Replizierbarkeit und Kollaboration im Forschungsalltag: Praktische Empfehlungen

Maximilian Sprengholz (HU Berlin & BIM)

Colloquium Empirische Methoden der Sozialforschung. 05/23

Problemstellung

- Wissenschaftliche Ergebnisse sollen reproduzierbar sein (Gayle & Connelly 2022):
    i. **Duplication:** Exakte Replikation gegeben derselben Daten, Methoden
    $\rightarrow$ rein technische Hürde
    ii. **Replication:** Freie Replikation mit anderen Daten, Methoden (siehe Breznau et al. 2022)
- **Never change a running system**: Das Problem entsteht erst, weil wir nicht wissen, warum es überhaupt lief
- Bei Kooperationen mit anderen Wissenschaftler*innen haben wir im Kleinen, was generelle Replizierbarkeit im Großen ist: Wir müssen auf robuste, weitestgehend universelle Standards einigen und diese bestmöglich dokumentieren
- Es ist kompliziert!

American Economic Review

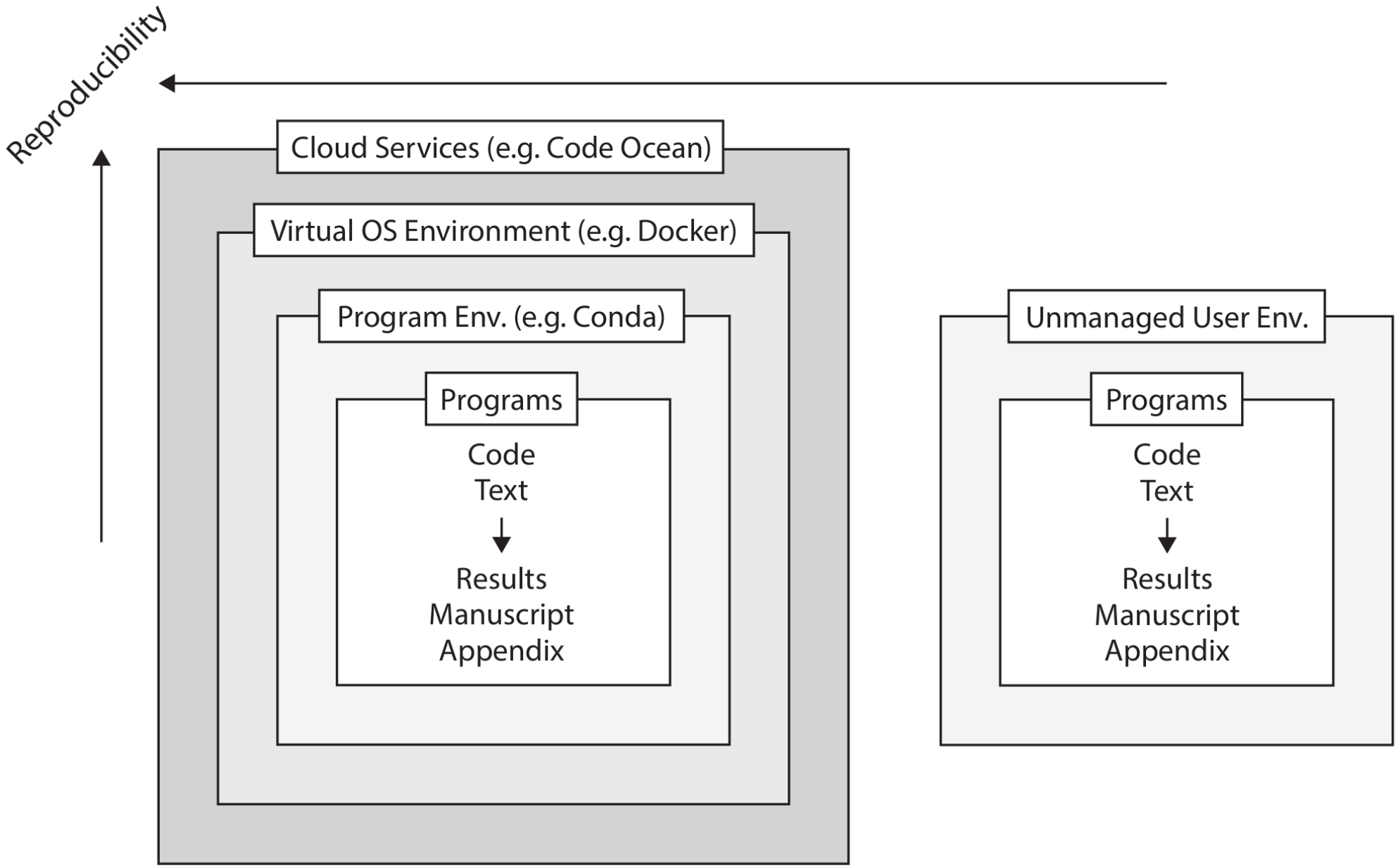
> It is the policy of the American Economic Association to publish papers only if the data and code used in the analysis are clearly and precisely documented and access to the data and code is non-exclusive to the authors.

European Sociological Review

> Where ethically feasible, European Sociological Review strongly encourages authors to make all data and software code on which the conclusions of the paper rely available to readers. Authors are required to include a Data Availability Statement in their article. This policy applies to all papers submitted to the journal on or after 5 September 2022.

Was soll replizierbar sein?

- **Im besten Fall alles!**
  - Datenaufbereitung
  - Analyse
  - Dokumente
- Replizierbarkeit zwar ist kein Kontinuum...
- ...aber mit ein paar Basics kommen wir schon ziemlich weit 👍


Reproducibility
Cloud Services (e.g. Code Ocean)
Virtual OS Environment (e.g. Docker)
Program Env. (e.g. Conda)
Programs
Code
Text
Results
Manuscript
Appendix
Unmanaged User Env.
Programs
Code
Text
Results
Manuscript
Appendix

Wir brauchen Struktur: Self-contained Projects (und gute Computerpraxis)

- Jedes Projekt sollte einen festen Ort mit einer sinnvollen (selbsterklärenden) **Ordnerstruktur** haben
- Wie *genau* diese aussieht, ist nicht so wichtig, so lange ein paar grundlegende Regeln beachtet werden (siehe exzellentes How-To zu "good enough practices", Wilson et al. 2017)
- Bei regelmäßig neuen Projekten empfiehlt sich eine Automatisierung, bspw. über `cookiecutter`

```
conda install cookiecutter -c conda-forge
cookiecutter gh:maximilian-sprengholz/good-enough-project
```

- Minimal notwendiger Input wenn Projekte transferiert werden / von mehreren Personen genutzt werden:
  - **working directory** `wd` +
  - **relative Pfade** `wd/src/00_master.do`, `wd/results/figures/myplot.pdf` (forward slashed + no spaces + no special characters)
- **Master Files:** Hier werden `wd` und alles andere defininiert und alle Syntaxen gestartet
- Bereitstellung aller notwendigen Files oder entsprechender Konfigurationen, welche die Files rekonstruieren können
- **Raw** Daten sind tabu!

Dependency management

- Egal ob `R`, `Stata`, oder `Python`: Software + Erweiterungen ändern sich ständig $\rightarrow$ es geht nicht ohne projektspezifische, replizierbare Environments!
- `conda` (Anaconda, Miniconda, Mamba) hilft dabei:
  - Im weitesten Sinne ein Environment & Paketmanager
  - Wurde für `Python` entwickelt, aber installierbare Pakete sind nicht auf `Python` beschränkt (bspw. `Pandoc`, `R`)
  - Hat ein automatisches *dependency management* (anders als bspw. `pip`)
- Nicht immer ist `conda` notwendig bzw. die beste Lösung:
  - `Stata` unterstützt keinerlei automatisches dependency management
  - `R`: Paket `renv` sieht sehr vielversprechend aus
  - `Python`: functioniert am besten mit `conda` bzw. `pip`

Version Control mit `git`

- Kollaborative (Software-)Entwicklung mit Versionskontrolle
- Versionskontrolle des eigenen Projektordners (repository) sehr nützlich: lokal arbeiten, Änderungen remote speichern
- Tracking ist zeilenbasiert: Git funktioniert am besten mit Dateien im Syntax/Markup-Format (alle Programmiersprachen, aber auch `LaTeX`, `Markdown`)
- Kollaboration: Erfordert händischen `merge` wenn mehrere Personen die gleiche Stelle des Files bearbeiten
- Die HU betreibt eine eigene GitLab Instanz, öffentliche und private repositories inklusive

- Fixe Grundstruktur + `git`: Projekte bleiben viel sauberer: Im Idealfall gibt es nur noch `myfile.do` mit gespeichertem Changelog und nicht mehr `myfile1.do`, `myfile2.do`, `myfile_final.do`, `myfile_FINALFINAL.do` ...
- Es kann trotzdem alles probiert werden lokal (oder in branches, forks): `commit` / `merge` selektiv + `.gitignore`
- Es geht nichts verloren!
- `git` ist ein Command Line Tool, aber es gibt sehr viele verschiedene GUIs, die wesentlich intuitiver sind (siehe Links am Ende)
- Plattformen wie `GitHub` und `GitLab`: Viele weitere nützliche Tools, z.B. GUI, Issues, Milestones, CI

Kollaboration beim Schreiben

- `Word` ist der Standard, hat aber folgende Nachteile:
  - Gleichzeitiges Arbeiten nur in der reduzierten Online-Version möglich
  - Offline: Keine zeilenbasierte Versionskontrolle
  - Dynamische Dokumente kompliziert, teilw. unmöglich
  - Die meisten Nachteile gelten auch für bspw. `OnlyOffice`
- Synchronisierung ist notwendig:
  - Der schlimmste Ansatz: Dokumente im "E-Mail-Umlaufverfahren"
  - Kein Problem mit `git`, aber auch mit jeder Cloud möglich (z.B. HU Box)
  - Dedizierte Services: Overleaf (Online `LaTeX`, HU Instanz), Authorea
- Trennung Analyse + Schreiben? Muss nicht: `RMarkdown`

Die Kür: Vollständige Automatisierung

- Im Idealfall müssen andere Wissenschaftler*innen *nichts* in den Projektfiles anpassen
- Da oft ein paar Infos übergeben werden müssen, kann dies aber über Argumente an ein Automatisierungstool geschehen, welches alles startet:
  Master vom Master 😵
- Es gibt viele verschiedene, z.B. `make`

Projektveröffentlichung

- Ein sauberes und gut dokumentiertes Repository ist im Idealfall direkt reproduzierbar und kann ohne viel zusätzlichen Aufwand veröffentlicht bzw. zur Verfügung gestellt werden
- Ziel: Open Access Hosting der Repositories mit Versionierung, Metadaten und DOI
- Mögliche Plattformen sind OSF, Zenodo und (kostenpflichtig) CodeOcean
- Einige der Plattformen haben Integration mit den großen `git` Plattformen, so dass eigene repositories bei release z.B. direkt von GitHub zu Zenodo gepusht werden
- OSF Beispiel, Zenodo Beispiel
- CodeOcean Beispiel

Links

- Großartiger Kurs von Simona Picardi (Basics generell nutzbar, Rest `R`)
- `cookiecutter`
- HU Git Instanz, GitHub
- Freie und plattformunabhängige Git GUI Clients: SmartGit, GitFiend, Gitnuro; VScode Extensions z.B. `gittree`
- Online LaTeX mit Overleaf: HU Instanz, Offizielle Domain
- `conda`
- `R`: `renv`

References

Breznau, N., Rinke, E. M., Wuttke, A., Nguyen, H. H. V., Adem, M., Adriaans, J., Alvarez-Benjumea, A., Andersen, H. K., Auer, D., Azevedo, F., Bahnsen, O., Balzer, D., Bauer, G., Bauer, P. C., Baumann, M., Baute, S., Benoit, V., Bernauer, J., Berning, C., … Żółtak, T. (2022). Observing many researchers using the same data and hypothesis reveals a hidden universe of uncertainty. Proceedings of the National Academy of Sciences, 119(44), e2203150119. https://doi.org/10.1073/pnas.2203150119

Gayle, V., & Connelly, R. (2022). The Stark realities of reproducible statistically orientated sociological research: Some newer rules of the sociological method. Methodological Innovations, 15(3), 207–221. https://doi.org/10.1177/20597991221111681

Wilson, G., Bryan, J., Cranston, K., Kitzes, J., Nederbragt, L., & Teal, T. K. (2017). Good enough practices in scientific computing. PLOS Computational Biology, 13(6), e1005510. https://doi.org/10/gbkbwp