
Chapter 6. Project Management and Software Engineering	351
Chapter 7. Requirements, Business Analysis, Architecture, Enterprise Architecture, and Design	437
Chapter 8. Programming and Code Development	489
Chapter 9. Software Quality: The Key to Successful Software Engineering	555
Index	645

- We don't discuss this!

Good practices in Statistical Software Development

- Instead I have browsed through some materials which focus on statistical software development



Good Practices in R Programming

Martin Mächler

maechler@project.org
The R Core Team

maechler@stat.math.ethz.ch
Seminar für Statistik
ETH Zurich, Switzerland

useR! – July 1, 2014

Google's R Style Guide

R is a high-level programming language used primarily for statistical computing and graphics. The goal of the R Programming Style Guide is to make our R code easier to read, share, and verify. The rules below were designed in collaboration with the entire R user community at Google.

Summary: R Style Rules

1. **File Names:** end in `.R`
2. **Identifiers:** `variable.name` (or `variableName`), `FunctionName`, `CONSTANTNAME`
3. **Line Length:** maximum 80 characters
4. **Indentation:** two spaces, no tabs
5. **Spacing**
6. **Curly Braces:** first on same line, last on own line
7. **else:** Surround `else` with braces
8. **Assignment:** use `<-`, not `=`
9. **Semicolons:** don't use them
10. **General Layout and Ordering**
11. **Commenting Guidelines:** all comments begin with `#` followed by a space; inline comments need two spaces before the `#`
12. **Function Definitions and Calls**
13. **Function Documentation**
14. **Example Function**
15. **TODO Style:** `T000(username)`

GP: Document your code

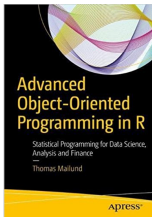
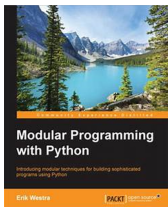
- **Comment** as much as you can.
- Write **help** for your programs.
- Create **user manuals** and **user guides**.
- Use tools to facilitate documentation building.

```
##' Add together two numbers  
##'  
##' @param x A number  
##' @param y A number  
##' @return The sum of \code{x} and \code{y}  
##' @examples  
##' add(1, 1)  
##' add(10, 1)  
add <- function(x, y) {  
  x + y  
}
```

GP: Modularize

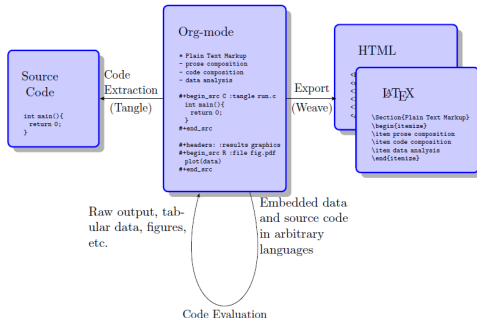
"Modular programming consists in separate the functionality of a program into independent and interchangeable modules."

- If a piece of code has to be used repeatedly or it is too long to affect code legibility it is recommended to encapsulate it into some type of *module*
- Modularization can happen at distinct levels.
 - *functions*
 - *classes and methods* (OOP)
 - *libraries or packages*



GP: Literate programming enhances reproducibility

- **Literate programming:** Enhances traditional software development by embedding code in explanatory essays and encourages *treating the act of development as one of communication*.
- **Reproducible research:** Embeds executable code in research reports and publications, with the aim of *allowing readers to re-run the analyses described*.



Reproducible research with R and Rstudio

- Using literate programming for reproducible research is a change of paradigm.
- Coding, Analyzing and Reporting become a unique piece of work.

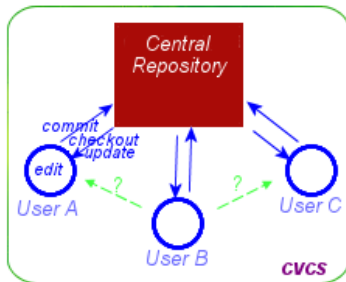


GP: Use Control Version Systems (CVS)

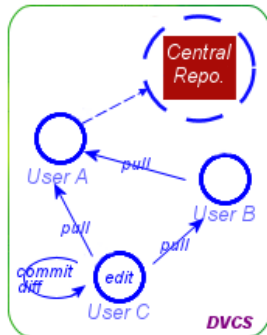


- **Version control systems** are a category of software tools that help a software developer/team manage changes to source code over time.
- **Version control software** keeps track of every modification to the code in a special kind of database.
 - If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to all team members.

Type of Control version systems



Decentralization



Git and Github: a standard *de facto*

- GitHub is a hosting service for Git repositories.
- GitHub offers all the Git functionality adding a series of own characteristics.
 - Web and Desktop-based graphical interface.
 - *Access control, bug tracking, task management, wikis ...*
- *GitHub brings together the worlds largest community of developers to **discover**, **share**, and **build** better software*

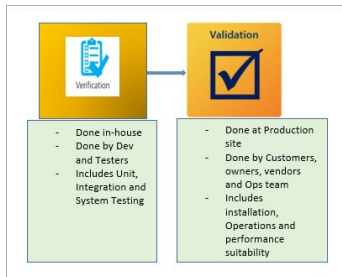


GP: Share your programs

- Sharing software is one great way to improve it.
 - Developers take care of details so that users will be satisfied with the program.
 - Users can provide feedback and can help improve the program.
- Sharing can be done in many different ways
 - Put a library in a repository
 - Create a web page
 - Sell the program
- Many scientist's work is better known by their software tools than by their papers

GP: Verify and Validate¹ your software

- How do we know that the code does what it is intended to do?
 - Prepare a battery of tests, apply them. Document it.
- How do we know that our code contains no mistakes?
 - Make someone else review your work. Document it.
- How do we know if the code can be installed and set to work?
 - Make someone else install and run the programs. Document it.



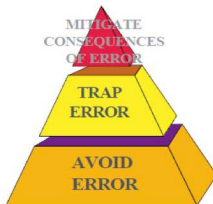
¹<https://www.softwaretestinghelp.com/difference-between-verification-vs-validation/>

GP: Optimize efficiently

- Optimization may improve a program's performance *if it doesn't consume more time than the time it saves*.
 - Before you start to optimise your code, ensure you know where the bottleneck lies; use a code profiler.
 - Use the appropriate data structures (e.g. matrices instead of data frames) and
 - Use the right functions to work with them (Use specialised row and column functions whenever possible)
- Learn to use techniques that can improve speed orders of magnitude. Apply if needed.
 - Parallelization can be great if many processes will be executed "in parallel" (e.g. Monte Carlo Simulations)
 - Learn to combine languages. Consider re-writing key parts of your code in C++.

GP: Manage error control and detection proactively

- Testing and reviewing can detect some errors
 - Some of these (e.g. programmers' errors) can be removed.
 - Others (e.g. wrong input) cannot be avoided.
- Learn **how to debug** your code
- **Plan reaction to errors** according to their severity.
- Learn how to do **Exception handling** to help avoid program crashes.



GP: Plan the process

Think, plan, code it! And NOT in another order!

This will not take much time but, however, will save us a significant amount of time.

- Software development is a **process**
- There exist a myriad of methodologies² and tools for planning and executing. Use the one you prefer.



- What do we want? How are we going to do it?

²<https://www.synopsys.com/blogs/software-security/top-4-software-development-methodologies/>

GP: Design

- Design or aesthetics may not seem so important, but nevertheless they are, especially in web applications.
- Sometimes, a very good software with a design that does not attract the user may not give the maximum performance that is expected of it. However, if the software is friendly and is accompanied by a good design, we can obtain better results.
- To achieve this goal, we suggest the use of html, css, markdown, rmarkdown, javascript, latex...

Conclusions

- Software development may be a “secondary” activity or may be our main source of work.
- The more we know about software development the better we can do the job.
- Applying a few good practices can considerably improve our performance.
- Remember that:

"You shouldn't feel ashamed about your code -if it solves the problem, it's perfect the way it is. But also, it could always be better"

-Hadley Wickam

Good Practices Summary

- Documentation
- Modularize
- Literate programming
- Version Control
- Share your programs
- Verify and Validate
- Optimize efficiently
- Error Control
- Planification
- Design