technische universität
dortmund

Department of Statistics

CRAN, R,
and the people moving it forward

Uwe Ligges
Uwe.Ligges@R-project.org

The R Project for Statistical Computing
TU Dortmund University

June 2019

Contents

- History
- Current state
- Challenges we are facing
- Recent and future changes

What is CRAN?

- Provides a decentralized and modularized way of creating software
- Based on R's standardized format of packages and package system and its ease of installing packages
- Accessibility by a broad audience:
  standardized repositories
- To support such a loosely coupled development model:
  verification of certain formal quality criteria via checks

What is CRAN?

- Provides a decentralized and modularized way of creating software
- Based on R's standardized format of packages and package system and its ease of installing packages
- Accessibility by a broad audience:
  standardized repositories
- To support such a loosely coupled development model:
  verification of certain formal quality criteria via checks
- nowadays more than 2^{13} packages – approaching 2^{14} packages

History of R and CRAN

1976 At the beginning was S, see Rick Becker's talk at useR!2016

1992 R development started (Ross Ihaka and Robert Gentleman)

1996 First paper about R

1997 R (Development) Core team

1997 CRAN founded (by Kurt Hornik and Fritz Leisch taking ideas from CPAN and CTAN)

1998 ...

???? Windows binary packages made available by Brian Ripley

???? Mac binary packages made available by Stefano Iacus, later by Simon Urbanek

History of R and CRAN

???? **R-1.0.0** was released (**S** version 3 compatibility)

History of R and CRAN

2000 **R-1.0.0** was released (**S** version 3 compatibility), **release date: February 29, 2000**

History of R and CRAN

2000 **R-1.0.0** was released (**S** version 3 compatibility), **release date: February 29, 2000**

2003 I took over maintenance of CRAN's Windows binary package repositories from Brian Ripley. We had 195 packages working under Windows (R-1.7.x), manually built and checked, when there was a submission ...

History of R and CRAN

2000 **R-1.0.0** was released (**S** version 3 compatibility), **release date: February 29, 2000**

2003 I took over maintenance of CRAN's Windows binary package repositories from Brian Ripley. We had 195 packages working under Windows (R-1.7.x), manually built and checked, when there was a submission ...

2004 **R-2.0.0**

2006 Some proposal by Insightful ...

History of CRAN (and R)

Proposed S-PLUS® Packages

- An S-PLUS® package is a collection of S-PLUS® functions, data, help files and other associated source files that have been combined into a single entity for distribution to other S-PLUS® users.
- This package system is modeled after the package system in R.
- Insightful Corporation hosts the Comprehensive S-PLUS® Archival Network (CSAN) site at http://csan.insightful.com/ to facilitate S-PLUS® package distribution.
- Packages can be downloaded from the CSAN websites in two forms: as raw source code or as Windows binaries.

History and Future of CRAN (and R)

2013 **R-3.0.0**

2019 The most recent version is **R-3.6.0**.

History and Future of CRAN (and R)

2013 **R-3.0.0**

2019 The most recent version is **R-3.6.0**.

2020 **R-3.7.0**???

From R-3.0.0 to R-4.0.0

One of the most important improvements in R-3.0.0 has been

> *There is support for vectors longer than* $2^{31} - 1$ *elements on 64-bit platforms.*

What else has happened since R-3.0.0?

- ALTREP changes
- Bytecode compiler and JIT
- Other speed and memory related improvements
- Countless bug fixes.
- $\approx$ 14200 svn commits

People

Douglas Bates
John Chambers
Peter Dalgaard
Robert Gentleman
Kurt Hornik
Ross Ihaka
Tomas Kalibera
Michael Lawrence
Friedrich Leisch
Uwe Ligges
Thomas Lumley
Martin Maechler
Martin Morgan
Paul Murrell
Martyn Plummer
Brian Ripley
Deepayan Sarkar
Duncan Temple Lang
Luke Tierney
Simon Urbanek
Heiner Schwarte up to October 1999
Guido Masarotto up to June 2003
Stefano Iacus up to July 2014
Seth Falcon up to August 2015
Duncan Murdoch up to September 2017

Current state of CRAN and its services

- More and more has been automated:
  R package management system has gone a long and successful way to also support repository maintainers.
- Key services:
  - distributes sources
  - provides check result summaries
  - builds and distributes binaries

Products

We can think of

- packages as ‘products’ (or apps),
- potential users as ‘customers’,
- package repositories like CRAN: ‘warehouse’-like storage areas (app stores).

Products

We can think of

- packages as ‘products’ (or apps),
- potential users as ‘customers’,
- package repositories like CRAN: ‘warehouse’-like storage areas (app stores).

If the ‘right’ product is not available:

- Easy for customers to become contributors: creating new package to fill the gap.
- Decentralized and modularized way of creating software.
- Rather then reinventing the wheel, package authors wisely reuse code of other packages.

Products, repositories and customers

Once a new customer downloads a package from a repository

- it is fine if it works as expected by the new user,
- it may be fair if it works as documented,
- the customer goes away if it does not work.

Hence we have to improve the quality assurance system and keep in mind that a bug in a non working package may not be caused by the package itself but by changes in another package it depends on.

Submitting to CRAN

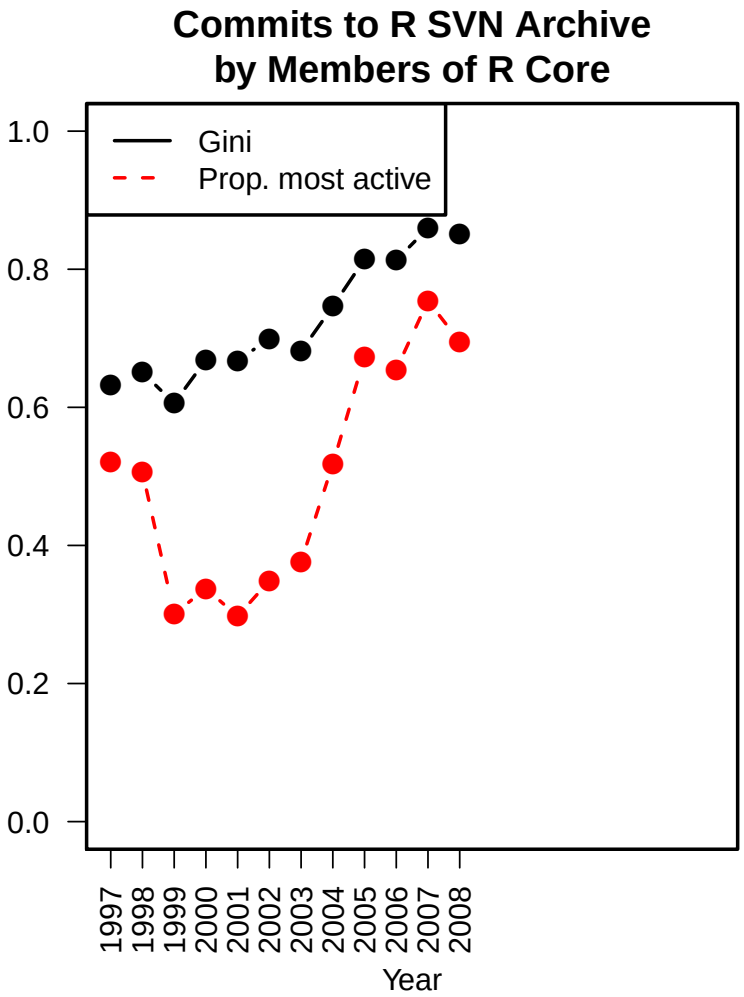
- Publication mechanism for CRAN:
  - Submit contributed source package via the web form.
  - Package checked by an incoming auto-check service.
  - (Package checked by a CRAN team member.)
  - If fine, package is included for provision.
  - Binary versions are created and checked.
  - Regular checks ...
- more later ...

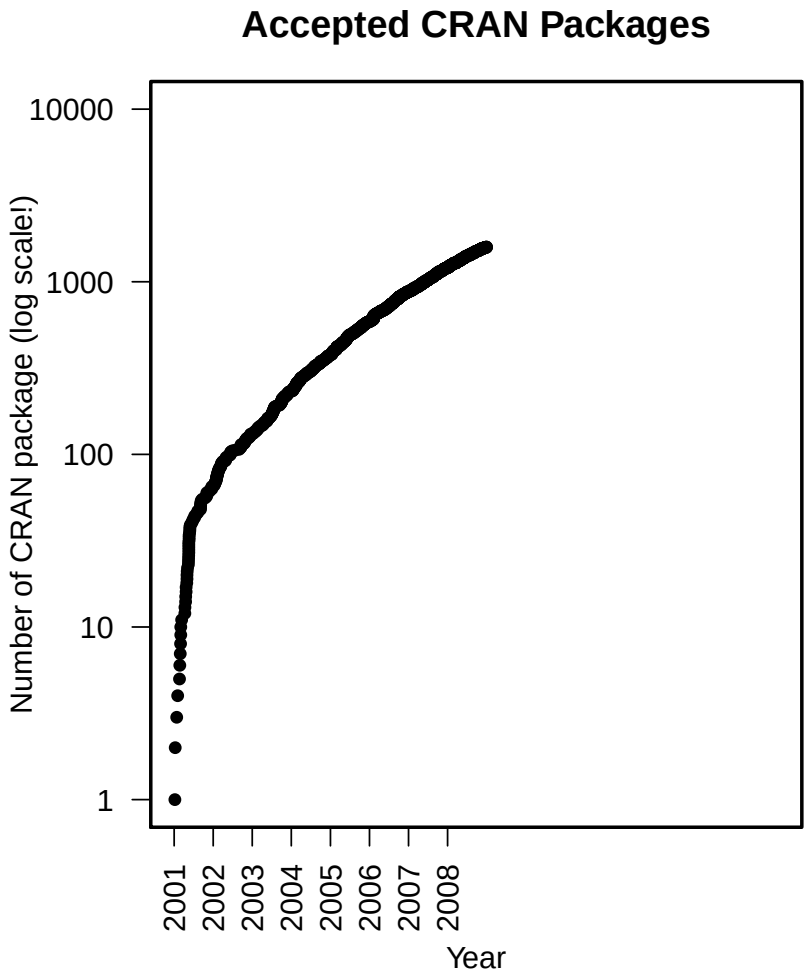
Activity: R and CRAN

11 years ago: useR!2008 in Dortmund

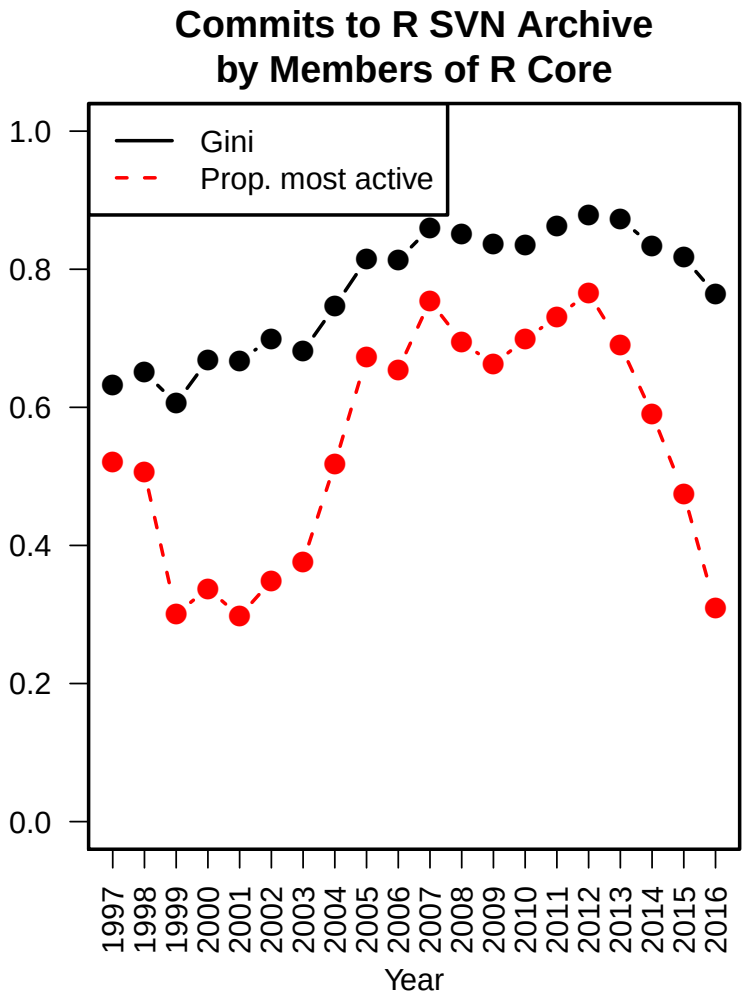
John Fox talked about "The Past, Present, and Future of the R Project: Social Organization of the R Project" (R Journal paper in 2009) and presented some data analysis on the topic (here slightly updated).

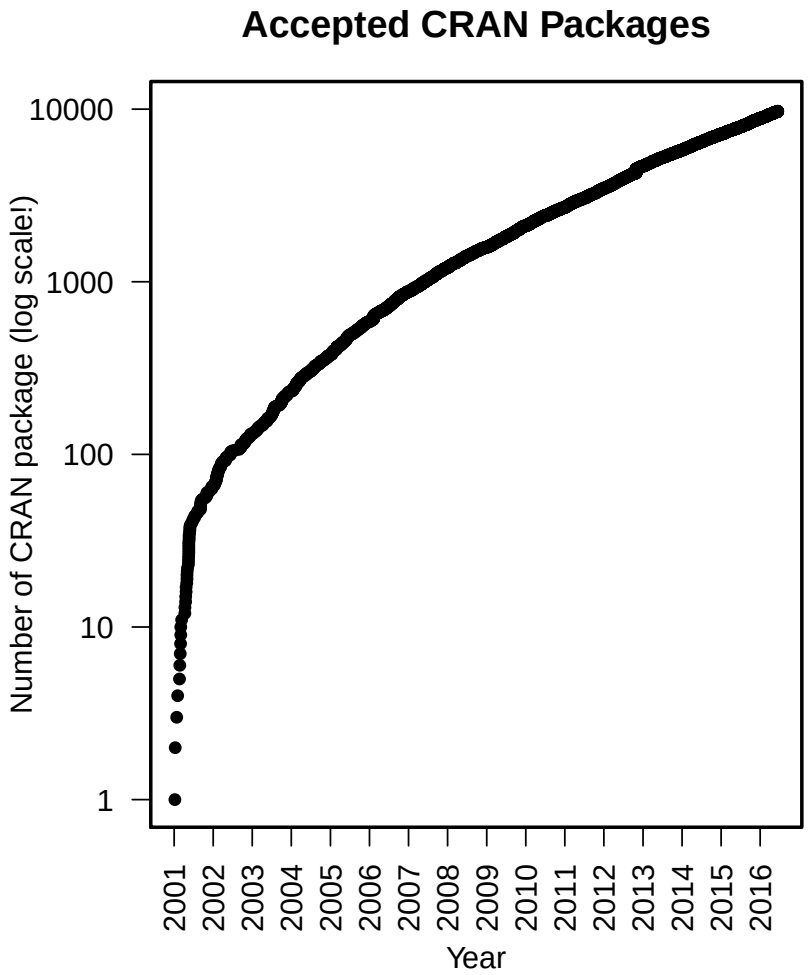
... and beyond? – Activity





... and beyond? – Activity





Dependencies

- Packages might depend on other packages that again depend on ...
- Hierarchy of dependencies could be broken by a simple bug in one of the packages:
  implications on the interoperability between packages.
- Check system that allows for recursive checking.
- Check system for huge package repositories should be parallelized:
  deal with thousands of packages while respecting dependency structures.

Dependency levels

Package maintainers specify dependencies in the DESCRIPTION file:

Depends: Package *B* depends on functionality in package *A* in such a way that package *A* is loaded in advance of *B*.

Imports: Package *B* imports (parts of) the namespace of package *A* into its own namespace.

LinkingTo: Package *B* makes use of header files in package *A*.

Suggests: Some functionality or examples in the documentation of package *B* depend on package *A*.

Enhances: Package *B* enhances packages *A* functionality but works without *A* being present.

Dependency levels

- *Depends*, *Imports* and *LinkingTo* must be fulfilled at installation time.
- *Suggests* must typically be available when a package is checked.
- Usually two different types of dependency graphs have to be calculated:
  - Graphs needed for finding the correct installation order.
  - Graphs needed for finding what have to be checked.

(Reverse) dependencies

Ignore the following two slides, they just contain some formulas to make the talk look scientific in a way ...

(Reverse) dependencies

Ignore the following two slides, they just contain some formulas to make the talk look scientific in a way ...
Consider

- package B depends on package A;
  formally denoted $A \in d(B)$, where $d(B)$ entails all dependencies of B,
- packages C and D depend on package B,
  i.e. $A \in d_R(C), A \in d_R(D)$,
- $d_R(D)$ denotes all recursive dependencies of package D,
  i.e. the transitive closure of all dependencies of D.

(Reverse) dependencies

Then, once A is updated, we definitely need to

- re-check packages A, B, C, and D,

because newly introduced features or changes in A could have broken something somewhere else.

- Reverse dependencies of A denoted $d^{-1}(A)$, i.e., all packages depending on A.
- Recursive reverse dependencies: $C \in d_R^{-1}(A)$ has to be considered for re-checking interoperability.
- With growth of CRAN: frequently an update of one package P breaks code in some dependent package(s) $d_R^{-1}(P)$.

(Reverse) dependencies

Then, once A is updated, we definitely need to

- re-check packages A, B, C, and D,

because newly introduced features or changes in A could have broken something somewhere else.

- Reverse dependencies of A denoted $d^{-1}(A)$, i.e., all packages depending on A.
- Recursive reverse dependencies: $C \in d_R^{-1}(A)$ has to be considered for re-checking interoperability.
- With growth of CRAN: frequently an update of one package P breaks code in some dependent package(s) $d_R^{-1}(P)$.
- Wonder why CRAN worked fairly well until 2008 without these checks.

(Reverse) dependencies

- We even may need – in addition to a new binary for *A* – some updated *binary* packages for *B*, *C* and *D*: e.g. if S4 classes are involved.
- Find out which other packages are 'involved':

```
tools::dependsOnPkgs(pkgs,
  dependencies = c("Depends", "Imports", "LinkingTo"),
  recursive = TRUE, lib.loc = NULL,
  installed = installed.packages(lib.loc,
                                 fields = "Enhances"))
```

(Reverse) dependencies

Number of CRAN packages with 0, 1, . . . , and max. number of (recursive / reverse) dependencies:

dependencies	0	1	2	3	10	max
flat	2025	1745	1671	1266	176	maGUI: 37
recursive	2025	682	432	524	391	RPANDA: 151
flat reverse	7663	1406	532	287	47	knitr: 2161
recursive reverse	?	?	?	?	?	?: ?

based on dependency levels

- *Depends*, *Imports*, *LinkingTo*,
- plus *Suggests* for flat reverse and recursive reverse dependencies.

(Reverse) dependencies

Some well known CRAN packages with extreme number of ¿ 1000 recursive reverse dependencies:

- knitr
- MASS
- Matrix
- testhat
- Rcpp
- survival

based on dependency levels *Depends*, *Imports*, *LinkingTo*, and *Suggests*.

(Reverse) dependencies

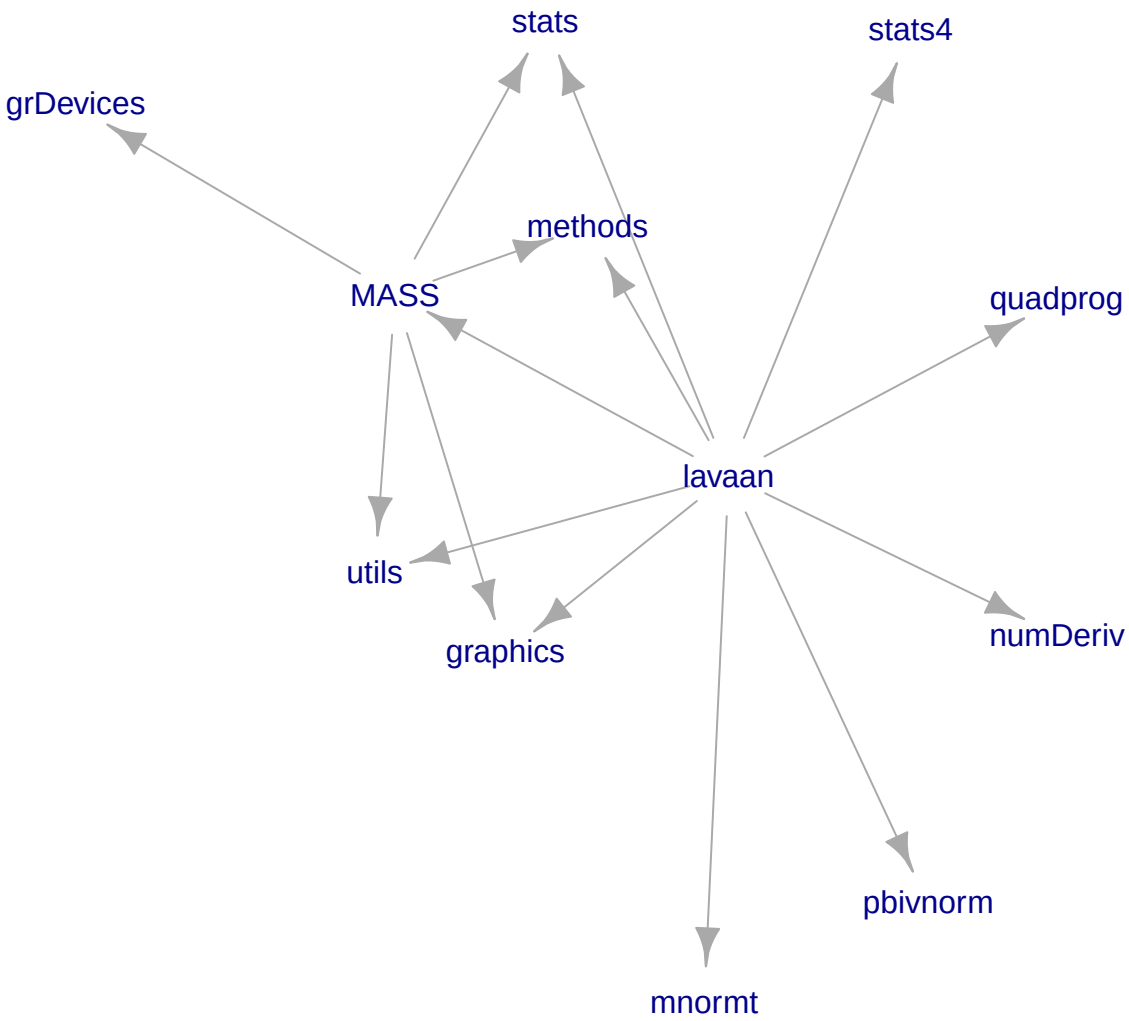
- CRAN's dependency matrix is sparse.
- Nevertheless, more than 75% (> 8000) of all CRAN packages 'depend' (recursively) on packages MASS, Matrix or survival.
- This is a problem given the runtime and many package updates a day — at least without allowing for parallel checks (and installs).

(Reverse) dependencies

- CRAN's dependency matrix is sparse.
- Nevertheless, more than 75% (> 8000) of all CRAN packages 'depend' (recursively) on packages MASS, Matrix or survival.
- This is a problem given the runtime and many package updates a day — at least without allowing for parallel checks (and installs).
- Let us take look at dependency graphs created with the help of package *igraph* (first forward, then reverse).

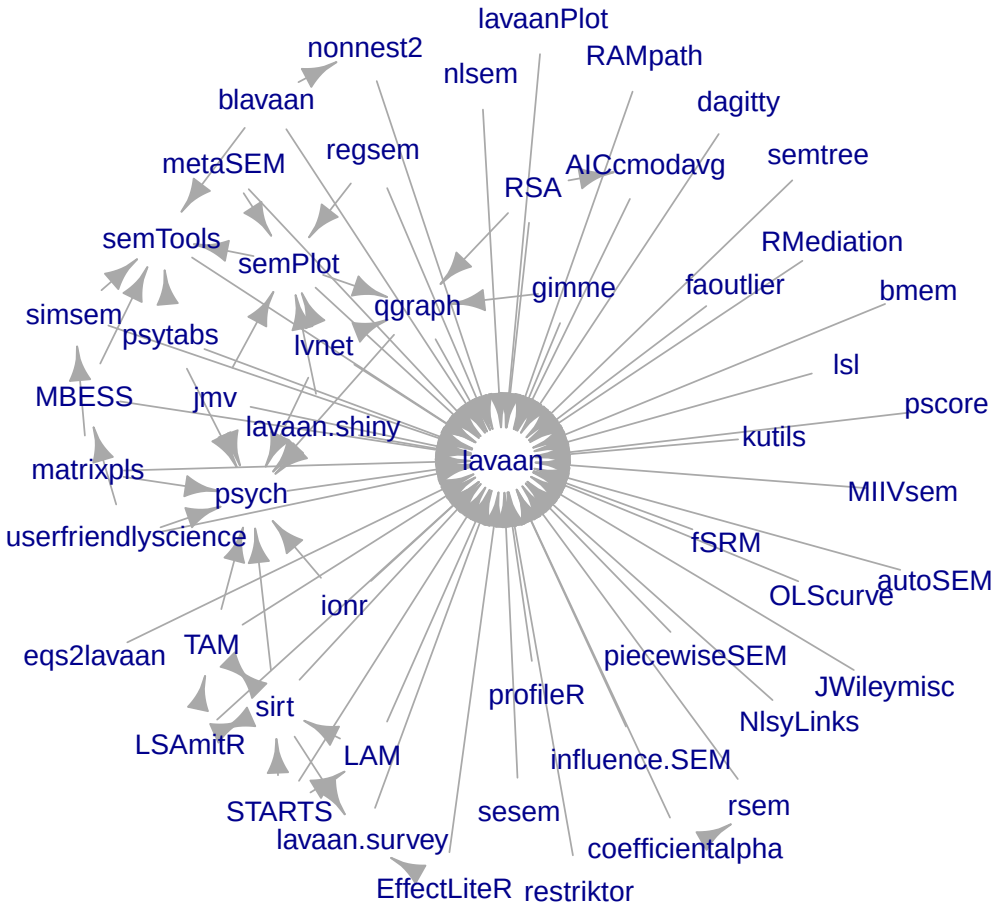
(Reverse) dependencies

Dependency graph for *lavaan*
dependency levels: *Depends*, *Imports* and *LinkingTo*.



(Reverse) dependencies

Graph representing the first level reverse dependencies of package *lavaan* as needed in every new check of the given package; dep. levels: *Depends*, *Imports*, *LinkingTo*, and *Suggests*.



Resources

- Combinations of different flavors of R:
  - Branches: *devel*, *patched*, and *release*.
  - Platforms: Linux, Mac OS X, (Solaris), Windows.
  - Architectures: (SPARC), ix86, x86_64.
- Building and checking of packages on all combinations can become rather time consuming, particularly for checking reverse recursive dependencies.
- Development version of R changes over time: regular checks (up to daily).

Computer resources

- Desirable to get check results for development processes early.
- Build/check system required that
  - **finished within (at least!) 24 hours**,
  - for each flavor of R in order to
  - provide the check results when needed, not thereafter, and
  - make binaries available in time during an R release cycle
- `http://CRAN.R-project.org/web/checks/check_timings.html`

Computer resources

Why should we 'improve' computer resources?

Tasks of a CRAN auto-build-and-check machine

- Make R-devel (daily).
- Build and check new and updated packages at least for R-release (or R-patched), R-devel, and R-oldrelease:
  - including reverse dependencies.
- Notifications for developers.
- Provide check summaries.

Computer resources

Why should we 'improve' computer resources?

Tasks of a CRAN auto-build-and-check machine

- Re-check all packages for R-devel (and R-patched?) on a regular (daily/weekly?) basis.
  Aim: make it possible to look out for errors for both R Core developers and package developers.
- Provide the on demand check system for package developers as a service.

CRAN Windows Binaries' Package Check

Last updated on 2006-06-13 17:51:39 **(useR! 2006)** (simplified)

No	Package	Version	R-2.3.1	Inst. time	Check time
...	...	...	...	...	...
742	wavelets	0.2-1	OK	26	88
743	waveslim	1.5	OK	58	109
744	wavethresh	2.2-8	OK	30	75
745	wccsom	1.1.0	OK	18	87
746	wle	0.9-2	OK	24	365
747	xgobi	1.2-13	**ReadMe**		
748	xtable	1.3-2	OK	22	52
749	zicounts	1.1.4	**WARNING**	26	46
750	zoo	1.1-0	OK	23	60
SUM (in hours) on **Xeon 3.06 GHz:**				6.34	19.77
				$\approx$ **26 hours**	

CRAN Windows Binaries' Package Check

Last updated on 2019-06-15 **(13 years later)** (simplified)

No	Package	Version	R-3.6.0	Inst. time	Check time
...	...	...	...	...	...
14451	zyp	0.10-1.1	OK	3	33
Sum (in hours)				$161/16 = 10$	$408/16 = 25.5$
2x Xeon E5-2670, 8-cores each				$\approx$ **36 hours**	

... or 3 weeks on a single core machine for (implicitly) 32-bit + 64-bit checks.

Human resources

It is all automated, so what?

Tasks of a repository maintainer

- Maintaining and adapting the scripts themselves.
- Maintaining the hardware of a devoted machine.
- Setting up repositories for new versions of R.
- Handling errors that were not covered by the scripts.
- Answering questions (for developers and users).
- Asking package maintainers to fix their packages.
- Asking maintainer to provide runtime checks, but checks that can be executed as quickly as possible given the computing resources.

Quality management can be improved by moving as many tasks as possible from human to computational resources.

Human resources

- more than 43300 messages in 2019 already, i.e. 261 messages per day

Move tasks from humans to other humans

A package maintainer submitting a package can help us:

- *Really* read *Writing R Extensions* and the *CRAN policies*.
- Check your package via `R CMD check --as-cran` *prior* to CRAN submission with a recent version of R-devel.
- In case you are in doubt try some on-demand check service at first.
- Ask questions on the R-package-devel mailing list.

Move tasks from humans to other humans

Unit test frameworks:

- If the R based tests implementation is not fancy enough for you, you may ask other unit test frameworks such as testthat, but use these properly!
- It is not helpful if they give an error message themselves that cannot be interpreted from looking at the check logs.

Solutions: Build and check systems

On demeand build and check systems:

- special service called *win-builder* (`http://win-builder.R-project.org`)
  - allows to upload source packages,
  - provides corresponding Windows binaries
  - provides CRAN equivalent check results
- See R-hub (`https://builder.r-hub.io/`) for a non-CRAN service that offers more than just the Windows platform.

... and beyond?

What to do beyond the current state?

- Further on support people to write packages by providing infrastructure and services for package maintainers and authors.
- Allow for value added services on top of CRAN.
- Further improve the quality (and the quality assurance system called R CMD check).

... and beyond?

What to do beyond the current state?

- Further on support people to write packages by providing infrastructure and services for package maintainers and authors.
- Allow for value added services on top of CRAN.
- Further improve the quality (and the quality assurance system called R CMD check).
- Keep R being the probably **most thoroughly validated statistics software product on earth**

... and beyond?

What to do beyond the current state?

- Further on support people to write packages by providing infrastructure and services for package maintainers and authors.
- Allow for value added services on top of CRAN.
- Further improve the quality (and the quality assurance system called R CMD check).
- Keep R being the probably **most thoroughly validated statistics software product on earth**
  — for some definition of validated

Questions?

Hence we are trying to find good answers:

- Infrastructure that supports calculation of the necessity of a binary re-built and tells update.packages() about it, too.
- Some better database like system for each library (no need to parse thousands of DESCRIPTION files).
- Even stricter and even more automation in CRAN maintenance.
- Spend some money for hardware and human maintenance work.

References

- Csardi G, Nepusz T (2006): The igraph software package for complex network research. InterJournal Complex Systems: 1695.
- Jackson I, Schwarz C, et al. (2010): Debian Policy Manual, version 3.8.4.0.
- Theußl S, Ligges U, Hornik K (2011): Prospects and Challenges in R Package Development. Computational Statistics 26(3), 395–404.
- Theußl S, Zeileis A (2009): Collaborative software development using R-Forge. The R Journal 1(1): 9–14.