CSE503: Software Engineering

David Notkin
University of Washington
Department of Computer Science & Engineering
Winter 2002

1

Software evolution

- Software changes
  - Software maintenance
  - Software evolution
  - Incremental development
- The objective is to use an existing code base as an *asset*
  - Cheaper and better to get there from here, rather than starting from scratch
  - Anyway, where would you aim for with a new system?

2

A legacy

- Merriam-Webster on-line dictionary
  - "a gift by will especially of money or other personal property"
  - "something transmitted by or received from an ancestor or predecessor or from the past"
- The usual joke is that in anything but software, you'd love to receive a legacy
  - Maybe we feel the same way about inheritance, too, especially multiple inheritance

3

Change

- "There is in the worst of fortune the best of chances for a happy change" —Euripides
- He who cannot dance will say, "The drum is bad" —Ashanti proverb
- "The ruling power within, when it is in its natural state, is so related to outer circumstances that it easily changes to accord with what can be done and what is given it to do" —Marcus Aurelius
- "Change in all things is sweet" —Aristotle

4

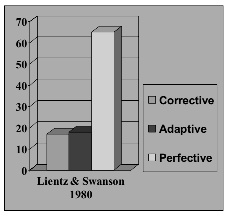
Why does it change?

- Software changes does not change primarily because it doesn't work right
  - Maintenance in software is different than maintenance for automobiles
- But rather because the technological, economic, and societal environment in which it is embedded changes
- This provides a feedback loop to the software
  - The software is usually the most malleable link in the chain, hence it tends to change
    - Counterexample: Space shuttle astronauts have thousands of extra responsibilities because it's safer than changing code

5

Kinds of change

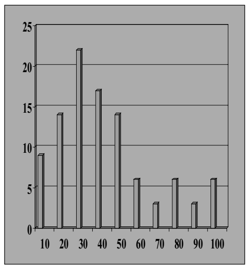
- Corrective maintenance
  - Fixing bugs in released code
- Adaptive maintenance
  - Porting to new hardware or software platform
- Perfective maintenance
  - Providing new functions
- Old data, focused on IT systems…now?



6

High cost, long time

- Gold's 1973 study showed the fraction of programming effort spent in maintenance
- For example, 22% of the organizations spent 30% of their effort in maintenance



7

Total life cycle cost

- Lientz and Swanson determined that at least 50% of the total life cycle cost is in maintenance
- There are several other studies that are reasonably consistent
- General belief is that maintenance accounts for somewhere between 50-75% of total life cycle costs

8

Open question

- How much maintenance cost is "reasonable?"
  - Corrective maintenance costs are ostensibly not "reasonable"
  - How much adaptive maintenance cost is "reasonable"?
  - How much perfective maintenance cost is "reasonable"?
- Measuring "reasonable" costs in terms of percentage of life cycle costs doesn't make sense

9

High-level answer

- For perfective maintenance, the objective should be for the cost of the change in the implementation to be proportional to the cost of the change in the specification (design)
  - Ex: Allowing dates for the year 2000 is (at most) a small specification change
  - Ex: Adding call forwarding is a more complicated specification change
  - Ex: Converting a compiler into an ATM machine is …

10

Question: relationship of reuse to evolution?

11

(Common) Observations

- Maintainers often get less respect than developers
- Maintenance is generally assigned to the least experienced programmers
- Software structure degrades over time
- Documentation is often poor and is often inconsistent with the code

- Is there any relationship between these?

12

Laws of Program Evolution
Program Evolution: Processes of Software Change (Lehman & Belady)

- Law of continuing change
- "A large program that is used undergoes continuing change or becomes progressively less useful."
  - Analogies to biological evolution have been made; the rate of change in software is generally far faster
- P-type programs
  - Well-defined, precisely specified
  - The challenge is efficient implementation
  - Ex: sort
- E-type programs
  - Ill-defined, fit into an ever-changing environment
  - The challenge is managing change
- Also, S-type programs
  - Ex: chess

13

Law of increasing complexity

- "As a large program is continuously changed, its complexity, which reflects deteriorating structure, increases unless work is done to maintain or reduce it."
  - Complexity, in part, is relative to a programmer's knowledge of a system
    - Novices vs. experts doing maintenance
  - Cleaning up structure is done relatively infrequently
    - Even with the recent interest in refactoring, this seems true. Why?

14

Reprise

- The claim is that if you measure any reasonable metric of the system
  - Modules modified, modules created, modules handled, subsystems modified, …
- and then plot those against time (or releases)
- Then you get highly similar curves regardless of the actual software system
- A zillion graphs on http://www.doc.ic.ac.uk/~mml/feast1/

15

Statistically regular growth

- "Measures of [growth] are cyclically self-regulating with statistically determinable trends and invariances."
  - (You can run but you can't hide)
    - There's a feedback loop
  - Based on data from OS/360 and some other systems
  - Ex: Content in releases decreases, or time between releases increases
- Is this related to Brooks' observation that adding people to a late project makes it later?

16

And two others

- "The global activity rate in a large programming project is invariant."
- "For reliable, planned evolution, a large program undergoing change must be made available for regular user execution at maximum intervals determined by its net growth."
  - This is related to "daily builds"

17

Open question

- Are these "laws" of Belady and Lehman actually inviolable laws?
- Could they be overcome with tools, education, discipline, etc.?
- Could their constants be fundamentally improved to give significant improvements in productivity?
  - Within the past two years, Alan Greenspan and others have claimed that IT has fundamentally changed the productivity of the economy: "The synergistic effect of new technology is an important factor underlying improvements in productivity."

18

Approaches to reducing cost

- Design for change (proactive)
  - Information hiding, layering, open implementation, aspect-oriented programming, etc.
- Tools to support change (reactive)
  - grep, etc.
  - Reverse engineering, program

19

Approaches to reducing cost

- Improved documentation (proactive)
  - Discipline, stylized approaches
  - Parnas is pushing this *very* hard, using a tabular form of specifications
  - Literate programming
- Reducing bugs (proactive)
  - Many techniques, some covered later in the quarter
- Increasing correctness of specifications (proactive)
- Others?

20

Program understand & comprehension

- Definition: The task of building *mental models* of the underlying software at various abstraction levels, ranging from models of the code itself to ones of the underlying application domain, for maintenance, evolution, and re-engineering purposes [H. Müller]

21

Various strategies

- Top-down
  - Try to map from the application domain to the code
- Bottom-up
  - Try to map from the code to the application domain
- Opportunistic: mix of top-down and bottom-up

- I'm not a fan of these distinctions, since it has to be opportunistic in practice
  - Perhaps with a *really* rare exception

22

Did you try to understand?

- "The ultimate goal of research in program understanding is to improve the process of comprehending programs, whether by improving documentation, designing better programming languages, or building automated support tools." —Clayton, Rugaber, Wills
- To me, this definition (and many, many similar ones) miss a key point: What is the programmer's task?
- Furthermore, most good programmers seem to be good at knowing what they need to know *and what they don't need to know*

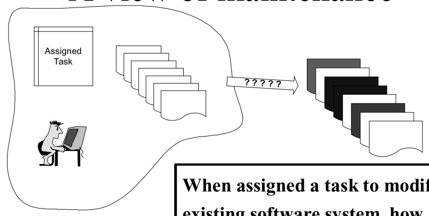
23

A scenario

- I'll walk through a simple scenario or two
- The goal isn't to show you "how" to evolve software
- Rather, the goal is to try to increase some of the ways in which you think during software evolution

24

A view of maintenance



When assigned a task to modify an existing software system, how does a software engineer choose to proceed?

25

Sample (simple) task

- You are asked to update an application in response to a change in a library function
- The original library function is
  - `assign(char* to, char* from, int cnt = NCNT)`
  - Copy `cnt` characters from `to` into `from`
- The new library function is

  -
    ```
    assign(char* to, char* from, int pos,
             int cnt = NCNT)
    ```
  - Copy `cnt` characters starting at `pos` from `to` into `from`
- How would you make this change? (In groups)

26