Basic concepts and preliminaries

CS 515, Spring 2020

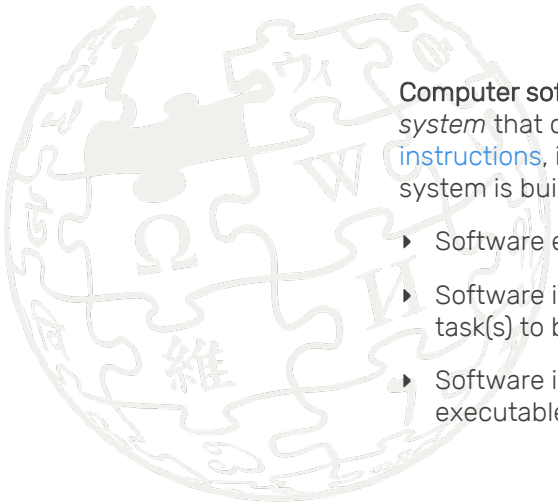
Laura Moreno
lmorenoc@colostate.edu

Colorado State University

@lymorenoc

1

What is software?



Computer software, or simply **software**, is that part of a *computer system* that consists of encoded information or computer instructions, in contrast to the physical *hardware* for which the system is built

- Software enables a computer to perform a specific task
- Software is a collection of instructions that describe a (set of) task(s) to be carried out by a computer
- Software includes computer programs, libraries and related non-executable data, such as documentation or digital media

Colorado State University

@lymorenoc

2

Some facts about software

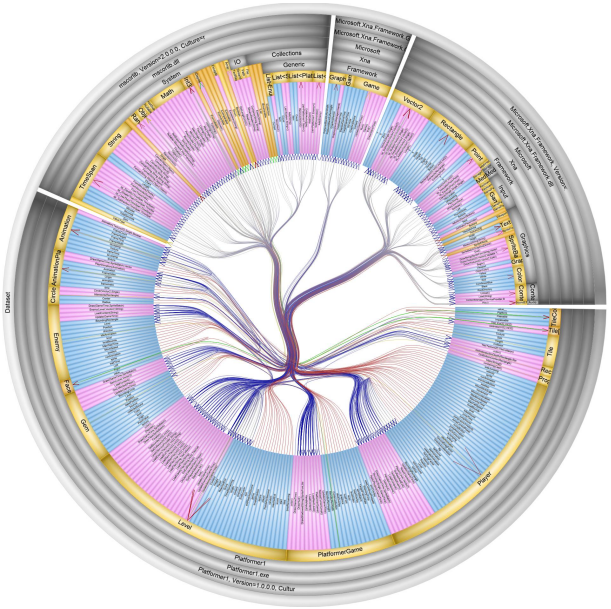
Society increasingly depends on software
… but it is often unreliable and of low quality

Software is regarded like a classical engineering product
… but it is more complex than any other human artifact

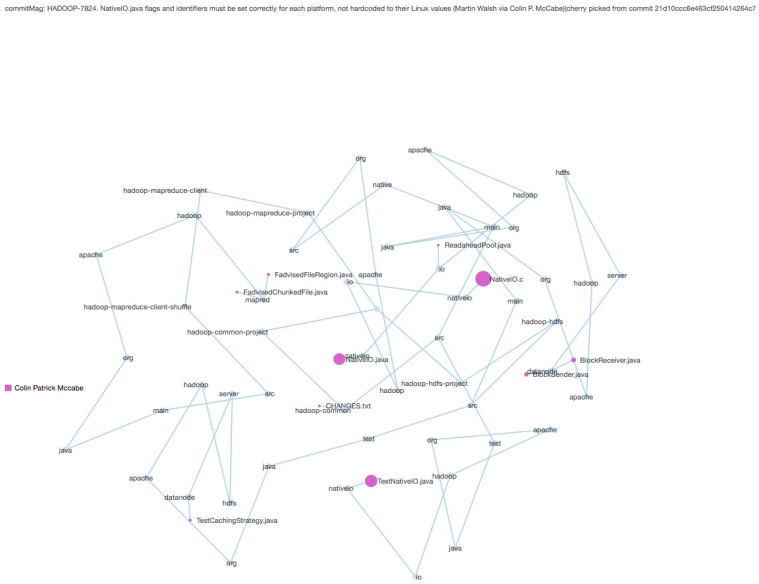
Maintenance is regarded like a lowly activity
… but 75% - 95% of cost is spent on maintenance

Software evolves due to business & technology drivers
… systems that do not change are *dead*

More facts: software is complex



More facts: software evolves!



Colorado State University

@lymorchoc

5

(mis)conceptions

Colorado State University

@lymorchoc

6

Some history…

1965	The concept *software evolution* was introduced by Halpern
1970	"Maintenance engineers" or "maintainers" at IBM
1972	*The Maintenance 'Iceberg'* by Canning
1976	Corrective, adaptive, and perfective maintenance by Swanson
1976-1980	*Laws of software evolution* by Beladi and Lehman
2000	Software maintenance and evolution landscape by Bennett and Rajlich
2003	Further distinction between maintenance and evolution by Bennett and Xu

Colorado State

@lymorchoc

7

Software maintenance vs. software evolution

The terms are used interchangeably but there is a semantic difference

According to Bennett and Rajlich (2000):

- "Software maintenance means to preserve from failure or decline"
- "Software evolution means a continuous change from lesser, simpler, or worse state to a higher or better state"

According to Bennett and Xu (2003):

- Software maintenance refers to "all forms of post-delivery support"
- Software evolution refers to "perfective changes, i.e., those driven by changes in requirements"

Colorado State

@lymorchoc

8

… so what's the difference?

Maintenance	Evolution
▸ Fixing behavior but preserving functionality, improving design to facilitate future change	▸ Supporting new functionality, improving performance, and supporting other environments
▸ Does not involve major changes to the architecture	▸ Creating new but related designs from existing ones
▸ Tasks are mostly planned	▸ "Over time what changes is not the software but our knowledge about a particular type of software" (Jazareki)

Colorado State University

@lymorchoc

9

Why the confusion?

Considering maintenance as another activity in software development

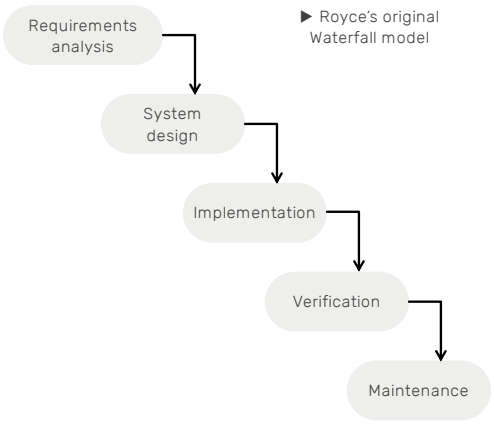
- Maintenance is the last phase of Waterfalll (Royce)
  - "The traditional view of the software life cycle has done a disservice to maintenance by depicting it solely as a single step at the end of the cycle" (Schneidewind)
- In iterative models, maintenance and evolution activities are closely intertwined

Software maintenance should have its own life cycle

- Understanding the code
- Modifying the code
- Revalidating the code

The *staged model of software* (Rajlich and Bennett) makes a clear distinction between evolution and maintenance

- Initial development, evolution, servicing, phaseout



▶ Royce's original Waterfall model

Colorado State University

@lymorchoc

10

Onto software evolution

Real software systems (aka E-type systems) mimic real-world business processes

Real-world business processes change

- passively through adaptions to new constraints (new laws, market situations, introduction of the Euro, etc.)
- actively through the introduction of new products and processes (Business Process Reengineering)

Software changes because the world changes

Colorado State

@lymorchoc

11

Lehman's laws of evolution

A program that is used in a real-world environment must change or become progressively less useful in that environment.

Lehman's First Evolution Law, 1980

1. Continuous change
2. Increasing complexity
3. Self-regulation
4. Conservation of organizational stability
5. Conservation of familiarity
6. Continuous growth
7. Declining quality
8. Feedback system

Colorado State

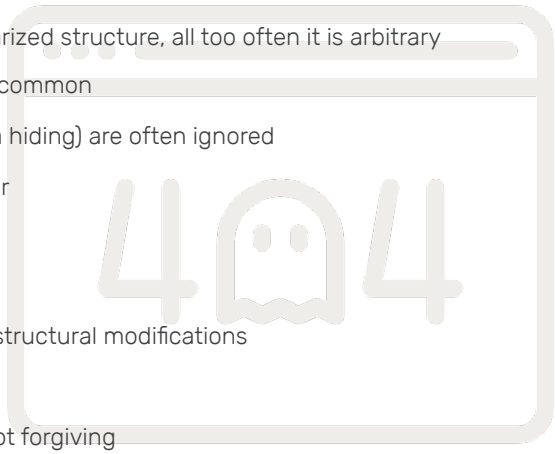
@lymorchoc

12

Onto software maintenance

Defects exist because software defect removal and quality control are far from perfect

- Programs seldom have a meaningfully modularized structure, all too often it is arbitrary
- Duplication of both data and functionality are common
- Golden, basic principles (e.g., abstraction, data hiding) are often ignored
- Changes to the code modify run-time behavior
- Sparse test suites, if at all
- Documentation? What?
- Functional modifications do not always imply structural modifications
- Bad names (int a = 0; method m1, m2, m3)
- Programming languages are permissive but not forgiving



Colorado State

@lymorchoc

13

Maintenance vs. development

Maintenance is like software development, ***but*** some unique skills and processes are required:

- Intimate knowledge of system structure and content
- Impact analysis and ripple effects
- Problem solving skills ("Programmers have become part historian, part detective, and part clairvoyant", Corbi, 1989)
- Change tracking and control
- Maintenance outsourcing
- Maintenance cost estimation

Colorado State

@lymorchoc

14

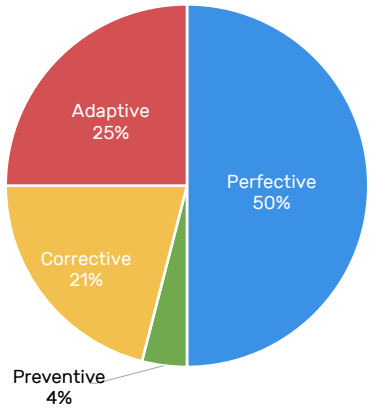
Types of software maintenance

Corrective: "bug fixing"

Preventive: finding mistakes before they show up, "good design"

Adaptive: adapting to new hardware, operating systems, requirements

Perfective: ameliorating performance and maintainability (restructuring, reverse engineering, reengineering re-documentation, etc.)



15

Why is maintenance hard?

Time pressure

Increasing software entropy

Determining the right change

Legacy systems (and their perks)

Little know-how, no tools

…

16

Software maintenance standards

Well-defined processes for software maintenance can be observed and measured, and thus improved

Process centric software maintenance is more of an engineering activity and less of an art

Standards for software maintenance

- ISO/IEC 14764: process implementation, problem and modification analysis, modification implementation, maintenance review/acceptance, migration, and retirement
- IEEE/EIA 1219: problem identification, analysis, design, implementation, system test, acceptance test, and delivery

17

Related concepts

18

Software configuration management (SCM)

SCM is the means by which the process of software evolution is managed

"Discipline of managing and controlling change in the evolution of software systems" (IEEE 1042)

SCM provides a framework for managing changes in a controlled manner

- To reduce communication errors among personnel working on different aspects of the software project
- To ensure that the released software is not contaminated by uncontrolled or unapproved changes.

SCM tools

- The early ones had limited capabilities in terms of functionality and applicability
- The modern ones provide advanced capabilities through which many different artifacts are managed
  - Identification, control, auditing, and accounting of software configurations

19

Reengineering

Software evolution is "the process of conducting continuous software reengineering" (Yang and Ward)

- Reengineering implies a single cycle of taking an existing system and generating from it a new system
- Evolution can go forever
- Reengineering transforms an existing "lesser or simpler" system into a new "better" system

Reengineering = Reverse engineering + Δ + Forward engineering

- *Reverse engineering:* defining a more abstract and easier to understand representation of the system
  - Lexical analysis, syntactic analysis, control flow analysis, etc.
  - *Goals/models/tools* (Benedusi et al.): Organizational paradigm for reverse engineering
- *Forward engineering:* moving from a high-level abstraction and logical, implementation-independent design to the physical implementation of the system
- *Δ:* captures alterations performed to the original system

20

Legacy systems

Old programs that continue to be used because they still meet the users' needs

Legacy systems are often vital to their organizations

- Yet, their code is difficult to understand
- They resist modification and evolution to meet new and constantly changing business requirements

Options to manage legacy systems

- *Freeze:* No further work is done on the system because its services are no longer needed, or a new system has completely replaced them
- *Outsource:* Supporting legacy software is not the core business, thus support service is outsourced to a specialist organization
- *Carry on maintenance:* The system is maintained for some more time, despite all the difficulties
- *Discard and redevelop:* The system is redeveloped from scratch, using new technologies
- *Wrap:* The system is wrapped around with a new software layer that hides unwanted complexity, but it still performs the actual computations
- *Migrate:* The system is moved to a new hardware and/or software platform, while still retaining the legacy system's functionality

Colorado State

@lymorchoc

21

Impact analysis

Task of identifying parts of the software that can potentially be affected if a proposed change is implemented

- Useful when planning for changes, making changes, and tracking the effects of changes

Impact analysis techniques

- *Traceability analysis:* High-level artifacts related to the feature to be changed are identified
- *Dependency analysis:* Assess the effects of a change on the semantic dependencies between program entities

Related notions

- *Ripple effect analysis:* Measures the impact of a change to a particular module on the rest of the program
- *Change propagation:* Ensures that a change made in one component is propagated properly throughout the entire system

Colorado State

@lymorchoc

22

Refactoring

Changing the structure of software to make it easier to comprehend and cheaper modify without changing the observable behavior of the system

- Preservation of the observable behavior is verified by ensuring that all the tests passing before refactoring must pass after refactoring

Without refactoring, the internal structure of software will eventually decay

In agile methodologies, refactoring is continuously applied

23

Program comprehension

The purpose of program comprehension is to understand an existing software system

It accounts for 50% of the effort invested in the life cycle of a software system (Corby, 1989)

Good understanding of the software is key to raising its quality

It involves building mental models of system at different levels of abstraction

- Mental model: a programmer's mental representation of a program

Mental models are built by applying cognitive processes

- Generating hypotheses and investigating their validity
- Strategies to arrive at meaningful hypotheses: bottom-up, top-down, and opportunistic

24

Software reuse

Using existing software knowledge or artifacts during the development of a new system

- Reuse is not constrained to source code fragments

Broad types of artifacts for reuse (Jones):

- Data reuse
- Architectural reuse
- Design reuse
- Program reuse

Reusability indicates the degree to which the software can be reused

- For a software component to be reusable, it must exhibit high cohesion, low coupling, adaptability, understandability, reliability, and portability

Code reuse leads to better quality and increased productivity

25

References

Tripathy, P., and Naik, K., Software Evolution and Maintenance: A Practitioner's Approach. 1st edition. 2014. Wiley. ISBN-13: 978-0470603413.

26