Software properties and software engineering paradigms

CS 515, Spring 2020

Laura Moreno
lmorenoc@colostate.edu

1

Software engineering paradigms

2

Software engineering

Set of recommendations on how to develop software

- Software is a result of software engineering

A discipline with a considerable body of knowledge and considerable importance in both academia and industry

Colorado State University

@lymorchoc

3

Paradigms & anomalies

Paradigm |ˈperəˌdīm|

noun

1 *technical* a typical example or pattern of something; a model: *there is a new paradigm for public art in this country*.

2 *linguistics* a set of linguistic items that form mutually exclusive choices in particular syntactic roles.

3 *science* coherent tradition of scientific research (by Kuhn).

4 *engineering* coherent tradition of engineering research and practice (by Rajlich).

Colorado State University

@lymorchoc

4

Paradigms & anomalies

Anomaly |əˈnäməlē|

noun (plural **anomalies**)

1 something that deviates from what is standard, normal, or expected.

2 *astronomy* the angular distance of a planet or satellite from its last perihelion or perigee.

3 an important fact that directly contradicts an old paradigm (by Kuhn).

Colorado State

@lvmorchoc

5

The beginnings of software

Software separated from the hardware in 1950s

- emerged as a distinct technology
- became independent product

Original programmers recruited from the ranks of electrical engineers and mathematicians

- used ad-hoc techniques from their former fields

Colorado State

@lvmorchoc

6

The change of the 70s

The birth of software engineering

The response to complexity

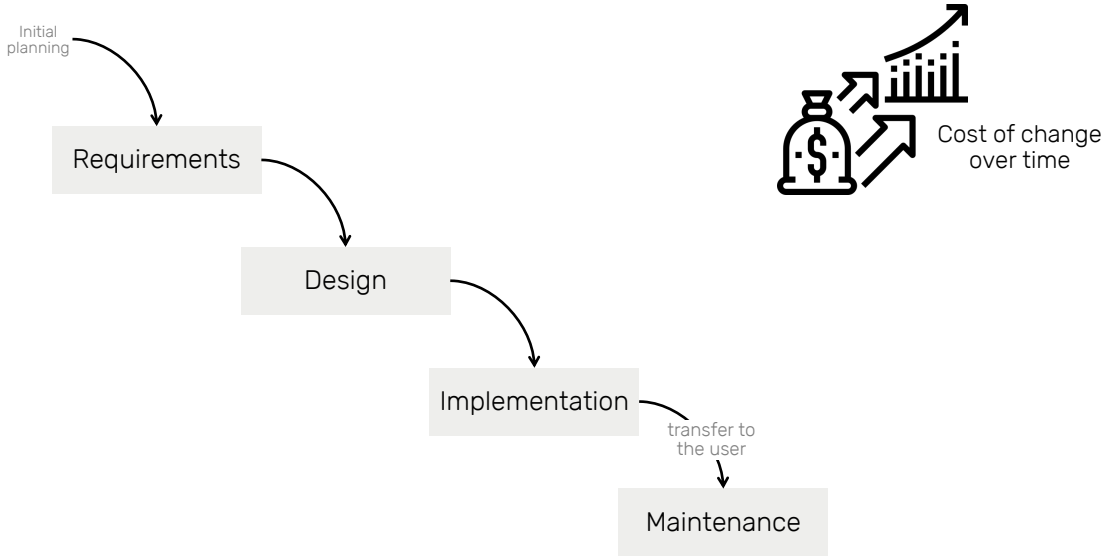
Anomaly

- Software complexity
- Previous (ad hoc) techniques did not scale up
- Brooks: "Mythical Man-Month"
  - Adding new people to a large software project that is already underway will slow it down instead of speed it up
  - Demands of the new operating system OS/360 taxed the limits of the programmers, project managers, and the resources of the IBM corporation

Paradigm change established discipline of software engineering (first ICSE)

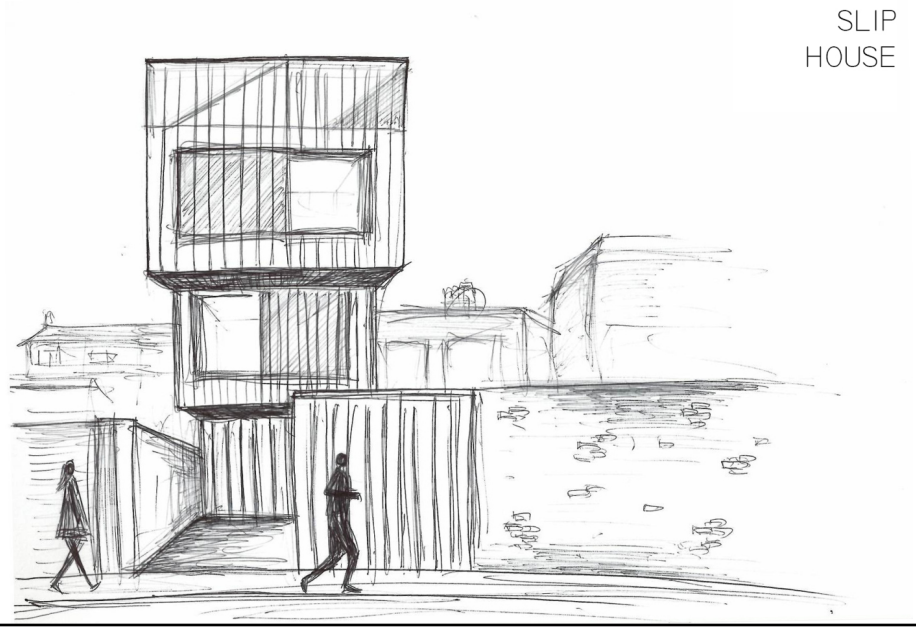
- Software requires people with specialized skills
- Dealt with complexity of software
- Software design established as an important consideration
- Introduced the waterfall metaphor

The waterfall model



9

The waterfall model in construction



11

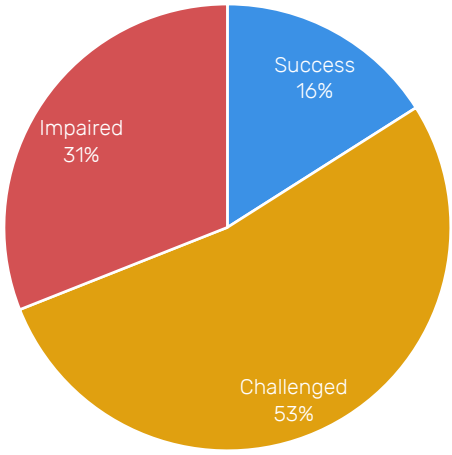
Waterfall is not enough in SE

The waterfall metaphor did not solve the problems of software development

It can't deal with volatility of requirements, technologies and knowledge

- Late software changes will happen

Different domains have different levels of volatility



Resolution of projects, The Standish Group report (1994)

The transition

Band-Aids

Band-Aid ::: Anticipation of changes

If changes can be anticipated at design time, they can be controlled by a parameterization, encapsulations, etc.

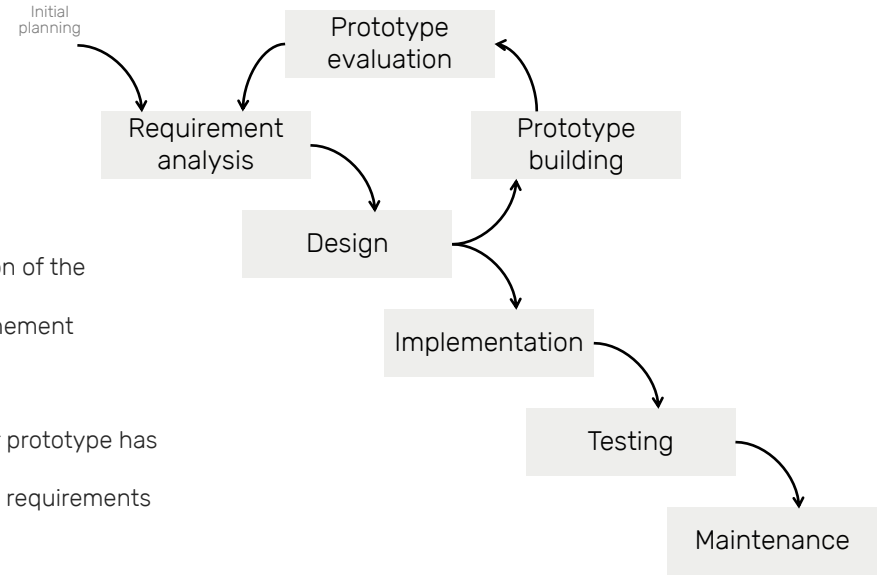
- Waterfall model still can be used

Experience confirms:

- Many changes are not anticipated by the original designers
- Inability to change software quickly and reliably means that business opportunities are lost

14

Band-Aid ::: Prototyping



Prototype

- A very preliminary version of the software
- Allows requirement refinement

Problem

- Volatility continues after prototype has been completed
- Only useful for the initial requirements elicitation

15

The third paradigm

The iterative/evolutionary approach

16

The response to volatility

Anomaly

- Volatility of requirements, technologies and knowledge
- The waterfall model can't accommodate volatility
- 30% or more requirements may change during development (Cusumano & Selby)
- Requirements for IT change at a rate of 2–3% per month (Caper-Jones)

Paradigm change shifted software engineers' priorities

- Focus on methods that allow to keep up with changes in requirements (technologies, knowledge)
- Plans are temporary
- Introduced the iterative and evolutionary approaches for software development

17

The paradigm change of the 2000s

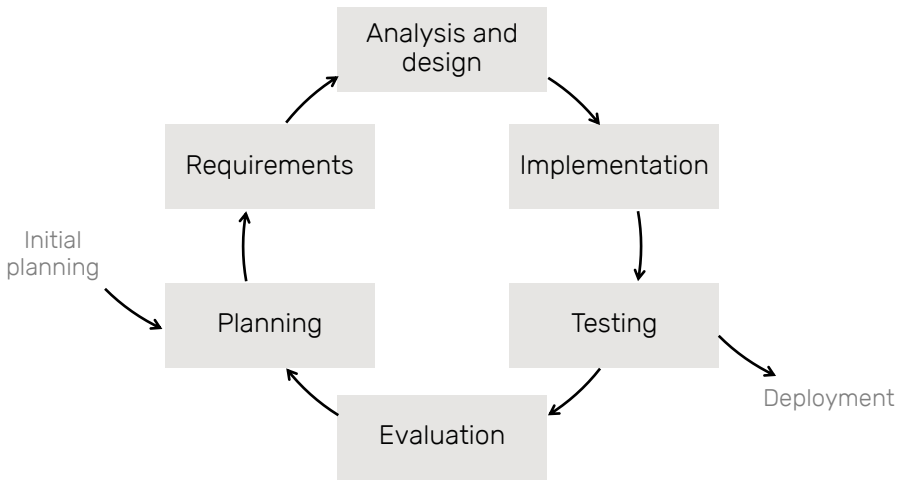
Iterative/evolutionary development
- Rational Unified Process
- …

New life-span models emphasize software evolution
- Staged model of software lifespan
- Based on data from long-lived systems

Agile development
- SCRUM
- Extreme programming
- …

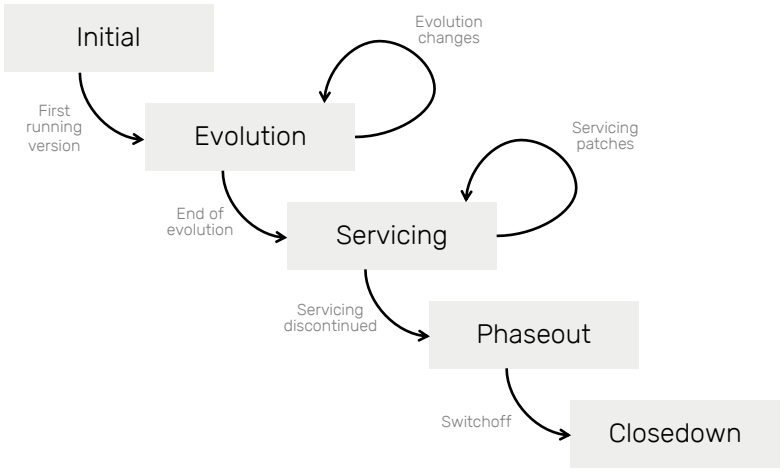
18

Iterative software development



19

Staged model



20

All paradigms currently coexist

The ad hoc paradigm is still used by some single programmers

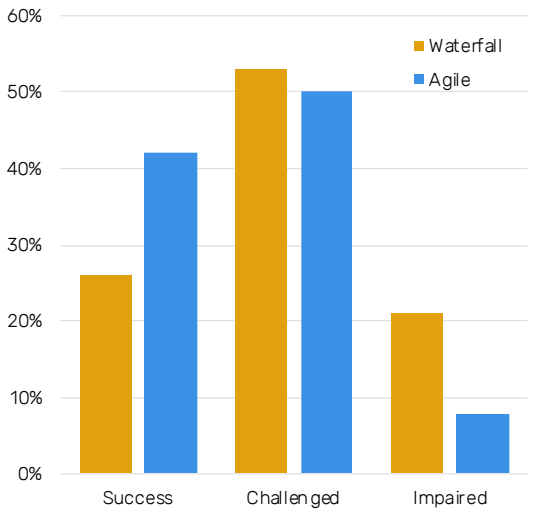
- Programming as an art rather than engineering
- Example: small games, some mobile apps

Waterfall works if there is no volatility

- Small or short-lived projects
- Unusually stable requirements and environments

The iterative/evolutionary/agile paradigms are the new mainstream

Evolution is the key process of the new paradigms



Resolution of projects,
The Standish Group report (2019)

21

Software properties

Software properties types

Essential	Accidental
Intrinsic to software (determine its nature)	Related to its production
Ever-present	Ephemeral
Mental crafting of the conceptual construct	Implementation process

Essential properties ::: Complexity

Programs are among most complex systems ever created

Limited short-term memory capacity

- "The magical number seven, plus or minus two" by George Miller (1956)
- "The magical number 4 in short-term memory: A reconsideration of mental storage capacity" by Nelson Cowan (2001)

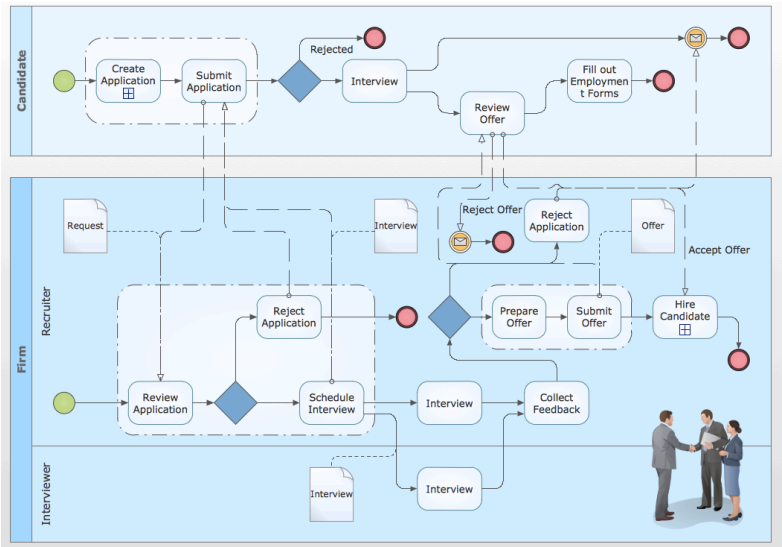
Cause of most of software problems

Several well-known strategies to deal with complexity

- Modeling
- Decomposition
- As-needed approach
- Abstraction

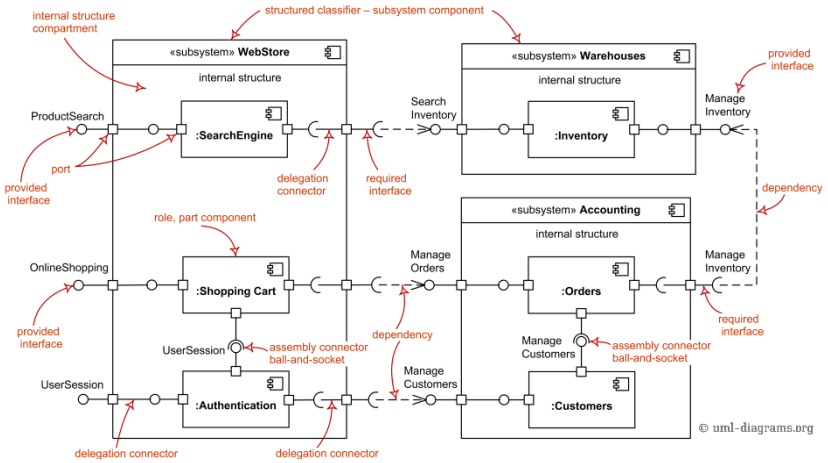
26

Modeling



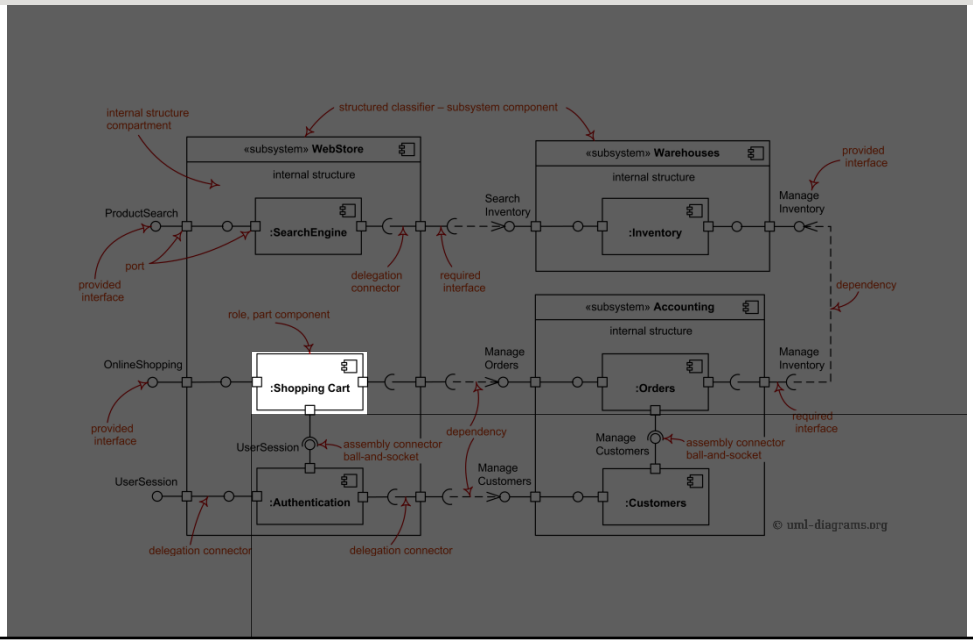
27

Decomposition



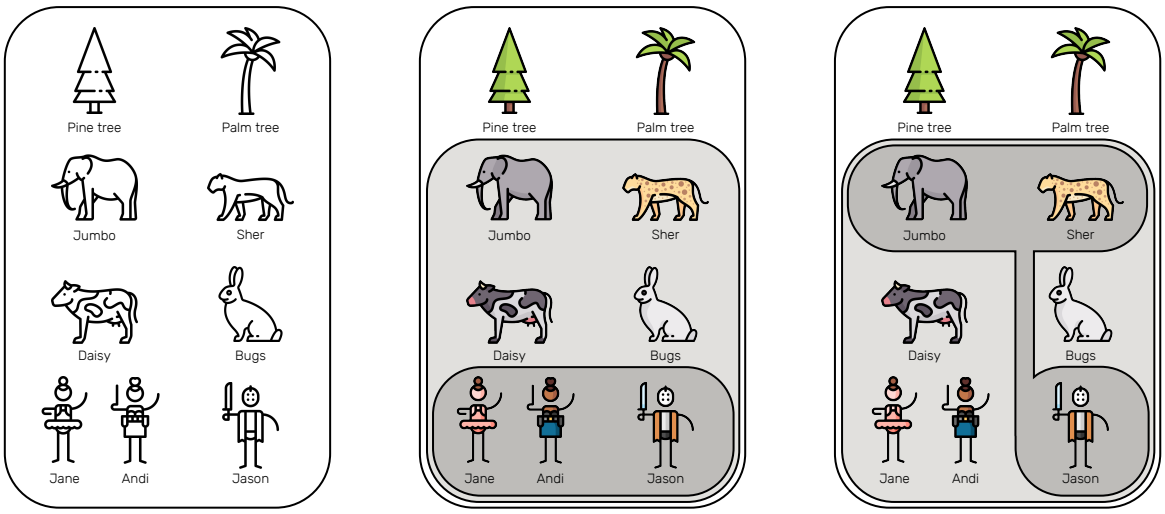
28

As-needed approach



29

Abstraction



unclassified "things"

organisms, mammals, humans

organisms, mammals, dangerous mammals

30

Essential properties ::: Conformity

Software is part of a larger system (domain) that consists of:

- Hardware
- Users
- Stakeholders
- Other systems

The software glues it all together, adding even more complexity

Changes in the domain force software change

Strategies: domain knowledge

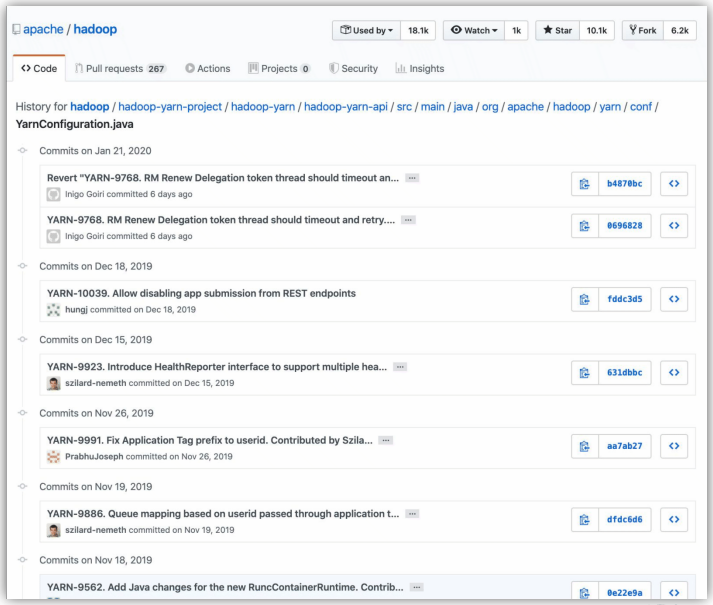
31

Essential properties ::: Changeability

Software is easy to change

- Ill-prepared or hasty changes are not uncommon

Yesterday's comprehension may be obsolete



32

Essential properties ::: Invisibility

Software belongs to the category of abstract concepts

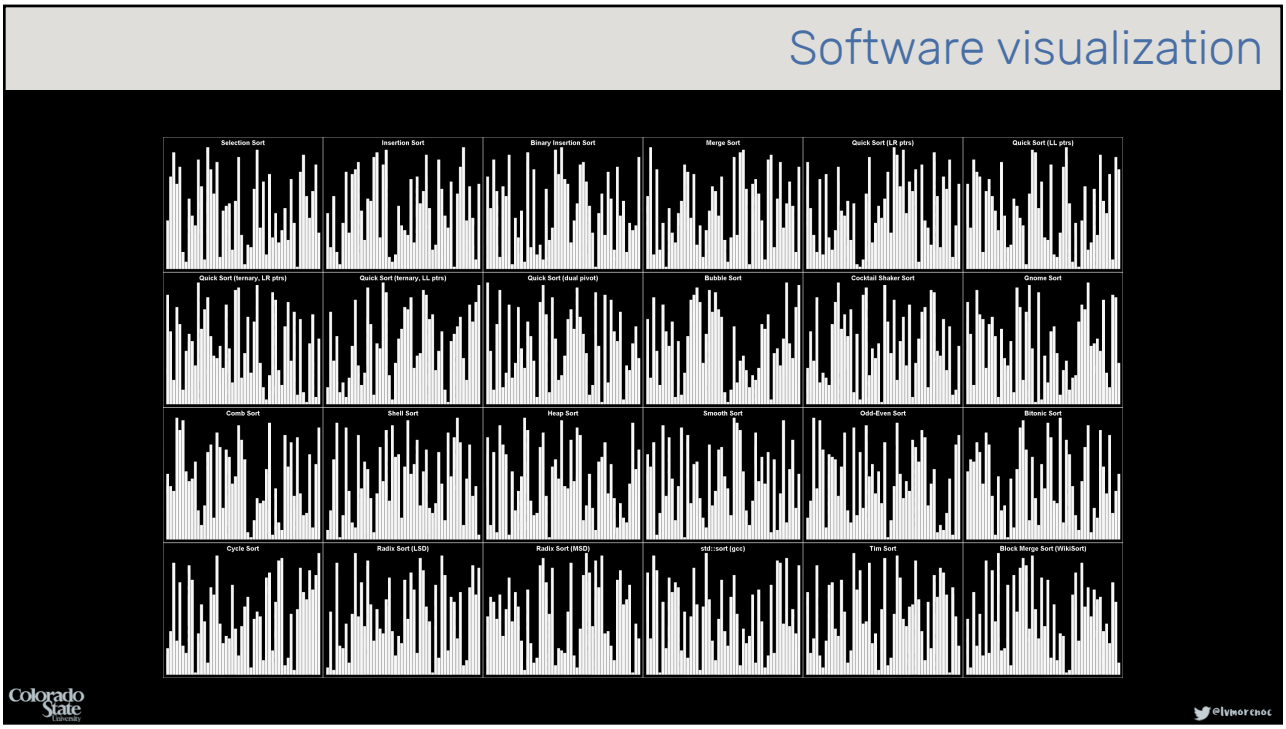
- Senses cannot be easily used in comprehension

Strategies to deal with invisibility (require lots of work)

- Visualization
- Sonification

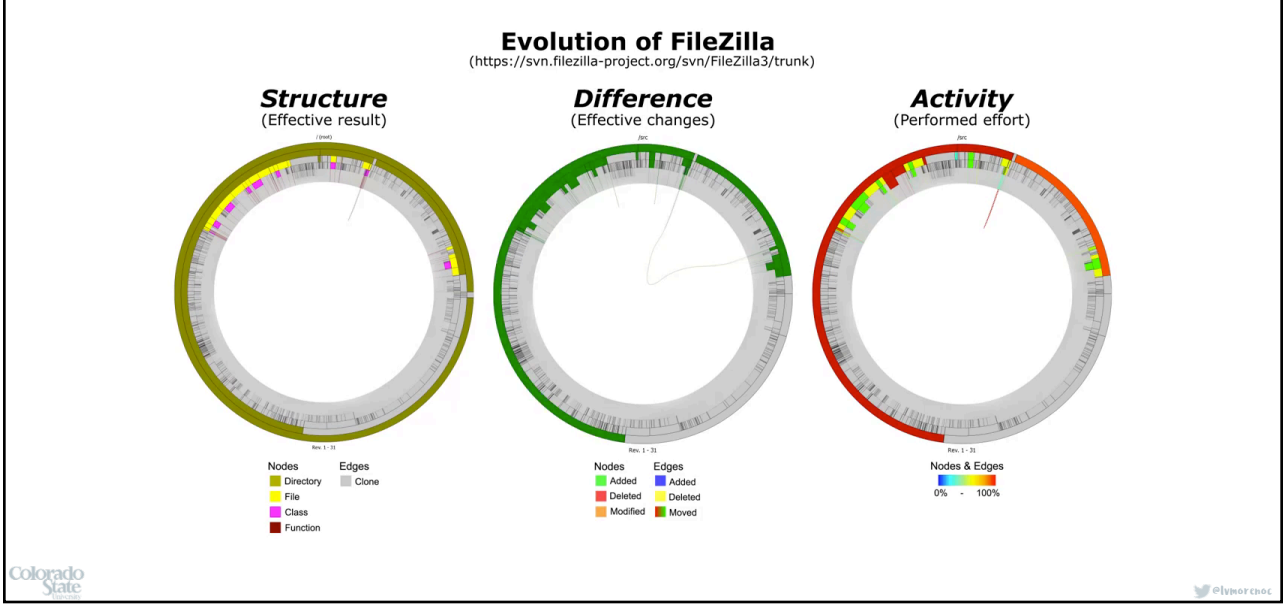
33

Software visualization



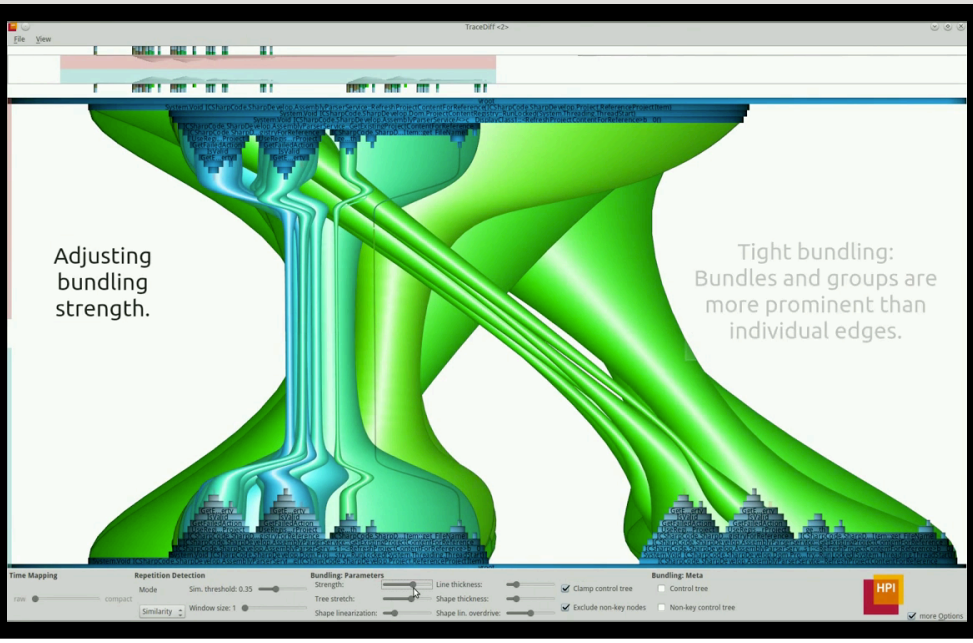
34

Software visualization



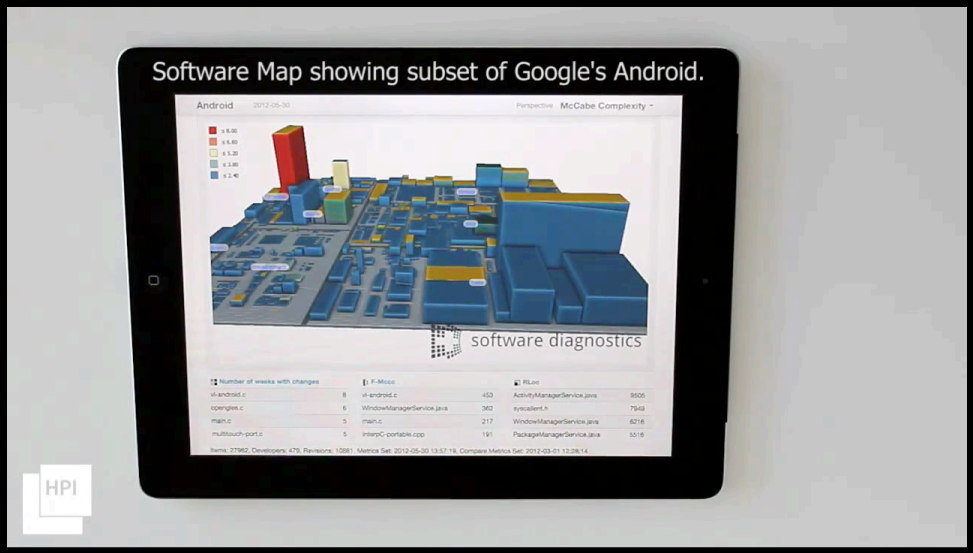
35

Software visualization



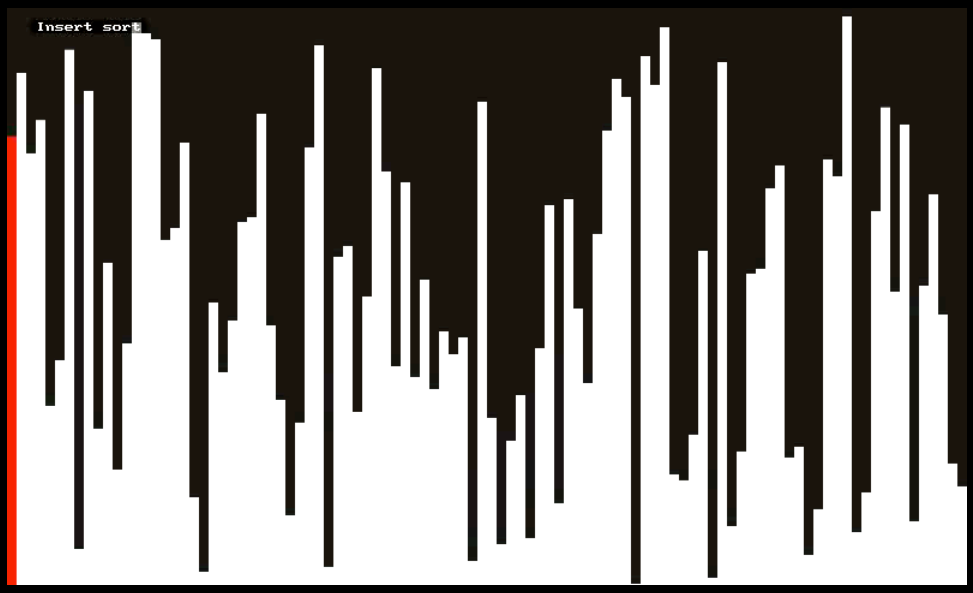
36

Software visualization



37

Software sonification

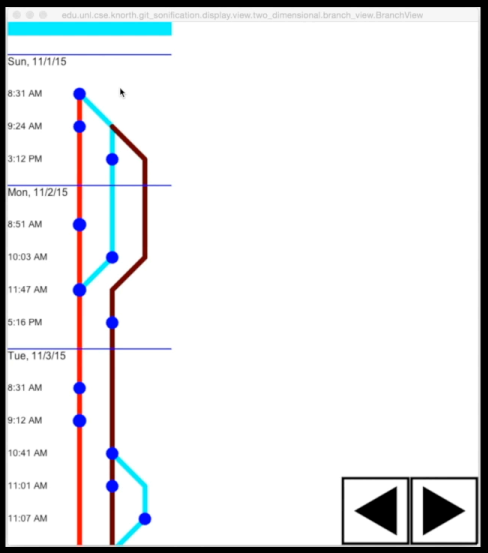


Colorado State University

@lvmorchoc

38

Software sonification



Colorado State University

@lvmorchoc

39

Essential properties ::: Discontinuity

Understanding of continuous semi-linear systems comes easier, but software is discontinuous

In discontinuous systems, a small change in the input can result in a huge change in the output

40

Some accidental difficulties

Past / current / future software technologies

Quirks of
- operating systems
- compilers
- programming languages
- processes

Hardware speed, memory size

Programming paradigm
- functional
- object oriented
- aspect oriented

Role of the software (critical, noncritical)

41

Accidental difficulties and solutions

Accidental difficulty solved	Breakthrough	Impacts
Slow turnaround of batch programming Development interruptions	Time sharing	Productivity
Low-level constructs proper of concrete machine programming	High-level programming languages	Productivity, reliability, simplicity, comprehensibility
Use of individual tools	Unified programming environments	Productivity, maintainability

42

Other solutions to accidental difficulties

- AI – expert systems
- Recommendation systems
- Automatic programming
- Graphical programming
- Formal methods – program verification
- …

43

Reading

Brooks, Frederick P., "No Silver Bullet: Essence and Accidents of Software Engineering," Computer, Vol. 20, No. 4 (April 1987) pp. 10-19