Inferring Causality with Data from Personal Software Process

William Nichols

Michael Konrad

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Evidence
How do we know?

"You can see a lot by just watching."

—Yogi Berra

"Science is what we do to keep from lying to ourselves."

—Richard Feynman

Myth vs Reality
Find causation from observational data

Important because, For Software Projects > $15 M	
Average cost overrun of 66%	Average schedule overrun of 33%
"Delivering Large scale IT projects on time, on budget, and on value", McKinsey-Oxford, 2010	

Staff factors? Testing? Tools? Estimation? What works?
We need better ways, but observations can be deceiving,

Testing hypotheses is hard because
No Controls -Experiments are impractical
Imprecise data constructs - Measures are inconsistent
Incomplete data - Combined data
Every project is different - Explosion of contextual factors
Unknown distributions – do statistical methods apply
Mixed causal systems

Can Causal Algorithms Help?

To control software development, we need factors that

1) Can be selected or manipulated
2) Have a causal effect (direct or indirect) on desired outcomes

New algorithms and techniques are becoming available

They are related to but distinct from multiple regression, Bayesian networks, and Machine Learning.

The methods have been successful in other domains.

How can we gain confidence when applying to Software Engineering?

Considerations for use of Causal Discovery

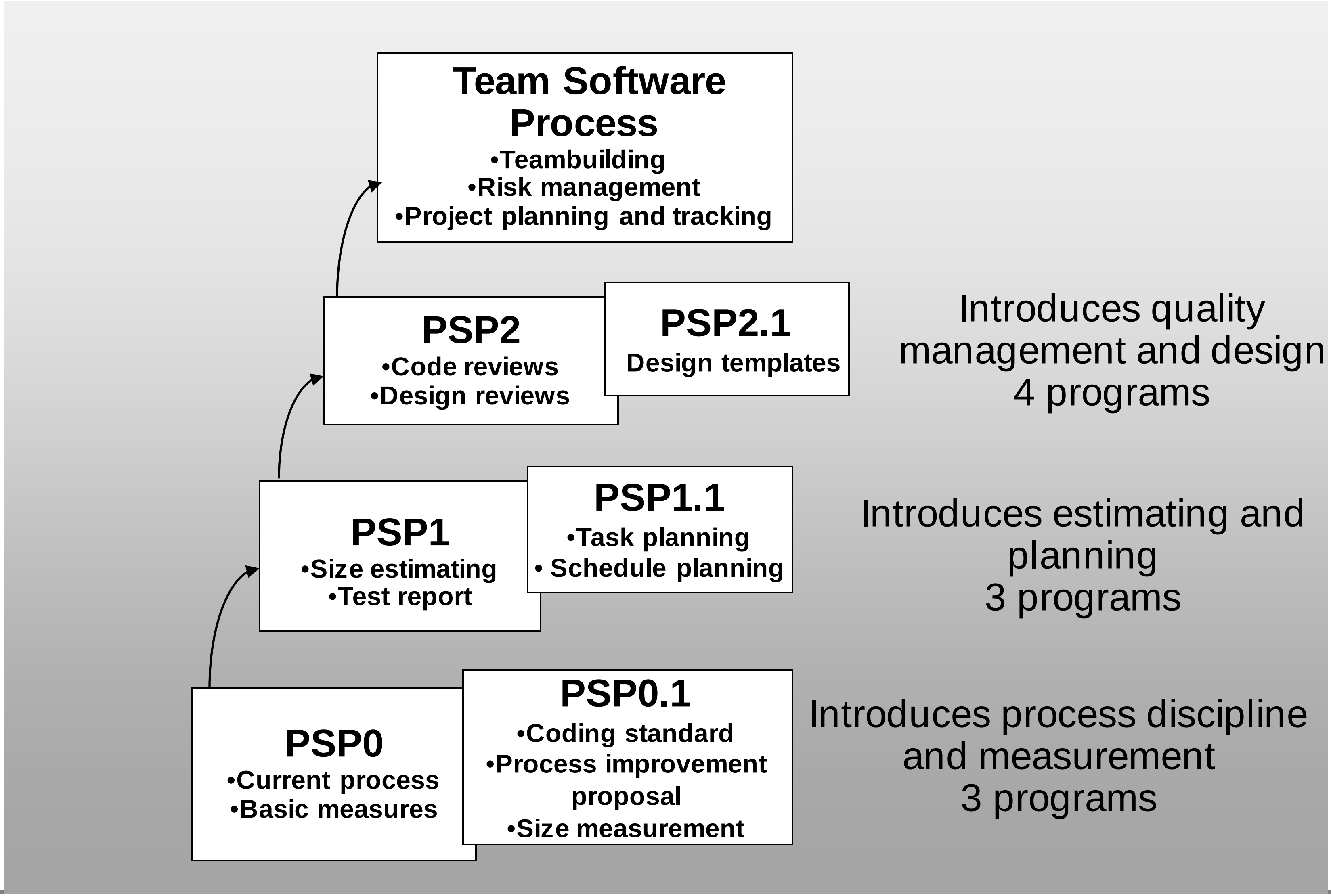
Data

- Precisely defined
- Continuous or convertible to continuous (for many algorithms)
- Large sample sizes
- Homogeneous context
- Inputs span a range of values
- Well understood

Additionally, algorithms may use assumptions about

- Gaussian or skewed distributions
- Linear relationships

The PSP Course, 10 Exercises



PSP Data

The PSP course has been taught for more than 20 years.

For the same ten-exercises in the course

- 3140 developers and 31,140 programs
- 3,355,882 lines of code
- 123,996.53 hours of work
- 221,346 defects

Each programmer developed the same 10 programs.

A great deal can be learned from analyzing these data.

Course results have been studied and analyzed using traditional methods

Of these, 494 sets of 10 programs are written in “C”.

We will only look at these data.

PSP Data has many desirable properties

When using the PSP, developers gather and use data. The data is generally of high quality.

Time data

- The time in minutes spent in each main development activity
- Stop-watch time (Interruption time is not included)

Size data

- Product size LOC, (can also use in db elements, pages, etc.)
- Categories: base, added, deleted, modified, reused

Defect data

- All defects removed in compile, test, review, etc.
- Type, phases injected & removed, fix time, description

We will use only continuous values.

We can also apply time based “prior knowledge” based on the process.

Data and Factors

scope		#	Variable	Description
Requirement (i)	I	10	AsgAveMin	How challenging are the requirements?
Student (j)	I	494	StuDAR	What is historical developer defect rate?
Student (j)	I	494	StuSize	What is historical developer Verbosity?
Student (j)	I	494	StuEffFactor	What is historical developer rate?
Assignment (i,j)	I	4940	ConstMin	Actual assignment design and code effort
Assignment (i,j)	O	4940	LOC	Actual size of program
Assignment (i,j)	O	4940	DefectTot	Number of program defects
Assignment (i,j)	O	4940	MinTot	Total effort expended

Expected Relationships

Size: $LOC_{ij} = ReqSize_i \times SSF_j$

Defects: $DefTot_{ij} = ConstMin_{ij} \times StuDAR_j$

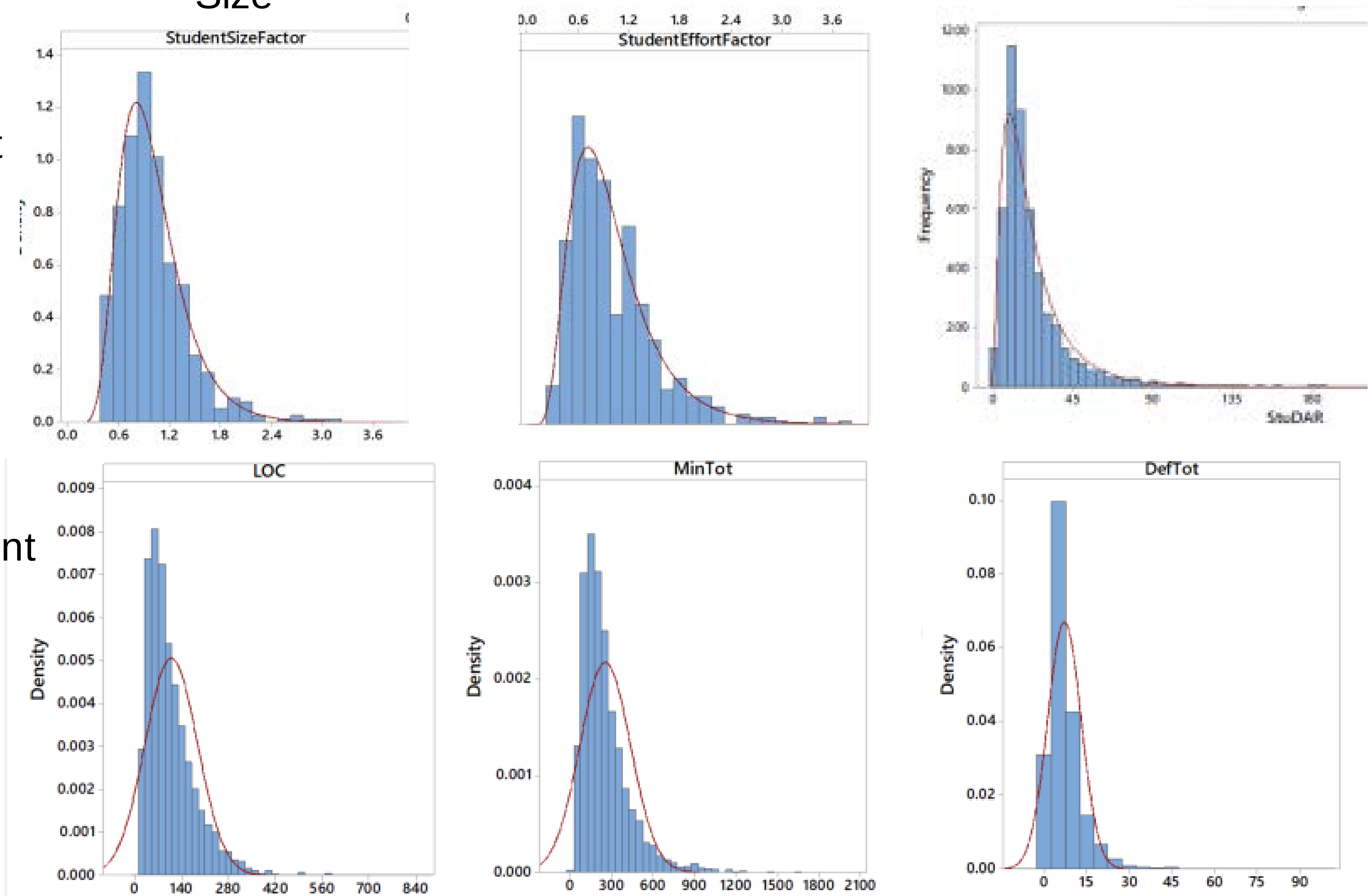
Effort: $MinTot_{ij} = ReqSize_i \times SEF_j$

Data is roughly log-normal, we apply log transforms

Size | Effort | Defects

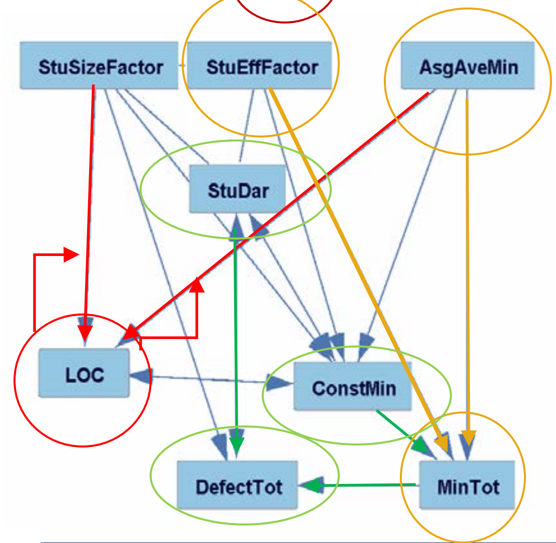
Student (j)

Assignment (i,j)

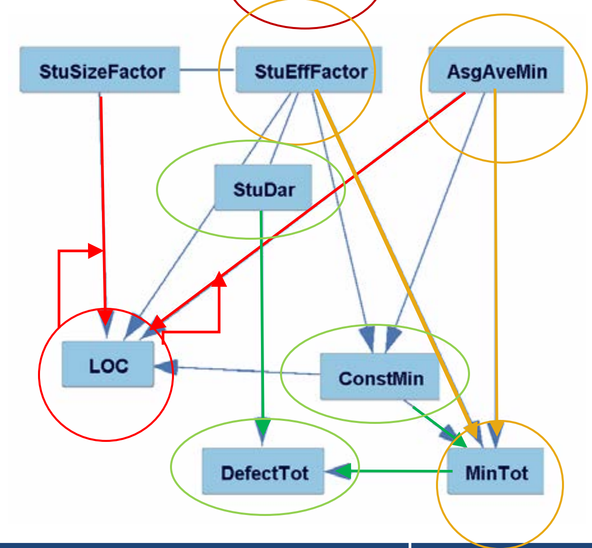


Results, Application of Causal Analysis

DAG from PC search



DAG from FGES search



Common Direct Causal Edges

1. StuSizeFactor → LOC
2. AsgAveMin → LOC
3. AsgAveMin → MinTot
4. StuEffFactor → MinTot
5. AsgAveMin → ConstMin
6. StuEffFactor → ConstMin
7. ConstMin → MinTot
8. MinTot → DefectTot

Expected Relation (log transformed for linear effects)	Edges found	PC	FGES
$\ln(LOC_{ij}) = C_0 \ln(AsgAveMin_i) + C_1 \ln(SSF_j)$	StuSizeFactor → LOC	Y	Y
	AsgAveMin → LOC	Y	Y
$\ln(MinTot_{ij}) = C_2 \ln(AsgAveMin_i) + C_3 \ln(SEF_j)$	AsgAveMin → MinTot	Y	Y
	StuEffFactor → MinTot	Y	Y
$\ln(DefectTot_{ij}) = C_4 \ln(ConstMin_{ij}) + C_5 \ln(StuDAR_j)$	StuDAR -> DefectTot ConstMin -> DefectTot	Bi I	Y I

Discussion

SWE data can have many of the necessary characteristics to apply these techniques

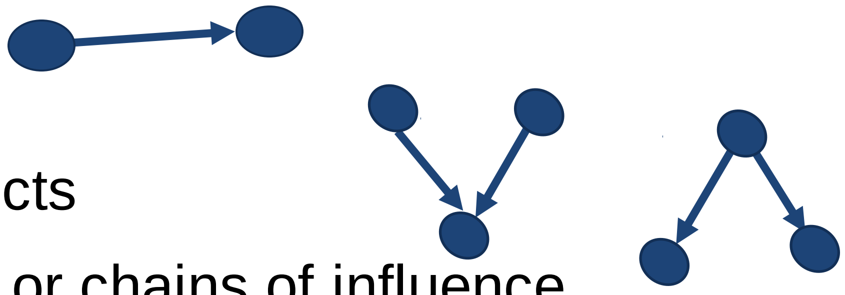
- Distinct algorithms found expected relationships
- Algorithms did not contradict each other

Evidence for "the rest of the story"

- Total effort is driven by product requirements
  - Personnel historical rates have a large influence
- Implemented size is driven by requirements
  - Personnel factors affect the actual implemented size
- Total defects are driven by requirements
  - Personnel and process factors affect defect levels

Conclusions, Causal algorithms have promise

From observational data



- Separate cause from effect
- Identify common causes or effects
- Recognize intermediate effects or chains of influence

Next Steps

- Refine the selection of causal factors (process, and so forth)
- Examine effect sizes and variation
- Build predictive models using the causal relationships
- Examine factor sensitivities
- Extend to additional data sets

If you would like to share data or collaborate

Contact Information

William R. Nichols

wrn@sei.cmu.edu

412-268-1727

Michael Konrad

mdk@sei.cmu.edu

412-268-5813

Selected PSP data analysis references

Vallespir, Diego. **Analysis of Design Defect Injection and Removal in PSP**. In *TSP Symposium 2011* (2011) (pp. 1–28). Pittsburgh: Carnegie Mellon Univeristy.

Vallespir, Diego, & Nichols, William R. **Quality Is Free , Personal Reviews Improve Software Quality at No Cost**. *Software Quality Professional* (2016), *18*(March), 4–13.

Vallespir, Diego, & Nichols, William. **An Analysis of Code Defect Injection and Removal in PSP**. In *Proceedings of the TSP Symposium 2012* (2012). Pittsburgh.

Valverde, Carolina, Grazioli, Fernanda, & Vallespir, Diego. **A Study of the Quality of Data Collected during the Using the Personal Software Process** (n.d.), 37–44.

Grazioli, Fernanda. ***An Analysis of Student Performance During the Introduciton of the PSP: An Empirical Cross Course Comparrison*** (2013). Universidad de la Republica.

Paulk, MC. ***An empirical study of process discipline and software quality*** (2005). Retrieved from http://d-scholarship.pitt.edu/8303/

Rombach, Dieter, Münch, Jürgen, Ocampo, Alexis, Humphrey, Watts S., & Burton, Dan. **Teaching disciplined software development**. *Journal of Systems and Software* (2008), *81*(5), 747–763. https://doi.org/10.1016/j.jss.2007.06.004