Causal Modeling, Discovery & Inference for Software Engineering

Rick Kazman
SEI/CMU and
University of Hawaii
Honolulu, HI
kazman@hawaii.edu

Robert Stoddard
Software Engineering
Institute/ CMU
Pittsburgh, PA
rws@sei.cmu.edu

David Danks
Department of Philosophy
CMU
Pittsburgh, PA
ddanks@cmu.edu

Yuanfang Cai
Computer Science Dept.
Drexel University
Philadelphia, PA
yfcai@cs.drexel.edu

I. WHY CAUSATION?

Much empirical research in software engineering has focused on studies of "naturalistic" phenomena (e.g., [3]). But these studies collect only observational data, and so traditional analysis techniques yield only correlations between project practices and characteristics (on the one hand) and measurable outcomes (on the other hand). Without knowing the causal effects, it is difficult for a manager to act upon correlational evidence. For example, as source files in a software project increase in size, they tend to have more bugs and be touched by more developers. That is, file size, bugs, and number of developers are all strongly positively correlated. If one mistakes correlation for causation, then one might be tempted to conclude that bug rates could be lowered just by reducing the number of developers who are working on a file!

While this example is simple, it captures the problem of relying on correlation data. While some correlations are spurious, others clearly are not. As Tufte said: "Correlation is not causation but it sure is a hint." [11] Certainly, if one project characteristic or practice causes another, it will not only be statistically correlated with it, but also provide a means to change it. Our goal, therefore, is to understand the *causal relations* between project characteristics and practices and the outcomes that we desire. In doing so, we can confidently create concrete, actionable recommendations for software project managers: adopt this language, or tool, or practice and something that you care about—code quality, bug rate, productivity, developer satisfaction—will improve.

II. A FRAMEWORK FOR CAUSAL MODELS

Over the past thirty years, a robust framework for causal modeling—*causal graphical models*—has been developed, with algorithms for discovery of causal structure from observational, experimental, and mixed datasets. In our research we employ the standard framework of causal graphical models (CGMs); causal Bayesian networks (CBNs) and causal structural equation models (SEMs) are two common types of CGMs, but not the only ones. At a high level, a CGM has two distinct components: (1) a graph for qualitative causal relations, and (2) a joint probability distribution or density for quantitative causal strengths. The graph is over nodes for the variables—V_1, …, V_n—with $V_c \rightarrow V_e$ if V_c is a (qualitative) cause of V_e. The notion of causation here is instrumental: $V_c \rightarrow V_e$ means that external interventions on V_c will probabilistically lead to changes in V_e, but not vice versa. The intensity of that causal connection is represented in the joint distribution or density $P(V_1, \ldots, V_n)$. These two components are connected through a pair of assumptions, commonly called *Markov* and *Faithfulness*, each of which uses one component to constrain the other. These assumptions encode standard, domain-general ways in which causal relations manifest in data. For a given CGM, there are fast, computationally efficient algorithms for inference and prediction given observations or interventions (including policy changes), many implemented in standard software packages [5], [10].

III. A CASE STUDY IN CAUSAL DISCOVERY

The dataset that we examined resulted from an exploration of architectural design flaws in a set of large, primarily open source software systems. Those systems were identified in two separate studies ([4] and [2]) that explored the relationship between design flaws and multiple outcomes that were (potentially) caused by design flaws. However, those studies drew only correlational conclusions, and so cannot be used to confidently justify policy or practice changes. We aim to determine whether architectural design flaws *caused* higher rates of bugs, higher rates of changes, and higher amounts of churn (committed lines of code).

We analyzed both sets of projects, and project versions, as listed in Table 1. These two studies analyzed a total of 15 distinct projects, and 20 distinct project versions, across a wide variety of application domains, and with greatly varying ages and sizes: the commercial project had just 56,000 lines of code, while the Google Chrome browser had over 5 million lines of code. We focused on nine systems for the present analysis: Avro, Camel, Cassandra, CXF, Hadoop, HBase, Ivy, OpenJPA, and PDFBox.

To collect the raw data, the original studies extracted the source code, revision histories, and issue-tracking databases from each of these projects and built a DRSpace (Design Rule Space) representation of each project's modular structure (as described in [12] and [13]). By analyzing the resulting DRSpaces, the design flaws associated with each file in each project were calculated, as described in [4].

978-1-5386-1589-8/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSE-C.2017.138

Using this data, those two studies calculated a number of correlations: between the number of design flaws that a file was implicated in, on the one hand, and four different *extrinsic* measures of a file's goodness—number of bugs, number of changes, bug churn, and change churn—on the other hand. To do this the Pearson Correlation Coefficient (PCC) was calculated between the four pairs of data sets.

Our final analyses focused on six variables:

Age [age in months]	*Devs* [# of developers]
LOC [lines of code]	*Violations* [total # of violations]
Churn_bugs [bug churn]	*Bugs* [# of bugs]

We applied the PC algorithm [10] to the datasets for the nine systems shown in bold in Table 1. The PC algorithm is an asymptotically reliable, constraint-based causal discovery method that efficiently determines the set of structures over $V_1, \ldots, V_n$ that imply the observed pattern of (un)conditional independencies. The PC algorithm has a free parameter—the alpha level used in independence tests—that controls the relative proportions of Type I and Type II errors. For each of the nine datasets, as well as the concatenation of all nine, we applied the PC algorithm at α=0.05. The PC algorithm does not explicitly search for causal structures with unobserved common causes, in contrast with algorithms such as FCI that do have this ability. Thus, a causal edge in the output could potentially be explained in other ways. Importantly, though, *absences* of causal edges are robust against the possibility of unobserved common causes: if there is no $A \rightarrow B$ edge, then we can reliably conclude that A does not directly cause B (subject to the usual errors due to statistical noise).

As an example of the algorithm output, the causal graph for all nine concatenated datasets is shown in Figure 1. This causal graph includes a bidrected edge $Dev \leftrightarrow LOC$, indicating the likely existence of an unobserved common cause of the two (plausibly, something like project size). Although the PC algorithm does not explicitly search for such unobserved factors, it does sometimes include them when required to explain the observed data.

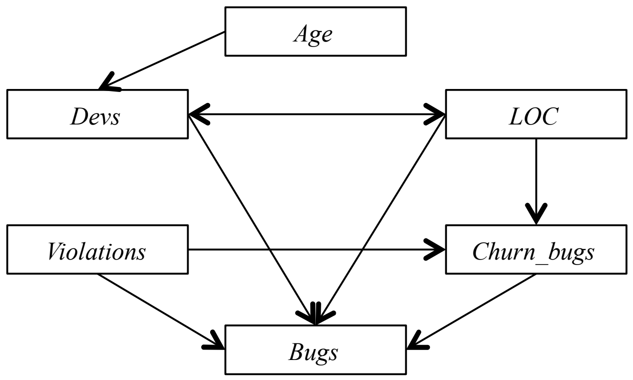

Figure 1: Output graph for all nine datasets

Some aspects of Figure 1 provide useful "sanity checks." For example, *LOC* → *Bugs* is predicted by essentially every model (and experience) of software development. It is also sensible that *Age* only influences other variables through the mediating factor of *Devs*. The age of a project should not directly cause problems, but only because older projects tend to have had more people working on them.

We *post hoc* identified groups of projects based on shared causal structure. Apache Avro was different from all the others, so we exclude it from this analysis. The projects fell into two groups: {Camel, Cassandra, Hadoop, OpenJPA, PDFBox} vs. {CXF, HBase, Ivy}. Table 1 lists the "characteristic" causal structures for each cluster (i.e., the edges that appear in most graphs for those projects).

Table 1: Characteristic edges for project-groups

Camel, Cassandra, Hadoop, OpenJPA, PDFBox	CXF, HBase, Ivy
Devs → *LOC*	*Age* → *Devs*
LOC → *Violations*	*Devs* → *Bugs*
Violations → *Churn_bugs*	*Bugs* → *Churn_bugs*
Age → *Bugs*	*LOC* → *Churn_bugs*
Devs → *Bugs*	
LOC → *Bugs*	
Violations → *Bugs*	
Churn_bugs → *Bugs*	

These two groups exhibit different causal structures, e.g. the first group has more causal connections than the second. The details also vary between the groups. For example, they both posit a causal connection between *Bugs* and *Churn_bugs*, but the directions differ. More generally, *Bugs* is the "causal sink"—the variable that is caused by many other things in the system—in the first group of projects, but *Churn_bugs* is the sink for the second. With regards to our key question, we find different answers in the two groups of projects. In the first group, *Violations* is a direct cause of both *Bugs* and *Churn_bugs*; that is, we predict that focusing on violation reduction should lower the number of bugs, and their churn. In the second group, though, *Violations* is not causally connected with any of the other variables. We thus predict that a focus on violation reduction would not have a corresponding impact on bugs in those three projects.

Why are there these differences? We do not know. We focused on a limited set of variables, and presumably there are others that could explain these results. Nonetheless, causal discovery and analysis has provided a powerful new tool in our toolbox to ask and examine such questions.

IV. Conclusions and Next Steps

This causal discovery analysis is intended as an initial step, and is certainly not the final word. For example, one could apply multiple causal discovery algorithms to measure the sensitivity of the learned structures to the use of the PC algorithm. Moreover, software projects exhibit significant dynamics over time, as code is written, refined, refactored, and so forth. We used static datasets that provide snapshots of the projects at particular moments in time. If we collect longitudinal data about similar variables, then we could start to uncover the underlying causal dynamics. One might also suspect that those dynamics could shift over time, as the software practices and philosophies change, as project members enter and leave, etc. Longitudinal data could also enable us to test for this type of causal non-stationarity. The key point that we have established here, however, is the first demonstration of the applicability and usefulness of causal discovery algorithms applied to observational software engineering datasets.

References

[1] D. M. Chickering. "Optimal Structure Identification with Greedy Search", *Journal of Machine Learning Research*, 2002, 3, 507–554.

[2] Q. Feng, R. Kazman, Y. Cai, R. Mo, L. Xiao, "An Architecture-centric Approach to Security Analysis", *Proceedings of the 13th Working IEEE/IFIP Conference on Software Architecture (WICSA 2016)*, (Venice, Italy), April 2016.

[3] R. Kazman, D. Goldenson, I. Monarch, W. Nichols, G. Valetto, "Evaluating the Effects of Architectural Documentation: A Case Study of a Large Scale Open Source Project", *IEEE Transactions on Software Engineering*, 2016, 42:3, 220-260.

[4] R. Mo, Y. Cai, R. Kazman, L. Xiao, "Hotspot Patterns: The Formal Definition and Automatic Detection of Architecture Smells", *Proceedings of the 12th Working IEEE/IFIP Conference on Software Architecture (WICSA 2015)*, (Montreal, Canada), May 2015.

[5] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco: Morgan Kaufmann. 1988.

[6] D. Perry, A. Porter, L. Votta, "Empirical studies of software engineering: a roadmap", *Proceedings of the Conference on the future of Software Engineering*, 345-355, 2000.

[7] J. D. Ramsey, P. Spirtes, C. Glymour, "On Meta-analyses of Imaging Data and the Mixture of Records", 2011, *NeuroImage*, 57, 323-330.

[8] M. Shaw, "Prospects for an Engineering Discipline of Software", *IEEE Software*, 7 (6), 15-24, 1990.

[9] S. Shimizu, P. O. Hoyer, A. Hyvärinen, A. Kerminen, "A Linear Non-Gaussian Acyclic Model for Causal Discovery", *Journal of Machine Learning Research*, 2006, 7, 2003–2030.

[10] P. Spirtes, C. Glymour, R. Scheines. *Causation, Prediction, and Search* (2nd ed.). Cambridge, MA: The MIT Press. 2000.

[11] E. Tufte, *The Cognitive Style of PowerPoint: Pitching Out Corrupts Within*, 2nd ed, Graphics Press, 2006.

[12] L. Xiao, Y. Cai, R. Kazman, "Titan: A Toolset That Connects Software Architecture with Quality Analysis", *Proceedings of the 22nd ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE 2014)*, (Hong Kong), November 2014.

[13] L. Xiao, Y. Cai, R. Kazman, "Design Rule Spaces: A New Form of Architecture Insight", Proceedings of the International Conference on Software Engineering (ICSE) 2014, (Hyderabad, India), June 2014.