IT University of Copenhagen

Software Architecture Reconstruction

III: Evolutionary Analysis

Assoc. Prof. Mircea Lungu

mlun@itu.dk

github.com/mircealungu/reconstruction

Outline

- What is Evolutionary Analysis
- How can it help in architecture recovery?
- Challenges of Evolutionary Analysis

What is Software Evolution?

No man ever steps in the same river twice, for it is not the same river and he is not the same man.

-- Heraclitus

Software Systems Must Evolve

> *An e-type program that is used in a real-world environment must change, or become progressively less useful in that environment. (Lehman's Law of Continuing Change)*

What might be referring to when he talks about an e-type system?

Software Systems Must Evolve

> *An e-type program that is used in a real-world environment must change, or become progressively less useful in that environment. (Lehman's Law of Continuing Change)*

What might be referring to when he talks about an e-type system?

- an e-type system is *embedded* in the real world
- the world changes, so the system must change

Software Evolution

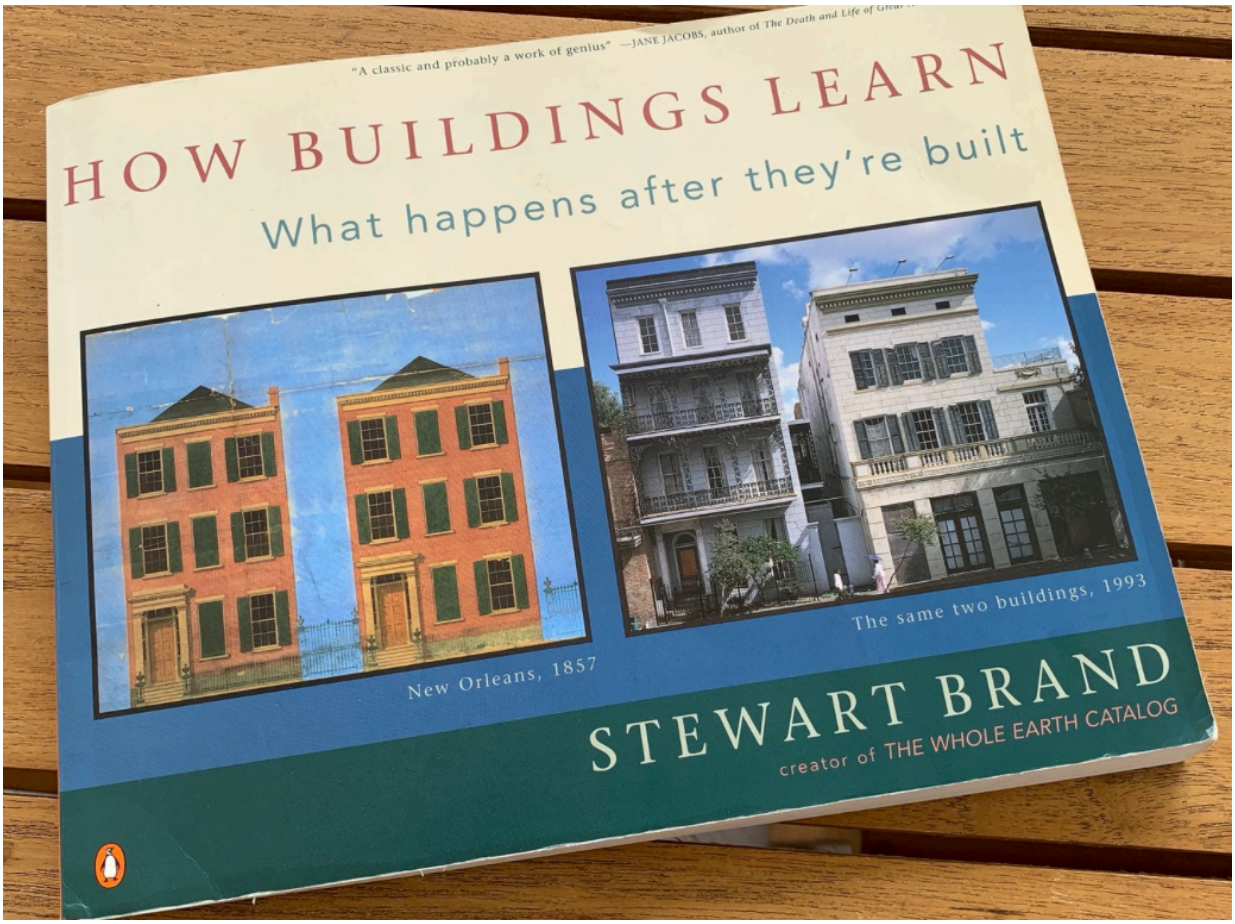
Software evolution is the continual development of a piece of software after its initial release to address changing stakeholder and/or market requirements

- we used to talk about *software maintenance*
- nowadays evolution is the preferred term

Metaphors?

From this POV, the architecture metaphor is not the best. Would a *garden* been a better metaphor?

Although, architecture is also not that bad.



So What is Evolution Analysis?

= the study of how a system evolves over time

Where can we find such information about its evolution?

So What is Evolution Analysis?

= the study of how a system evolves over time

Where can we find such information about its evolution?

Version control repository.

So What is Evolution Analysis?

= the study of how a system evolves over time

Where can we find such information about its evolution?

Version control repository.

Are you aware of tools that supports such analysis?

So What is Evolution Analysis?

= the study of how a system evolves over time

Where can we find such information about its evolution?

Version control repository.

Are you aware of tools that supports such analysis?

One example: Git-Truck

```
git clone git@github.com:zeeguu/api.git
npx -y git-truck-beta@latest
```

Why is it named git-truck?

So What is Evolution Analysis?

= the study of how a system evolves over time

Where can we find such information about its evolution?

Version control repository.

Are you aware of tools that supports such analysis?

One example: Git-Truck

```
git clone git@github.com:zeeguu/api.git
npx -y git-truck-beta@latest
```

Why is it named git-truck?

The truck-factor To think about

- Is a turck factor good if high or if low?
- How could you improve it?

In which way can this information be useful for architecture recovery?

In which way can this information be useful for architecture recovery?

To highlight the parts of the system that are most changed

Why?

In which way can this information be useful for architecture recovery?

To highlight the parts of the system that are most changed

Why?

> *"The value of anything is proportional to time invested in it." (M. Lungu)*

Practically:

- studies observe correlation between *code churn* and complexity metrics
- it's likely that they'll require more effort in the future (e.g. yesterday's weather [Girba et al.])
- high *code churn* predicts bugs better than size

Code Churn

= a metric that indicates how often a given piece of code—e.g., a file, a class, a function—gets edited.

- process metric (*as opposed to?*)
- can suggest relevance for the architecture (*in wjhich way?*)
- can be detected with **language independent analysis** (which is good for polyglot systems)

Viewpoint: Evolutionary Hotspots

= an architectural viewpoint that highlights those code entities where most commits are made

Notebook: Computing Evolutionary Hotspots with PyDriller

Challenges

- Taking into account developer styles
  - the micro-commits developer vs. the large chunk commiter
- Removing irellevant files that change frequently (`README.md`, or `LICENSE.md`)
  - Combine with static complexity metrics
  - Manual investigation
- Selecting the appropriate time-interval for the analysis
  - Weighting towards recency (discarding past changes more)
- Sometimes git loses track of file history
  - e.g. if you rename and make changes at the same time