Software Supply Chains: a Tutorial

Audris Mockus
University of Tennessee
audris@utk.edu

ISEC'22 *[2022-02-03 Thu]*

Outline

About the speaker

Where/Who/What

- Moscow Physical Technical Institute, Carnegie Mellon University
- 23 years at Bell Labs/Avaya Labs
- Since Fall'14 at the University of Tennessee
  - Professor of Digital Archeology and Evidence Engineering
  - or Data Science/Big data/Software Engineering
- Looking for interested students, postdocs, visitors

Software Supply Chains

- What are Open Source Software Supply Chains?
- Why should anyone care about Open Source Software Supply Chains?
- What are the main types of OSS supply chains?
- What are key risks associated with each type of OSS supply chain?
- How to measure and manage these risks?
- Practical examples
  - World of Code or WoC
    - Current, Complete, Curated, Cross-referenced Collection of Open Source Version Control Data (CCCCCosvCd)
    - "Research Ready" for Supply Chain Research

History

- The term is used in wide variety of circumstances
- Inspired by traditional supply chains
  - “Supply Chain Management” (SCM) Keith Oliver (1982) [25] supply chain management [2], i.e., planning, organizing, and controlling in business logistics
  - (1999) [28] “a system of organizations, people, activities, information, and resources involved in moving a product or service from supplier to customer”

History

- Not consistently used for software
    - “software supply chain management” Jacqueline (1995) [14] instead of “system development life cycle”
    - Greenfield (2003) [11], “software components are created by software factories, and software

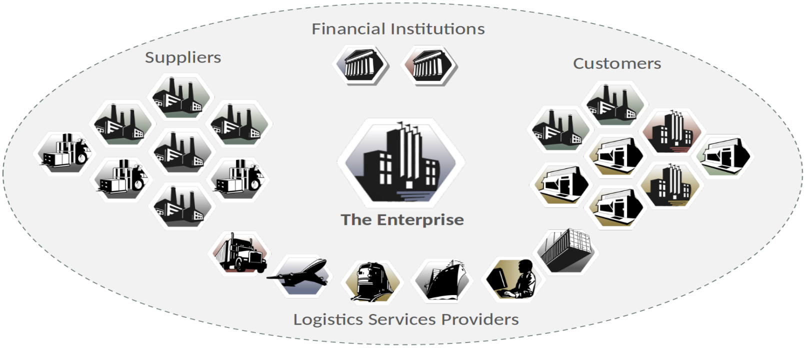
  supply chain is used to create standards to ease the alignment and assembly”
    - Aparna (2005) [4] “how to select from multiple suppliers” in “software focused supply chain.”
    - Ellison(2010) [9] “how risks can be introduced in coding, control management, deployment and operations.”

Traditional Supply Chain

is a set of three or more companies directly linked by one or more of the upstream or downstream flows of products, services, finance, and information from a source to a customer

Key features:

- Complex interdependencies
- Distributed decisions



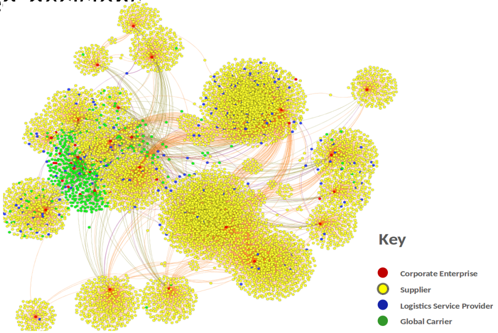
Graphics courtesy of Diane Palmquist and Greg Kefer, GT Nexus, UT Supply Chain Forum, November 12, 2014

Traditional Supply Chain

is a set of three or more companies directly linked by one or more of the upstream or downstream flows of products, services, finance, and information from a source to a customer

Key features:

- Complex interdependencies
- Distributed decisions



Graphics courtesy of Diane Palmquist and Greg Kefer, GT Nexus, UT Supply Chain Forum, November 12, 2014

Key concepts

- We define SSC by making analog of components in traditional SC.
    - Nodes:
        - Actors: developers and groups (companies)
        - Support/motivation: corporate backers supporting these developers or groups ("financing")
        - Software projects/packages/libraries
    - Links: relationships among software projects or packages
    - Product: Code, Effort, Knowledge
    - Actions: modifying/copying/learning/implementing the source code

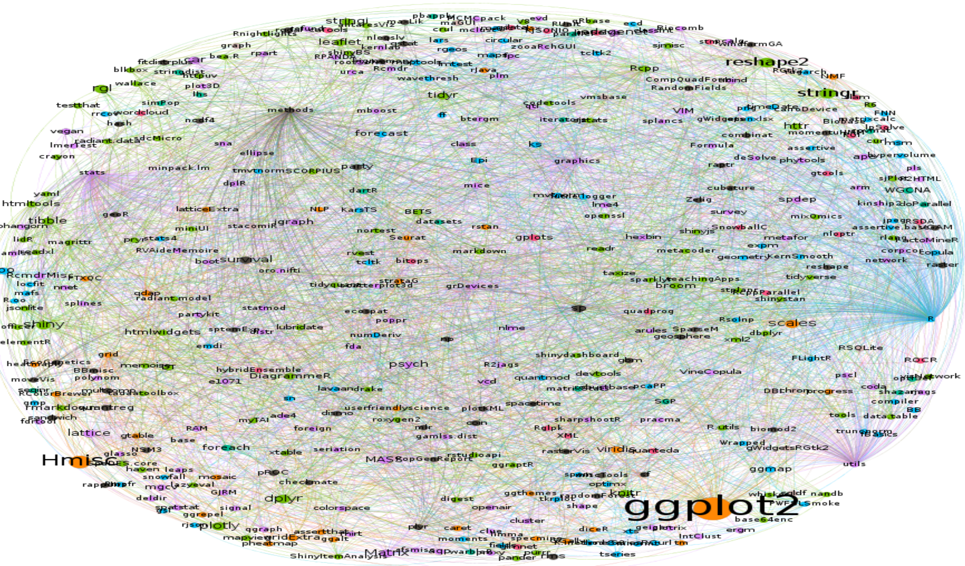
SSC of the first kind: “dependencies”

- Technical dependencies among projects with change effort as product flow: upstream effort either reducing or increasing the need for downstream effort
- Risks: unknown vulnerabilities, breaking changes, lack of maintenance, lack of popularity
- Measure: no way to tell how many projects use a package/library

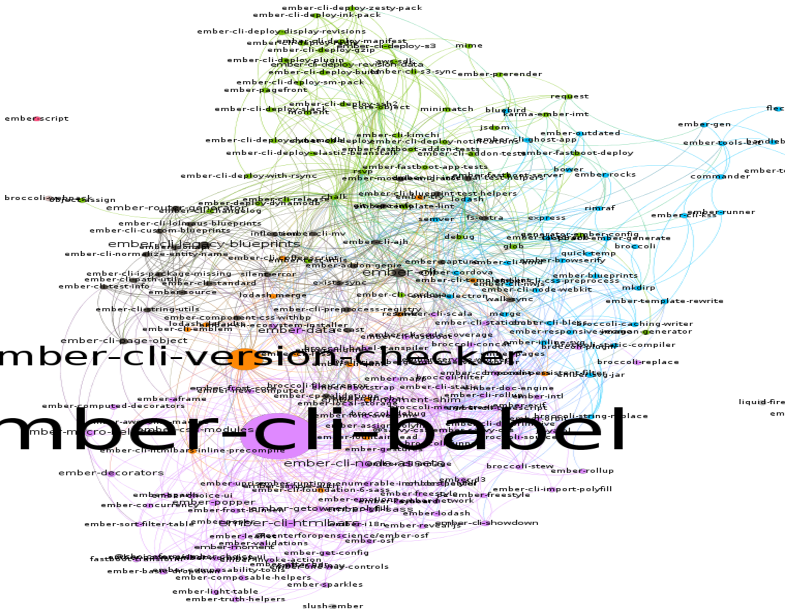
Examples

- Python: import re
- Java: import java.util.Collection;
- JavaScript: package.json

I(a) chain: CRAN



I(a) chain: ember



SSC of the first kind (b): “toolchain”

- Tool-chain needed to build the project
- Risks: tool support waining, changing standards, obsolescence of the tool chain

Examples

- make, ant, cmake, junit, gcc, jenkins, git, . . .
- LAMP, MEAN, . . .

SSC of the second kind: “copy”

- Copying of the source code from project to project as product flow (22 percent of files copied to, on average, 14 other repositories)
- Risks: license, unfixed bugs, no new functionality
- Measure: no way to identify what has been copied

Examples of SSC of the second kind

- Implementation of a complex algorithm
- Useful template
- Build configuration

Code-reuse-induced dependencies

- Collect all the blobs for a project
- Look for all other projects that contain the same blobs
- Investigate blobs that span many projects
- What are these code reuse patterns?

Code-reuse-induced — emberjs

- Build tools: rake — for Ruby on Rails
- Testing: qunit — testing framework
- Static: jQuery – a JavaScript library
- Framework: epf – Emberjs Persistence Foundation

Code-reuse-induced — emberjs

- Prior incarnations: SproutCore/Amber.js - early name for the EmberJS project,
- Hard forks: innoarch/bricks.ui - a hard fork of EmberJS that was developed as a separate project.
- Tutorials cookbooks/nodjs: early code examples
- Package manager: package.json — for npm

SSC of the third kind: “knowledge”

- Knowledge (product) flow through code changes as developers learn from and impart their knowledge to the source code
- Risks: developers may leave, companies may discontinue support, bad practices may propagate
- Measure: no way to know what projects a developer worked on

Examples of SSC of the third kind

- Developers gaining skills with tools/packages/practices
- Developers spreading practices, e.g., testing frameworks

What are Knowledge Flows

Transfer of tacit and explicit information, culture, behaviors, customs, or values among individuals or groups that occurs as a result of activities that are based on shared or interdependent artifacts, interests, or objectives [22]. It is not simply an exchange of explicit information as on, for example, StackExchange.

The three kinds do not cover patch deployment

- SolarWind and other patch deployment schemes

Might traditional SCs be relevant to SSCs?

- Developers are distributed and scattered all of the world geographically, but cooperate on the development of software products through virtual internet.
- A majority of software products are built on top of one or more mature software products by directly reusing source code of other projects, following successful design in other projects, involving core developers from other projects, etc.

Why should anyone care about Open Source Software Supply Chains?

- Why should one care about OSS?
  - underpins all other software via standard platforms, tools, and components
- Why should one care about SSCs?
  - no software is standalone: everything depends on decisions made by others

Why should anyone care about OSS?

- Source code available
- Software development is transparent
- If provider leaves, community will continue maintaining it
- No licensing costs
- Extremely rich resource of libraries components and frameworks in almost any language
- Provides most widely used industry-standard platforms (linux)
- Underpins most commercial software

Why should anyone care about OSS?

- Flexibility and Agility
- Share Maintenance Costs
- Cost-Effectiveness
- Transparency
- Speed
- Better Security

Why should anyone care about Software Supply Chains?

- Why should developers care about SSCs?
  - No software is standalone: need platform, tools, components, frameworks, libraries
- Why should users care about SSCs?
  - Will the stuff be fixed?
  - Can I access the source code (by whom)?
  - Will it continue to work as it does now?
- Why should enterprises care about SSCs?
  - Save costs: writing from scratch not an option
  - Follow standards/interoperability (e.g., linux)
- Why should governments care about SSCs?
  - Good for business
  - Good for training citizens

Why should anyone care about Software Supply Chains?

- With a supply chain perspective in OSS development, developers can better evaluate software components on various properties, e.g., risks, maintenance and quality, thus being enabled to make more wise decisions on the selection of downstream packages to u

se. Besides, as the increment of transparency and visibility in SSC, developers are enabled to seek for more talented software engineers to cooperate with, which leads to a faster growth in knowledge and learning experience for developers, more mature and successful software products, and meanwhile, speeds up the evolution of software ecosystems.

What are key risks of Software Supply Chains?

- Type Ia: unknown vulnerabilities, breaking changes, lack of maintenance, lack of popularity
- Type Ib: tool support, changing standards, obsolescence of the tool chain
- Type II: license compliance, known vulnerabilities/bugs, missing updated functionality
- Type III: developers may leave, companies may discontinue support

Visibility

Visibility
Information that developers have about the inputs, processes, sources and practices used to bring the product to consumers/market. This includes complete supply chain visibility including traceability for the entire supply chain. Visibility is, generally, inwardly/developer focused.

Transparency
Information that developers share with their consumers about the inputs, processes, sources and practices used to bring the product to the consumer. It is more outwardly focused/from the consumer perspective than visibility.

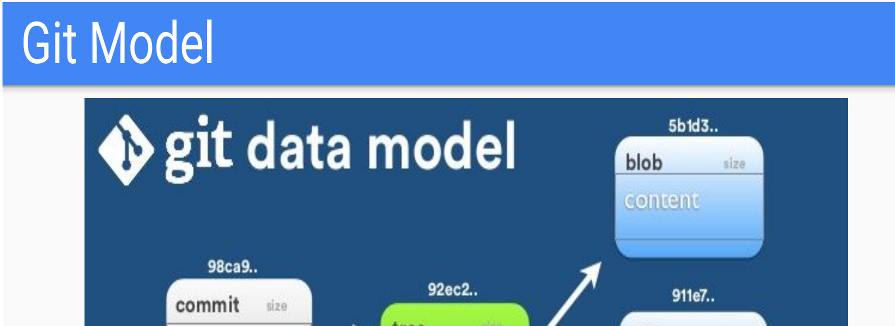
How to measure Software Supply Chains?

- Need all versions of all source code
  - Discover all git repositories
  - Clone/download these project repositories
  - Extract/correct necessary data
  - Construct SSCs

Aside on git

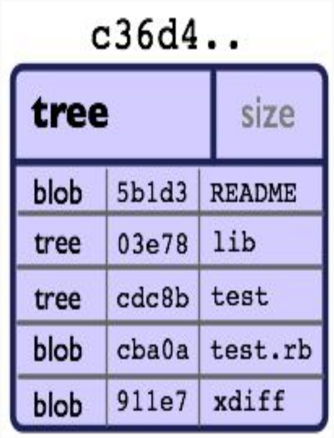
- git is a database containing sha1 indexed objects:
  - blobs: a source code file
  - trees: a directory
  - commits: a change
- sha1 of last commit (HEAD) reflects on entire past

**



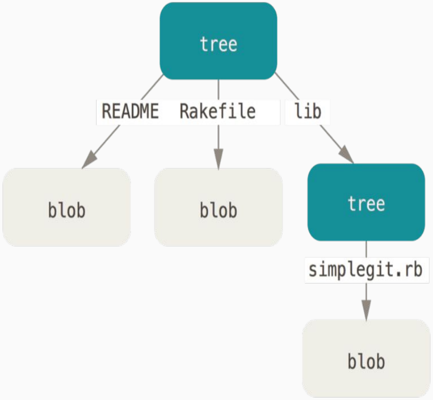
Git Model

Git Model - Trees

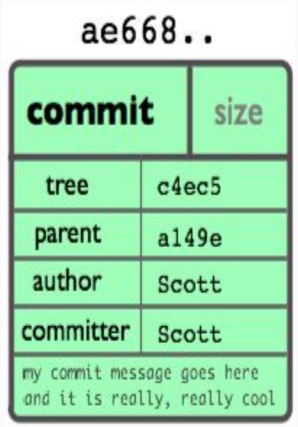


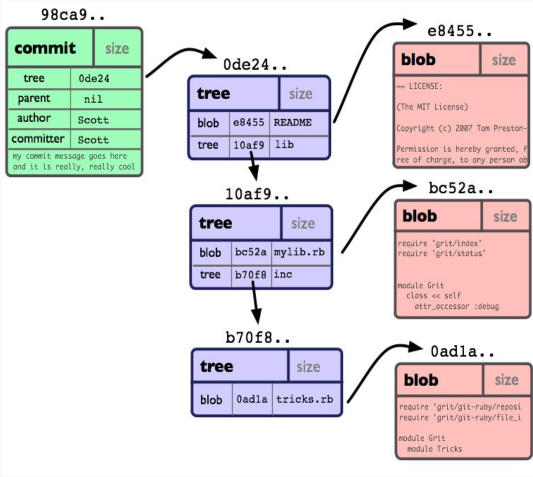
c36d4..

tree		size
blob	5b1d3	README
tree	03e78	lib
tree	cdc8b	test
blob	cba0a	test.rb
blob	911e7	xdiff



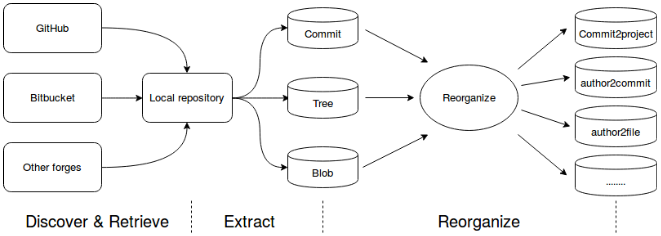
Git Model - Commits





Measurement infrastructure

- WoC: Discover, Retrieve, Store, Analyze, Update [21, 17]



- All: repositories over 50PB
- 20 percent semi-annual growth

Discover

- Forge discovery
  - History (since 1998)
    - Search/Developer News/Aggregators/Reviews
    - Major projects
    - Class assignments (Digital Archeology)
- Within-forge discovery
  - Use forge-specific APIs
  - Develop tricks to overcome limits
- Compare to other efforts
  - Softwareheritage.org: UNESCO sponsored
    - sole objective to capture all code
    - has 28 percent fewer commits than WoC

Storage/retrieval tricks

- Impossible to store all 50PB
  - De-duplicated: 250TB
- Can not transfer 50PB
  - Do only updates: use last commit
  - Approximately 200TB/quarter

WoC: What is it?

- Complete: discover git repos
  - approx 100 forges, 170M repos, 60+M Authors)
- Current: quarterly releases
- Curated/research ready
- WoC version U numbers are at bitbucket.org/swsc/overview/
- Projects, authors, blobs, APIs, commits fully cross-referenced
- How to use: github.com/woc-hack/tutorial
- Web interface: worldofcode.org

Version Control Data Challenges

- Not experimental data
  - Multiple contexts
  - Missing events
  - Incorrect, filtered, or tampered with
- Continuously changing
  - Systems and practices are evolving
- Challenges measuring or defining accuracy
- Potential for misinterpretation

Commits in VCS

- Context:
  - Why: merge/push/branch, fix/enhance/license
  - What: e.g, code, documentation, build, binaries
  - Practice: e.g., centralized vs distributed
- Missing: e.g., private VCS, no links to defect
- Incorrect: tangled, incorrect comments, wrong dates, authors
- Filtered: small projects, import from CVS
- Tampered with: `git rebase`

OSS data are rather bad

"productive" authors

Number of commits	Author
10960000	one-million-repo < *mikigal.acc@gmail.com* >
4400778	datakit < *datakit@docker.com* >
2463758	greenkeeper[bot] < *greenkeeper[bot]@users.noreply.github.com* >
2063212	Auto Pilot < *noreply@localhost* >
1864730	Your Name < *you@example.com* >

Silly competitions

- Fake commits with everyone's name to have most contributors

Authors	Repo
389993	cirosantilli/imagine-all-the-people

- The longest chain of commits

Length of commit chain	Repo
9959999	github.com/one-million-repo/biggest-repo-ever

More examples at bitbucket.org/swsc/overview/fun/

Data That Needs to be Curated

- Author IDs
- Repository forks
- Code dependencies (dozens of languages)
- Project types
- Link to external data sources
- Many other challenges

Curated Data: Partially Solved Problems

- Author IDs [1, 8, 10]
- Repository forks [24]
- Code dependencies (dozens of languages)
- Linking to external data sources
- Many other challenges

Example: Identifying Authors

- We have 34M strings
- Are these two the same person?
  - Aaron Lee < aaron.lee@rackspace.com>
  - Aaron Lee < wwkeyboard@gmail.com>
- Text similarity (adjusted for common names)
- Behavioral fingerprints:
  - Similarity of written text (Doc2Vec embedding)
  - Change the same source code files
  - Work in the similar time-zones
- Trained machine learning: 99.99 accuracy

Results: Identifying Authors

- Out of 16007 distinct author strings 10950 distinct authors
- Up to 14 ids for a single author
- Situation is much worse for more productive authors
- Dramatic differences in network measures

Eliciting WoC Requirements: History

- 2000-2007: Proof of concept [3, 20, 21]
- 2010-2016: A company-wide WoC [13, 12]
- 2016-2021: Conceptualizing and investigating SSCs [16, 18]
- 2020-: WoC community planning
  - Hackathons
  - Advisory Board

How to participate?

- worldofcode.org
- bitbucket.org/swsc/overview:
  - related publications / data / analysis
- Next hackathon
  - woc-hack.slack.com : sign up at `http://bit.ly/WoC-Hack`
  - github.com/woc-hack/tutorial
  - Signup form `http://bit.ly/WoCSignup`

WoC APIs

For a more comprehensive experience during Software Supply Chains tutorial at ISEC'22 please have your computer available during the tutorial. Also please sign up in advance at `http://bit.ly/WoC-Signup` and, once your account is created, log in and try out the World of Code tutorial: `https://github.com/woc-hack/tutorial`

What types of nodes are there?

- a - Author id; A - aliased author id
- p - repository (project); P - deforked project
- c - commit
- b - blob; ob - “old” blob (a predecessor version in a commit changing a file)
- f - filename (including full path)
- Pkg - imported package (from parsed blob)
- Def - implemented package (from parsed blob)
- Attributes
  - t - time stamp (of the commit)
  - l - language of the code in the blob

What types of links are there?

- SSC I:
  - b to def,pkg
  - A to Def, Pkg
  - P to Def, Pkg
- SSC II:
  - b to f,t[AP], also to f[aA] (first blob)
- SSC III:
  - [aA] to [pPcf] also to fb (first blob)
  - [pP] to [aAcf] also to fb (first blob)
- [aA] to [pPcbf] also to fb (first blob)

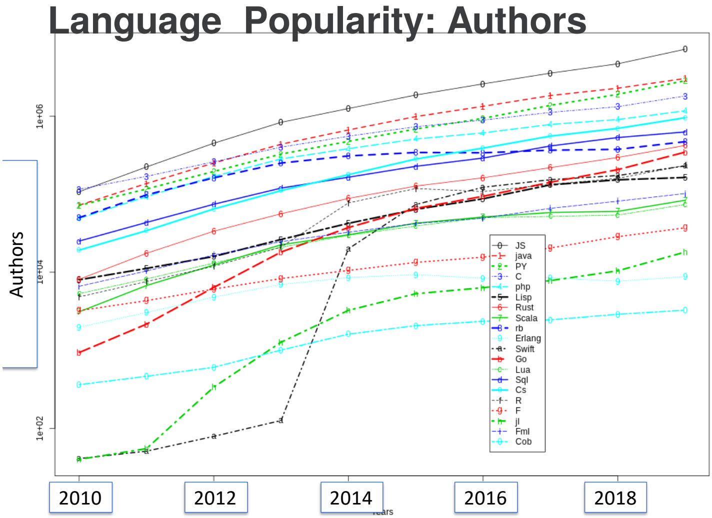
Why a resource like WoC is necessary?

- To improve research quality
    - Most software development is not for isolated projects
    - But current research practices ignore SSC relationships and lead to inadequate models/tools/practices
- To do entirely new types of research
    - What made OSS so successful so far?
    - Create fundamental theories in software engineering enabled by WoC and similar observatories

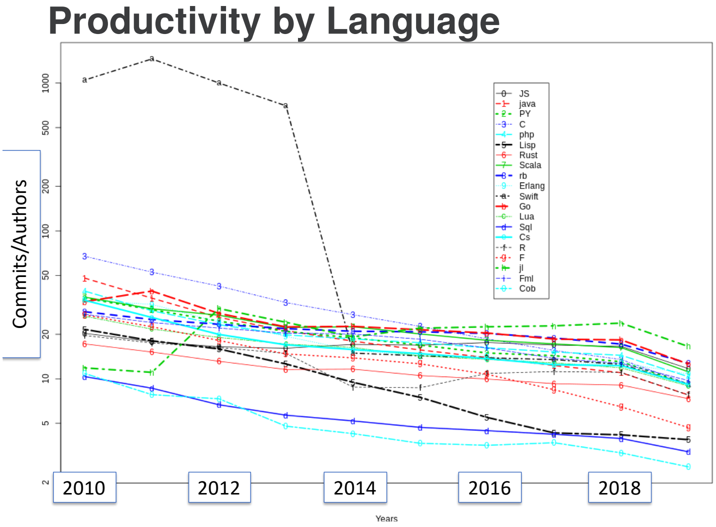
Unique features

- Key advantages WoC-like infrastructure
- Completeness: proper instead of convenience sampling
- Cross-referencing:
  - first time ever measure/study Type II and III SSCs and downstream of Type I SSCs
- Curation: don't need to spend a lifetime cleaning data Not accounting for activities downstream

What Languages are Popular?



How Difficult Each Language is?



Can we predict which technology will prosper/fail?

- Two data-science technologies: abstraction of data.frame in R
  - tidy vs data.table
- Sample: all R-language files in public repositories
  - 1.5M files, 5M versions
- When was the first time tidy or data.table included?
  - 17,536 projects use data.table
  - 7,032 projects use tidy

Results (Choice and Decision Maker)

- Choice
  - Exposure: Recent and Cumulative deployments, Mentions on StackOverflow
  - Qualities: Activity, developers,responsiveness, open issues
- Decision maker
  - Activity, developers
  - Performance needs
  - Social network
  - Technical network

Note: Red- negative, Green- positive

VDiOS (Vulnerability Detection in Open Source)

- "Orphan vulnerabilities"
  - Vulnerabilities in copied code that still exist in a project after they are discovered and fixed in another project.
  - Code is cloned and committed, not imported from a library or package manager.
  - Link to the original code does not exist or is not readily available.
  - Often overlooked part of the software supply chain.

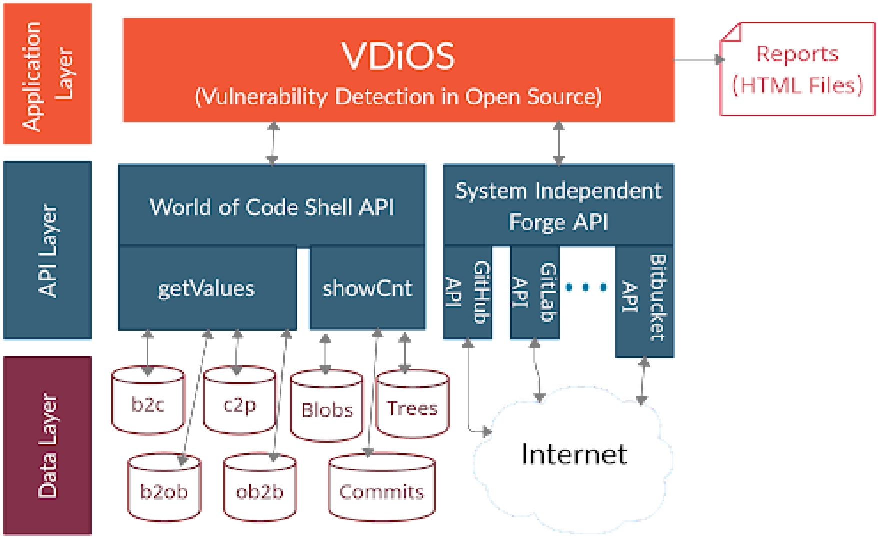
Risks from Orphan Vulnerabilities

- An exploit for such vulnerabilities may be widely known, making it easier to attack software with known vulnerabilities.
- Code in such repositories may be copied to other projects that may not be aware of the vulnerability.
- Code in such repositories may be built into applications and run by unsuspecting users.
- If a substantial number of OSS projects contain known and unfixed vulnerabilities, OSS may suffer reputational damage as a dump of low quality code where it may be hard to find high-quality projects

VDiOS operation

- Given a vulnerability fix in one project, identifies all other projects that:
  - Still contain the vulnerable code.
  - Used to contain the vulnerable code, but have now been fixed.
  - Used to contain the vulnerable code, have been changed, but we do not know if the change fixed the vulnerability.
- World of Code provides
  - Nearly complete collection of open source software
  - Allows VDiOS to find copied files at a scale that has traditionally been computationally infeasible.

VDiOS architecture



VDiOS summary

- Key Benefits:
  - Inform maintainers and users of still vulnerable projects about the risks of the vulnerability in their code.
  - Warn users that contemplate reusing such code about the unpatched vulnerabilities.

More Examples

- Contextualize/Correct/Impute [23]
  - Author aliasing/Bot detection: via behavioral fingerprinting [1, 8, 10]
  - De-forking/de-cloning via shared commits [24]
- Type I: dependencies
  - Models of spread [19]
  - Models of popularity [7]
  - Patterns of effort contribution [6]
- Type II: copying
  - Orphan vulnerabilities [26]
- Type III: knowledge
  - Knowledge at loss [27]
  - Developer impact [15]
  - Skill spaces [5]

Beyond SSCs

- Improve research quality, e.g.,
  - Avoid convenience sampling
  - Account for (often predominant) network effects
  - Conduct natural experiments
  - Increase productivity by sharing curated data
- Build tools for FLOSS developers
- Inform Enterprises

What has been done so far?

- Contextualize/Correct/Impute [23]
  - Author aliasing/Bot detection: via behavioral fingerprinting [1, 8, 10]
  - De-forking/de-cloning via shared commits [24]
- Type I: dependencies
  - Models of spread [19]
  - Models of popularity [7]
  - Patterns of effort contribution [6]
- Type II: copying
  - Orphan vulnerabilities [26]
- Type III: knowledge
  - Knowledge at loss [27]
  - Developer impact [15]
  - Skill spaces [5]
  - Eight implemented use cases
    - Relationships: code flow, technical and tool dependencies, knowledge flow [6, 7, 26]

Other names for SSC risks

- License/Regulatory
- Breaking Changes
- Lack of updates
- Corporate Involvement
- Exploits
- Truck factor
- Technology advancements

Takeaways

- OSS: why it is worthy of study
- SSC: why relevant for present software, types and risks
- OSS Observatory (WoC): help speed discovery in this novel area
- You can benefit too!

Bio

Audris Mockus worked at AT&T, then Lucent Bell Labs and Avaya Labs for 21 years. Now he is the Ericsson-Harlan D. Mills Chair professor in the Department of Electrical Engineering and Computer Science of the University of Tennessee.
He specializes in the recovery, documentation, and analysis of digital remains left as traces of collective and individual activity. He would like to reconstruct and improve the reality from these projections via methods that contextualize, correct, and augment these digital traces, modeling techniques that present and affect the behavior of teams and individuals, and statistical models and optimization techniques that help understand the nature of individual and collective behavior. His work has improved the understanding of how teams of software engineers interact and how to measure their productivity.
Dr. Mockus received a B.S. and an M.S. in Applied Mathematics from Moscow Institute of Physics and Technology in 1988. In 1991 he received an M.S. and in 1994 he received a Ph.D. in Statistics from Carnegie Mellon University.

Abstract

Software engineering (SE) studies practices employed by individual projects. The emergence of and extensive reliance on open source software (OSS) make that traditional SE focus too narrow to comprehend and support key developer decisions in the highly interconnected network of OSS. These networks have least three types of relationships: runtime or tool-chain dependencies, copying of the source code, and transfer of code maintenance expertise. These three relationships are similar to those in traditional supply chains (SCs) with maintenance effort, source code, and knowledge representing the product flow from supplier to consumer in traditional SCs. As in traditional SCs, the decisions are taken in a decentralized manner and risks may materialize because of events at nodes far away from the consumer or producer in the supply chain. Importantly, each of the three types of SSCs has unique advantages and risks. This tutorial will:

- Conceptualize these three OSS dependencies as types of software supply chains (SSCs),
- Present ways to operationalize the measurement of OSS SSCs,
- Go over several examples of the new insights, research questions, and applications of OSS SSCs
- Introduce World of Code (WoC) infrastructure designed to support research on OSS SSCs

Expected outcomes: Participants will be able to articulate the nature of the three types of OSS SSCs, understand primary risks and benefits of each type of SSC, and will gain basic skills needed to measure the SSCs using WoC infrastructure.

References I

[1] Sadika Amreen, Yuxia Zang, Chris Bogart, Russell Zaretzki, and Audris Mockus.
Alfaa: Active learning fingerprint based anti-aliasing for correcting developer identity errors in version control systems.
International Journal of Empirical Software Engineering, 2019.

[2] Ronald H Ballou.
Business logistics/supply chain management: planning, organizing, and controlling the supply chain.
Pearson Education India, 2007.

[3] Hung-Fu Chang and Audris Mockus.
Constructing universal version history.
In *ICSE'06 Workshop on Mining Software Repositories*, pages 76–79, Shanghai, China, May 22-23 2006.

[4] Aparna A Chhajed and Susan H Xu.
Software focused supply chains: Challenges and issues.
In *Industrial Informatics, 2005. INDIN'05. 2005 3rd IEEE International Conference on*, pages 172–175. IEEE, 2005.

[5] Tapajit Dey, Andrey Karnauch, and Audris Mockus.
Representation of developer expertise in opensource software.
In *ICSE 2021*. ACM Press, May 2021.

[6] Tapajit Dey, Yuxing Ma, and Audris Mockus.
Patterns of effort contribution and demand and user classification based on participation patterns in npm ecosystem.
In *Proceedings of the 15th International Conference on Predictive Models and Data Analytics in Software Engineering*. ACM, 2019.

[7] Tapajit Dey and Audris Mockus.
Are software dependency supply chain metrics useful in predicting change of popularity of npm packages?
In *Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering*, pages 66–69. ACM, 2018.

References II

[8] Tapajit Dey, Sara Mousavi, Eduardo Ponce, Tanner Fry, Bogdan Vasilescu, Anna Filippova, and Audris Mockus.
Detecting and characterizing bots that commit code.
In *IEEE Working Conference on Mining Software Repositories*, May 2020.

[9] Robert J Ellison and Carol Woody.
Supply-chain risk management: Incorporating security into software development.
In *System Sciences (HICSS), 2010 43rd Hawaii International Conference on*, pages 1–10. IEEE, 2010.

[10] Tanner Fry, Tapajit Dey, Andrey Karnauch, and Audris Mockus.
A dataset and an approach for identity resolution of 38 million author ids extracted from 2b git commits.
In *IEEE Working Conference on Mining Software Repositories: Data Showcase*, May 2020.

[11] Jack Greenfield and Keith Short.
Software factories: assembling applications with patterns, models, frameworks and tools.
In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 16–27. ACM, 2003.

[12] Randy Hackbarth, Audris Mockus, John Palframan, and David Weiss.
Assessing the state of software in a large enterprise.
Journal of Empirical Software Engineering, 10(3):219–249, 2010.

[13] Randy Hackbarth, Audris Mockus, John Palframan, and David Weiss.
Assessing the state of software in a large enterprise: A 12-year retrospective.
In *The Art and Science of Analyzing Software Data*, pages 411–451. Elsevier, 2016.

[14] Jacqueline Holdsworth.
Software Process Design.
McGraw-Hill, Inc., 1995.

[15] Andrey Karnauch, Sadika Amreen, and Audris Mockus.
Developer reputation estimator (dre).
In *ASE'19*, 2019.

References III

[16] Yuxing Ma.
Software Supply Chain (SSC) Development and Application.
PhD thesis, 2020.

[17] Yuxing Ma, Chris Bogart, Sadika Amreen, Russell Zaretzki, and Audris Mockus.
World of code: An infrastructure for mining the universe of open source vcs data.
In *IEEE Working Conference on Mining Software Repositories*, May 26 2019.

[18] Yuxing Ma, Tapajit Dey, Chris Bogart, Sadika Amreen, Marat Valiev, Adam Tutko, David Kennard, Russell Zaretzki, and Audris Mockus.
World of code: Enabling a research workflow for mining and analyzing the universe of open source vcs data.
International Journal of Empirical Software Engineering, 2020.

[19] Yuxing Ma, Audris Mockus, Russell Zaretzki, Bogdan Bichescu, and Randy Bradley.
A methodology for analyzing uptake of softwaretechnologies among developers.
IEEE Transactions on Software Engineering, 2020.

[20] Audris Mockus.
Large-scale code reuse in open source software.
In *ICSE'07 Intl. Workshop on Emerging Trends in FLOSS Research and Development*, Minneapolis, Minnesota, May 21 2007.

[21] Audris Mockus.
Amassing and indexing a large sample of version control systems: towards the census of public source code history.
In *6th IEEE Working Conference on Mining Software Repositories*, May 16–17 2009.

[22] Audris Mockus.
Succession: Measuring transfer of code and developer productivity.
In *2009 International Conference on Software Engineering*, Vancouver, CA, May 12–22 2009. ACM Press.

References IV

[23] Audris Mockus.
Engineering big data solutions.
In *ICSE'14 FOSE*, 2014.

[24] Audris Mockus, Diomidis Spinellis, Zoe Kotti, and Gabriel John Dusing.
A complete set of related git repositories identified via community detection approaches based on shared commits.
In *IEEE Working Conference on Mining Software Repositories: Data Showcase*, May 2020.

[25] R Keith Oliver, Michael D Webber, et al.
Supply-chain management: logistics catches up with strategy.
Outlook, 5(1):42–47, 1982.

[26] David Reid, Mahmoud Janshani, and Audris Mockus.
The extent of orphan vulnerabilities from code reuse in open source softwar.
In *ICSE 2022*. ACM Press, May 2022.
accepted.

[27] Peter C Rigby, Yue Cai Zhu, Samuel M Donadelli, and Audris Mockus.
Quantifying and mitigating turnover-induced knowledge loss: case studies of chrome and a project at avaya.
In *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*, pages 1006–1016. IEEE, 2016.

[28] Ernest L. Nichols Robert B. Handfield.
Introduction to supply chain management.
New York: Prentice-Hall, 1999.