Lecture 6 & 7

Empirical Studies of Software Evolution: Code Decay

Announcement

- Your proposals have been graded.
- Literature Survey & Tool Evaluation: Each in the range of 1-4
- Project Proposal: Each in the range of 1-8

Today's Agenda

- Divya's presentation on the code decay paper
- Discuss Belady & Lehman's paper
- Discuss Code Decay paper
- Q and A session on my feedback to your proposals

Today's Presenter

- Divya Gopinath (Skeptic)

A Model of Large Software Development

- L.A Belady and M.M Lehman
- 1976
- IBM OS/360
- Seminal empirical study paper in software evolution

What was it like in 1976?

- IBM PC came out around 1984
- Apple introduced PC in 1970s
-

What was it like in 1976?

- E.W. Dijkstra: a program must as a mathematical theorem should and can be provable
- Increasing cost of building and maintaining software was alarming

Subject Program & Data

- OS/360
- 20 years old
- 20 user-oriented releases
- Starting with the available data, they attempted to ***deduce*** the nature of consecutive releases of OS/360

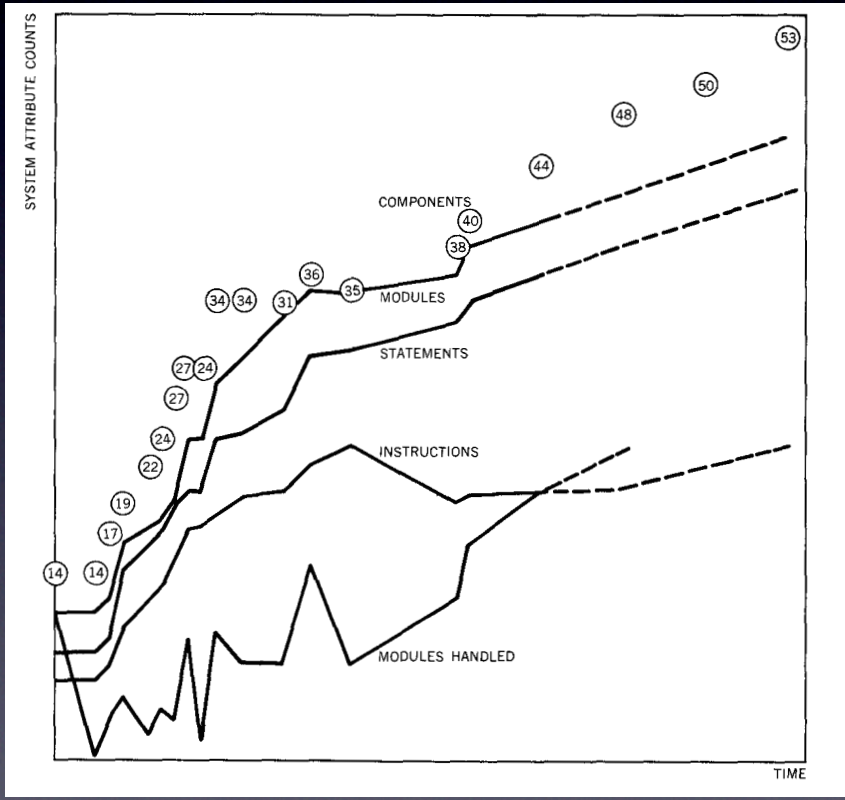
When observing software evolution, what can you measure?

- # of bugs (reported problems)
- # of modules => # of directories, # of files
- performance metrics => times, memory usage, CPU usage, IPC
- change types: corrective, adaptive, perfective
- # of developers working in the organization, # of deveopers per module
- size of code changes => # of lines of changed code
- how wide spread the changes are => # of files or modules touched by the same change
- age in calendar year, days, months / age in logical unit such as release, check-in, version

Variables observed in Belady and Lehman 1976

- The release number
- Days between releases
- The size of the system
- The number of modules added, deleted, and changed
  - Complexity: the fraction of the released system modules that were handled during the course of release MH_r / M_r
- Manpower, machine time, and costs involved in each release

Growth Trends of System Attribute Counts With Time



Growth Trends

Figure 2 Average growth trends of system attributes

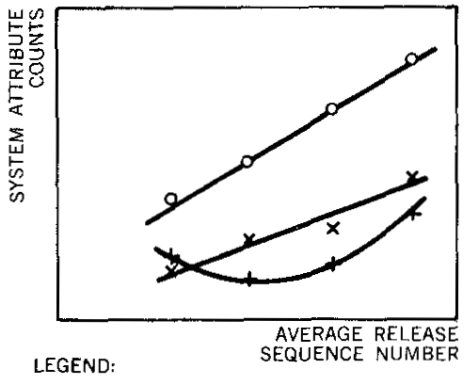


Figure 3 Average growth trends of system attributes compared with planned growth

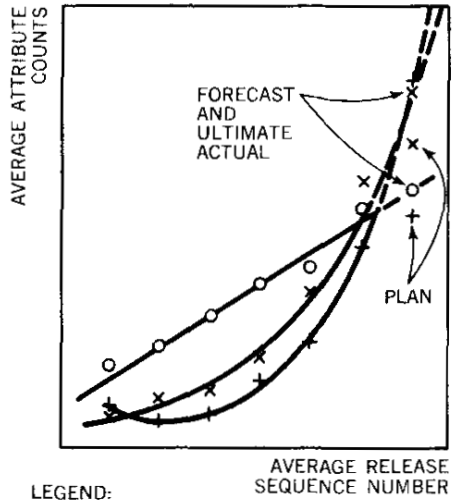
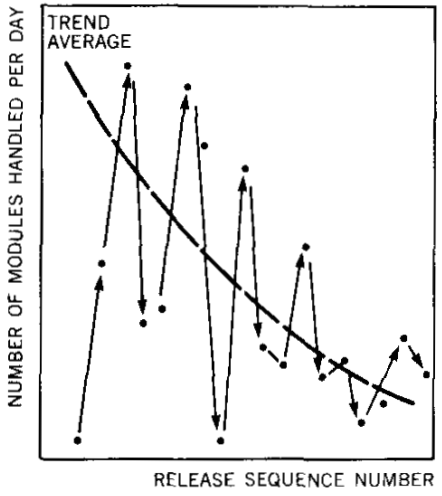


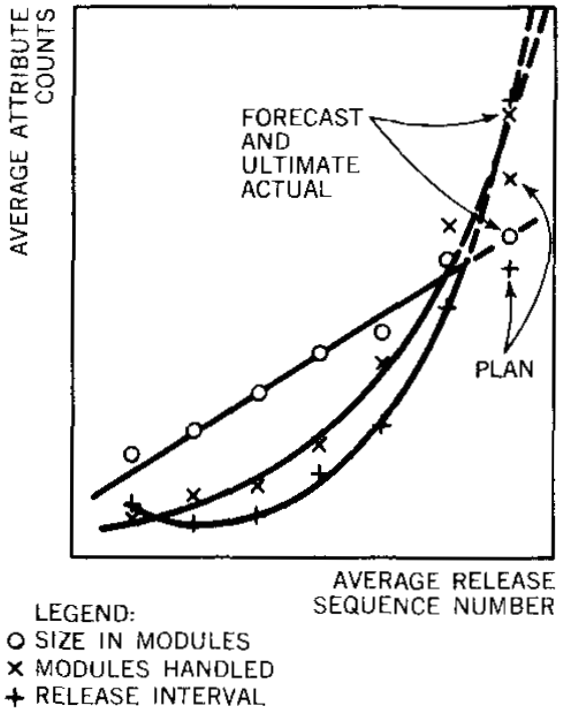
Figure 4 Serial and average growth trends of a particular attribute



What can you deduce from these graphs?

- It takes longer and longer to release the next release
- The size of increases over time
- It requires modifying more and more modules per each release

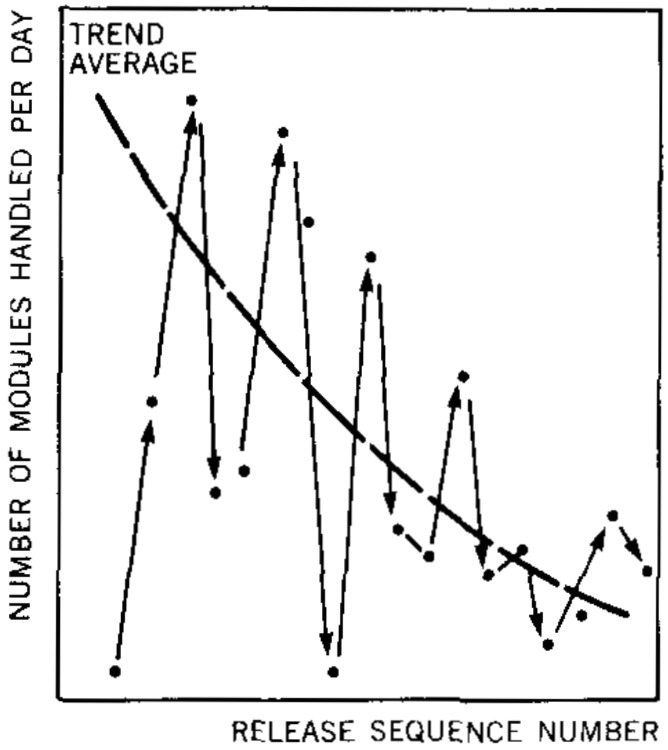
Figure 3 Average growth trends of system attributes compared with planned growth



What can you deduce from these graphs?

- Handling fewer number of modules as the software evolves
-

Figure 4 Serial and average growth trends of a particular attribute



Belady & Lehman: the Law of Program Evolution Dynamics

1. Law of continuing change: a system that is used undergoes continuing change until it is judged more cost effective to freeze and recreate it
2. Law of increasing entropy: the entropy of a system (its unstructuredness) increases with time, unless specific work is executed to maintain or reduce it.

Belady & Lehman: the Law of Program Evolution Dynamics

3. Law of statistically smooth growth: Growth trend measures of global system attributes may appear to be stochastic locally in time and space, but statistically, they are cyclically self-regulating, with well-defined long-range trends

Belady & Lehman: the Law of Program Evolution Dynamics

- Law of continuing change: a system that is used undergoes continuing change until it is judged more cost effective to freeze and recreate it
- $\Delta M_r = 200 + S1 + Z1$
- Law of increasing entropy: the entropy of a system (its unstructuredness) increases with time, unless specific work is executed to maintain or reduce it.
- $C_r = 0.14 + 0.0012R^2 + S2 + Z2$
- Law of statistically smooth growth: Growth trend measures of global system attributes may appear to be stochastic locally in time and space, but statistically, they are cyclically self-regulating, with well-defined long-range trends
- $M_r = 760 + 200 R + S + Z$ (where S and Z represents cyclic and stochastic components)

As a skeptic:

- Laws are presumptive
- How do you use for real daily software development?
- External validity: does the law hold for other projects?
- Factors:

As a skeptic:

- What is the unit of a module and a component?
- What is the granularity of a release? Do they have the same amount of functionality addition per each release?
- What types of changes does each release include?
- Any changes in the organization structures & developers?
- Are they laws or just hypotheses?
- ***What are potential contributions / benefits of understanding software evolution?***

My general thoughts on Belady & Lehman

- Very insightful paper at the time of 1976
- The first use of statistical regression for characterizing software evolution
- Discussed the nature of software evolution, characterized it using their empirical data
- Deduction of laws from one system's evolution --- very weak external validity, perhaps hasty conclusions

Does Code Decay?

- Eick et al.
- TSE 2001 (almost 25 years after Belady & Lehman's Study)

Problem Definition

- What do the authors mean by "code decay?"
  - ***it is harder to change than it should be***
  - related to Belady & Lehman's second law: the entropy of a system increases with time, unless specific work is executed to maintain or reduce it.

Discussed Problem

- Check whether ***code decay*** is real: "Does Code Really Decay?"
  - how code decay can be characterized
  - the extent to which each risk factor matters
- ****Empirical Study* Paper***

Hypotheses

- What the authors are ***trying or expecting to find***?
  - The span of files increases over time (age)
  - Effort has some relations to many measurable variables.
  - Modularity breaks over time
  - Fault potential has some relation to many measurable variables.

Hypotheses

- What the authors are ***trying or expecting to find***?
  1. The span of changes increases over time
  2. Breakdown of modularity increases over time
  3. Fault potential, the likelihood of changes to induce faults has some relations to ...
  4. Efforts has some relationship too
- Usually *good* empirical study paper either finds surprising empirical evidence that contradicts conventional wisdom or provides thorough empirical evidence that validates well known hypotheses.

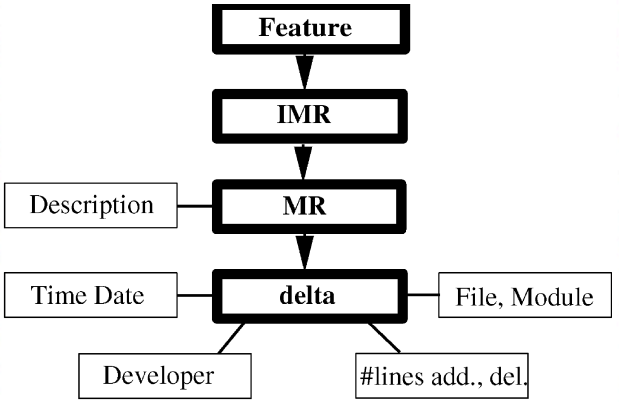
Study Approach

- Data selection
- Selection of measurement variables (so called independent variables)
- Study method that finds *relationships* among the measurement variables

Study Approach:
(1) Data Selection

- Rich data set
  - Telephone switching system
  - 100 million LOC
  - 5000 modules
  - 50 major subsystems
  - in C and C+

This system evolved by following
a well-defined process.
Structured, manually labeled data
Easy to group related changes



Study Approach: (2) Measure Independent Variables

- c denotes changes (mostly a MR)
- Variables
  - DELTAS(c) = # of deltas associated with c
  - ADD(c) = # of lines added by c
  - DEL(c) = # of lines deleted by c
  - DATE(c) = the date on which c is completed
  - INT(c) = the interval of c (calendar time required to implement c)
  - DEV(c) = number of developers implementing c

Study Approach: (2) Measure Independent Variables

- Derived variables
  - FREQ(m, I) = $\sum_{c \rightsquigarrow m} 1\{\mathrm{DATE}(c) \in I\}$, / I
  - FILES(c) = # of files touched for change c
  - NCSL (m) = # of non-commentary source lines per module
  - AGE(m) = average age of its consequent lines

Study Approach:
(3) Finding Correlation

- Linking risk factors to symptoms
- Statistical regression
  - This requires designing some template models

Study Approach: (3) Finding Correlation

- Fault Potential I. The number of faults that will have to be fixed in module m in the future. Change effects are dampened over time

$$\mathrm{FP_{WTD}(m,t)} = \gamma_1 \sum_{c \rightsquigarrow m,\ \mathrm{DATE(c)<t}} e^{-\alpha[t-\mathrm{DATE(c)}]} \times \log[\mathrm{ADD(c,m)} + \mathrm{DEL(c,m)}]$$

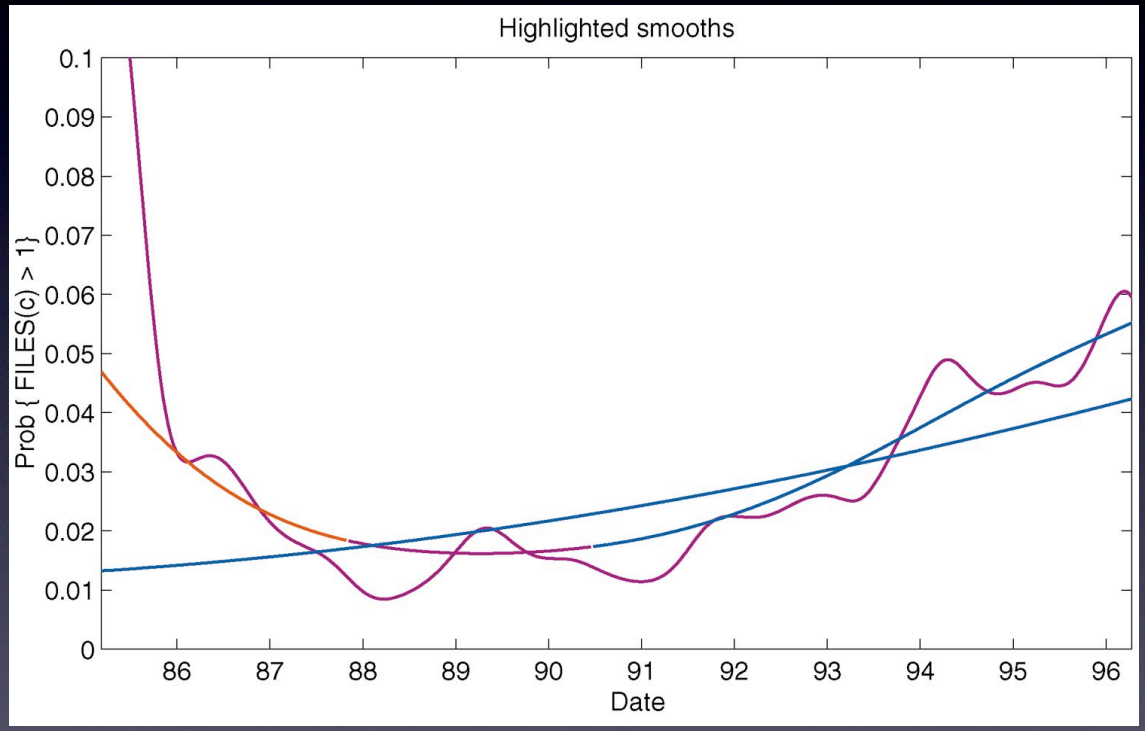
- Fault Potential II. The number of faults that will have to be fixed in a module in the future. Faults are less likely in older code (when beta is <1)

$$\mathrm{FP_{GLM}(m,t)} = \gamma_2 \times \sum_{c \in \Delta} \mathbf{1}\{c \rightsquigarrow m\} \times \beta^{\mathrm{AGE(m)}},$$

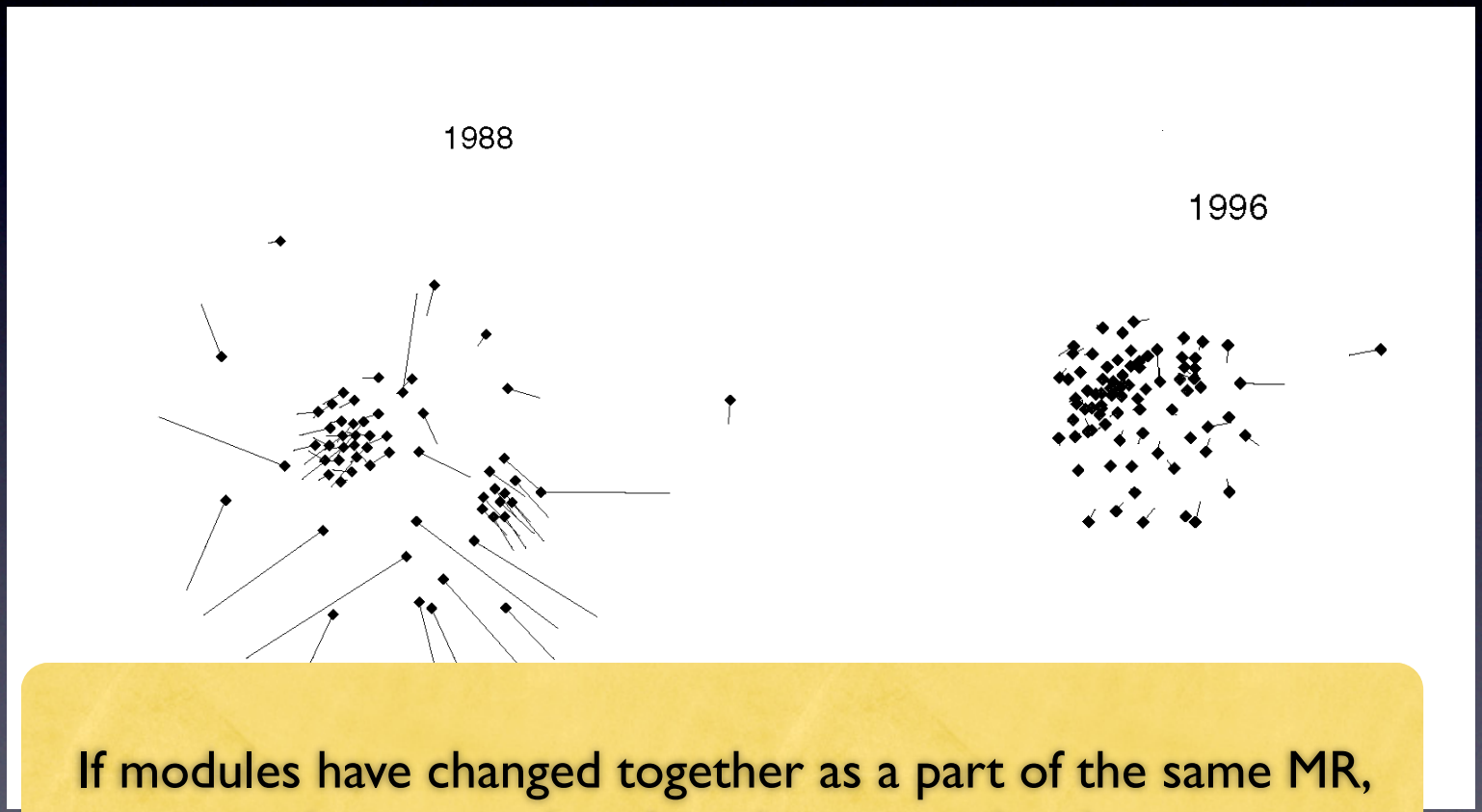
- Effort Model: predictors of the person-hours

$$\mathrm{EFF(c)} = a_0 + a_1\mathrm{FILES(c)} + a_2 \sum_f 1\{c \rightsquigarrow f\}|f| + a_3\mathrm{ADD(c)} + a_4\mathrm{DEL(c)} + a_5\mathrm{INT(c)} + a_6\mathrm{DEV(c)}.$$

Results: (1) The span of changes increases over time?



Results: (2) Breakdown of modularity increases over time?



If modules have changed together as a part of the same MR, they were placed to close to each other.

Results: (3) Fault potential, the likelihood of changes to induce faults increases over time

$$\mathrm{FP_{WTD}(m,t)} = \gamma_1 \sum_{c \rightsquigarrow m,\ \mathrm{DATE(c)<t}} e^{-\alpha[t-\mathrm{DATE(c)}]} \times \log[\mathrm{ADD(c,m)} + \mathrm{DEL(c,m)}]$$

$$\mathrm{FP_{WTD}(m)} \propto \sum_{c \rightsquigarrow m} e^{0.75\times \mathrm{DATE(c)}} \times \log[\mathrm{ADD(c,m)} + \mathrm{DEL(c,m)}],$$

$$\mathrm{FP_{GLM}(m,t)} = \gamma_2 \times \sum_{c \in \Delta} \mathbf{1}\{c \rightsquigarrow m\} \times \beta^{\mathrm{AGE(m)}},$$

$$\mathrm{FP_{GLM}(m)} = .017 \times \sum_{c} \mathbf{1}\{c \rightsquigarrow m\} \times .64^{\mathrm{AGE(m)}}.$$

Large, recent changes add the most to fault potential.

Code having many lines that have survived for a long time is likely to be relatively free of faults.

Results: (4) Prediction of efforts increases over time

$$\mathrm{EFF(c)} = a_0 + a_1\mathrm{FILES(c)} + a_2 \sum_f 1\{c \rightsquigarrow f\}|f| + a_3\mathrm{ADD(c)} + a_4\mathrm{DEL(c)} + a_5\mathrm{INT(c)} + a_6\mathrm{DEV(c)}.$$

$$\log(1+\mathrm{EFF(c)}) = .32 + .13\ (\log[1+\mathrm{FILES(c)}])^2 - .09(\log[1+\mathrm{DEL(c)}])^2 + .12\log[1+\mathrm{ADD(c)}]\log[1+\mathrm{DEL(c)}] + .11\log[1+\mathrm{INT(c)}] - .47\log[1+\mathrm{DELTAS(c)}].$$

File span has positive correlation.
Large deletions are implemented rather easily.
Hardest changes require both additions and deletions.
Large number of editing changes are rather easy to implement.

Expected results?

- ...

Any unexpected results?

- The number of developers does not have much impact
- Complexity metrics did not play much roles compared to size and number of changes
-

Expected results?

- File span increases over time.
- Size and time of changes have some positive correlation with fault potential
- Modularity degrades over time.

Any unexpected results?

- Once size and time of changes are taken into account, other variables (e.g. # developers, complexity metrics, span of files) did not play much roles in predicting faults.
- Large number of editing changes are rather easy to implement.
- In the beginning of evolution, file span was relatively large.

Threats to Validity? Limitations?

-

Four types of threats to validity

- External Validity: Can we ***generalize*** to other situations?
- Internal Validity: Assuming that there is a relationship in this study, is the relationship a ***causal*** one?
- Construction Validity: Assuming that there is a causal relationship in this study, can we claim that the program reflected well our construct of the program and measure?
- Conclusion Validity: Is there a relationship between the cause and effect?

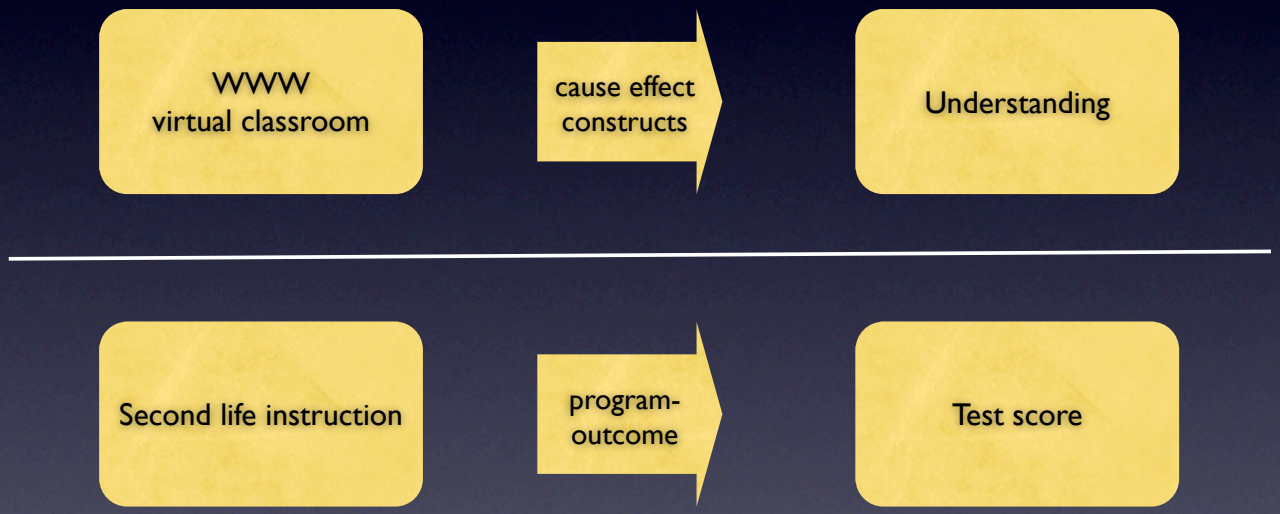
Another way of looking at Validity

Theory: what you *think*

Observation: what you *test*

Example: WWW => Student Learning

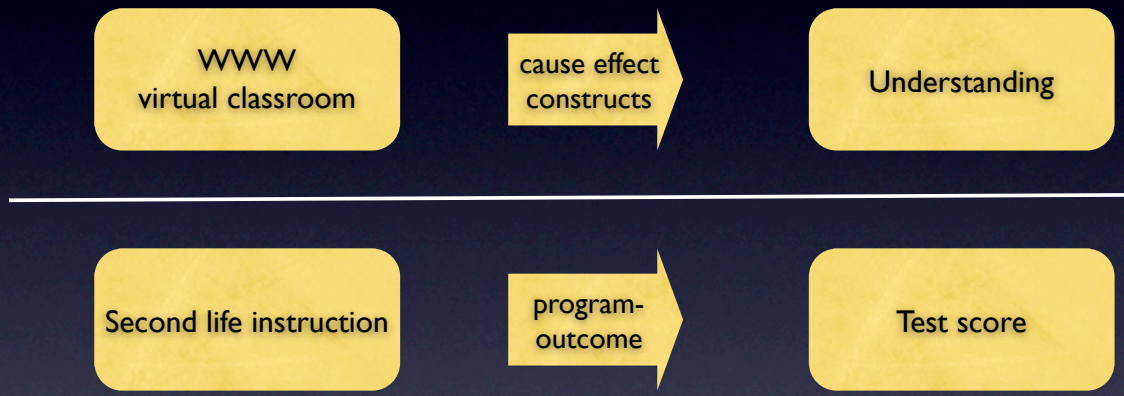
Theory: WWW virtual classroom improves student understanding of course materials



Observation: Let one half of EE382V to use second-life virtual class room and let the other half come to regular lecture. Compare their test scores at the end.

External Validity

Theory: WWW virtual classroom improves student understanding of course

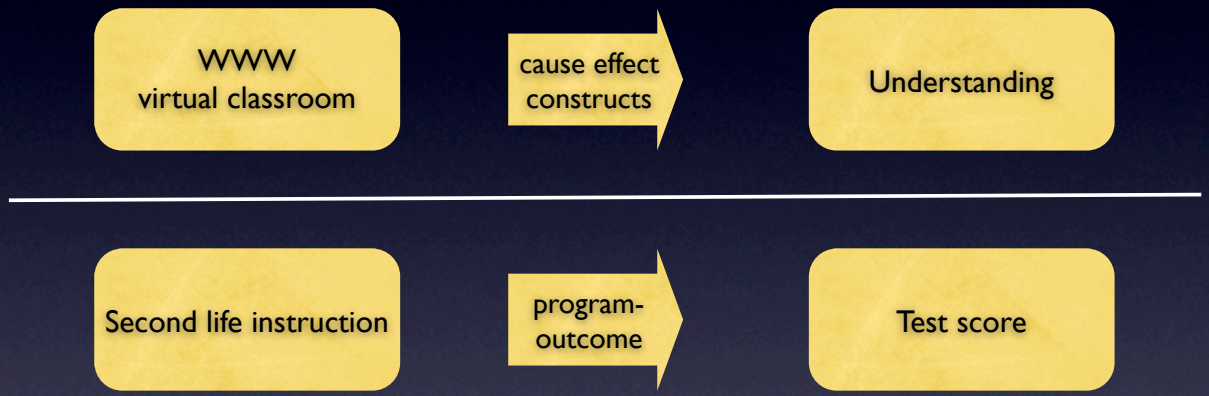


Observation: Let one half of EE382V to use www site and let the other half

External Validity: Does this study generalize to students of EE322c?

Internal Validity

Theory: WWW virtual classroom improves student understanding of course

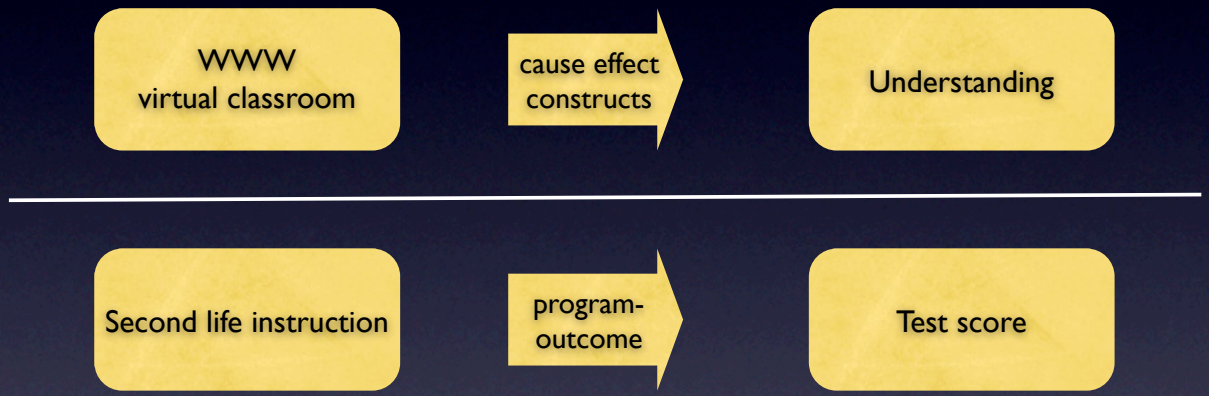


Observation: Let one half of EE382V to use www site and let the other half

Internal Validity: Assuming that students using WWW did better in their test, isn't it because these students have more money (apparently they have computers & high-speed internet) and rich students have more experiences with objective tests (due to their parents sending them to prep-schools.)

Construct Validity

Theory: WWW virtual classroom improves student understanding of course

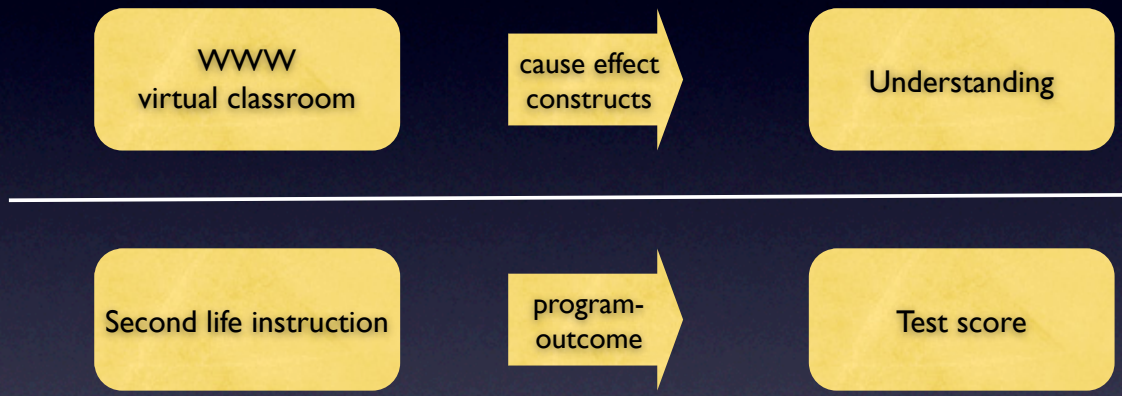


Observation: Let one half of EE382V to use www site and let the other half

Construct Validity: Is the operationalization method valid? Do objective test scores truly reflect students' understanding of core concepts? Don't students who are familiar with second life interface just test better?

Conclusion Validity

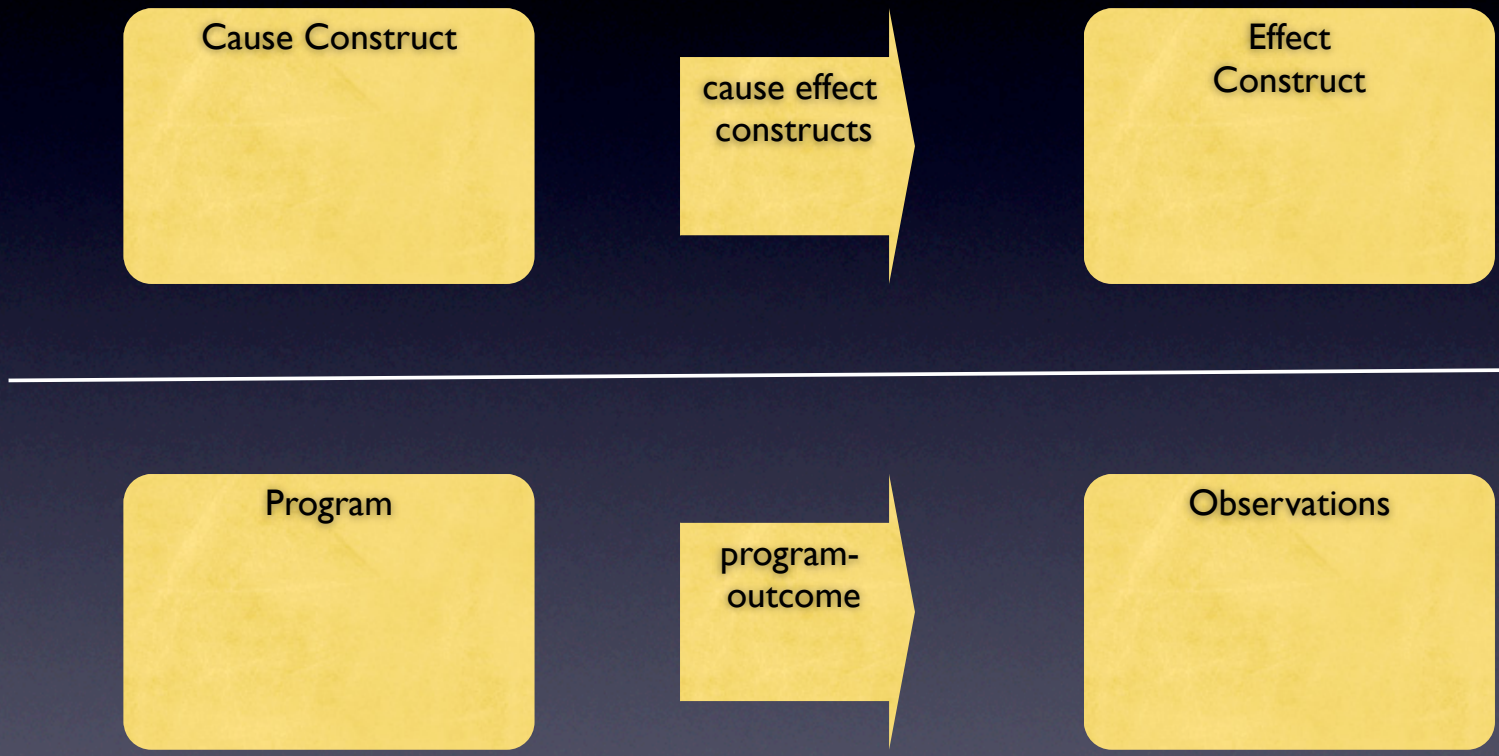
Theory: WWW virtual classroom improves student understanding of course



Observation: Let one half of EE382V to use www site and let the other half

Conclusion Validity: Are the correlation between second-life virtual classroom use and test scores significant?

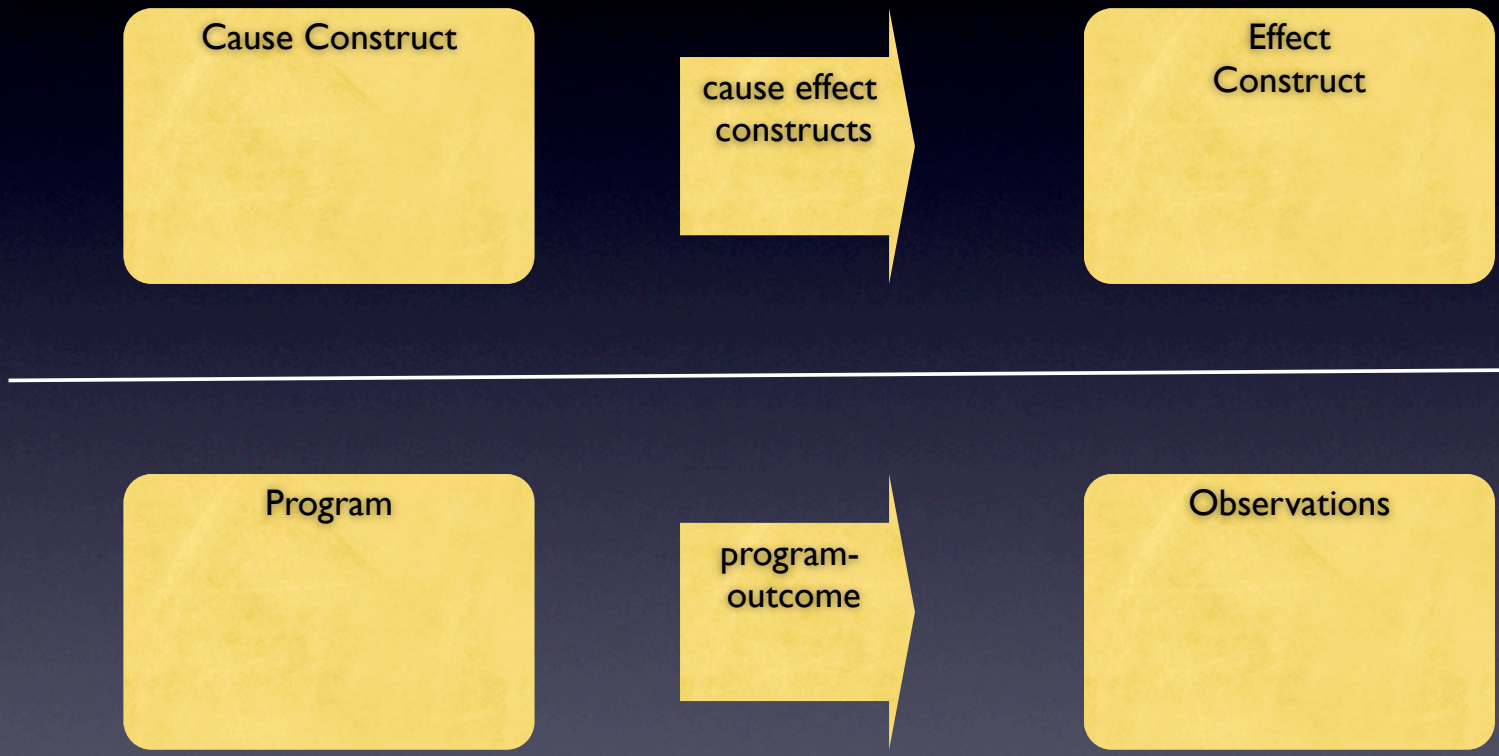
1. Temporal Behavior of the Span of Code Changes



1. Temporal Behavior of the Span of Code Changes

- External Validity
- Internal Validity
- Construct Validity
- Conclusion Validity

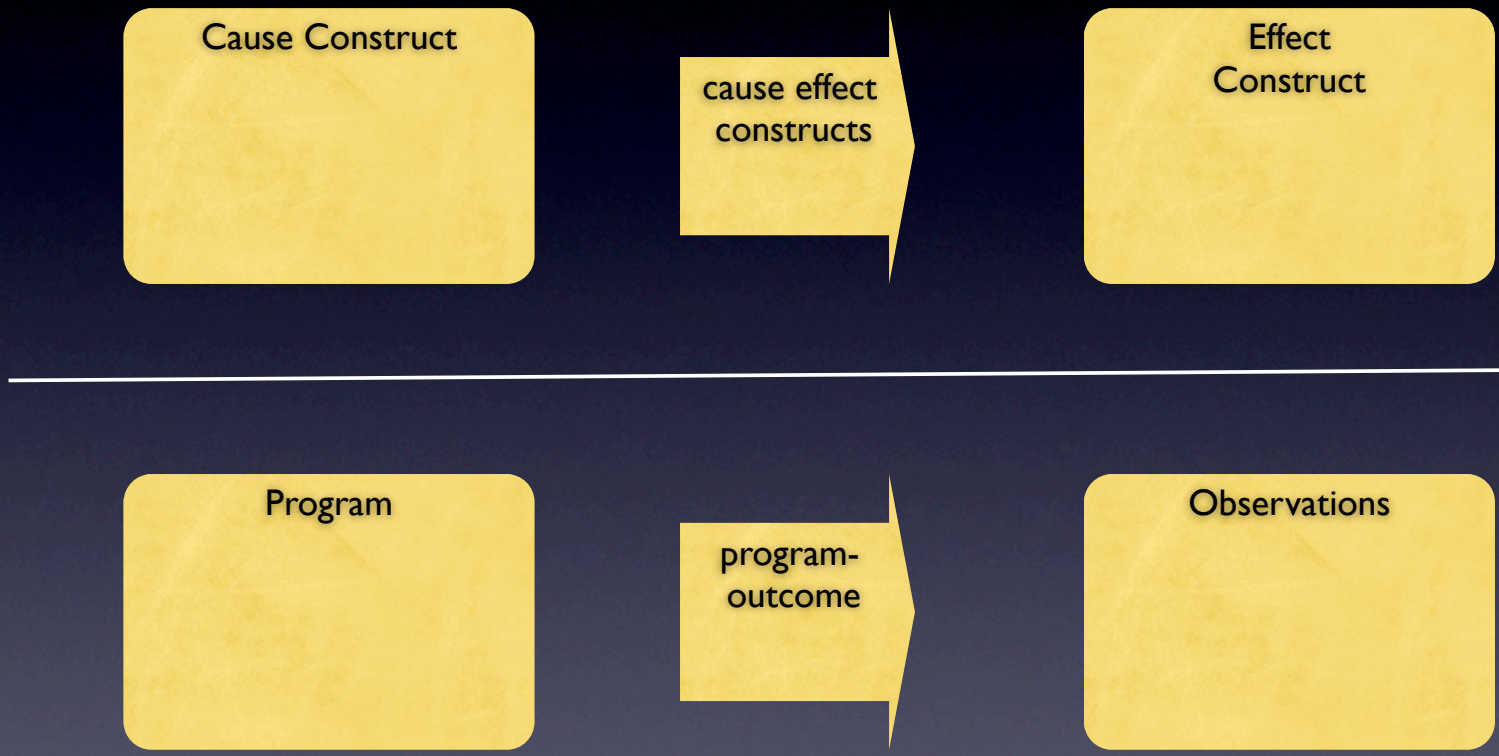
2. Time Behavior of Modularity



2. Time Behavior of Modularity

- External Validity
- Internal Validity
- Construct Validity
- Conclusion Validity

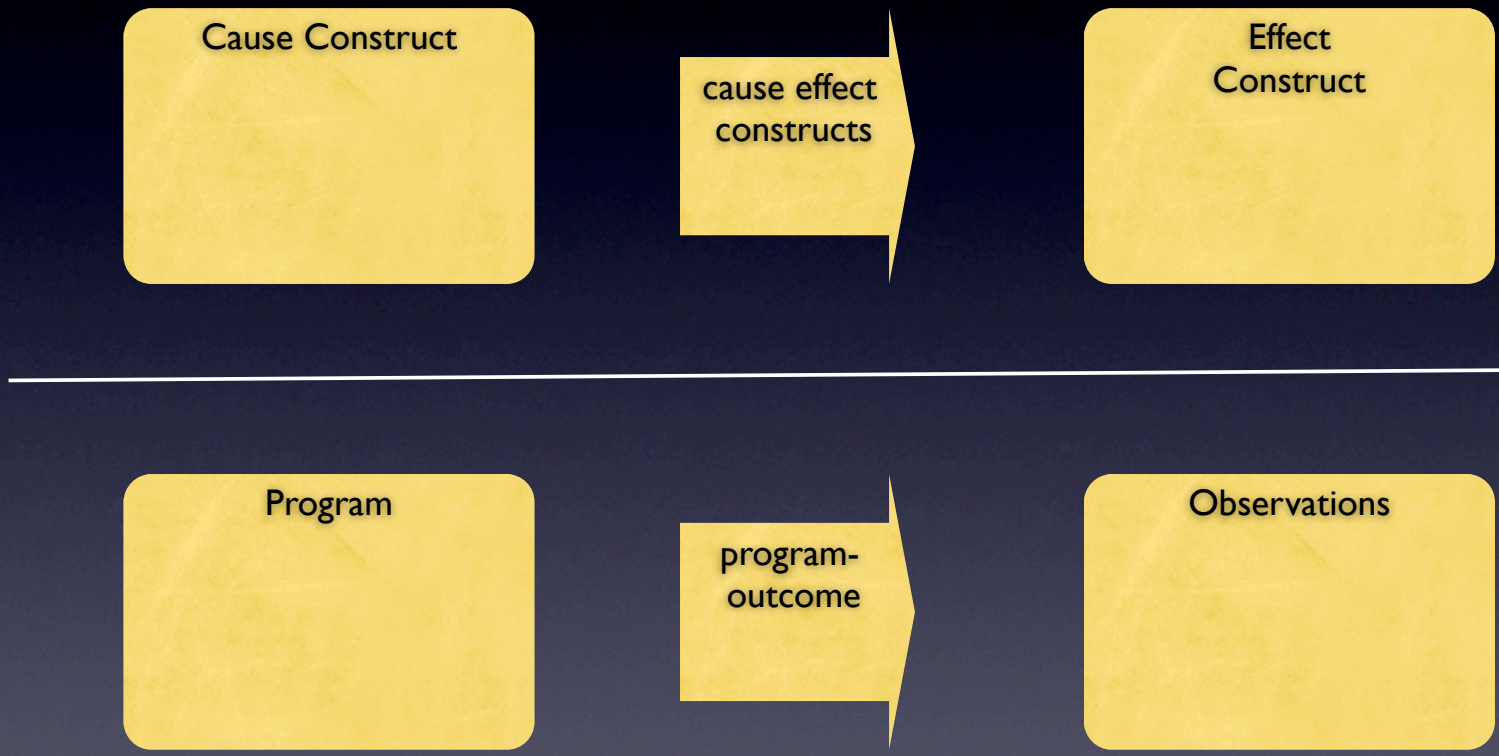
3. Prediction of Faults



3. Prediction of Faults

- External Validity
- Internal Validity
- Construct Validity
- Conclusion Validity

4. Models of Effort



4. Models of Effort

- External Validity
- Internal Validity
- Construct Validity
- Conclusion Validity

My general thoughts on Code Decay Paper

- Rich data set!!!
- Scientific research method
  - Identification of hypotheses => identify key variables and measure them = > create statistical models => statistical regression
- What do identified coefficients real mean?
- Can programmers use any of these findings for daily development activities?

Recap

- Code decay can be mapped to specific measured / derived variables.
  - e.g., span of changes => file span, non-localized changes => changes that spans module boundaries
- Early mining software repositories research in late 90s that is based on statistical regression analysis and visualization
- These types of research require having good insights.
  - e.g., weighted time dampened model
- Identified which factors do mater! => some surprising results that complexity metrics do not matter

Limitations

- Carefully designed model that may have over-fitted data?
- Their model did not consider change types or change content
- Their model cannot handle module specific information
- Their results do not generalize to other systems because most changes in open source system does not map to a logical software change