R&D Meets Production: The Dark Side

J.P.Lewis

zilla@computer.org

Disney The Secret Lab

R&D ⟶ Production Issues

- R&D ⟷ Production interaction is not always easy. In fact...
  - R&D team: not completely sure if it can be done, or how long it will take.
  - Producers: need to get it done and know how long it will take

Can we improve this situation?

Topics

- Humor: anecdotes in course book (+ some R&D successes)
- Math: Paradox meets math: Halting, Godel incompleteness, meets...

  Liar paradox: person from Canada says, “all people from Canada are liars.”
- ...paradox + math meets software R&D
- what is creativity?

Large Limits to Software Estimation

J. P. Lewis, Large Limits to Software Estimation

ACM Software Engineering Notes

Vol 26, No. 4 July 2001 p. 54-59

- How I came to this...
- R&D i.e. *software* (in general sense- including shaders, scripting, ...)

Big Failures of Software Estimation

- An unpublished review of 17 major DOD software contracts found that the average 28-month schedule was missed by 20 months, and no project was on time.
- Air traffic control AAS system: $6.5 billion. "The greatest debacle in the history of organized work...we learned nothing from it"

*Software: It’s Chaos

- GAO-93-13 on major software challenges: “We have repeatedly reported on cost rising by millions of dollars, schedule delays of not months but years, and multi-billion-dollar systems that don’t perform as envisioned.”
- California child support: $100 million, US medical claims: $92 million, IRS: several billion

What is Software Estimation

- Estimation of development schedules, program complexity, programmer productivity, program reliability
- Software Process Management: managing the software development process
- Capability Maturity Model, ISO-900x

*Capability Maturity Model

5 Levels:

1. Initial ("unpredictable")
2. Repeatable
3. Defined
4. Managed
5. Optimizing

*CMM Levels

- At the Defined Level, the standard process for developing and maintaining software across the organization is documented, including both software engineering and management processes, and these processes are integrated into a coherent whole. ... The organization exploits effective software engineering practices when standardizing its software processes.
- At the Managed Level, the organization sets quantitative quality goals for both software products and processes. ...An organization-wide software process database is used to collect and analyze the data available from the projects' defined software processes. Software processes are instrumented with well-defined and consistent measurements at Level 4.

Process Management evaluated

- Good intentions
- Engineering or philosophy? ("coherent whole", "effective software engineering", etc.)
- Not always effective: One spectacular development failure came from one of the few CMM Level 4 organizations

Strong Claims?

- A software process manifesto: "In an immature organization, there is no objective basis for judging product quality or for solving product or process problems...

  [In a mature organization] There is an objective quantitative basis for judging product quality and analyzing problems with the product and process. Schedules and budgets are based on historical performance and are realistic."

*More Claims

- Quality framework document:: "Consistent measurements provide data for doing the following: Predicting the software attributes for schedules, cost, and quality. ..."
- Course title: "Productivity Improvement through Defect-Free Development"

Still More Claims

- Handbook of Quality Assurance: "In the Certainty state [of quality management], the objective of software development and software quality management, producing quality software on time with a set cost everytime, is possible."
- Book promoting a software estimation package: "...software estimating can be a science, not just an art. It really is possible to accurately and consistently estimate costs and schedules for a wide range of projects."

Empirical Studies

- Kemerer: 4 estimation algorithms on 15 large projects for which historical data was available. Post facto error in predicted development time ranged from 85% to >700%.
- DeMarco and Lister Programming Benchmark: Size of code (loc) written by different programmers to a single specification varied by more than a factor of 10.

Problem!

- Estimation procedures take as input an estimate of the complexity of the project – this was obtained from historical data by Kemerer.
- How do we obtain this estimate for a new project?

Absurd Example

- Gather data: the average programmer completes a small programming exercise in 3.7 hours.
- Therefore, a new operating system release can be completed by an average programmer in 3.7 hours?
- Historical data do not help without an estimate of the complexity of the future project!

Algorithmic Complexity (AC)

- Kolmogorov Complexity
- KCS Complexity: Kolmogorov, Chaitin, Solomonoff
- Complexity of a digital object:

The length of the shortest program that produces that object.

AC is intuitive

- Consider

```
1111111...:      for i to n print "1"
1313131...:      for i to n print "13"
3423314...:*     print "3423314........."
```

- * algorithmically random

What about Language?

AC is defined in the large:

$$K_u(x) \leq K_p(x) + O_p(1)$$

Pick any language. A translator from that language to any other is a fixed size, e.g. 100K bytes.
In the limit of large objects, the choice of language is insignificant.

Algorithmic Complexity

- Objective (mathematical) definition complexity
- Intuitive
- Supports precise reasoning about related issues
- Addresses limitations of source code metrics (loc, fp): that such metrics do not reflect the complexity of the code

*AC Simplified

- Prefix Complexity
- Li and Vitanyi, *Kolmogorov Complexity*, Springer

Flavor of AC Reasoning

- "WinZipper2000 is guaranteed to compress any file"
- FALSE: there are 2^N unique files of size N bits. There are fewer than 2^N possible files of (compressed) size less than N bits. Not all 2^N files can be uniquely recovered.
- *Almost all objects are algorithmically random.

Complexity Tower

- Impossible
- Intractable

  (how much work is 2^{64}? 2^{32} is “4 giga”, so if 4Ghz proc takes 60 instructions $\longrightarrow$ 4 giga-minutes = 8181 years!)
- Polynomial, Linear

Incompleteness

- Godel Incompleteness
- Halting problem, Rice’s theorem: there is no program that can determine extensional properties of all programs
- C(x) is not computable

AC Proof of Godel Incompleteness

- A formal theory with N bits of axioms and statements

  $$C(x) > L$$

  contains many such statements that cannot be proved when L is much greater than N.
- If $C(x) > L$ is proved, save the particular x that was found. This allows $x : C(x) > L$ to be generated with $N + O(1)$ bits - contradiction.

Berry Paradox

- “The first number that requires more than a thousand words to specify”
- is 12 words

*Incompleteness

- Out of an infinity of expressible true statements $C(x) > L$, only a fixed number are provable.
- A supposed ‘complexity’ software metric written in 500loc cannot accurately characterize most programs larger than this.

Church-Turing thesis

- (‘Objective’: a step-by-step process that leads you to a common result)
- An objective process is essentially an algorithm, whether undertaken by human or computer.

Claim 1

- Program size and complexity cannot be objectively and feasibly estimated a priori.

Because

- Claim 1: Program size and complexity cannot be objectively and feasibly estimated a priori.
- In fact complexity cannot be feasibly determined, period. (The size of a program is $\geq$ its complexity.)

*AC vs. the real world

- AC is output only
- Function arguments: AC of a large table containing input-output pairs ('tabular size')
- State: consider as implicit argument to any routines that are affected
- Interactivity: bake the user input into the program

Claim 2

- Claim 2: Development time cannot be objectively predicted
- Claim 1: Program size and complexity cannot be objectively and feasibly estimated a priori.

Because

- Claim 2: Development time cannot be objectively predicted
- Objective development time estimate depends on an objective estimate of the complexity (recall absurd 3.7 hour example).

Claim 3

- Claim 3: Absolute productivity cannot be objectively determined
- Claim 2: Development time cannot be objectively predicted
- Claim 1: Program size and complexity cannot be objectively and feasibly estimated a priori.

Because

- Claim 3: Absolute productivity cannot be objectively determined
- Productivity: LOC / time? No, complexity/time: finish a difficult (complex) program quickly = high productivity.
- *Proviso: relative productivity can be objectively estimated by experiment

Claim 4

- Claim 4: Program correctness cannot be objectively determined.
- Claim 3: Absolute productivity cannot be objectively determined
- Claim 2: Development time cannot be objectively predicted
- Claim 1: Program size and complexity cannot be objectively and feasibly estimated a priori.

Because

- Claim 4: Program correctness cannot generally be proved.
- Suppose a proof $F(P, S)$ that program P correctly implements spec S. Then S is formal and $C(S) \approx C(P)$. (Write a program that exhaustively queries S to determine the right output for a given input).

*Approximate Estimator?

- Find $E : C(x) <= E(x) <= C(x) + b$?
- Apply triangle inequality

$$K(a|b) \leq K(a|x) + K(x|b) + O(1)$$

to the two-part description:

$$K(K(p)|p) \leq K(K(p)|B) + K(B|p) + O(1)$$

(B - set of programs $[C(x) \ldots C(x) + b]$)

- $K(*|B) \leq \log|B|$
- But $K(K(p)|p) \neq O(1)$

*(note)

- Note on this:

$$K(K(p)|B) \leq \log|B| + O(1)$$

- The complexity is known to be within finite bounds, so there are a finite number of programs that can be run dovetail, one of them is guaranteed to produce p.

Claim 5

- "$K(B|b) \neq O(1)$", meaning,
- Claim 5: There is no estimator which produces a correct fixed bound on the complexity of all inputs (programs).

Math = computation

$\boxed{\boxed{\text{Math = computation}}}$

- Axioms ↔ program input or initial state
- rules of inference ↔ program interpreter
- theorem(s) ↔ program output
- derivation ↔ computation

Godel ↔ halting ↔ $C(x) \neq O(1)$

Math = Computation

- Every even number is the sum of two primes?
- How long would it take you to write a program to prove or disprove this? Write a program that tests even numbers of increasing size. If this program *halts*...
- Math = programming $\neq$ manufacturing!

Conclusions

- Claims of *objective* estimation are wrong
- I did not say that estimation / process management efforts are not helpful!
- Social responsibility
- Union: lighting is creative, programming not. But if ‘creative’ is ‘that which cannot be automated’, then programming is art, while lighting may not be.

End

The phrase

> “is self-referential, when preceeded by itself”

is self-referential, when preceeded by itself.

“Research”

- *Peer review*

Large Limits to Software Estimation

- Producers need estimates of software development times, but:
- Some of the stronger claims of Software estimation/Software process management advocates are directly contradicted by Kolmogorov complexity.