

# Statistical Software Development

## Principles and war stories

Peter Dalgaard

Department of Biostatistics  
University of Copenhagen

December 1, 2005 / Dina Workshop, Tune

# Outline

## Introduction

The R Project

## Technical Issues and Tools

Version Control

Cross-Platform Issues

Dealing with Complexiy

## Development dynamics

General Principles

Release Planning

## Bugs and Bug Tracking

## What is R?

- ▶ R is an environment for statistical computing and graphics
- ▶ It consists of an interpreted programming language, predefined statistical functions, and a highly flexible graphical subsystem
- ▶ However, that is not the topic for today. . .

## The R Sources

- ▶ 260000 lines of C
- ▶ 146000 lines of Fortran (mostly not ours)
- ▶ 100000 lines of R
- ▶ 80000 lines of Rd (R documentation format)
- ▶ Perl, Java, shell scripts, Makefiles, Texinfo files, . . .
- ▶ About a million lines in all

## The Development Process

- ▶ R is a fairly large software project
- ▶ It runs on multiple computing platforms
- ▶ Maintained and developed by a group of 17 volunteers from all over the world
- ▶ How to keep such a project together?

## Themes of this Talk

- ▶ Which dangers/issues do we need to tackle?
- ▶ Which tools do we use to make the task manageable?
- ▶ Development psychology
- ▶ Bugs and chasing them down
- ▶ *The talk is intended as relatively lightweight, describing general concepts and tools, illustrated with specific case stories,*

## Version Control Systems

17 people working on the same set of source files – how to coordinate?

- ▶ Joint source code repository (p.t. in Zürich, earlier Auckland and Madison)
- ▶ Originally used CVS (Concurrent Versioning System) over secure shell logins
- ▶ Now using Subversion - WebDAV based (safer)
- ▶ Checkout/Commit/Conflict mode of operation
- ▶ Multiple *branches*

## Computing Platforms

- ▶ R works on Windows, MacOS X, Linux, and multiple Unix variants
- ▶ ... *and* their variants over time
- ▶ Graphical User Interfaces (GUIs)
- ▶ Operating system interfaces
- ▶ Help file formats
- ▶ Binary packaging formats

## Build Configuration

- ▶ Things that are basically the same, but still vary between (and within) platforms
- ▶ Compiler flags (optimization, position-independent code)
- ▶ External libraries (and versions thereof)
- ▶ Dynamic linking
- ▶ Compiler bugs, often caused by optimization

## GCC 2.96

In the autumn of 2000, Red Hat Linux had acquired Cygwin, home of the GCC compiler development.

RH 7.0 contained a “2.96” version of GCC, which was basically a snapshot of the development sources. This caused a major uproar; R was directly affected in at least two areas

- ▶ `rpois` went into an infinite loop
- ▶ Double precision `.Machine$double.eps` underestimated

Similar things happened to other projects. Red Hat was forced to release multiple compiler updates `gcc2.96-54` → `2.96-113`

## Automating Things

- ▶ The `make` program describes how to build targets from subtargets using *makefiles*
- ▶ The GNU Autotools create the `configure` script that sorts out all the little system dependencies. Special configuration needed for R.
- ▶ Common help file format (`.Rd`), tools to generate HTML, latex, plain text, and windows help files
- ▶ Package format specification and build tools

Ideally

```
./configure; make; make install
```

## Contributed R packages

- ▶ CRAN – Comprehensive R Archive Network (center in Vienna)
- ▶ Raising standards for software distributions, avoiding “bitrot”
- ▶ Making packages
  - ▶ Portable
  - ▶ Reliable
  - ▶ Easy to install
  - ▶ Documented
- ▶ Strategy
  - ▶ API relatively unchangeable
  - ▶ Standard package format
  - ▶ Provide tools for consistency checking

## Testing and Checking

- ▶ Formal check — documentation consistent with function definitions, all fields present in package descriptions, all examples runnable, etc
- ▶ Run all examples and check for errors
- ▶ Specific testing, regression tests, targeted consistency checks
- ▶ Automated checks of all (600+!) packages: Catches errors both in packages and in R.

## Development Principles

- ▶ Statistical software development is slightly special compared to other software
  - ▶ High emphasis on correctness
  - ▶ Many users not proficient in computer development tools
  - ▶ Large amount of legacy code to integrate
  - ▶ Dependency on many tools
- ▶ Conservative, carefully tested releases
- ▶ Only one release at the time (as opposed to other projects' stable/unstable releases)

## Development Psychology

- ▶ Developers are volunteers and usually have variable workloads from other causes. Besides, they are only human. . .
- ▶ Deadline riding – putting programming off until a release is approaching, then frantically trying to finish in time
- ▶ Some jobs are more fun than others (e.g. coding vs. documentation)

## Deadline Dangers

- ▶ Sometimes very small changes have far-ranging and unexpected consequences.
- ▶ Fixing one bug may uncover another.
- ▶ Last-minute changes will often be rash and poorly thought through
- ▶ It can be less important to fix a bug than to have the test suites run to completion.

## Release Scheduling

- ▶ Users want a relatively modest rate of change
- ▶ Teachers want R available at start of semesters
- ▶ Developers typically get things done in connection with Summer and Winter breaks
- ▶ Release cycle, x.y.0 every half year (April, October)
- ▶ Patch releases (x.y.z) as necessary

## Release Process

- ▶ Need feedback from users, especially those on obscure platforms
- ▶ (R-2.1.0+: Also need translation teams to react to changed messages)
- ▶  $T - 4$ weeks Grand feature freeze. Start of alpha releases
- ▶  $T - 2$ weeks Feature freeze. Start of beta releases
- ▶  $T - 1$ week Code freeze.
- ▶  $T$  Release

## The Best Laid Plans...

In the series of releases leading up to version 1.0.0 (Feb 29, 2000) we had tested the system of feature and code freeze. The actual release of 1.0.0 went smoothly, but then I had to send the following note to Martin Maechler:

```
Martin,  
Please fix your spam filter so that it only catches  
repeated instances of capital 'X'. It is holding  
back the release announcement now!!  
Peter
```

The NEWS file had an entry about “R CMD xxx” and the filters took it for pornographic spam.

# Bugs

- ▶ Bugs are inevitable in any large project by Murphy's law
- ▶ Finding and fixing bugs is a major part of software maintenance
- ▶ Often our own fault
- ▶ Sometimes found in very old code
- ▶ R and its user community tends to be very good at finding bugs

## The Bug Repository

- ▶ The biggest problem with bug reports is to avoid forgetting about them
- ▶ R uses a system called Jitterbug
- ▶ Simple filing system
- ▶ Web interface (PD's desktop machine)
- ▶ Email interface, interfaces with the R-devel mailing list
- ▶ Unfortunately, Jitterbug is now unmaintained and is susceptible to spam injection

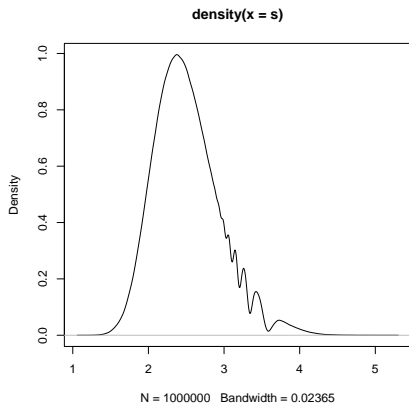
## The Usual Suspects

- ▶ Some frequently encountered bugs:
- ▶ Out-of-bounds errors (underdimensioned arrays)
- ▶ Floating point accuracy – to fuzz or not to fuzz
- ▶ Boundary cases (0 columns, 1 columns,  $\text{ppois}(n,0)$ , infinite DF, etc.)
- ▶ Failing to PROTECT objects from garbage collection

## A Story of Random Normals

- ▶ Around version 1.6, R's random number generation was getting a bad reputation.
- ▶ At this point we were using
  - ▶ the *Marsaglia-Multicarry* method as the base uniform generator
  - ▶ the *Kinderman-Ramage* method for normal variate generation
- ▶ Most reports were related to extreme value distributions

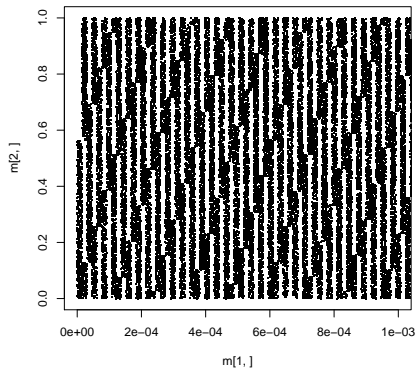
```
RNGversion("1.6.0")  
s <- replicate(1e6, max(rnorm(100)))  
plot(density(s))
```



## Random Normals, cont'd

- ▶ Replacing the uniform generator removed the problem
- ▶ Replacing the normal variate method *also* removed the problem
- ▶ The Kinderman-Ramage method is quite complicated, but part of it is that
  - ▶ one uniform number is used to decide whether the random value is out in the tail
  - ▶ a second independent uniform number is used to decide where in the tail
- ▶ The Marsaglia-Multicarry method has unfortunate pattern formation near the edges.

```
m <- matrix(runif(8e7),2)
plot(m[1,],m[2,],xlim=c(0,1e-3),pch=".")
```



## A Little Later

- ▶ G.Tirler found a bug in R's Kinderman-Ramage routine
- ▶ A constant was wrong and a code branch was missing creating some deviation from the true normal density.
- ▶ Fixing that bug and retesting revealed that it still wasn't quite right
- ▶ This turned out to be an error in the original 1976 paper!

## The Bug that Wouldn't Die

PR#862,1123,1175 (Pearson residuals in binomial models)

PD: *“Could someone please ram a wooden stake through this one and prevent it from haunting us again?”*

Thomas Lumley (in bug repos): *“not the same as 1123, and still there after the fixes for that. Fixed, and with three regression tests added to make check.”*

## Summary

- ▶ I have described some **technical tools** which we use to keep the R project together
- ▶ Basic **development principles** have been discussed
- ▶ User interaction helps to find bugs, sometimes **bugs in very old and trusted code**