# Sampling & Metrics

# Survey & MSR

- Many software repositories are avalaible (Source code, bugs, tests, requirements)
- Why not using them to perform a survey?
  - Goal, Null hypothesis, operation, analysis

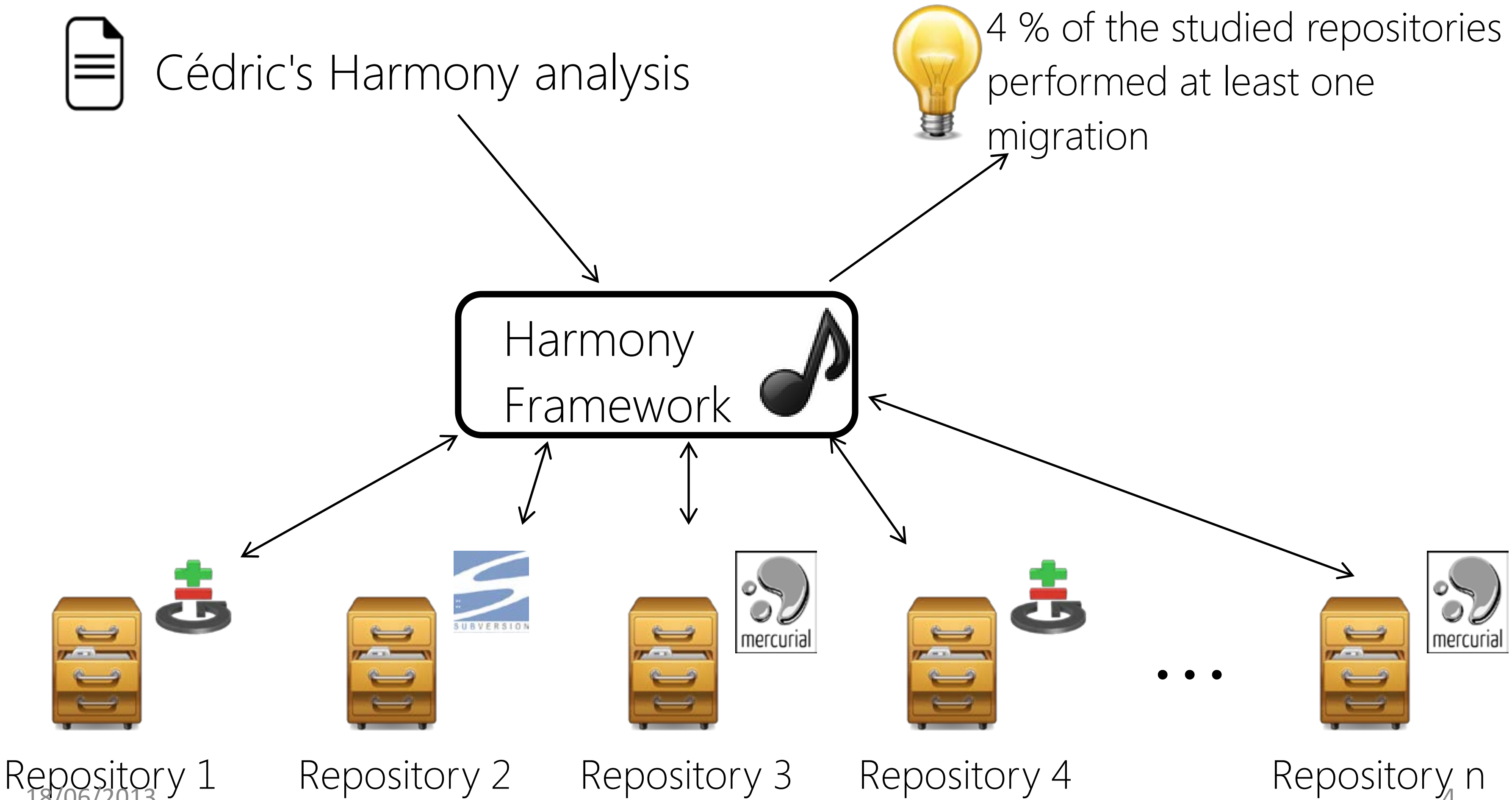
# Example of Survey

Cédric

Are library migrations frequent ?


Repository 1 Repository 2 Repository 3 Repository 4 … Repository n

# Harmony: Browse Model



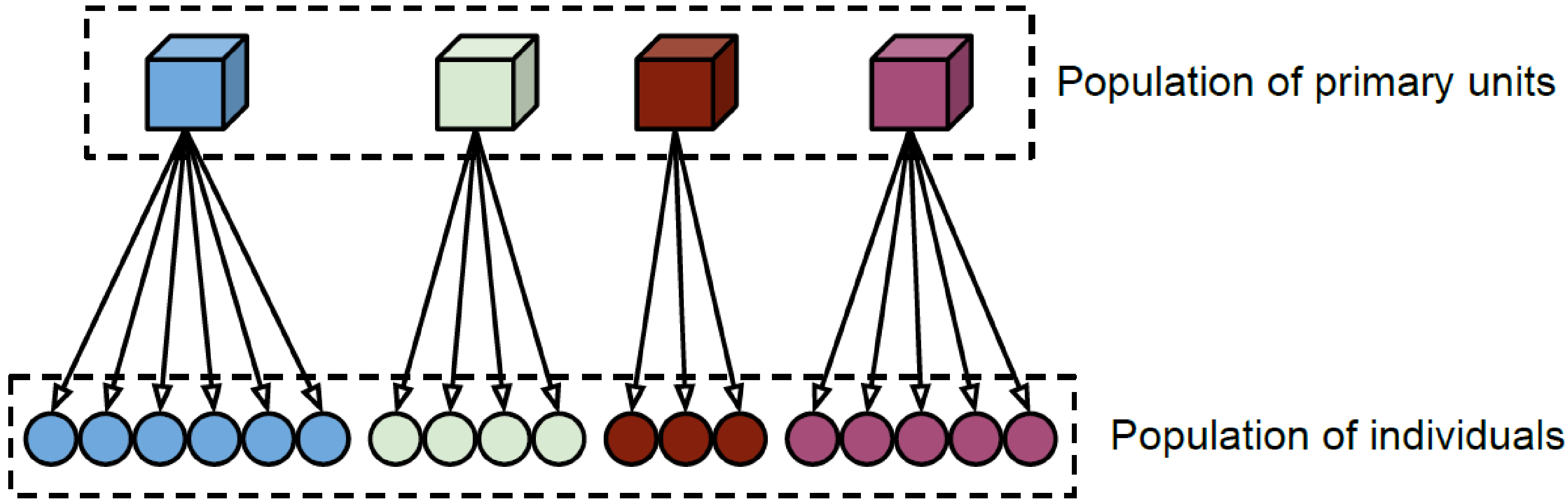
# Issues

- Sampling
  - The more repositories the best the result of the survey?
  - Which repositories (big, old, small, young, active, with large community, …)?
  - How many repositories? How much of their resources?
- Metrics
  - Re-using existing measures?
  - How to define new measures?

# Principles of statistics

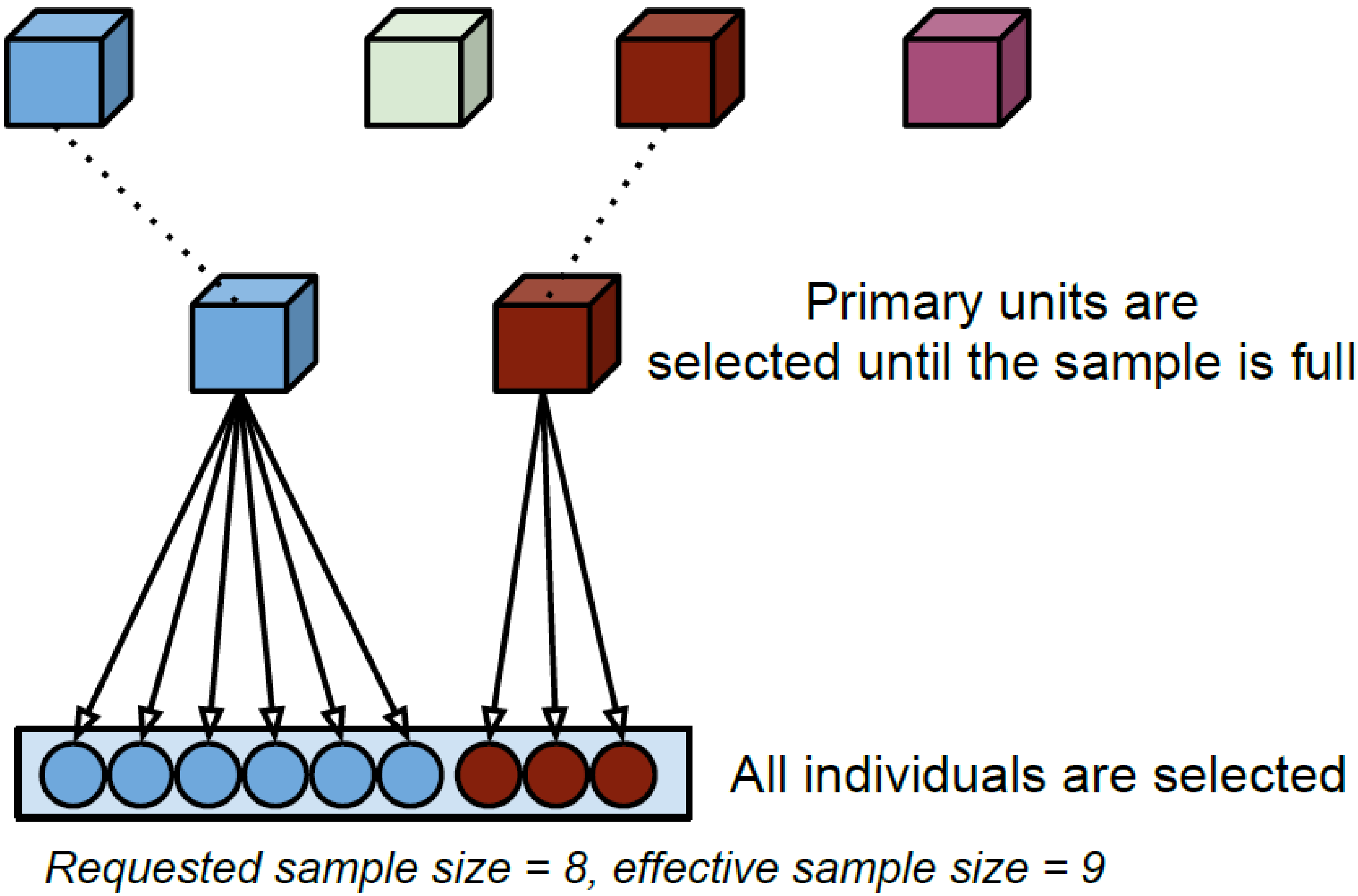
- Individual (Unit of interest)
  - Object to measure (class, project, developer, …)
- The distributivity of the measures has an impact on the number of individuals to measure
  - Normal law ≈ 30 individuals
  - Flip a coin. With 10%. With 10 tests.
    - P(1 A and 9 B) = P(1/10) = 0.0097
    - P(2/10 or less) = 0.097
    - P(3/10 or less) = 0.449

# Sampling - Principes

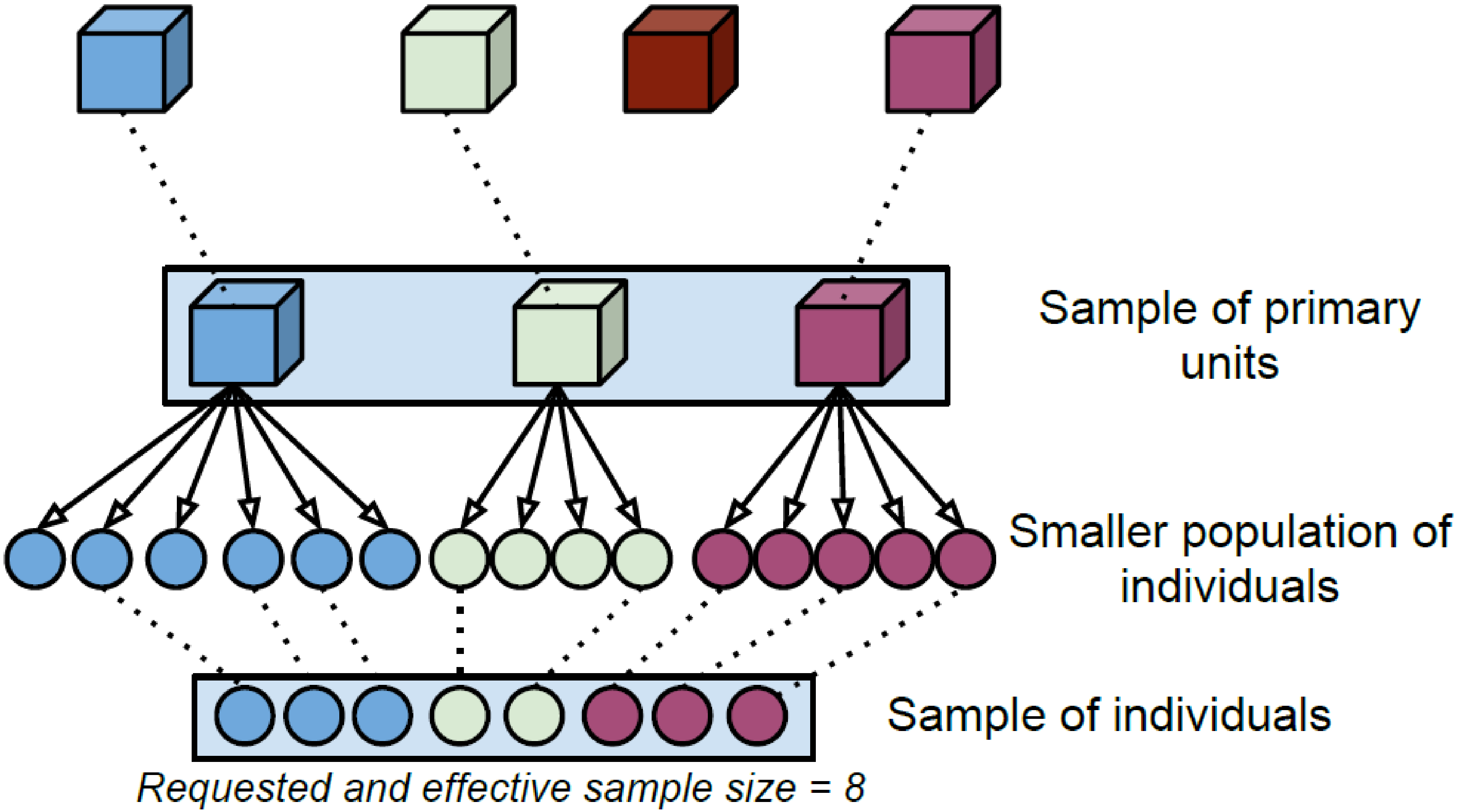



**Primary unit selection has a major impact**

# Cluster Sampling



# Double Sampling



# Example on NM+NA

- NM + NA = Number of methods + Number of attributes
- 80/100 quantile
- 3 large projects
  - NM+NA = 27 (but no trust !)
- Double Sampling on GitHub (400 projects) + Bootstrap on 1000 classes
  - Nm+NA = [23-27] with 95% of confidence

# Metrics

- Measure the quality of software development
- Analyse the impact on maintenance
- Identify anti-patterns
- Examples
  - Source Code: LOC, CBO, DIT, DIT, …
  - Workload: NbCommits, Touches, CHURN
  - Bugs: NbOfOpenBug, TimeToFix
  - …

# Metrics, effects and aggregation

- A metric should represent something
- Correlation to measure the « effect » of the metrics
- Aggregation of metrics

# Detecting Fault Prone Component

- Use metrics to identify fault prone components
- Focus maintenance on these components

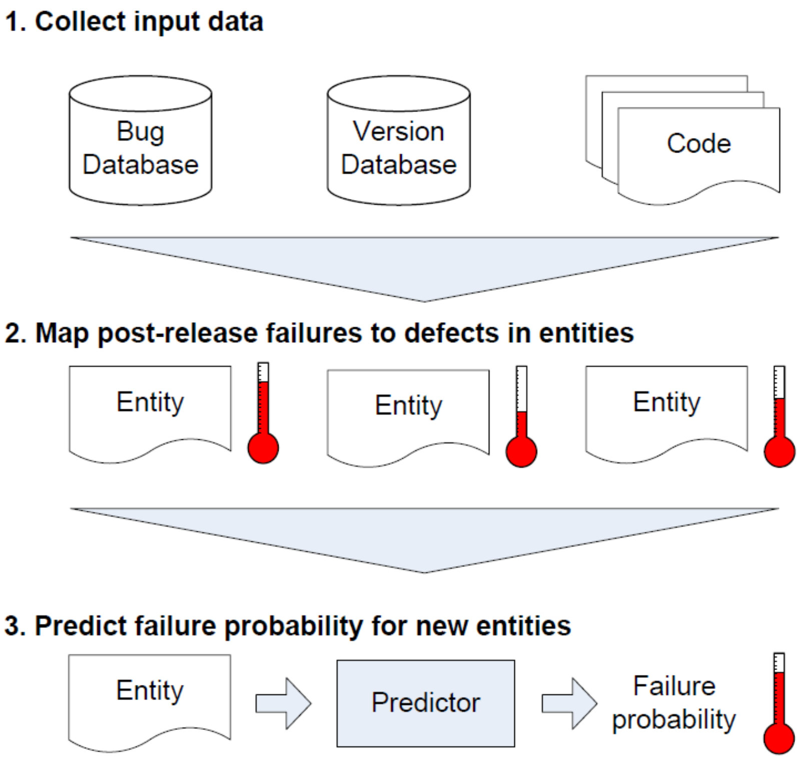



Figure 1. After mapping historical failures to entities, we can use their complexity metrics to predict failures of new entities.

# Correlation

| Metric | Description | | Correlation with post-release defects of *M* | | | | |
|---|---|---|---|---|---|---|---|
| | | | A | B | C | D | E |
| **Module metrics** — correlation with metric in a module *M* | | | | | | | |
| *Classes* | # Classes in *M* | | **0.531** | **0.612** | **0.713** | 0.066 | 0.438 |
| *Function* | # Functions in *M* | | 0.131 | **0.699** | **0.761** | 0.104 | **0.531** |
| *GlobalVariables* | # global variables in *M* | | 0.023 | **0.664** | **0.695** | 0.108 | 0.460 |
| **Per-function metrics** — correlation with maximum and sum of metric across all functions *f*() in a module *M* | | | | | | | |
| *Lines* | # executable lines in *f*() | Max | -0.236 | **0.514** | **0.585** | **0.496** | **0.509** |
| | | Total | 0.131 | **0.709** | **0.797** | 0.187 | **0.506** |
| *Parameters* | # parameters in *f*() | Max | -0.344 | 0.372 | **0.547** | 0.015 | 0.346 |
| | | Total | 0.116 | **0.689** | **0.790** | 0.152 | 0.478 |
| *Arcs* | # arcs in *f*()'s control flow graph | Max | -0.209 | 0.376 | **0.587** | 0.527 | 0.444 |
| | | Total | 0.127 | **0.679** | **0.803** | 0.158 | **0.484** |
| *Blocks* | # basic blocks in *f*()'s control flow graph | Max | -0.245 | 0.347 | **0.585** | 0.546 | 0.462 |
| | | Total | 0.128 | **0.707** | **0.787** | 0.158 | 0.472 |
| *ReadCoupling* | # global variables read in *f*() | Max | -0.005 | **0.582** | **0.633** | 0.362 | 0.229 |
| | | Total | -0.172 | **0.676** | **0.756** | 0.277 | 0.445 |
| *WriteCoupling* | # global variables written in *f*() | Max | 0.043 | **0.618** | **0.392** | 0.011 | 0.450 |
| | | Total | -0.128 | **0.629** | **0.629** | 0.230 | 0.406 |
| *AddrTakenCoupling* | # global variables whose address is taken in *f*() | Max | 0.237 | **0.491** | **0.412** | 0.016 | 0.263 |
| | | Total | 0.182 | **0.593** | **0.667** | 0.175 | 0.145 |
| *ProcCoupling* | # functions that access a global variable written in *f*() | Max | -0.063 | **0.614** | **0.496** | 0.024 | 0.357 |
| | | Total | 0.043 | **0.562** | **0.579** | 0.000 | 0.443 |
| *FanIn* | # functions calling *f*() | Max | 0.034 | **0.578** | **0.846** | 0.037 | **0.530** |
| | | Total | 0.066 | **0.676** | **0.814** | 0.074 | **0.537** |

Depends on both the metrics and software

# Combine Metrics

**Table 5. Regression models and their explanative power**

| Project | Number of principal components | % cumulative variance explained | $R^2$ | Adjusted $R^2$ | F - test |
|---|---|---|---|---|---|
| A | 9 | 95.33 | **0.741** | 0.612 | 5.731, $p < 0.001$ |
| B | 6 | 96.13 | **0.779** | 0.684 | 8.215, $p < 0.001$ |
| C | 7 | 95.34 | **0.579** | 0.416 | 3.541, $p < 0.005$ |
| D | 7 | 96.44 | **0.684** | 0.440 | 2.794, $p < 0.077$ |
| E | 5 | 96.33 | **0.919** | 0.882 | 24.823, $p < 0.0005$ |

# But …

**Table 6. Predictive power of the regression models in random split experiments**

| Project | Correlation type | Random split 1 | Random split 2 | Random split 3 | Random split 4 | Random split 5 |
|---|---|---|---|---|---|---|
| A | Pearson | **0.480** | **0.327** | **0.725** | -0.381 | **0.637** |
| | Spearman | **0.238** | **0.185** | **0.693** | -0.602 | **0.422** |
| B | Pearson | -0.173 | **0.410** | **0.181** | **0.939** | **0.227** |
| | Spearman | -0.055 | **0.054** | **0.318** | **0.906** | **0.218** |
| C | Pearson | **0.559** | -0.539 | -0.190 | **0.495** | -0.060 |
| | Spearman | **0.445** | -0.165 | **0.050** | **0.190** | **0.082** |
| D | Pearson | **0.572** | **0.845** | **0.522** | **0.266** | **0.419** |
| | Spearman | **0.617** | **0.828** | **0.494** | **0.494** | **0.494** |
| E | Pearson | -0.711 | **0.976** | -0.818 | **0.418** | **0.007** |
| | Spearman | -0.759 | **0.577** | -0.883 | **0.120** | **0.152** |

*Predictors are accurate only when obtained from the same or similar projects.*

# Properties [CK94]

- Noncoaseness
  - For each P there exists Q such that $m(P) \neq m(Q)$
- Nonuniqueness
  - There can exist distinct classes P and Q such that $m(P) = m(Q)$
- Design Details are important
  - Given P and Q, which are similar, this does not imply that $m(P)=m(Q)$
- Monotonicity
  - Given P and Q, $m(P) \leq m(P+Q)$, $m(Q) \leq m(P+Q)$
- Noequivalence of Interaction
  - Given P, Q and R, $m(P) = m(Q)$, does not imply that $m(P+R) = m(Q+R)$
- Interaction increases complexisty
  - $m(P)+m(Q) < m(P+Q)$

# Conclusion

- Use existing data (OSS Repository) to perform survey
- Statistics and sampling
- Metrics and correlation