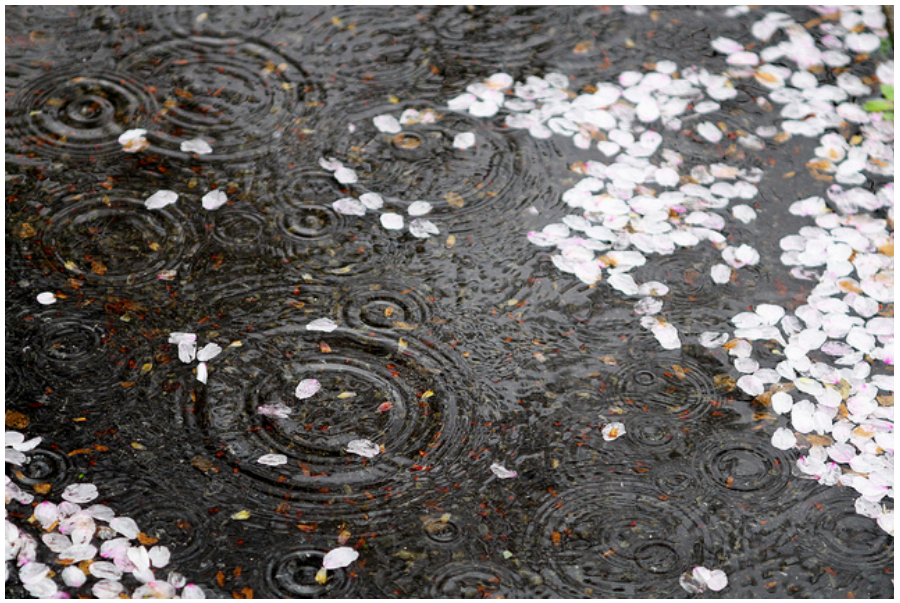
Software Ecosystems: Spreading the work around


https://www.flickr.com/photos/mrhayata/6933963596

James Howison
School of Information
University of Texas at Austin

(based in part on joint work with Jim Herbsleb at Carnegie Mellon)

This material is based upon work supported by the US National Science Foundation under Grant Nos. SMA- 1064209 (SciSIP), OCI-0943168 (VOSS) and ACI-145348 (CAREER).

Reuse

- Digital information can be copied
  - High design costs
  - Ultra-low instantiation costs
  - Cheap network distribution
- Implications:
  - "Write once, run anywhere"
  - Everyone gets a car!

Recombination

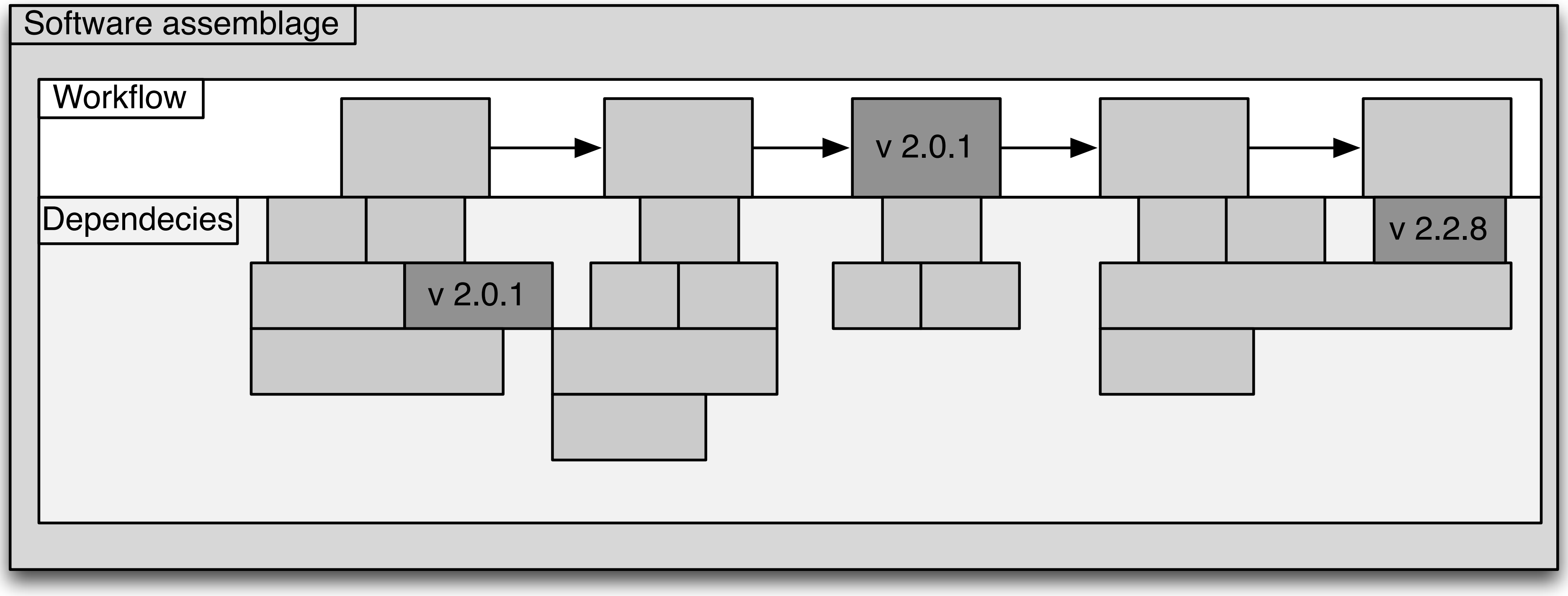
- Digital information is very flexible
  - Patched
  - Wrapped
  - Extended
  - Recombined
- Re-combinability is great for innovation
  - Lots of new ways to do things
  - But a sting in the tail?

https://www.flickr.com/photos/44551921@N04/

Today's questions

1. How does recombination make software work harder?
2. What sort of work needs to be done to cope?
3. How do different ways of working spread that work around?

How do scientists use software?



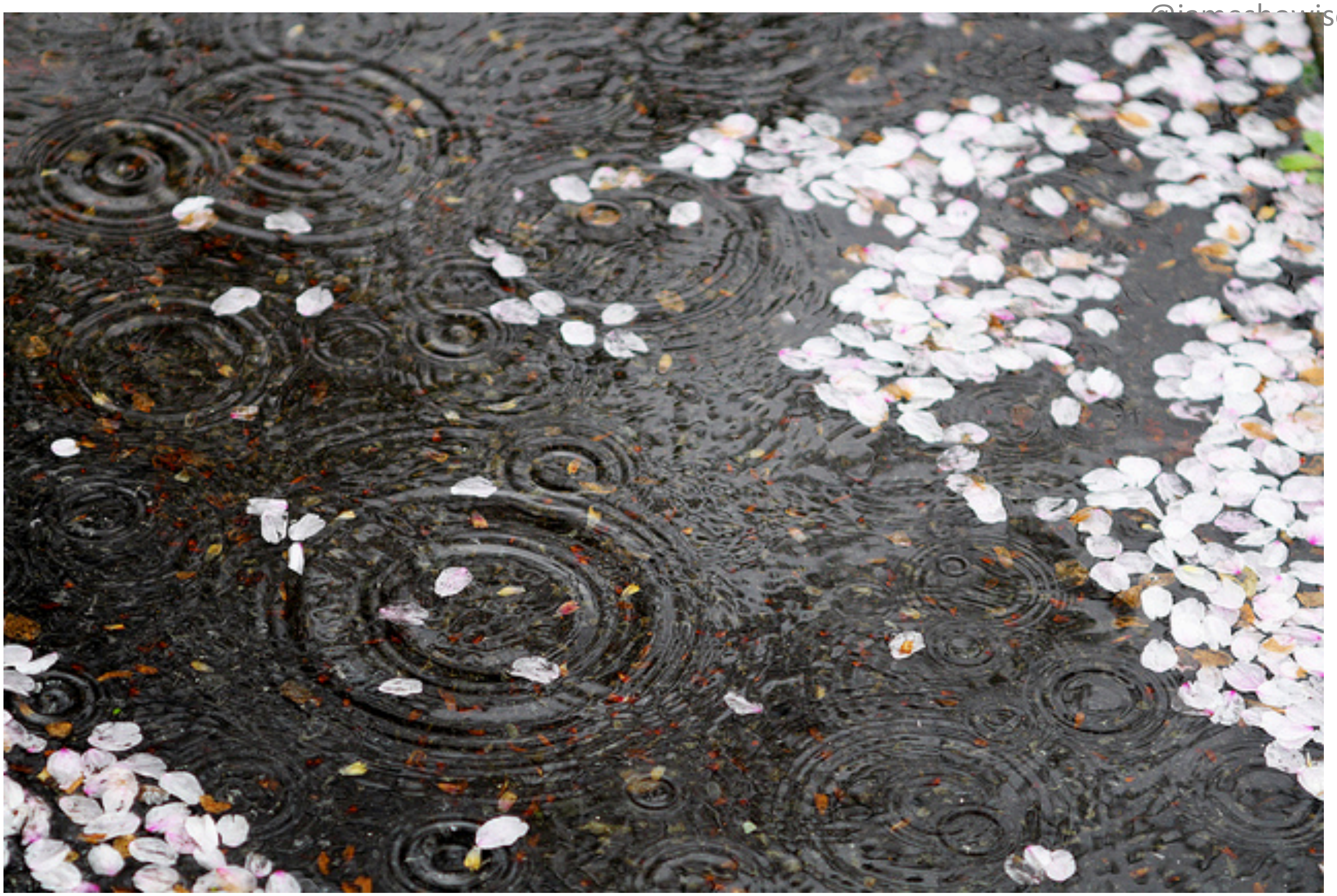
Edwards and Batcheller, Deelman, Bietz and Lee, Segal, Olson and Olson, De Roure and Gobels, Ribes and Finholt, Howison and Herbsleb

Re-animating assemblages

- Scientists pull an assemblage together, “get the plots” and often then leave it, often for months or years.
- When they return they return to *extend*; to use the software assemblage for new purposes, for new science, not simply to replicate.

But the world changes …

- Reanimation encounters change in the software ecosystem
  - Updated packages, New packages, New interfaces
- And not just in the immediate components of a workflow, but in the dependencies and complements.
- This work is echoed at component producers, since components are themselves assemblages.



https://www.flickr.com/photos/mrhayata/6933963596

What work holds a software ecosystem together (if anything)?

- Sensing work
  - knowing how workflows/software "out there" are changing
- Adjustment
  - making appropriate changes to account for changing surroundings
- Synchronization
  - ensuring that changes in multiple components make sense together, avoiding cascades.

Holding things together is hard work

But you can’t unlock the potential of cyberinfrastructure without it

How is this work spread around?

- If your software neighborhood works at all then this work is being done, by someone. In some fashion, not necessarily efficiently.

- How does this work happen:
  – What do we do within our project?
  – What work do we send to our users?
  – What work do we send to neighboring projects?

What work happens within our project?

Challenges

- Knowing what each other intended or intends
- Choosing dependencies with care and monitoring those projects
- Synchronizing our work internally
- Knowing who is doing the adjustment work to bring things into alignment

Techniques

- **“Atomic commits”**
  – Small, frequent contributions easier to follow than large “code bombs”
- **Test suites** communicate intentions
  – A failed integration test is a misunderstanding
- **Continuous integration** can help monitor dependencies
  – Pull new versions of dependencies and complements

What work do we send to our users?

Challenges

- How do we know:
  - What users do with our code?
  - What tools they combine it with?
  - How and how often they adjust our code (inc. wrappers, pre-processing)?
  - Where they go for help?
  - Do we talk to both users and their sys admins?
- How do we shape our users use?
  - Can't control them, but we can guide them.

Techniques

- Registration of users
- Build use reporting into our software (e.g., Thain, 2006) including reporting complements.
- Know "lead-users" but know longer tail as well.
- Concentrate, monitor, and *curate* help discussions.
- Documentation can nudge user behavior.

What work do we send to neighboring projects?

Challenges

- Know who we depend on (our upstream)
- Know who depends on us (our downstream)
- How do they adjust to our changes? Can we synchronize releases?
- Is our use of other components visible?
- Can we pass our adjustments upstream (saving others time)?

Techniques

- Be part of upstream and downstream projects (Read lists, attend events, work with their code). Include others in your events.
- Can your adjustment work be passed upstream?
- Are we the bug reporters that we'd like to have?

Different "organizational forms" spread the work out differently

1. The grant startup
2. The "service center"
3. The "merely open" project
4. The ecosystem player

The grant startup

- A new grant to a lab
- An internal team, low external transparency (like a "stealth mode" VC-funded startup)
- Experimenting, not wanting to bother others until "it works"
- From an ecosystem perspective, this can be very much like one big code dump, pushing sensing, adjustment, and synchronization onto end users and other projects.
- Two key actions:
  - Identify potential users and know what else they use
  - Exchange people with dependencies and complements

The “service center”

- “We’re funded to do the work for our users”
- Characteristics:
  - One way code membrane (we release, but don’t take contributions)
  - A Helpdesk (individual tickets solved in private)
- But: the team doesn’t scale to all the sensing, adjustment and synchronization needed
- Key actions:
  - Enroll our users, accept and learn from patches
  - Take support out into public, leveraging “active users”

The “passively open” project

- Projects have all the trappings of open source
  - Open source license
  - Hosted on github, maybe even Jenkins
  - Core team might even work in public
- Outside contributions are possible, but don’t often occur
- Openness at least means you can be “sensed” by others, but that puts the work on others.
- Key actions:
  - Go from passive to actively open, encouraging contributions
  - Sense how the code is used and what it is combined with

The ecosystem player

- Funded to build community and to cultivate the work of others
- Knows their “ecosystem” neighbors
- Actively senses how users use their code (e.g., actively curating help discussions where ever they occur)
- Has dense project co-memberships (upstream and downstream)
- Works to synchronize releases with neighbors.
- Key actions:
  - Document what you do so that others can learn
  - Document the value of what you do!

Takeaways

- Recombination is a key affordance of software, and it means that projects exist in a "neighborhood" of direct and indirect dependencies
- Over time dependencies lead to new kinds of work
  - **Sensing**: knowing what is nearby and how it is changing
  - **Adjusting**: changing to account for nearby changes
  - **Synchronizing**: gathering adjustments in time to avoid cascades
- If your neighborhood works, *someone* is doing that work.
- The way your project organizes its work spreads this work around your neighborhood.

Suggested sessions

- Who in this room is in your neighborhood? Grab them and talk about who does sensing, adjustment and synchronization?
- Is there scope for a software distribution in our neighborhood?
- Tell someone a story of epic adjustment work that you did.
  - Could it have been avoided? Could it have been shared with others to avoid them doing it? Did you publicize it?
- Map your neighborhood; especially complementary packages.
- Brainstorm how to sense users better
  - Do we concentrate and curate our users' help discussions? Do they stackexchange? Can we be more active there?
- How could a funded Software Institute help?

References

Batcheller, A. L. (2011). *Requirements Engineering in Building Climate Science Software.* (Ph.D. Dissertation). University of Michigan. Retrieved from http://deepblue.lib.umich.edu/handle/2027.42/86438

Bietz, M. J., Baumer, E. P., & Lee, C. P. (2010). Synergizing in Cyberinfrastructure Development. *Computer Supported Cooperative Work*, *19*(3-4), 245–281. http://doi.org/10.1007/s10606-010-9114-y

Borgman, C. L., Wallis, J. C., & Mayernik, M. S. (2012). Who's Got the Data? Interdependencies in Science and Technology Collaborations. *Computer Supported Cooperative Work (CSCW)*, *21*(6), 485–523. http://doi.org/10.1007/s10606-012-9169-z

Brown, D. A., Brady, P. R., Dietz, A., Cao, J., Johnson, B., & McNabb, J. (2007). A Case Study on the Use of Workflow Technologies for Scientific Analysis: Gravitational Wave Data Analysis. In I. J. Taylor, E. Deelman, D. B. Gannon, & M. Shields (Eds.), *Workflows for e-Science* (pp. 39–59). London: Springer. Retrieved from http://link.springer.com/chapter/10.1007/978-1-84628-757-2_4

Edwards, P. N. (2010). *A vast machine computer models, climate data, and the politics of global warming*. Cambridge, Mass.: MIT Press. Retrieved from http://site.ebrary.com/id/10424687

Edwards, P. N., Mayernik, M. S., Batcheller, A. L., Bowker, G. C., & Borgman, C. L. (2011). Science friction: Data, metadata, and collaboration. *Social Studies of Science*, *41*(5), 667–690. http://doi.org/10.1177/0306312711413314

Goble, C., De Roure, D., & Bechhofer, S. (2013). Accelerating Scientists' Knowledge Turns. In A. Fred, J. L. G. Dietz, K. Liu, & J. Filipe (Eds.), *Knowledge Discovery, Knowledge Engineering and Knowledge Management* (pp. 3–25). Springer Berlin Heidelberg.

Howison, J., Deelman, E., McLennan, M. J., Silva, R. F. da, & Herbsleb, J. D. (2015). Understanding the scientific software ecosystem and its impact: Current and future measures. *Research Evaluation*, *rvv014*(First published online: July 27, 2015), 17 Pages. http://doi.org/10.1093/reseval/rvv014

Howison, J., & Herbsleb, J. D. (2011). Scientific software production: incentives and collaboration. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work* (pp. 513–522). Hangzhou, China. http://doi.org/10.1145/1958824.1958904

Howison, J., & Herbsleb, J. D. (2013). Incentives and Integration in Scientific Software Production. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work* (pp. 459–470). New York, NY, USA: ACM. http://doi.org/10.1145/2441776.2441828

Howison, J., & Herbsleb, J. D. (2014). *The sustainability of scientific software production.* Working Paper, University of Texas at Austin.

Lee, C. P., Bietz, M. J., Derthick, K., & Paine, D. (2012). A Sociotechnical Exploration of Infrastructural Middleware Development. Presented at the CSCW.

Segal, J. (2009). Software Development Cultures and Cooperation Problems: A Field Study of the Early Stages of Development of Software for a Scientific Community. *Computer Supported Cooperative Work (CSCW)*, *18*(5), -606. http://doi.org/10.1007/s10606-009-9096-9

Segal, J., & Morris, C. (2008). Developing Scientific Software. *IEEE Software*, *25*(4), 20.