UPTAKE

#ODSC

OPEN
DATA
SCIENCE
CONFERENCE

R Packages for Team Collaboration

Stephanie Kirmer, Data Scientist

@data_stephanie
www.stephaniekirmer.com
https://github.com/skirmer/odsc2018/

About Me and Uptake

My role includes: modeling, internal tool development, exploratory analysis, customer interaction

Uptake:

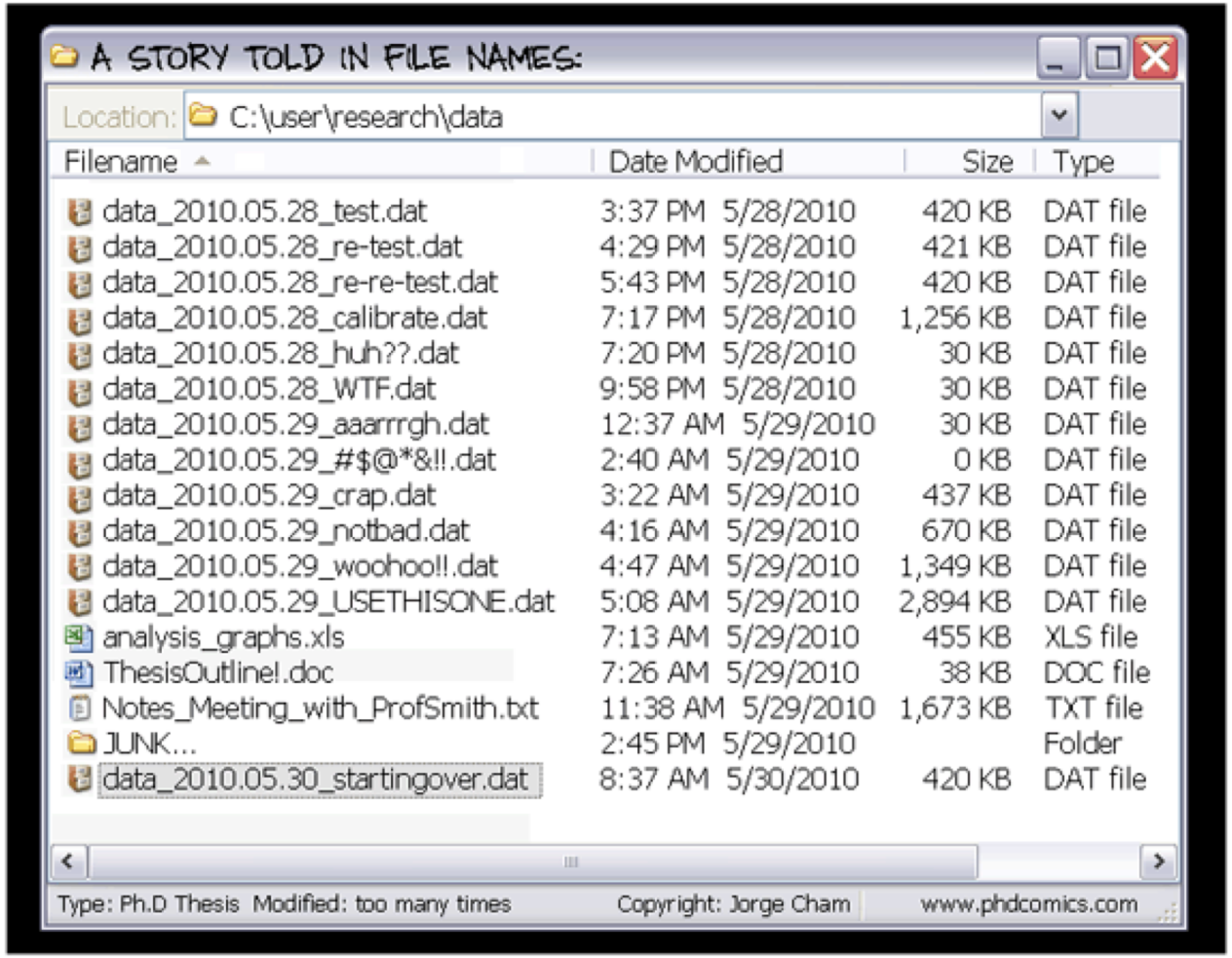
- Analytics company serving a variety of areas of industry (rail, mining, oil & gas, trucking, etc)
- We provide data science and analytics insights to help reduce downtime, improve safety, and increase efficiency for our customers.
- 60+ person data science team, with expertise across the field
- Data science is at the core of what our company does!

With a team this big, we have strong techniques for collaboration and sharing knowledge.

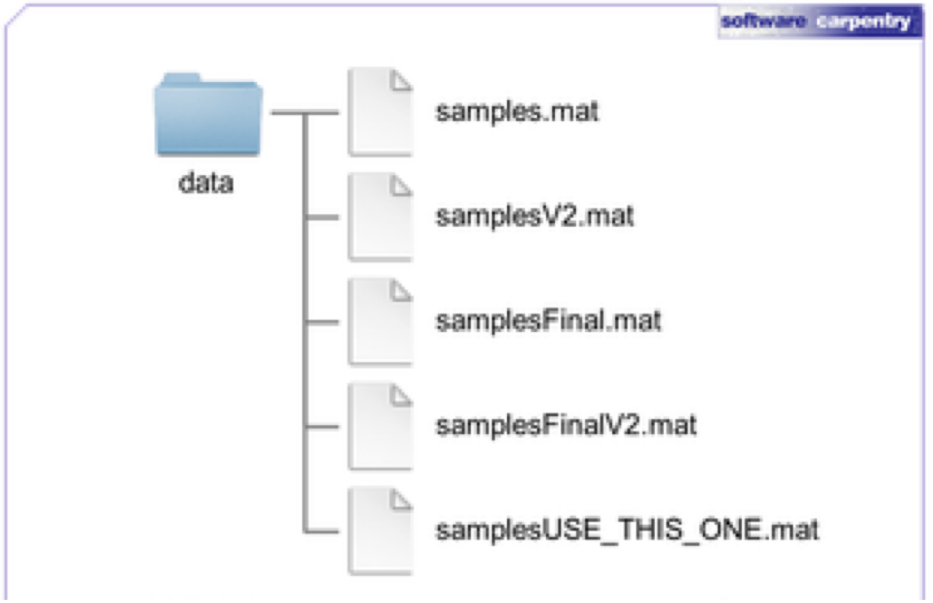
Problem:

Data science teams without structured, intentional collaboration leak knowledge and waste resources.

Don’t do this.



http://phdcomics.com/comics/archive.php?comicid=1323



http://v4.software-carpentry.org/data/mgmt.html

Not in your personal life.

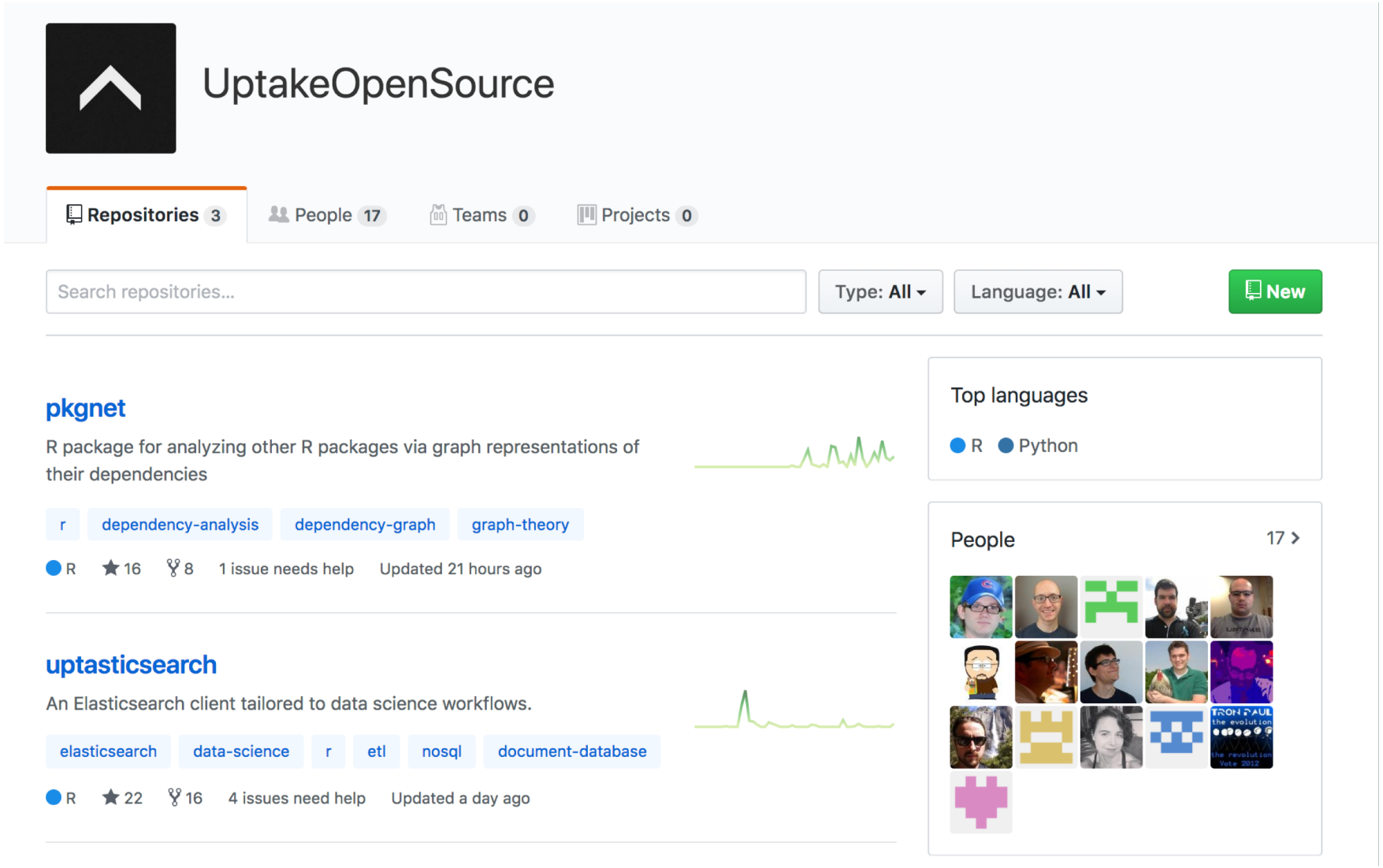
Definitely not in business.

Solution:

Build your team's culture, infrastructure, and tools so that retaining and sharing knowledge is easy and valued.

R packages are a great tool to help you get there.

Do this instead.

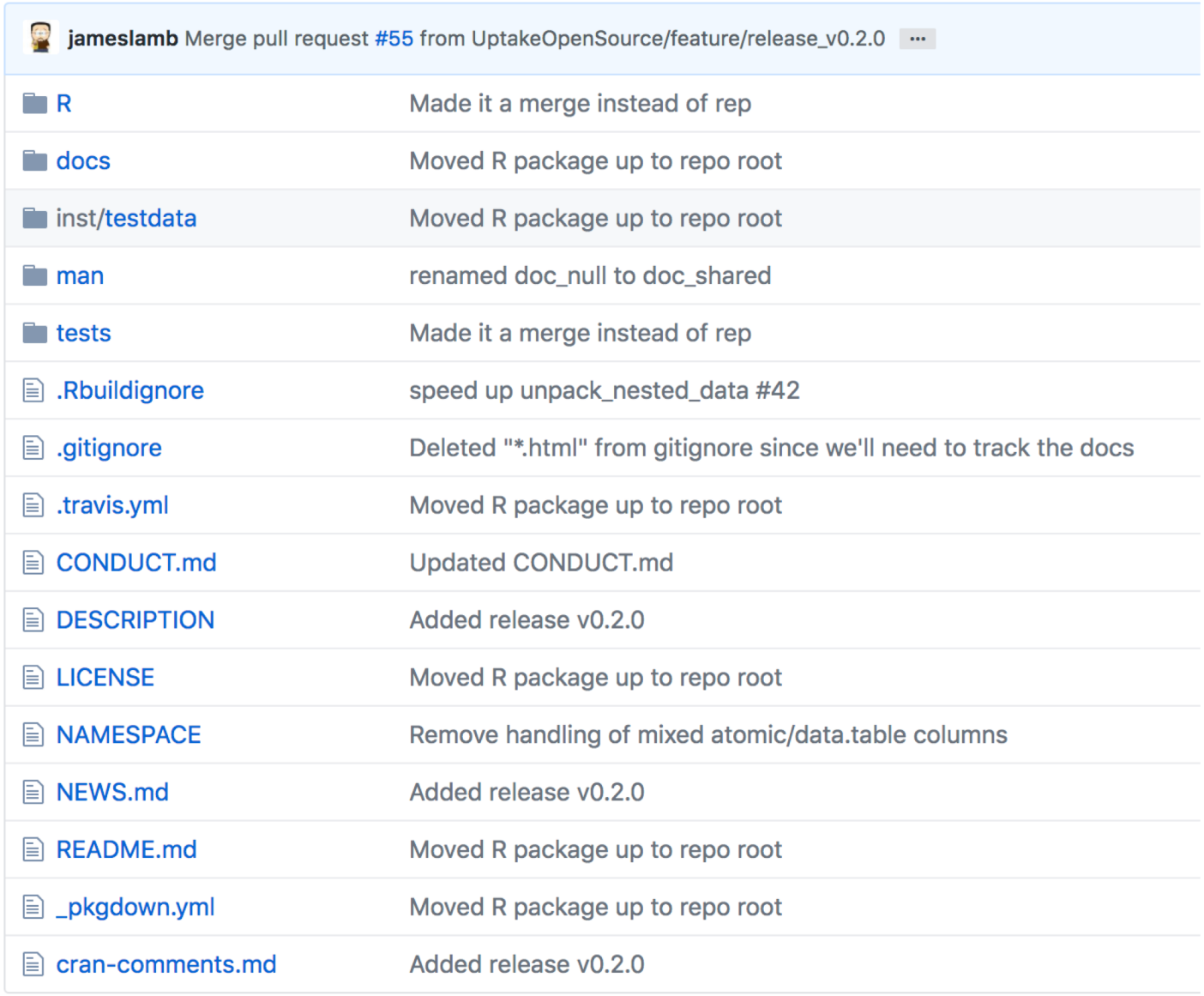


Version control

Collaborative development

Packaged tools

https://github.com/UptakeOpenSource



https://github.com/UptakeOpenSource/uptasticsearch

Code can all be installed with a single command.

Versions are automatically managed and are always accessible.

Folders have standard organizing scheme.

Progression of Team Collaboration Infrastructure

None	Little	Moderate	High
No version control systems.	Minimal, unsystematic use of version control.	Git is in place, but used inconsistently.	Git is an integral part of the workflow for the entire team.
Code sharing via email, when it is shared at all.	Code sharing via email or links/filepaths on a shared file space.	Code sharing is done with git when it happens.	Code is shared & reviewed through git, and continuous integration (CI) system is used.
Code stored locally on hard drives, not accessible to colleagues.	Code stored on shared drive space or a cloud service.	Git is the ideal place to share code, but shared drives are still used.	Packages are shared through an internal CRAN-type system.

Progression of Team Collaboration Infrastructure

None	Little	Moderate	High
No version control systems.	Minimal, unsystematic use of version control.	Git is in place, but used inconsistently.	Git is an integral part of the workflow for the
Code sharing via email, when it is shared at all.	Code sharing via email or links/filepaths on a shared file space.	Code s with git happen	it, tem is used.
Code stored locally on hard drives, not accessible to colleagues.	Code stored on shared drive space or a cloud service.	Git is the ideal place to share code, but shared drives are still used.	Packages are shared through an internal CRAN-type system.

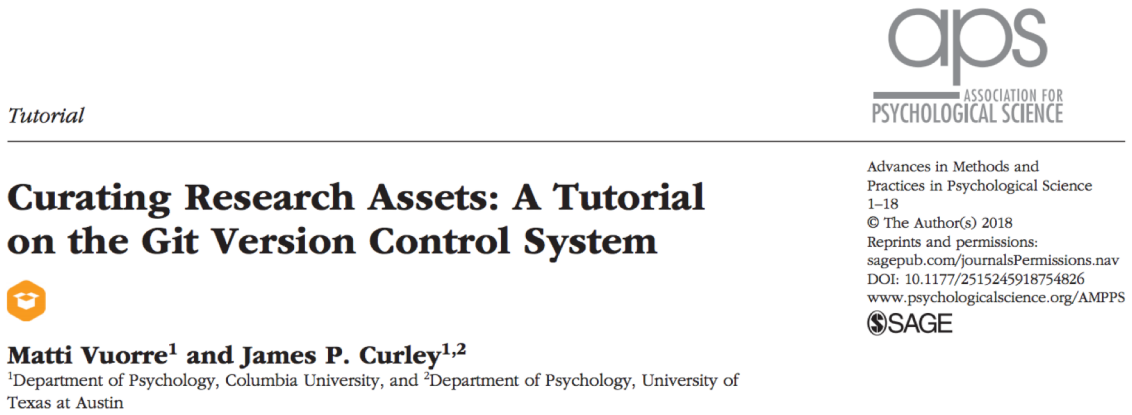
This is where using team R packages becomes a real possibility

Why?

Improve quality and reproducibility by using peer reviewed, validated tools in package form.

- Rewriting code = opportunity for mistakes.

- A shared set of tools creates consistency across projects.
  - Definitions of measures and metrics
  - Data cleaning and management steps



Tutorial

aps ASSOCIATION FOR PSYCHOLOGICAL SCIENCE

Curating Research Assets: A Tutorial on the Git Version Control System

Advances in Methods and Practices in Psychological Science
1–18

DOI: 10.1177/2515245918754826
www.psychologicalscience.org/AMPPS
SAGE

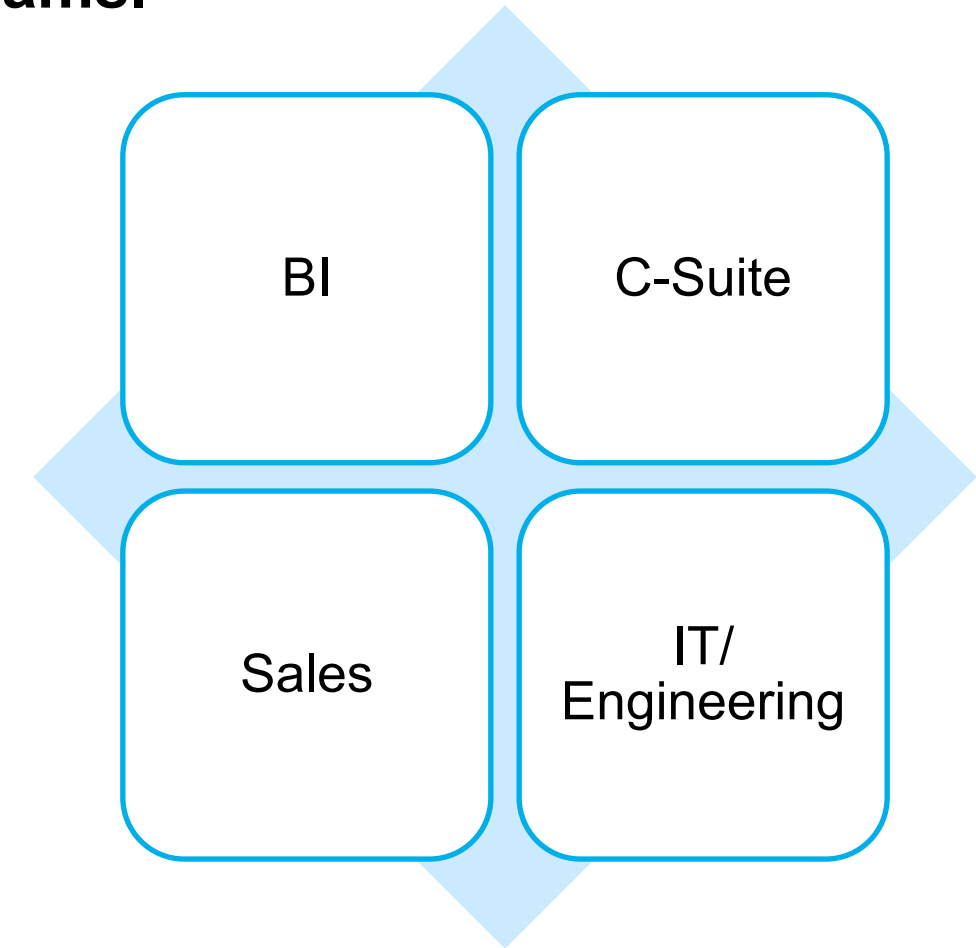
Matti Vuorre[1] and James P. Curley[1,2]
[1]Department of Psychology, Columbia University, and [2]Department of Psychology, University of Texas at Austin

http://journals.sagepub.com/eprint/CxmDEDrWCmqw6sZTGNEh/full

Build package elements for use by other teams.

- Package functionalities can include:
  - Data ETL
  - Basic analysis
  - Visualizations
  - .. and more!

- Teams can also build useful tools for their specific tasks



Empower your colleagues to do routine work, and free up data science to do the complex and innovative projects.

Don’t wait until personnel changes to think about your information.

With robust information infrastructure:

New Hire

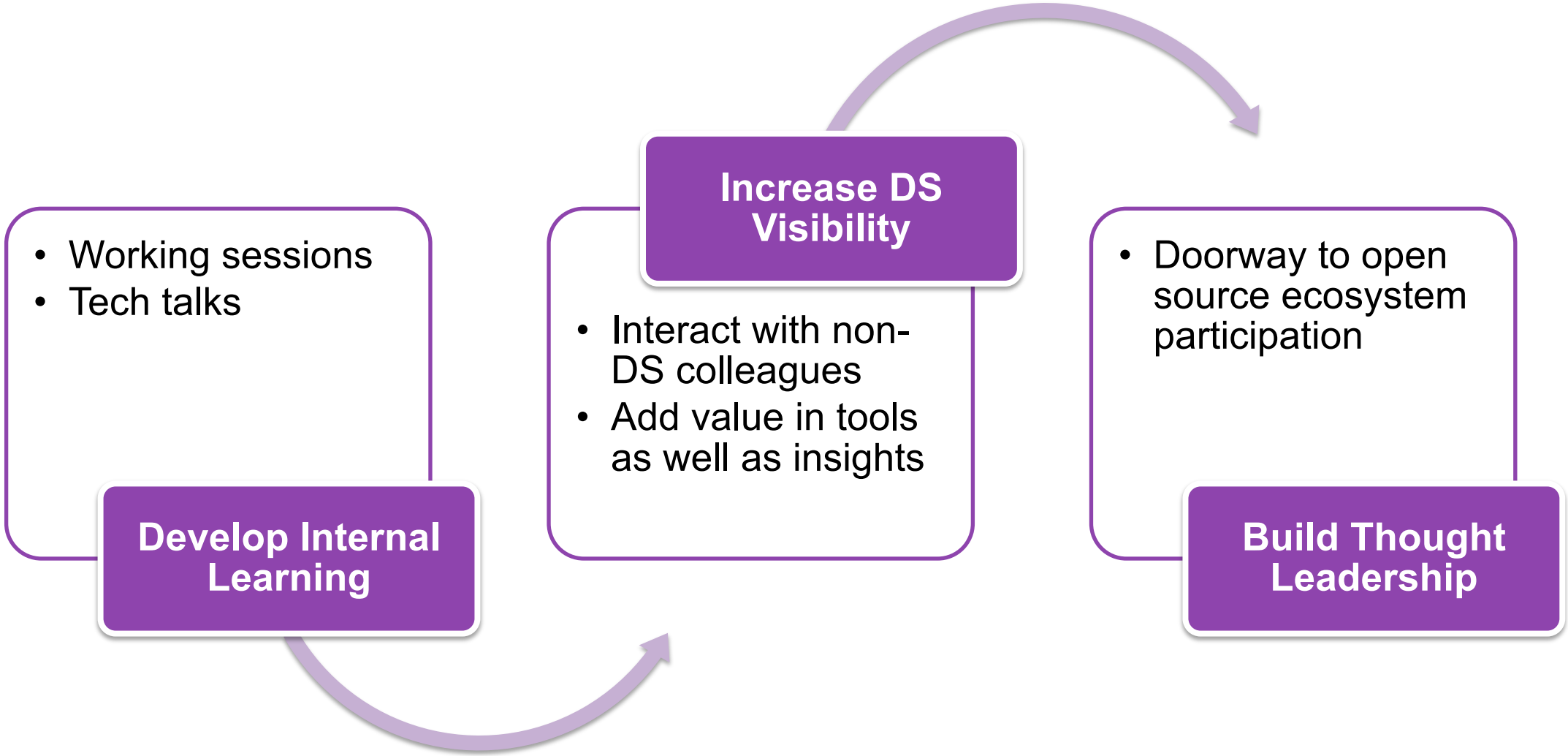
- Goes from beginner to contributor – fast!
- Has a welcoming onboarding experience.

Staff Departure

- Causes no panic or scramble.
- Results in no lost information or knowledge.

Community tools increase the stability of your team for the long term.

Expand a culture of education across your organization.



How?

A group of individual data scientists =/= a collaborative team of data scientists.

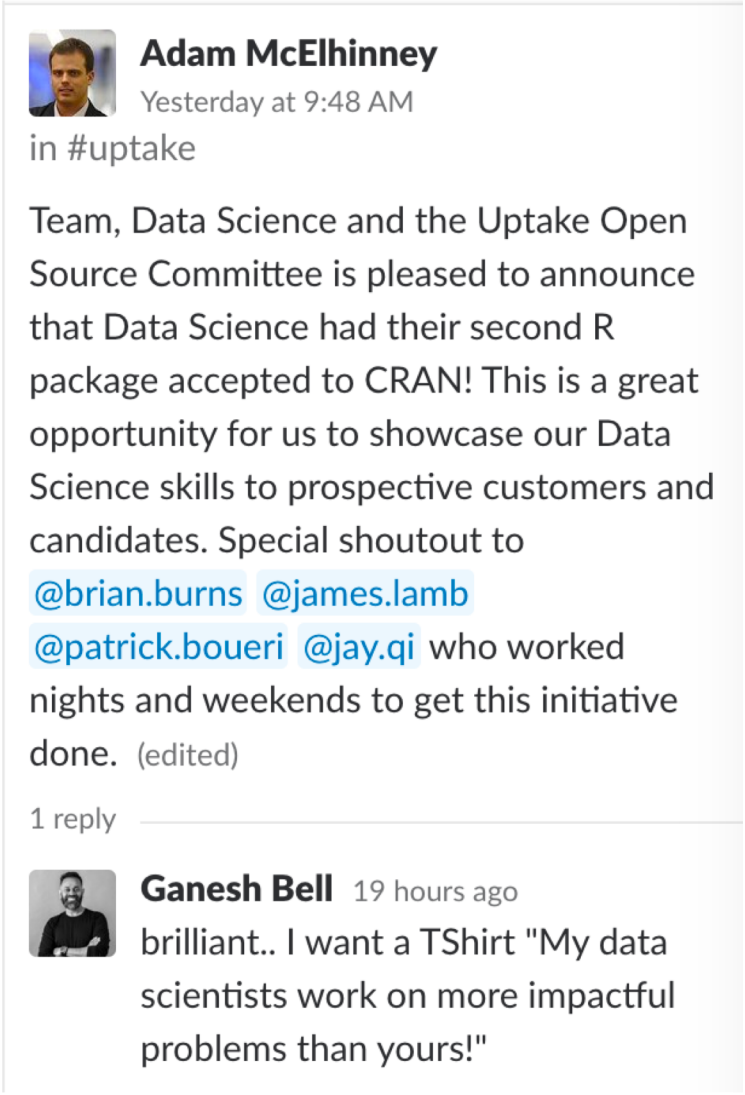
Develop a paradigm of thinking that is productive and works towards your objectives.

- Common, understood goals
- Trust in colleagues
- Willingness to share information and help each other

Leadership has to commit to changes.

People need to be incentivized to maintain and contribute to new tools through:

- Time
- Compensation
- Recognition



Have a vision for what sort of system you want your team to have, and build infrastructure that can scale and evolve.

Start by thinking and planning.

Explore your undocumented resources	• Learn your institutional history/knowledge. • Investigate existing best practices.
Catalog user needs	• Who do you want using your tools? What will they need?
Think for the long haul	• Your needs today will change tomorrow.

Build out the tech infrastructure you need thoughtfully.

Don’t take on trendy tools you don’t really need!

Internal git infrastructure

- Set up a system for versioning and code sharing, and communicate it to your team.

Building functions and packages

- After planning, construct the actual code.
- Many useful tutorials exist to help you along (see reference slide)

Write functions and documentation intelligently.

Make modular functions, follow best practices

- One operation/task = one function.
- Think about readability and future editability.
- Parameterize any aspects the user may want to specify.

(Remember to plan for growth and change!)

Good jobs for one function:

- Calculate a value
- Generate one plot
- Produce one dataframe

BAD job for one function:

- Calculate seventeen values, produce six dataframes, and produce four plots based on those dataframes

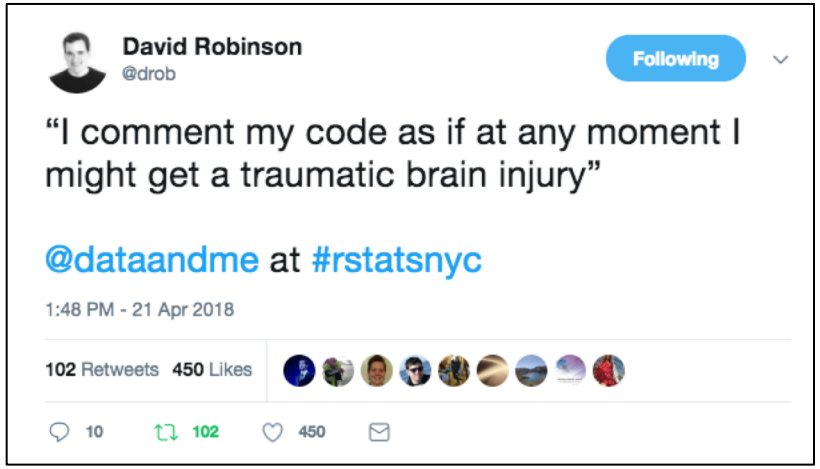
Make robust documentation a norm and a habit.

At the earliest stages of team infrastructure, documentation is haphazard.

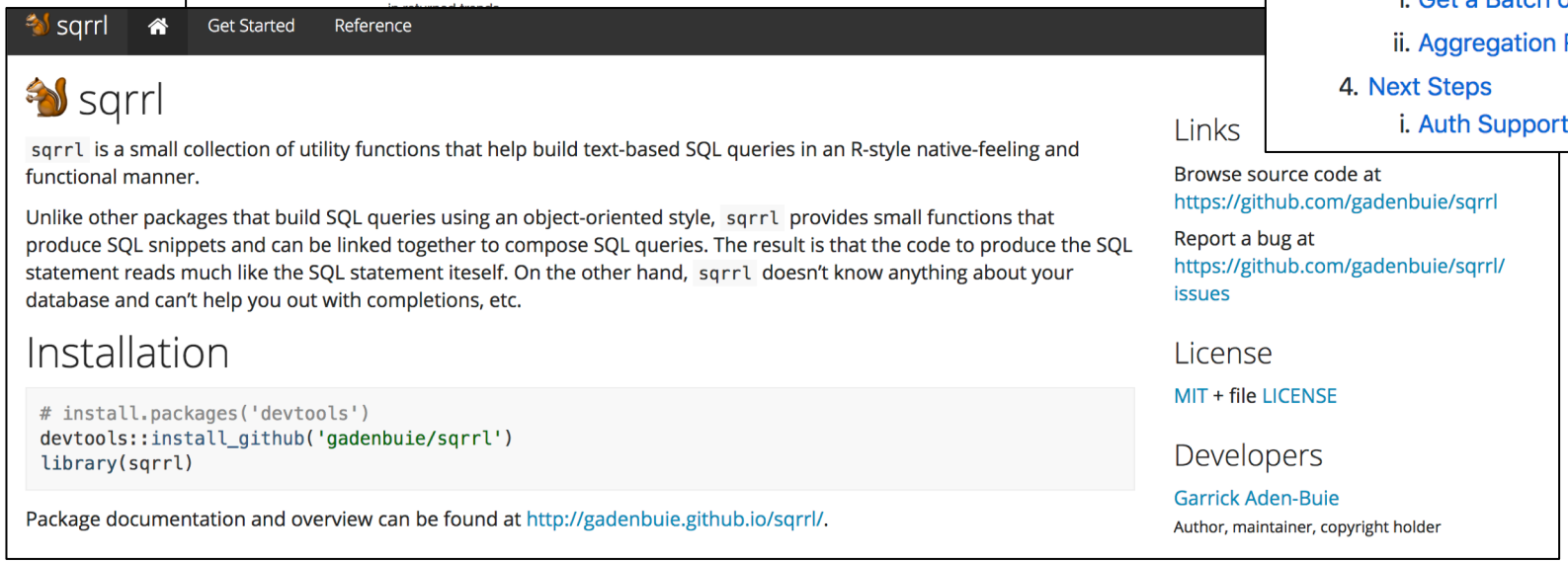
Write for the layperson or for ‘future you’.

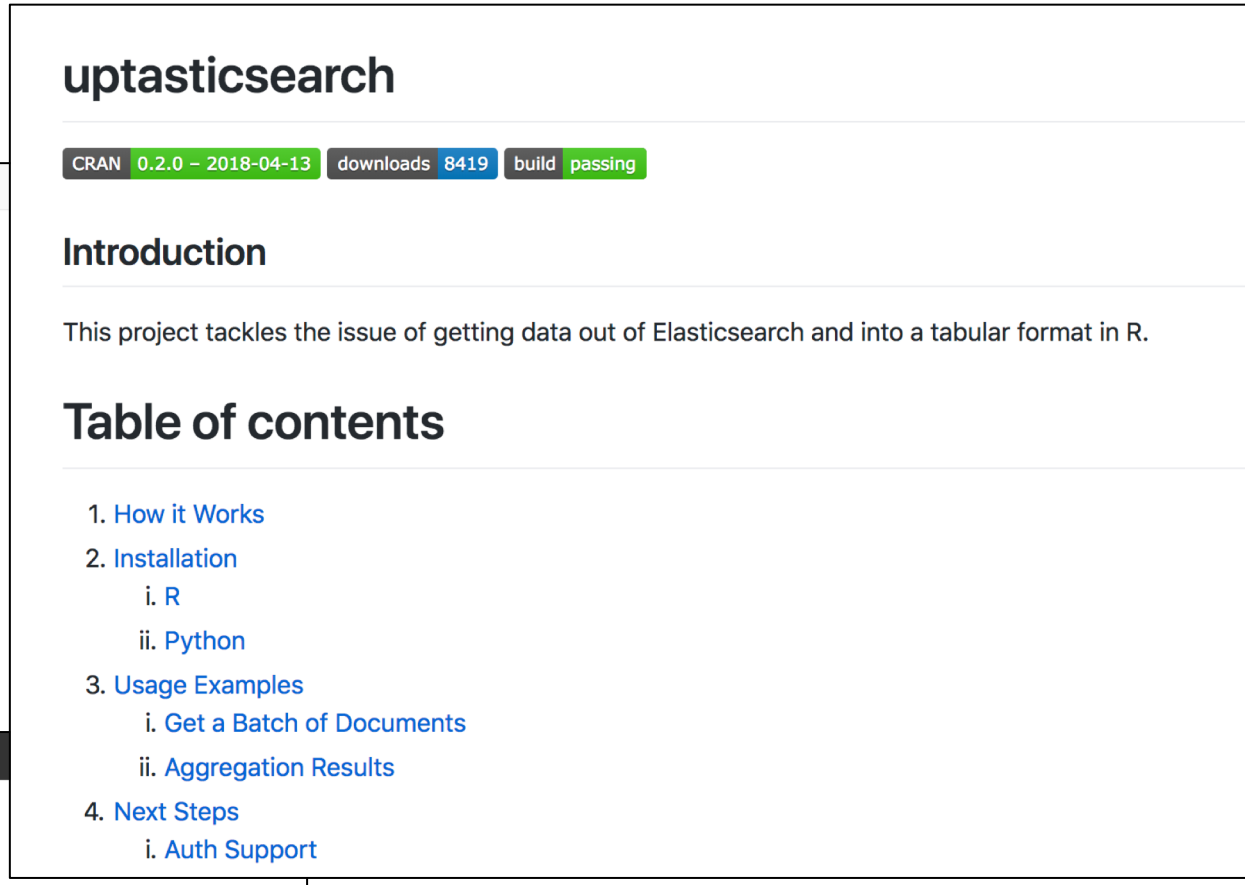
R packages have a huge assortment of documentation possibilities!

- Function help docs
- Vignettes
- Package level descriptive documents
- Websites via pkgdown (http://pkgdown.r-lib.org/)



Great documentation



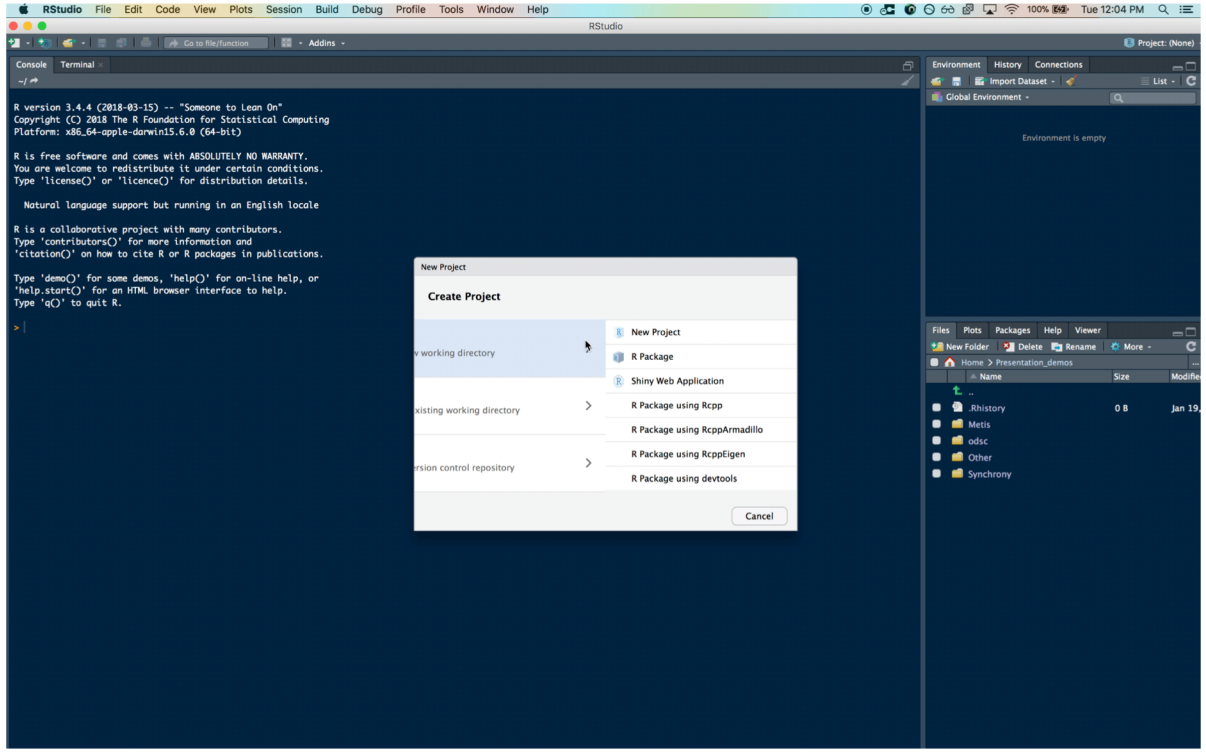


http://rtweet.info/reference/get_trends.html
https://gadenbuie.github.io/sqrrl/index.html
https://github.com/UptakeOpenSource/uptasticsearch

Building a Simple R Package - Tutorial

Essential Steps

1. Write or collect some functions to start with
2. Environment setup (libraries)
3. Initialize package skeleton
4. Write documentation (each function and the overall package)
5. Build
6. Share!



See my video tutorial and scripts at https://github.com/skirmer/odsc2018 to follow along step by step.

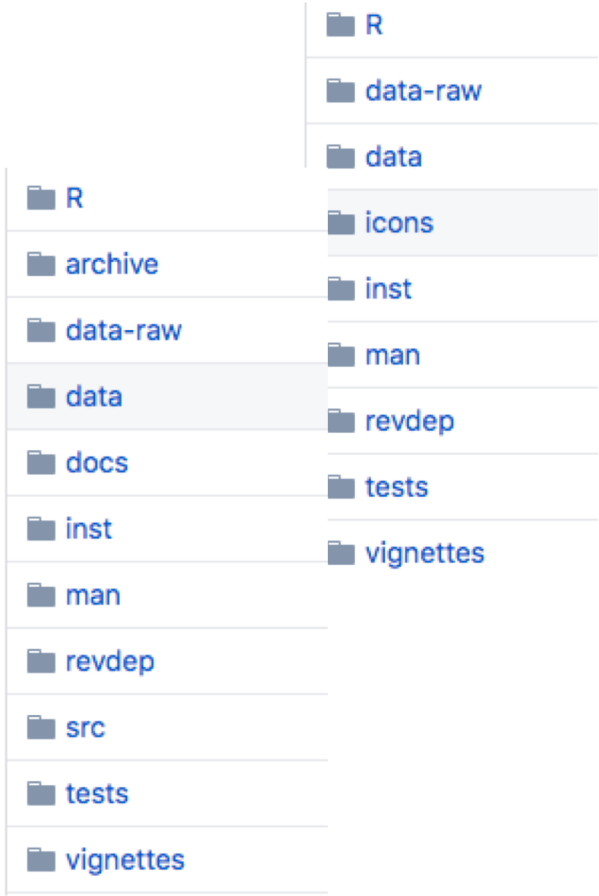
R Package Infrastructure

Files:

- DESCRIPTION: package overview
- NAMESPACE: dependencies/exporting management

Folders:

- R: Function scripts
- man: Function help docs
- vignettes: Rmd files explaining the functions' operation in depth
- tests: unit testing code to ensure correct function behavior



AND THEN: you can include pretty much anything else you like.

inst, data, images, docs, etc.

Even the most complex, powerful R packages out there all use this same construction.

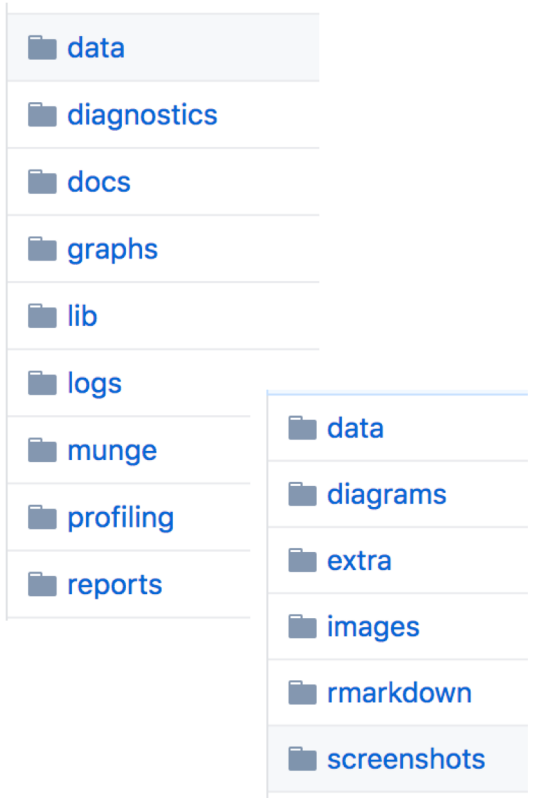
When you have tools ready to be used, teach your colleagues.

Let data scientists who authored functions teach their peers.

Proactively teach BI, analytics, executive, and sales staff

- What is this, and why should they care?
- Empower your colleagues to be your collaborators.

Over time, add more things that people need.

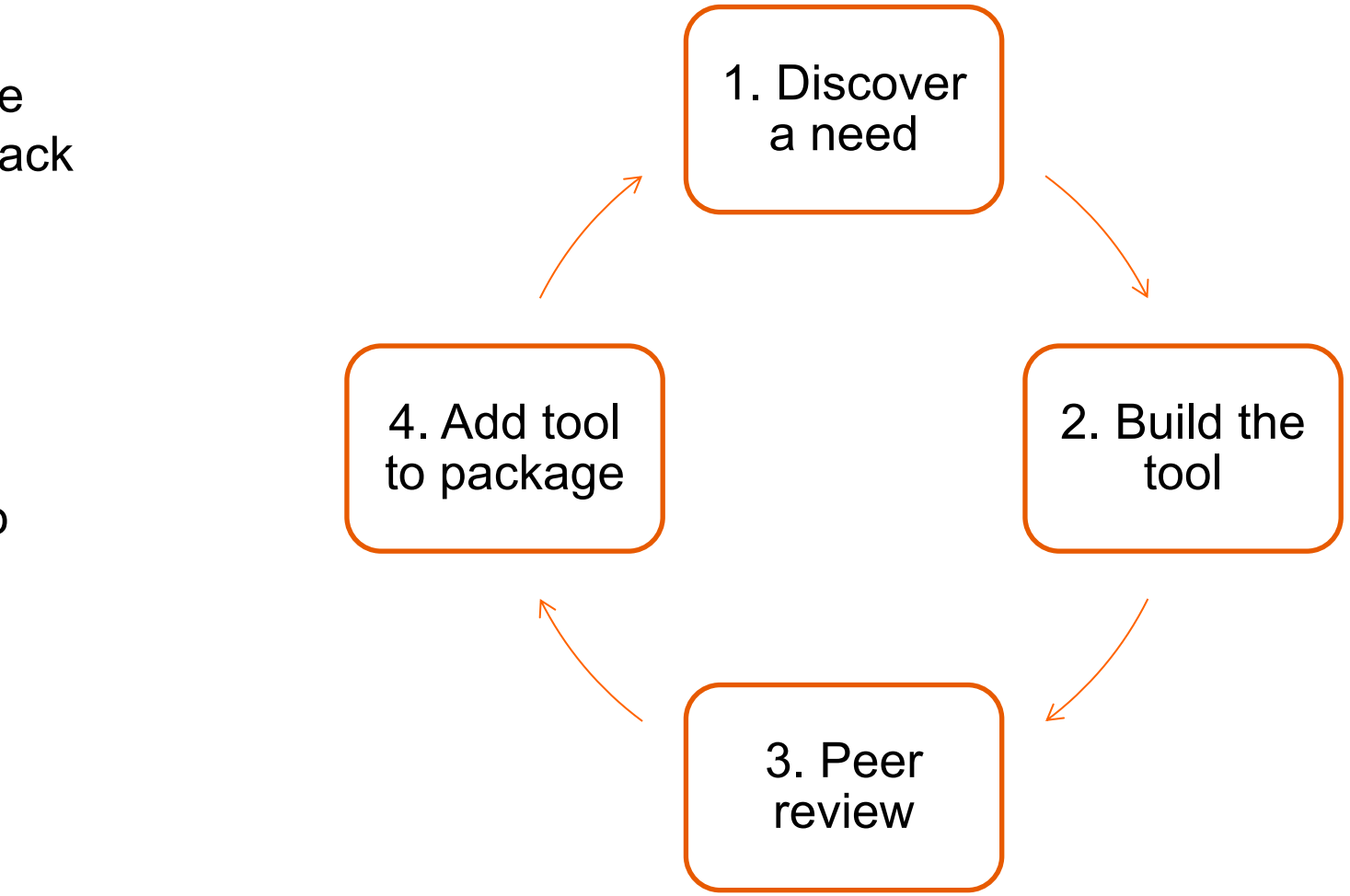


Add new features when the need arises, in a systematic way.

Ideas for new features can come from any user – welcome feedback

Don’t skimp on the peer review element!

Do occasional housecleaning to ensure things are still working, efficient, and relevant



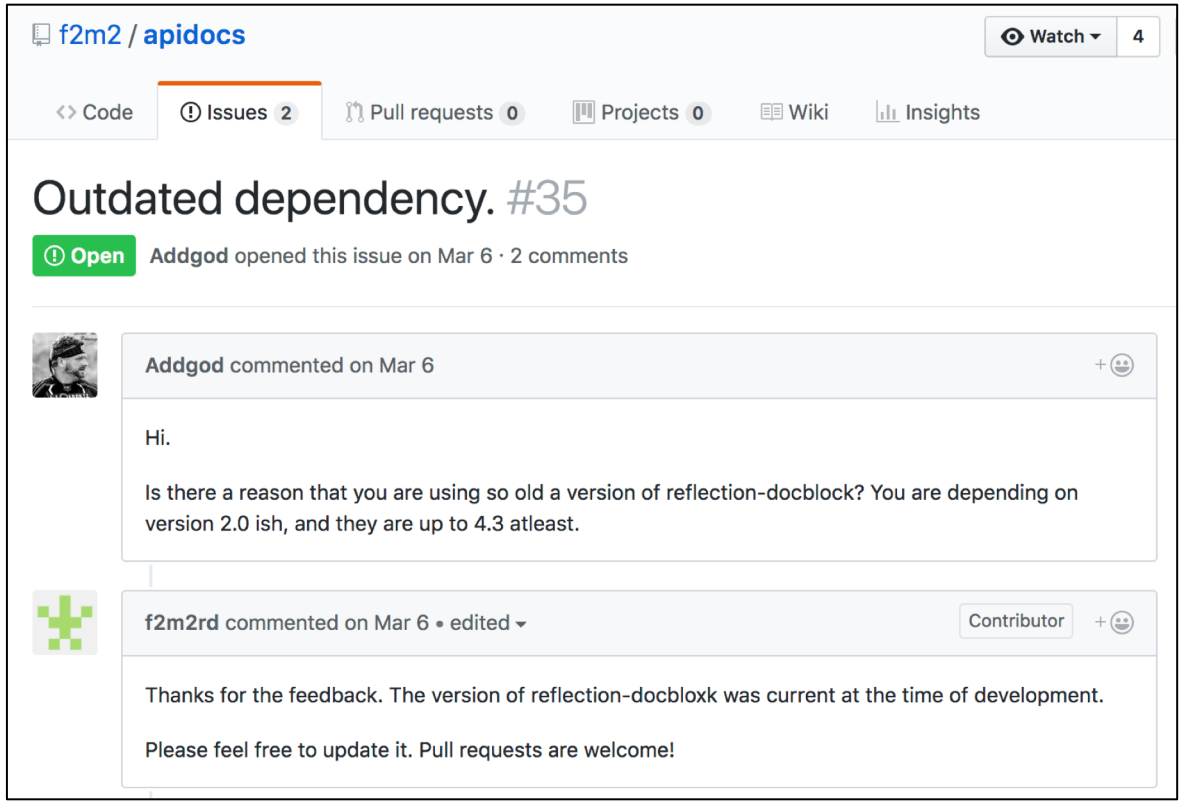
Make collaboration a core priority across the team.

Support improving the package, not just creating the package.

Stagnating documentation/tools < than no tools at all.

Move from single user ownership to team ownership.

- How you talk about the tool matters. Get people invested in it!



Other Technical Notes

Unit testing

- Programmatic checks that automatically run when a package is built
- Prevents unintentional breaking changes
- When to write unit tests, and how many, is a topic of debate.

Continuous Integration

- Contributions are built automatically in test environment to ensure constant compatibility

Internal CRAN

- Set up an internal-only repository where your packages live
- They can be installed with a traditional `install.packages()` call with a few small setup steps
- https://cran.r-project.org/doc/manuals/R-admin.html#Setting-up-a-package-repository

Wrapping Up

Having smooth functioning, collaborative data science is possible!

You can do work that is reproducible, high quality, and less effort.

Your entire organization can be more data literate and self supporting, freeing up data scientists for innovation.

You can build a culture of learning and education for your whole organization, starting from data science.

Suggested Roadmap

1. Develop collaborative culture;
2. Get leadership support;
3. Plan and think ahead;
4. Build tools and write code thoughtfully;
5. Don’t skip documentation;
6. Educate your organization;
7. Maintain the tools!

Reference Material

How-To:

- Write your own functions: https://nicercode.github.io/guides/functions/
- Write great documentation: http://r-pkgs.had.co.nz/man.html
- Construct an R package:
  - https://github.com/skirmer/r_packages
  - https://github.com/jtleek/rpackages
- Build websites from package documentation: http://pkgdown.r-lib.org/
- Get familiar with git:
  - https://product.hubspot.com/blog/git-and-github-tutorial-for-beginners
  - https://guides.github.com/
  - https://try.github.io/levels/1/challenges/1
- Write unit tests:
  - https://katherinemwood.github.io/post/testthat/
  - http://r-pkgs.had.co.nz/tests.html

Case Studies:

Capital One: https://github.com/emilyriederer/references/blob/master/tidycf_Riederer_rstudio.pdf

Airbnb: https://peerj.com/preprints/3182/

Tools:

- https://bitbucket.org/
- https://about.gitlab.com/
- https://jenkins.io/
- https://travis-ci.org/

@data_stephanie
www.stephaniekirmer.com
https://github.com/skirmer/odsc2018/