Adaptation, Selection, and Intelligent Design: The Forces Behind Software Evolution

University of Waterloo

Michael W. Godfrey

Software Architecture Group (SWAG)
University of Waterloo
D.R. Cheriton School of Computer Science

Overview

- A quick introduction to biological evolution
  - Natural selection, evo-devo, and spandrels
- What, exactly, is software?
  - Passive theorem or hungry beast?
- Software change: maintenance vs evolution
  - Adaptation, selection, and intelligent design
- Lehman's laws, the S curve, and the Linux kernel
  - What has history taught us?
- Where do we go from here?

Disclaimers

- I dropped biology after grade 9
  - A little knowledge is a dangerous thing.
- I'm not (yet) an accredited engineer
- Some of these are very personal opinions
- Please try this at home!

Sponsorship

- Today's talk is brought to you by
  - The friends of Charles Darwin
  - NSERC
  - SWAG and the University of Waterloo
- We do research in these questions at UW
  - If this talk is interesting to you, please consider applying for grad school at UWaterloo!
  - Software engineering applicants are especially valued by our group (SWAG)!

What is evolution?

Essential change in a population over time

3 fundamental ideas of evolution

1. Mechanisms that increase variation
   (i.e., act as an agent for essential change)

2. Mechanisms that decrease variation
   (i.e., act as an inhibitor for essential change)

3. It's all relative, man.
   - Change occurs within an environment, which also evolves.
   - That is, what succeeds today may not succeed tomorrow (but may succeed the day after).

Bio. evolution in a nutshell

"[Bio.] evolution is change in a gene pool of a population over time."
[`talk.origins` FAQ]

- Genes/alleles are the essential encoding that is inherited by living creatures; they comprise the *genotype* of the individual.
  - A gene is the dedicated "slot" or memory location
  - Alleles are the set of abstract values at those genes
  - DNA are the bits that encode the allele messages (ACGT)

- The *phenotype* is the totality of what an individual becomes.
  - The phenotype is determined by the genotype PLUS interaction with the environment.
  - Non-genotypic change is not inheritable! [Lamarck was wrong!]

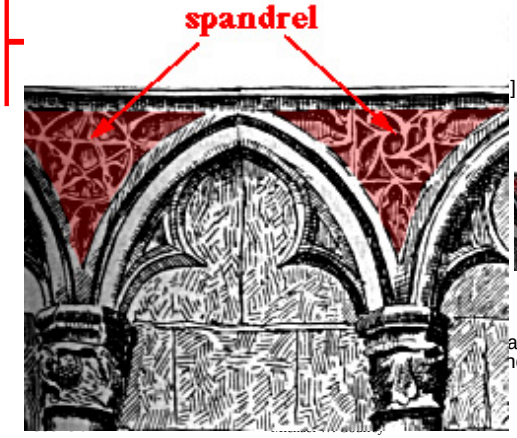
Bio. evolution in a nutshell

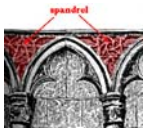
1. *Mechanisms that increase genetic variation*
   - Mutation (new values)
   - Recombination (new combinations)
   - Gene flow
   - Genetic drift (chance increase in existing allele freq.)

2. *Mechanisms that decrease genetic variation*
   - Natural selection (!!)
   - Sexual selection
   - Genetic drift (chance decrease in existing allele freq.)

3. *Evolution happens in an environment, which changes*

Fun facts about bio. evolution





alances, and
ney are able

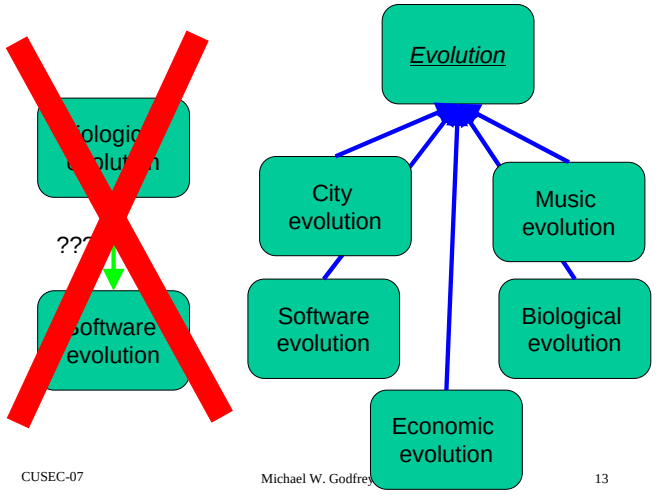
Evolutionary Development

- Evolution:
  - What happens to a *population* over time
- Development:
  - What happens to an *individual* over time
- Evolutionary development:
  - How development evolves over time

[ref: Endless Forms Most Beautiful, by Sean Carroll. *"The new science of evo-devo"]*

A common pattern in Evo-Devo

1. Replication
   - Legs are good, let's have more
   - *Servers, VMs, processors are good*
2. Specialization
   - Front legs need pincers
   - *DB server, web server, file server; intelligent controllers*
3. Pruning, if needed
   - Three pairs of legs are enough, it seems
   - *If consuming needed resources (memory, power), consider retirement or redeployment*



What, exactly, is software?

- "Executable"
- Written in a prog. language by a human
- Data transformer
- Hardware scripting tool
- Manages Platonic artifacts (*e.g.,* spreadsheet)
- Changeable
- Can do "anything"

What, exactly, is software?

- Can do "anything":
  - Pure design
  - Create / manage intellectual abstractions
    - Limited only by your imagination
  - Has mathematically precise semantics
    - Can be proven correct

How systems fail

- Physical systems often fail because of *materials failure*
  - So maintenance is about understanding materials, using redundancy, replacing "worn out" parts
- Software system usually fail because of *design flaws*
  - Sw systems don't wear out thru "bit rot"
  - Sw systems are so complex that most have many bugs that are never discovered
  - So maybe we should concentrate on "correctness" and our problems will go away?

Passive theorem or angry beast?

- *A sw system is a passive theorem!*
  - *It just needs to be beaten with formal methods so that correctness will be guaranteed!*
    - We know that this view doesn't scale up well
    - Formal methods certainly have their uses, but they are not a panacea
- *Accept that bugs will exist!*
  - The power of prog langs and the complexity of the systems pretty much guarantee this [Brooks]
  - Not all bugs are fatal: some can be fixed, some ignored
  - The use of established engineering design techniques can help *e.g.,* redundancy, periodic sanity checks, sandboxing

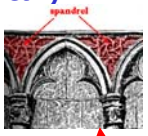
Sw evolution in a nutshell

1. Forces that encourage variation
   - User demand, new platforms, marketing, emergent uses
2. Forces that limit variation
   - Complexity, market saturation, politics
3. Sw is embedded in an environment
   - And that environment (development, user, political) forms a complex feedback loop that affects its evolution [Lehman]

Responding to evolutionary pressures

- Basic pressure on sw to evolve
  - Lehman's first law: Adapt or die
  - Software doesn't decay physically
    - Rather, the environment and our expectations change
- "Intelligent design"
  - Parnas: Design for Change
    - Info hiding, virtualize likely hotspots, design reviews
  - OO dev, frameworks, AOSD
  - ... but you can't anticipate everything
  - ... and flexibility has a cost

Responding to evolutionary pressures



- Selection and adaptation
  - The deployment environment (users) "selects" individuals and features for success
  - Tho, unlike bio, this can also be planned and evaluated
- Software systems often exhibit *emergent* properties (cf. spandrels)
  e.g., vmware, XML, WWW, IM as a debugger

Bio. vs. software evolution

- *One important difference:*
  - Unlike biology, the prime agent for new features of software is not chance, it's marketing, *etc.*!
    - Rate of change is much, much faster!
      - Modern humans have been through about 500 generations in 10,000 years since first agrarians
      - Linux has been thru more than 500 versions since 1994
    - Software evolution is (partly) Lamarckian!

Bio. vs. software evolution

- *One important observation:*
  - Not all of the effects of software change are planned or foreseen.
    - Software reacts with its environment (development, deployment, political) and forms a complex feedback system that influences the future evolution of the software [Lehman]
    - Software systems exhibit interesting emergent phenomena (*e.g.,* the Java platform)

Summary: Bio. vs. sw evolution

- If you take away nothing else from this, at least remember this:
  - *"We shape our tools, then our tools shape us."* [McLuhan]
  - We can plan and execute changes ...
    - ... but we cannot understand in advance all of the effects of the changes, both on the environment and feeding back into the development of the software system.
  - Understanding how software evolves requires studying both the planned and unplanned phenomena of software change.

Maintenance vs. Evolution

- *Maintenance* connotes
  - Fixing, rather than intellectually enhancing
  - Short-term, not long-term goals
- Many prefer the term *evolution:*
  - *Fundamental* change and adaptation
  - Short- and long-term change
  - Planned *and* unplanned phenomena [my two cents]

Why study software evolution?

- Improved understanding:
  - Why is your system is designed as it is?
    c.f. the "temporal layers" architectural pattern
  - Quality assessment of third-party software
  - Spot recurring problems, development bottlenecks
- To better anticipate change and reduce risk
- Because we can :-)

Research in software evolution

- We are still at the (early) stage of formulating theories and performing case studies.

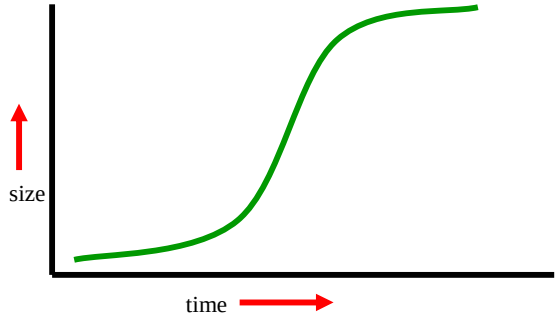
Does open source software evolve differently from industrial closed-source software?
How? Why?

How common is code cloning in industrial software?
Are all kinds of clones equally problematic?
What are the long term effects of cloning?
Is it better to fix old clones or let them be?

Lehman's Laws in a nutshell

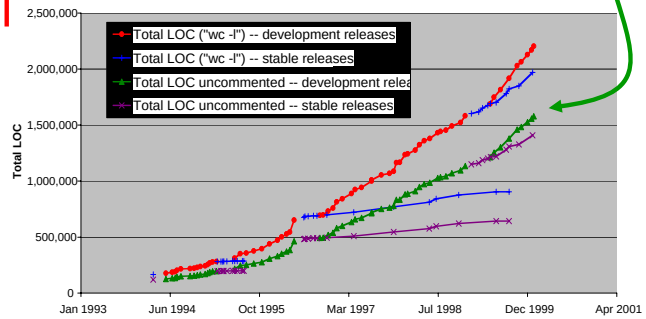
- **Observations:**
  - (Most) useful software must evolve or die.
  - As a software system gets bigger, its resulting complexity tends to limit its ability to grow.
  - Development progress/effort is (more or less) constant; growth is *at best* constant.
    Lehman/Turski's model: $y' = y + E/y^2 \sim (3Ex)^{1/3}$
    where y= # of modules, x = release number
- **Advice:**
  - Need to manage complexity.
  - Do periodic redesigns.
  - Treat software and its development process as a feedback system (and not as a passive theorem).

The S curve

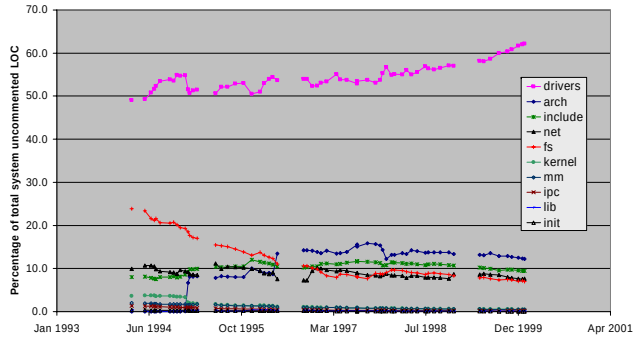


Growth of Lines of Code (LOC)

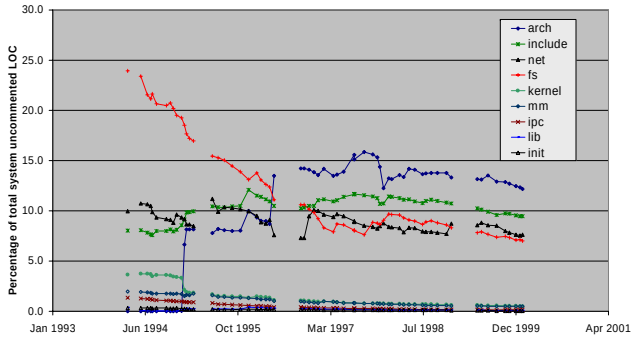
$y = .21*x^2 + 252*x + 90{,}055 \quad r2=.997$



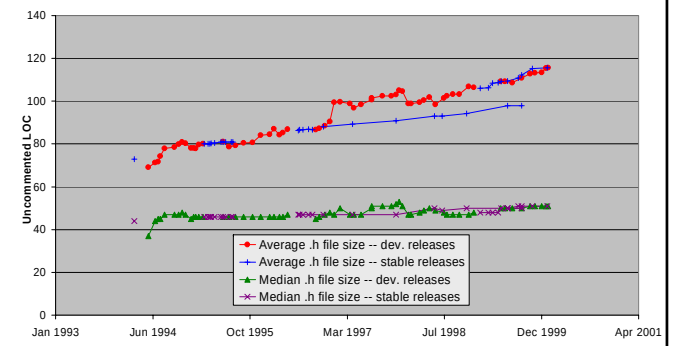
SS LOC as %age of total system



SS LOC as %age of total system



Average / median .h file size



Change patterns and evolutionary narratives

- "Band-aid evolution"
  - just add a layer, temporal architecture
- "Vestigial features"
- "Convergent evolution"
- "Adaptive radiation" [Lehman]
  - When conditions permit, encourage wild variation
  - Later: evaluate, prune, and let "best" ideas live on

Change patterns and evolutionary narratives

- Phenomena observed in Linux evolution
  - Careful control of core code; more flexibility on contributed drivers, experimental features
    - Linus has many lieutenants
  - "Aunt Tillie" effect
    - Simplicity and scrutability of code, development processes, approval process, *etc.*
  - "Mostly parallel" enables sustained growth
    - "Hard interfaces" make good neighbours.
    - Loadable modules makes feature development easier
  - "Clone and hack" makes sense!

Change patterns and evolutionary narratives

- Phenomena observed in Linux evolution
  - Amazing social phenomenon of OSD
    - "You can try this at home"
      - and they did!
    - Anti-MS sentiments,
      - *"We can build it ourselves!"*
      - Enlightened self-interest for many large computer industry companies
        - » *"If we can't own the standard, no one should."*
    - Bandwagon effect (both OS developers and industry)
      - Support for Linux as deployed OS by IBM, Dell, Sun, ...
      - Lots of contributed *production-quality* third party code from industry (IBM S/390, drivers)

The future of sw evol research

- Software *development* is about creating and managing abstractions
  - Software *analysis* is about extracting abstractions from the artifacts
  - Sometimes the abstractions have to been delivered kicking and screaming with forceps
- Much latent info about sw development hides in the artifacts
  - Mining sw repositories, finding useful latent info
  - Think like a scientist and an engineer
    - Common sense is still required (Daniel German)

Summary

- Tortoises, software, and economies evolve
  - Software evolution research is about uncovering latent abstractions
- Understanding both planned and unplanned change is key
  - Dig into the evolutionary history, find lumps under the carpet, ask "why"
- Science is about asking questions, engineering is about using what you've learned to practical ends
  - We need both software scientists and software engineers!