# Understanding the Challenges of Legacy Projects

Week 1and 2

Dr. Fatima Sabir



**CSE -350 Special Topics in SOftware Engineering**

# What are the legacy Software Systems

- In Oxford English Dictionary (OED) the word “legacy” is defined as
  - A sum of money, or a specified article, given to another by will; anything handed down by an ancestor or predecessor.
  - Legacy software systems are large software systems that we don’t know how to cope with but that are vital to our organization.
- Supporting the definitions are a set of acceptable features of a legacy system:
  - large with millions of lines of code.
  - geriatric, often more than 10 years old.
  - written in obsolete programming languages.
  - lack of consistent documentation.
  - poor management of data, often based on flat-file structures.
  - degraded structure following years of modifications.
  - very difficult, if not impossible, to expand.
  - runs on old processor.

# 1.1 Legacy Projects

As developers, we tend to focus on the code, but a project encompasses many other aspects, including

- Build tools and scripts
- Dependencies on other systems
- The infrastructure on which the software runs
- Project documentation
- Methods of communication, such as between developers, or between developers
- and stakeholders

# Characteristics of legacy projects

1. Old
2. Large
3. Inherited
4. Poorly Documented
5. Using Obsolete and Conventional Methodologies
6. Without any standardization

# Example

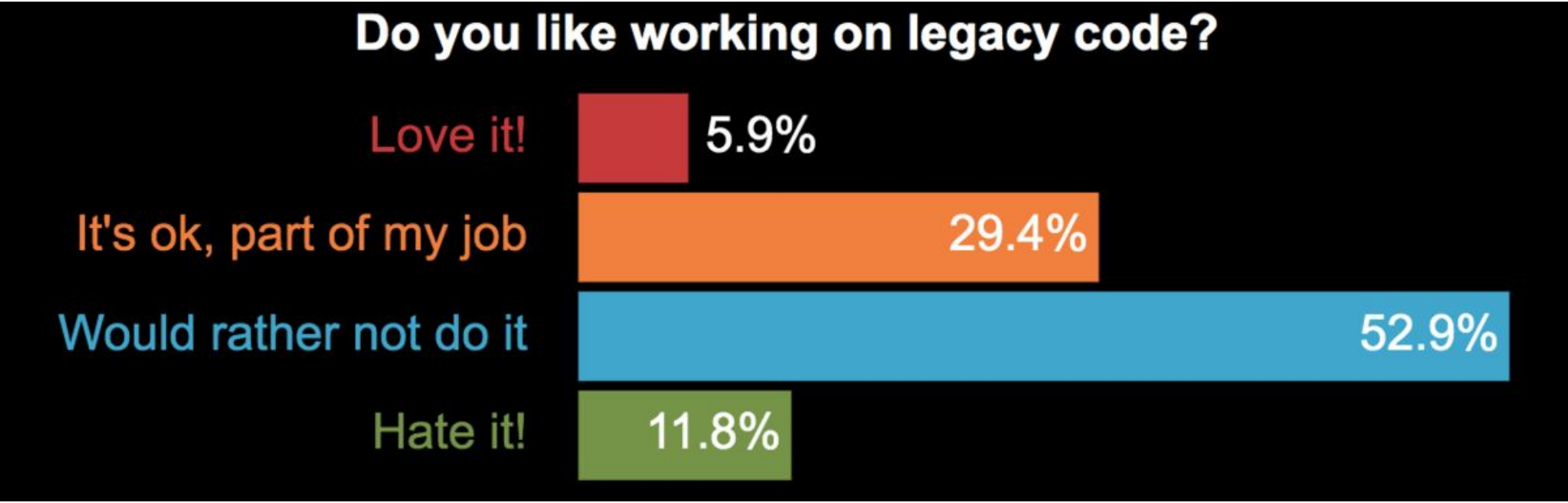
The Coverity report is available at http://wpcme.coverity.com/wp-content/uploads/2012-Coverity-Scan-Report.pdf.

I think the primary reason for Linux's continued success as a software project Is,

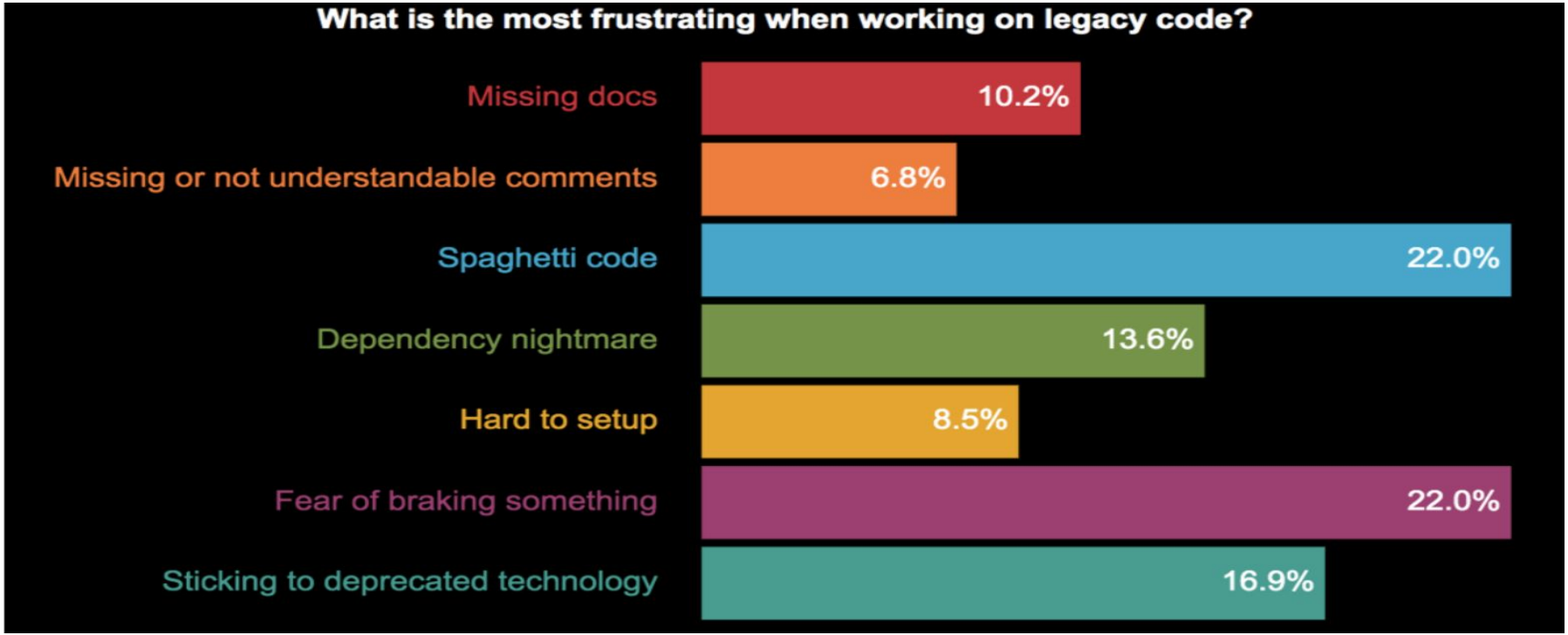
1. culture of open and frank communication
2. incoming changes are thoroughly reviewed
3. information-sharing
4. code review.
5. Bugs location and correctness

code review in an interview with LWN in 2008 (https://lwn.net/Articles/285088/)

# Legacy Code


Do you like working on legacy code?
Love it!
5.9%
It's ok, part of my job
29.4%
Would rather not do it
52.9%
Hate it!
11.8%

# What is so Hard?



# 1.1 What is so Hard about Legacy Code ?

a) Un-tested or Un-testable Code
b) Inflexible code
c) Code encumbered by technical debt

# A ) Problems with Un-Tested Code

- Non existent or Unreliable Technical documentation for software projects
- Tests are often the best place to look for clues about the system's behavior and design assumptions.
- A good test suite can function as the de facto documentation for a project.
- A socially responsible developer will take care to fix any tests that were broken by their changes
- Legacy projects are usually written without testing in mind, so retroactively adding tests is extremely difficult.

# Solution

1. Best Possibility to use Mock Downloader
2. Junit
3. Checking Function Call using OOP Structure
4. Save any existing value of the system property.
5. Set the system property to the value you want.
6. Run the test.
7. Restore the system property to the value that you saved. You must make sure to do this even if the test fails or throws an exception.

# b) In Flexible Code

Implementing new features or changes to existing behavior is inordinately difficult.

```
public void deleteWibble(Wibble wibble)
                    throws NotAuthorizedException {
    if (!loggedInUser.isAdmin()) {
        throw new NotAuthorizedException(
            "Only Admins are allowed to delete wibbles");
    }
        ...
}
```

```
public void deleteWibble(Wibble wibble)
                    throws NotAuthorizedException {
    if (!(loggedInUser.isAdmin() || loggedInUser.isPowerUser()) {
        throw new NotAuthorizedException(
            "Only Admins and Power Users are allowed to delete wibbles");
    }
        ...
}
```

# 1.2 Legacy Infrastructure

a) Development environment
b) Outdated Dependencies
c) Heterogeneous Environment

# a) Development Environment

- ❏ View and edit the code in your IDE
- ❏ Run the unit and integration tests
- ❏ Run the application on your local machine

**Setting up a legacy project often involves a combination of** ,

- Downloading, installing, and learning how to run whatever arcane build tool the project uses
- Running the mysterious and unmaintained scripts you found in the project's /bin folder
- Taking a large number of manual steps, listed on an invariably out-of-date wiki page

# b) Heterogeneous Environment

1. Developers run the software on their local machines.
    a. Upgrades trickle down from production
    b. Different Tools in Different Environment
2. They deploy it to a test environment for automatic and manual testing.
3. It is deployed to a staging environment that mimics production as closely as Possible.
4. It is released and deployed to the production environment (or, in the case of packaged software, it’s shipped to the customer).

# c) Legacy Culture

Fear of Change

Knowledge Soils

# 1.3 Fear of Change

**Table 1.1 Changes, benefits and risks for a legacy project**

| Changes | Benefits | Risks | Actions required |
|---|---|---|---|
| Removing an old feature | · Easier development<br>· Better performance | · Somebody is still using the feature | · Check access logs<br>· Ask users |
| Refactoring | · Easier development | · Accidental regression | · Code review<br>· Testing |
| Upgrading a library | · Bug fixes<br>· Performance improvements | · Regression due to a change in the library's behavior | · Read the changelog<br>· Review library code<br>· Manually test major features |

# 1.4 Knowledge Soils

1. Domain information about the users' requirements and the functional specifications of the software
2. Project-specific technical information about the software's design, architecture,
3. Internals general technical knowledge such as efficient algorithms, advanced language features, handy coding tricks, and useful libraries

# Go A head —----

- Create & view code issues directly from your editor
- Track & prioritise code improvements like technical debt
- Add key issues to your sprints with our Jira integration
- guide to legacy software  migration
- Stepsize VS Code and JetBrains extensions for Technical Debt
- Use Statis Analysis Tool Code sonar ,CAST ,Klocwork ,SonarQube

# Class Work

Explore your project and find all options discussed in lecture

# References


1. Miller, H. W. (2016). *Reengineering legacy software systems*. Digital Press.
2. The Engineer's Complete Guide to Legacy Code - Stepsize by CATE LAWRENCE
3. Sommerville, I. (2011). *Software Engineering, 9/E*. Pearson Education India.