[CS3704] Software Engineering

Dr. Chris Brown

Virginia Tech

11/8/2023

Announcements

- PM3 due Friday (11/10) at 11:59pm

Maintenance

Maintainability
Refactoring
Debugging
Code Reviews

Learning Outcomes

By the end of the course, students should be able to:

- **Implement a software system following the software life cycle phases**
- **Develop software engineering skills working on a team project**
- **Identify processes related to phases of the software lifecycle**
- **Explain the differences between software engineering processes**
- **Discuss research questions and studies related to software engineering**
- **Communicate (via demo and writing) details about a developed software application**

Deployment/Maintenance

Goal: release, upgrade, and fix the software

- **Maintenance:** The process of changing, modifying, and updating software to keep up with customer needs.
- When software is completed, it must be *deployed* or delivered to the customer. Additionally, software must be *maintained* such that user problems are addressed after operationalization.
- *Software Artifacts:* All!

Deployment/Maintenance (cont.)

- After you complete the implementation and testing phases, what’s next?
  - Both occur after the product is in full operation
  - Software requires continual maintenance to ensure the program operates correctly and frequent deployment to make updates available to users.
- **TODO:** Discuss a time when you had to maintain older software or deliver software to users? How did it go?

Challenges of Large Code Bases

How can you ensure…

- Maintainable code?
- Reuseable code?
- Readable code?
- Bug-free code?

Average defect detection rate for various types of testing

- Unit testing: 25%
- Function testing: 35%
- Integration testing:45%

How can this be improved?

Maintainability

- The ease with which existing software can be corrected, adapted, or enhanced *after delivery*. [Pressman]
- **Maintenance:** The process of changing, modifying, and updating software to keep up with customer needs.
  - i.e. debug, correct faults, improve the performance, versioning, deployment, etc.
- **Supportability:** the capability of supporting a software system over its whole product life [Pressman]

Why is Maintenance Important?

- Creating new software and deploying it is exciting, however great software must evolve over time.
- Most of software engineering resources, time, money, and effort is consumed in tasks necessary to maintain a software product.
- Without maintenance, software will become obsolete and essentially useless

How to write maintainable code?

- Still an open research questions

Some tips:

- Design!
- Testing!
    - Unit tests
    - Integration tests
- Documentation

Why is Maintenance Important? (cont.)

Well-maintained software allows for:

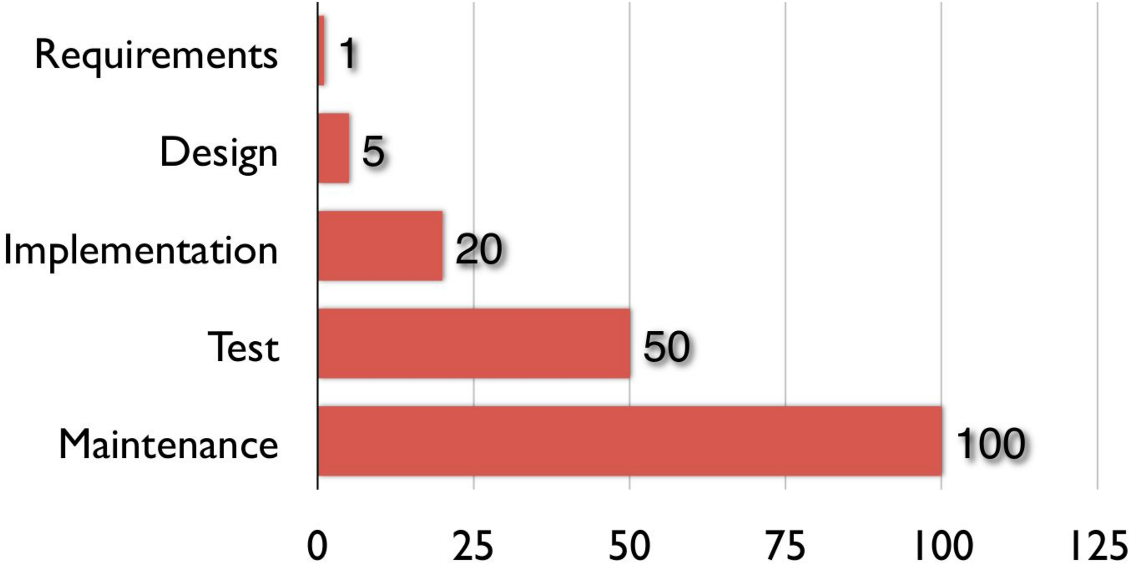
- Better planning
- Improved design
- Less volatile project costs
- Managed products
- Limited bugs
- Faster bug fixes
- Increased productivity
- …

Problem

The cost of software maintenance is high!

- Can take up to two-thirds of budget and over 50% of software development process.
- The older the software, the more maintenance will cost

- Much cheaper and easier to modify software and fix bugs earlier in the SDLC.



Why is maintenance so hard?

Code Modifications

- Software changes over time.
  - Changes to code base, bug fixes, new features, etc.

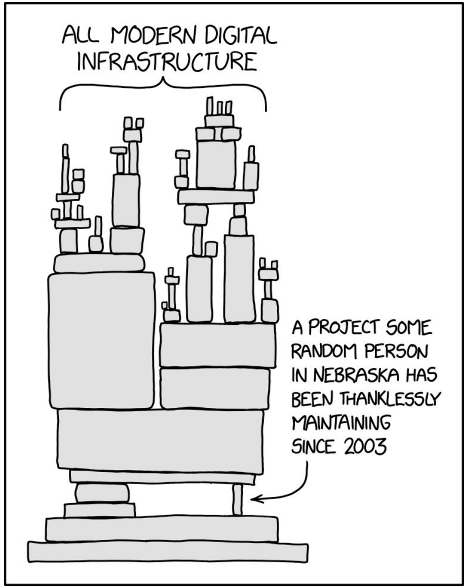
Technological Advances

- Becomes increasingly complex over time.
  - New hardware and software paradigms for compatibility (Smartphones, VR headsets, cloud,...)
  - Greater expectations for non-functional requirements (performance, speed, memory, etc.)

Why is maintenance so hard? (cont.)

Dependencies

- Software is not created in isolated silos…


ALL MODERN DIGITAL INFRASTRUCTURE
A PROJECT SOME RANDOM PERSON IN NEBRASKA HAS BEEN THANKLESSLY MAINTAINING SINCE 2003

Why is maintenance so hard? (cont.)

Maintainability is a mystery

- Subjective concept

“In order to make software architecture and components more maintainable, mysteries of maintainability have to be solved.”

[Molhotra]

Types of Software Maintenance

1. Corrective
2. Preventative
3. Perfective
4. Adaptive

[Thales]

Corrective Software Maintenance

- Fixing something that goes wrong in a piece of software.
    - Ex.) Finding faults and errors

** This is the most common form of software maintenance!*

Preventative Software Maintenance

- Predicting changes to keep software working for as long as possible.
    - Ex.) Upgrades, bug prevention, security updates, latent faults
    - **Latent faults:** Small issues that may lack significance at a given time, but may turn into larger problems in the future

Perfective Software Maintenance

- Improving software to adjust to user needs and stay relevant on the market.
  - Ex.) Adding new features, removing less effective features

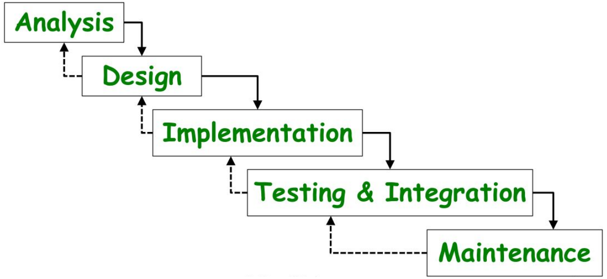
Adaptive Software Maintenance

- Changing non-functional aspects of the system related to your software.
    - Ex.) Compatibility on new operating systems, cloud storage, hardware, policies and rules for usage

How Maintenance Relates to SE

Software maintenance motivates many of the concepts we’ve discussed so far in class!

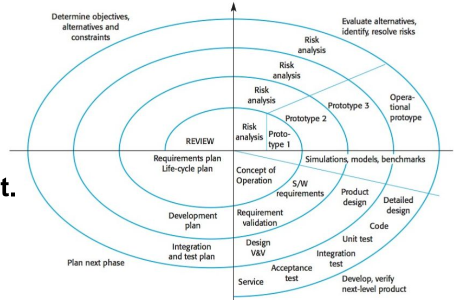
Ex.) ***Processes:*** One motivation for introducing SE processes, and moreso iterative processes, is the ability to improve maintenance.



Stand-up Meetings

TODO: Another scrum meeting!

What I did.
What I need to do next.
What is blocking me.



How Maintenance Relates to SE (cont.)

Ex.) ***Design Patterns:*** One of the main benefits of design patterns is **maintainability!** (better organization, easier search, faster debugging,...)

Ex.) ***Metrics and Analysis:*** Code metrics and analysis tools increase awareness for need and help support software maintenance work.

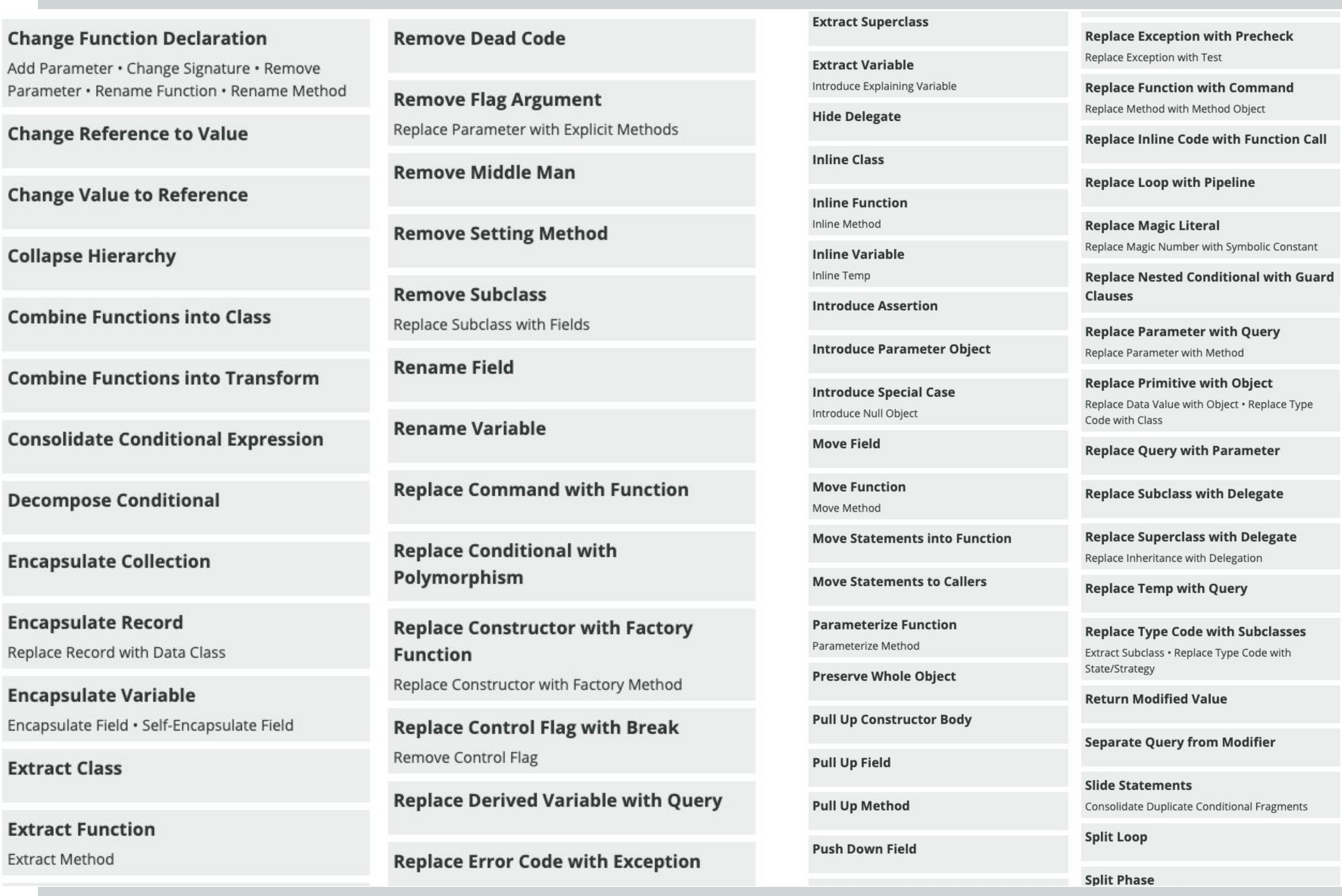
Ex.) ***Testing:*** Good tests find errors

Requirements Analysis, Code Reviews, etc.

Refactoring

The process of modifying code for improvement *without* changing the underlying functionality and behavior of the program.

- Refactoring is not just modifying code
- There are many different types of refactoring



Change Function Declaration
Add Parameter • Change Signature • Remove Parameter • Rename Function • Rename Method

Change Reference to Value

Change Value to Reference

Collapse Hierarchy

Combine Functions into Class

Combine Functions into Transform

Consolidate Conditional Expression

Decompose Conditional

Encapsulate Collection

Encapsulate Record
Replace Record with Data Class

Encapsulate Variable
Encapsulate Field • Self-Encapsulate Field

Extract Class

Extract Function
Extract Method

Remove Dead Code

Remove Flag Argument
Replace Parameter with Explicit Methods

Remove Middle Man

Remove Setting Method

Remove Subclass
Replace Subclass with Fields

Rename Field

Rename Variable

Replace Command with Function

Replace Conditional with Polymorphism

Replace Constructor with Factory Function
Replace Constructor with Factory Method

Replace Control Flag with Break
Remove Control Flag

Replace Derived Variable with Query

Replace Error Code with Exception

Extract Superclass

Extract Variable
Introduce Explaining Variable

Hide Delegate

Inline Class

Inline Function
Inline Method

Inline Variable
Inline Temp

Introduce Assertion

Introduce Parameter Object

Introduce Special Case
Introduce Null Object

Move Field

Move Function
Move Method

Move Statements into Function

Move Statements to Callers

Parameterize Function
Parameterize Method

Preserve Whole Object

Pull Up Constructor Body

Pull Up Field

Pull Up Method

Push Down Field

Replace Exception with Precheck
Replace Exception with Test

Replace Function with Command
Replace Method with Method Object

Replace Inline Code with Function Call

Replace Loop with Pipeline

Replace Magic Literal
Replace Magic Number with Symbolic Constant

Replace Nested Conditional with Guard Clauses

Replace Parameter with Query
Replace Parameter with Method

Replace Primitive with Object
Replace Data Value with Object • Replace Type Code with Class

Replace Query with Parameter

Replace Subclass with Delegate

Replace Superclass with Delegate
Replace Inheritance with Delegation

Replace Temp with Query

Replace Type Code with Subclasses
Extract Subclass • Replace Type Code with State/Strategy

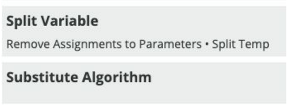
Return Modified Value

Separate Query from Modifier

Slide Statements
Consolidate Duplicate Conditional Fragments

Split Loop

Split Phase

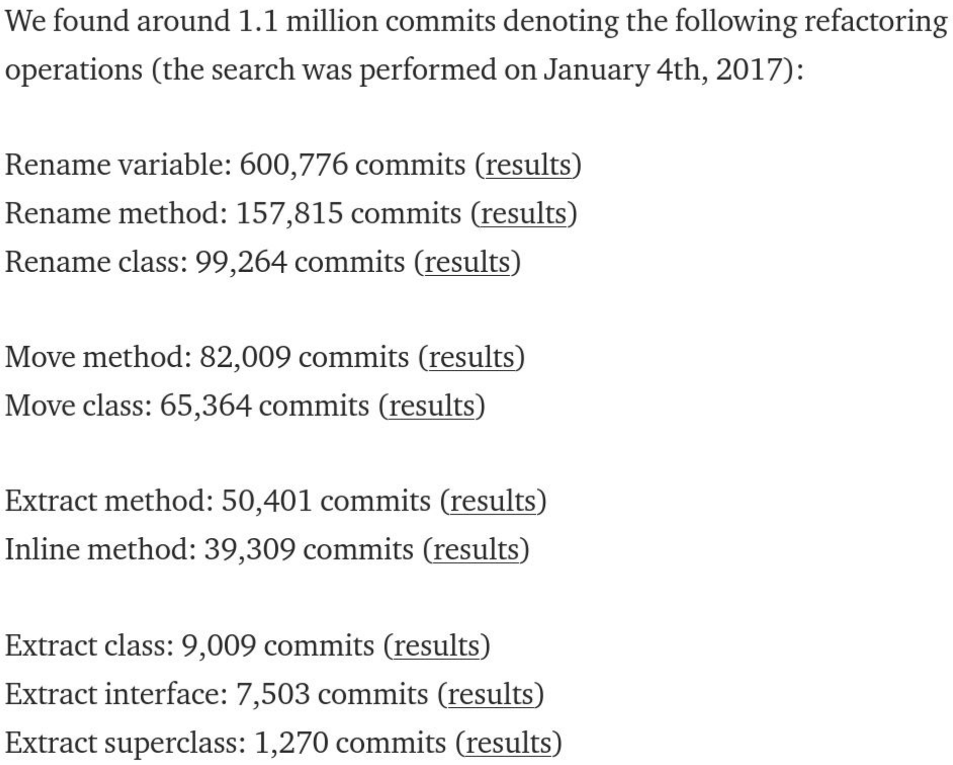


Split Variable
Remove Assignments to Parameters • Split Temp

Substitute Algorithm

72 refactorings introduced ed by Fowler et al.

Most Common Refactoring



We found around 1.1 million commits denoting the following refactoring operations (the search was performed on January 4th, 2017):

Rename variable: 600,776 commits (results)
Rename method: 157,815 commits (results)
Rename class: 99,264 commits (results)

Move method: 82,009 commits (results)
Move class: 65,364 commits (results)

Extract method: 50,401 commits (results)
Inline method: 39,309 commits (results)

Extract class: 9,009 commits (results)
Extract interface: 7,503 commits (results)
Extract superclass: 1,270 commits (results)

77%!

[Silva]

Refactoring Example

```
class Gorilla {
  int paws() {
     return 4;
  }
}
```

INTRODUCE EXPLAINING VARIABLE

```
class Gorilla{
  int paws(){
     int pawCount = 4;
     return pawCount;
  }
}
```

EXTRACT INTERFACE

```
class Gorilla implements Primate {
  int paws() {
    int pawCount = 4;
    return pawCount;
  }
}
…
interface Primate {
  abstract int paws();
}
```

RENAME METHOD

```
class Gorilla implements Primate{
  int feet() {
    int pawCount = 4;
    return pawCount;
  }
}
…
interface Primate {
  abstract int feet();
}
```

Why Refactor?

- The idea behind refactoring is to acknowledge that *it will be difficult to get a design right the first time!*
- As a program's requirements change, the design may also need to change.
- Refactoring provides techniques for evolving the design in small incremental steps.

Benefits

- Often code size is reduced after refactoring
- Confusing structures are transformed into simpler structures
    - ***which are easier to maintain and understand***

When to Refactor?

Refactor when you add functionality

- do it before you add the new function to make it easier to add the function
- or do it after to clean up the code after the function is added

Refactor when you need to fix a bug

Refactor as you do a code reviews

When you identify code that needs to be refactored

- **Identify “Bad Smells” in Code**

A Few Bad Code Smells

Duplicated Code

- bad because you modify one instance of duplicated code but not the others; not all versions fixed

Long Method

- long methods are more difficult to understand; performance concerns with respect to lots of short methods are largely obsolete

Message Chains

- a client asks an object for another object and then asks that object for another object etc.
- Bad because client depends on the structure of the navigation

Dispensables

- Pointless or unneeded code whose absence would make code cleaner, more efficient, and easier to understand

…

Floss vs. Root Canal Refactoring

Describes when refactoring is done and why.

The distinction will help you: 🦷

- Identify the tools that will be most useful to you
- Figure out whether your refactoring is “best practice”

[Parnin][Murphy-Hill]

Root Canal Refactoring

Large changes to software code.

- Painful, expensive, the result of long periods of neglect

When: Refactoring for protracted periods; time specifically set aside

Why: Typically after code has gotten difficult to maintain

- Not considered best practice ✗

Floss Refactoring

Smaller and more frequent code changes

- Easier to do, regular, something software engineers know they *should* do

When: Continuously

- As often as “every few minutes”

Why: It helps achieve an immediate goal (to “clean up” or “improve” is not a goal)

- Considered best practice ✅

Code Changes

In addition to refactoring, there are a variety of other processes for modifying source code for a program including:

- Restructuring
- Reengineering

[Pressman]

Restructuring

- Modifying source code and or data to make it amenable to future changes and produce a higher quality application than the original.

What’s the difference from refactoring?

Restructuring vs. Refactoring

- *Restructuring* is general and doesn’t imply particular methods; *Refactoring* consists of very specific transformations
- *Restructuring* is generally larger; *Refactoring* is usually smaller
- *Refactoring* focuses on code; *Restructuring* can deal with code and/or data
- *Refactoring* is small behavior-preserving changes; *Restructuring* can be changes that modify program behavior

Reengineering

- A radical redesign in order to achieve dramatic improvements.
  - **Scenario:** *“An application has served the business needs of a company for 10 or 15 years. During that time, it has been corrected, adapted, and enhanced many times. People approached this work with the best intentions…Now the application is unstable. It still works, but every time a change is attempted, unexpected and serious side effects occur. Yet the application must continue to evolve. What to do?”* [Pressman]
  - Obliterate it and start over

Reengineering Activities

Reengineering is a rebuilding activity. Before rebuilding you should:

1. **Inspect to determine if it is in need of rebuilding**
    a. List of criteria
    b. Is it possible to “re-model” instead of rebuild
2. **Understand how the original was built**
3. **Be disciplined about rebuild**
    a. Use more in-depth materials and processes (may cost more now, but prevent expenses and time later)
    b. Focus on high-quality now and for the future

Debugging

The process of finding and fixing errors in code.

This is widely regarded as the most expensive and time-consuming part of SE! [Alaboudi]

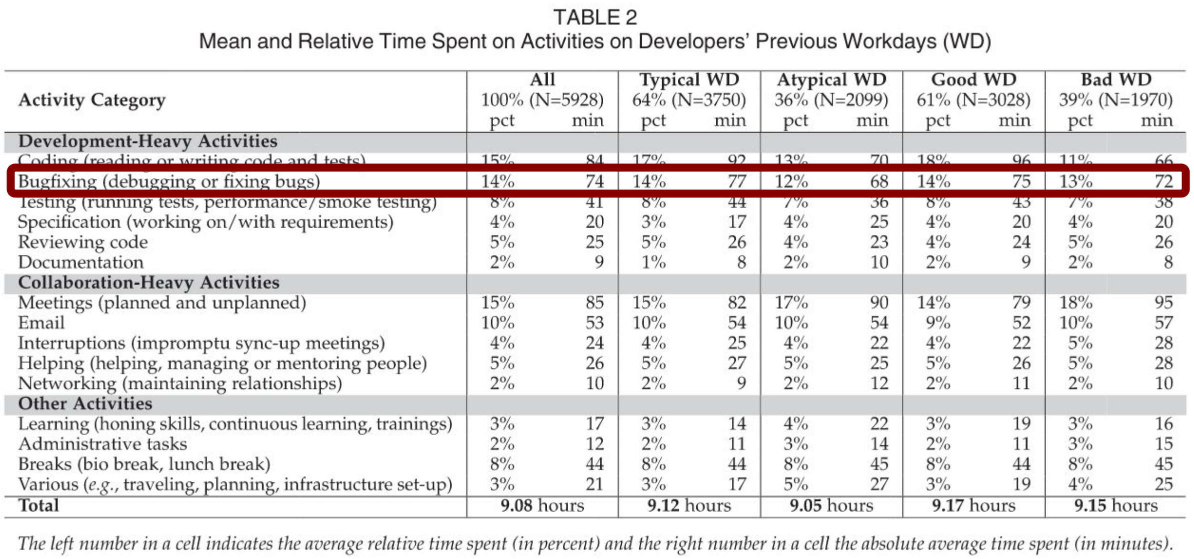


TABLE 2
Mean and Relative Time Spent on Activities on Developers' Previous Workdays (WD)

Activity Category	All 100% (N=5928) pct	 min	Typical WD 64% (N=3750) pct	 min	Atypical WD 36% (N=2099) pct	 min	Good WD 61% (N=3028) pct	 min	Bad WD 39% (N=1970) pct	 min
Development-Heavy Activities										
Coding (reading or writing code and tests)	15%	84	17%	92	13%	70	18%	96	11%	66
Bugfixing (debugging or fixing bugs)	14%	74	14%	77	12%	68	14%	75	13%	72
Testing (running tests, performance/smoke testing)	8%	41	8%	44	7%	36	8%	43	7%	38
Specification (working on/with requirements)	4%	20	3%	17	4%	25	4%	20	4%	20
Reviewing code	5%	25	5%	26	4%	23	4%	24	5%	26
Documentation	2%	9	1%	8	2%	10	2%	9	2%	8
Collaboration-Heavy Activities										
Meetings (planned and unplanned)	15%	85	15%	82	17%	90	14%	79	18%	95
Email	10%	53	10%	54	10%	54	9%	52	10%	57
Interruptions (impromptu sync-up meetings)	4%	24	4%	25	4%	22	4%	22	5%	28
Helping (helping, managing or mentoring people)	5%	26	5%	27	5%	25	5%	26	5%	28
Networking (maintaining relationships)	2%	10	2%	9	2%	12	2%	11	2%	10
Other Activities										
Learning (honing skills, continuous learning, trainings)	3%	17	3%	14	4%	22	3%	19	3%	16
Administrative tasks	2%	12	2%	11	3%	14	2%	11	3%	15
Breaks (bio break, lunch break)	8%	44	8%	44	8%	45	8%	44	8%	45
Various (*e.g.*, traveling, planning, infrastructure set-up)	3%	21	3%	17	5%	27	3%	19	4%	25
Total	**9.08** hours		**9.12** hours		**9.05** hours		**9.17** hours		**9.15** hours	

The left number in a cell indicates the average relative time spent (in percent) and the right number in a cell the absolute average time spent (in minutes).

Debugging (cont.)

Debugging activities

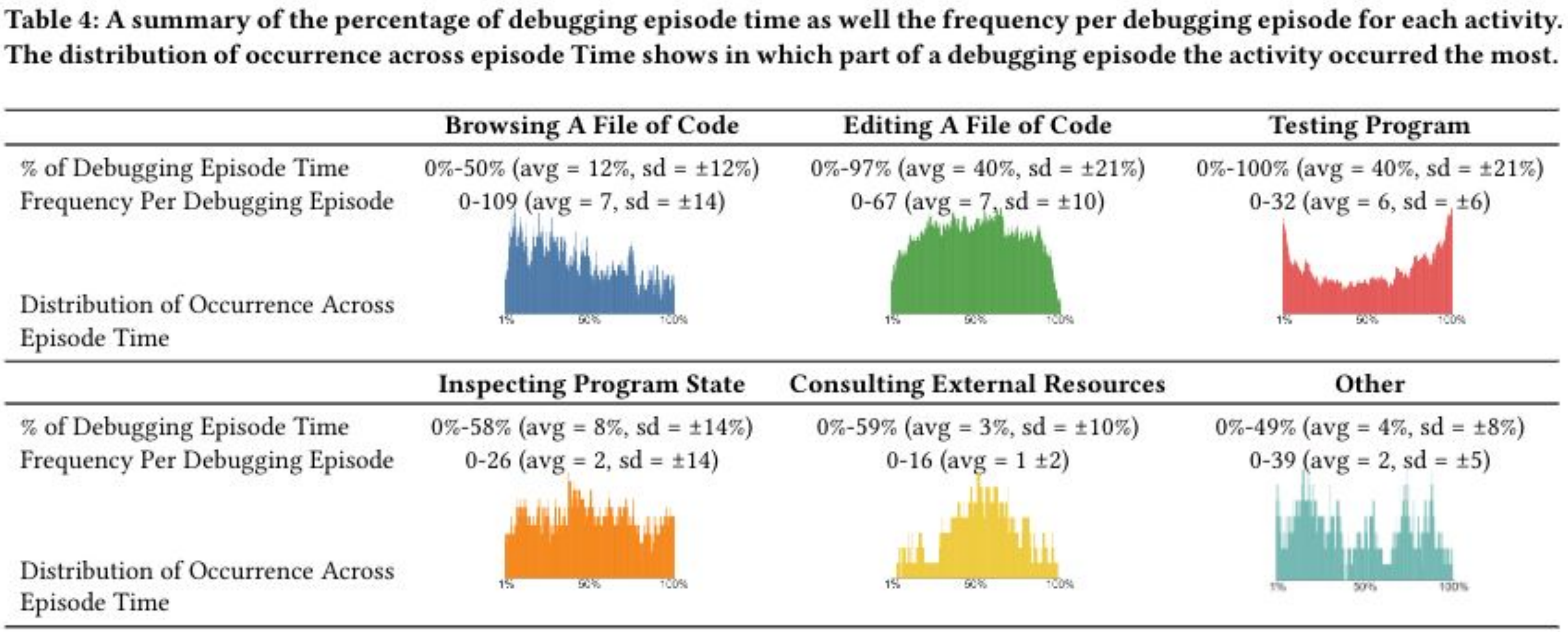


Table 4: A summary of the percentage of debugging episode time as well the frequency per debugging episode for each activity. The distribution of occurrence across episode Time shows in which part of a debugging episode the activity occurred the most.

	Browsing A File of Code	Editing A File of Code	Testing Program
% of Debugging Episode Time	0%-50% (avg = 12%, sd = ±12%)	0%-97% (avg = 40%, sd = ±21%)	0%-100% (avg = 40%, sd = ±21%)
Frequency Per Debugging Episode	0-109 (avg = 7, sd = ±14)	0-67 (avg = 7, sd = ±10)	0-32 (avg = 6, sd = ±6)
Distribution of Occurrence Across Episode Time			
	Inspecting Program State	**Consulting External Resources**	**Other**
% of Debugging Episode Time	0%-58% (avg = 8%, sd = ±14%)	0%-59% (avg = 3%, sd = ±10%)	0%-49% (avg = 4%, sd = ±8%)
Frequency Per Debugging Episode	0-26 (avg = 2, sd = ±14)	0-16 (avg = 1 ±2)	0-39 (avg = 2, sd = ±5)
Distribution of Occurrence Across Episode Time			

Tips for Debugging

- Avoid random or extensive changes to the program! First, examine the code to figure out what went wrong.
- Think before you change anything
- Figure out *why* it went wrong
- Always address the first compiler error listed before addressing the other errors.
- Always recompile after making changes to code because fixing one error can fix other errors as well (or introduce new ones).
- Use automated tools when possible.

Types of Errors

1. **Syntax errors**
   a. Programming language misuse (usually caught by compiler).
2. **Logic errors**
   a. Code is syntactically correct, but there is incorrect output/behavior.
3. **Runtime errors**
   a. The program experiences an error and stops during execution.

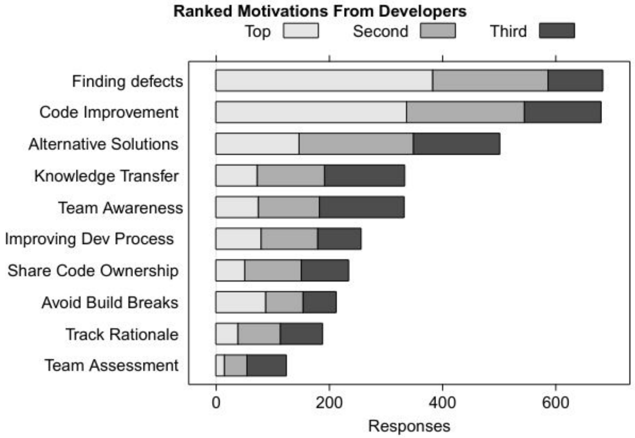
Code Reviews

- The process of manually inspecting source code changes.
  - Human code analysis (also known as peer code reviews)
  - Usually performed by a developer other than the author
- Goal: Inspect code before integration to improve software quality.

Why Code Reviews?

- Ensures requirements are met
- Ensures consistent design
- Ensures consistent implementation

Also provides many benefits to development teams…



[Bacchelli]

Mechanics of Code Reviews

- ***Who:*** Original developer and reviewer, sometimes together in person, sometimes offline.
- ***What:*** Reviewer gives suggestions for improvement on a logical and/or structural level, to conform to previously agreed upon set of quality standards.
  - Feedback leads to refactoring, followed by a 2nd code review.
  - Eventually reviewer approves code.
- ***When:*** When code author has finished a coherent system change that is otherwise ready for merging
  - change shouldn't be too large or too small
  - before committing the code to the repository or incorporating it into the new build

Why Code Reviews?

- > 1 person has seen every piece of code
  - Prospect of someone reviewing your code raises quality threshold.
- Forces code authors to articulate their decisions
- Hands-on learning experience for rookies without hurting code quality
  - Pairing them up with experienced developers

Why Code Reviews (cont.)?

- Team members involved in different parts of the system
  - Reduces redundancy, enhances overall understanding
- Author and reviewer both accountable for committing code

Code Review Variations

inspection: A more formalized code review with:

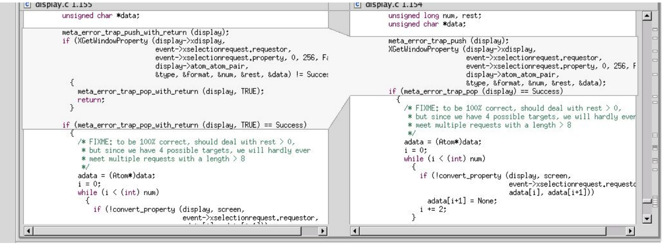
- roles (moderator, author, reviewer, scribe, etc.)
- several reviewers looking at the same piece of code
- a specific checklist of kinds of flaws to look for
  - possibly focusing on flaws that have been seen previously
  - possibly focusing on high-risk areas such as security
- specific expected outcomes (e.g. report, list of defects)

walkthrough: informal discussion of code between author and a single reviewer

code reading: Reviewers look at code by themselves (possibly with no actual meeting)

Code Reviews in Industry

- Code reviews are a **very** common industry practice.
- Made easier by advanced tools that:
  - integrate with configuration management systems
  - highlight changes (i.e., diff function)
  - allow traversing back into history



display.c 1.155

```
        unsigned char *data;

        meta_error_trap_push_with_return (display);
        if (XGetWindowProperty (display->xdisplay,
                                event->xselectionrequest.requestor,
                                event->xselectionrequest.property, 0, 256, Fa
                                display->atom_atom_pair,
                                &type, &format, &num, &rest, &data) != Succes
          {
            meta_error_trap_pop_with_return (display, TRUE);
            return;
          }

        if (meta_error_trap_pop_with_return (display, TRUE) == Success)
          {
            /* FIXME: to be 100% correct, should deal with rest > 0,
             * but since we have 4 possible targets, we will hardly ever
             * meet multiple requests with a length > 8
             */
            adata = (Atom*)data;
            i = 0;
            while (i < (int) num)
              {
                if (!convert_property (display, screen,
                                       event->xselectionrequest.requestor,
```

display.c 1.154

```
        unsigned long num, rest;
        unsigned char *data;

        meta_error_trap_push (display);
        XGetWindowProperty (display->xdisplay,
                            event->xselectionrequest.requestor,
                            event->xselectionrequest.property, 0, 256, F
                            display->atom_atom_pair,
                            &type, &format, &num, &rest, &data);
        if (meta_error_trap_pop (display) == Success)
          {
            /* FIXME: to be 100% correct, should deal with rest > 0,
             * but since we have 4 possible targets, we will hardly ever
             * meet multiple requests with a length > 8
             */
            adata = (Atom*)data;
            i = 0;
            while (i < (int) num)
              {
                if (!convert_property (display, screen,
                                       event->xselectionrequest.requesto
                                       adata[i], adata[i+1]))
                  adata[i+1] = None;
                i += 2;
              }
```

Code Reviews at Google

- "All code that gets submitted needs to be reviewed by at least one other person, and either the code writer or the reviewer needs to have readability in that language. Most people use Mondrian to do code reviews, and obviously, we spend a good chunk of our time reviewing code."

-- Amanda Camp, Software Engineer, Google

Code reviews at Yelp

- "At Yelp we use review-board. An engineer works on a branch and commits the code to their own branch. The reviewer then goes through the diff, adds inline comments on review board and sends them back. The reviews are meant to be a dialogue, so typically comment threads result from the feedback. Once the reviewer's questions and concerns are all addressed they'll click "Ship It!" and the author will merge it with the main branch for deployment the same day."

-- Alan Fineberg, Software Engineer, Yelp

Code reviews at WotC

- “At Wizards we use Perforce for SCM. I work with stuff that manages rules and content, so we try to commit changes at the granularity of one bug at a time or one card at a time. Our team is small enough that you can designate one other person on team as a code reviewer. Usually you look at code sometime that week, but it depends on priority. It’s impossible to write sufficient test harnesses for the bulk of our game code, so code reviews are absolutely critical.”

-- Jake Englund, Software Engineer, MtGO

Code reviews at Facebook

- "At Facebook, we have an internally-developed web-based tool to aid the code review process. Once an engineer has prepared a change, she submits it to this tool, which will notify the person or people she has asked to review the change, along with others that may be interested in the change -- such as people who have worked on a function that got changed.

  At this point, the reviewers can make comments, ask questions, request changes, or accept the changes. If changes are requested, the submitter must submit a new version of the change to be reviewed. All versions submitted are retained, so reviewers can compare the change to the original, or just changes from the last version they reviewed. Once a change has been submitted, the engineer can merge her change into the main source tree for deployment to the site during the next weekly push, or earlier if the change warrants quicker release."

- Ryan McElroy, Software Engineer, Facebook

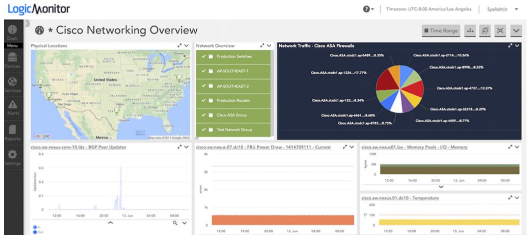
Monitoring Software

- The process of observing, tracking, and reporting on the operations and activities of:
    - Users
    - Applications
    - Network or system services
    - Other non-functional requirements (performance, security, etc.)

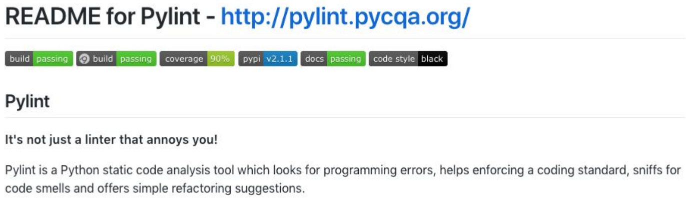
Software Monitoring Examples

Many ways to approach monitoring software:

Visualization Tools


LogicMonitor
Cisco Networking Overview
Time Range
Physical Locations
United States
Network Overview
Production Switches
AP-SOUTHEAST-1
AP-SOUTHEAST-2
Production Routers
Cisco ASA Group
Test Network Group
Network Traffic - Cisco ASA Firewalls
cisco.sw.nexus-core-10.lax - BGP Peer Updates
cisco.sw.nexus.07.dc10 - FRU Power Draw - 1414709111 - Current
cisco.sw.nexus01.lax - Memory Pools - I/O - Memory
cisco.sw.nexus.01.dc10 - Temperature
cisco.sw.nexus.01.lax - Top 10 Interfaces by bandwidth

GitHub Badges



README for Pylint - http://pylint.pycqa.org/

build passing | build passing | coverage 90% | pypi v2.1.1 | docs passing | code style black

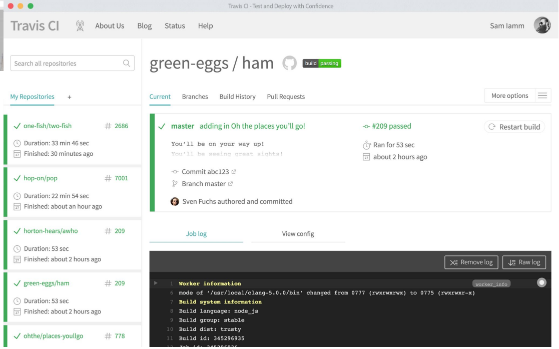
Pylint

It's not just a linter that annoys you!

Pylint is a Python static code analysis tool which looks for programming errors, helps enforcing a coding standard, sniffs for code smells and offers simple refactoring suggestions.

System Administrator

Project Builds



green-eggs / ham build passing

✓ one-fish/two-fish # 2686
Duration: 33 min 46 sec
Finished: 30 minutes ago

✓ hop-on/pop # 7001
Duration: 22 min 54 sec
Finished: about an hour ago

✓ horton-hears/awho # 209
Duration: 53 sec
Finished: about 2 hours ago

✓ green-eggs/ham # 209
Duration: 53 sec
Finished: about 2 hours ago

✓ ohthe/places-youllgo # 778

✓ master adding in Oh the places you'll go! #209 passed

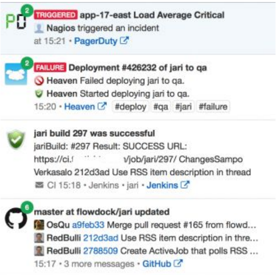
```
You'll be on your way up!
You'll be seeing great sights!
```

Commit abc123
Branch master
Sven Fuchs authored and committed

Ran for 53 sec
about 2 hours ago

```
Worker information
mode of '/usr/local/clang-5.0.0/bin' changed from 0777 (rwxrwxrwx) to 0775 (rwxrwxr-x)
Build system information
Build language: node_js
Build group: stable
Build dist: trusty
Build id: 345296935
```

Issue Trackers



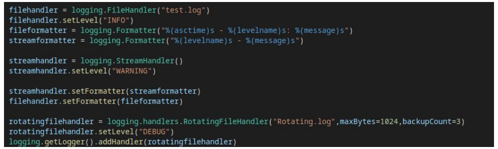
TRIGGERED app-17-east Load Average Critical
Nagios triggered an incident
at 15:21 · PagerDuty

FAILURE Deployment #426232 of jari to qa
Heaven Failed deploying jari to qa.
Heaven Started deploying jari to qa.
15:20 · Heaven #deploy #qa #jari #failure

jari build 297 was successful
jariBuild: #297 Result: SUCCESS URL:
https://ci.f v/job/jari/297/ ChangesSampo
Verkasalo 212d3ad Use RSS item description in thread
CI 15:18 · Jenkins · jari · Jenkins

master at flowdock/jari updated
OsQu a9feb33 Merge pull request #165 from flowd...
RedBulli 212d3ad Use RSS item description in thre...
RedBulli 2788509 Create ActiveJob that polls RSS ...
15:17 · 3 more messages · GitHub

Logging



```
filehandler = logging.FileHandler("test.log")
filehandler.setLevel("INFO")
fileformatter = logging.Formatter("%(asctime)s - %(levelname)s: %(message)s")
streamformatter = logging.Formatter("%(levelname)s - %(message)s")

streamhandler = logging.StreamHandler()
streamhandler.setLevel("WARNING")

streamhandler.setFormatter(streamformatter)
filehandler.setFormatter(fileformatter)

rotatingfilehandler = logging.handlers.RotatingFileHandler("Rotating.log",maxBytes=1024,backupCount=3)
rotatingfilehandler.setLevel("DEBUG")
logging.getLogger().addHandler(rotatingfilehandler)
```

Next Time…

- **Project Milestone 3 due Friday (11:59pm)**
    - Upload on Canvas

- Discussion Presentations on *Implementation and Maintenance* [11/10]
- Discussion Presentations on *Testing* [11/17]

References

- Abdulaziz Alaboudi et al. *“An exploratory study of debugging episodes”.* 2017
- RS Pressman. *“Software engineering: a practitioner's approach”.*
- Martin Fowler, Kent Beck, John Brant, William Opdyke, and Donald Roberts, *“Refactoring: Improving the Design of Existing Code”*, 1999.
- Danilo Silva et al. *“Why We Refactor? Confessions of GitHub Contributors”.* 2016
- Ruchika Malhotra and Anuradha Chug, *“Software Maintainability: Systematic Literature Review and Current Trends”*, 2016.
- The Thales Group. *“The 4 Types of Software Maintenance”.*
- Emerson Murphy-Hill. *“How we Refactor, and How We Know It”.* 2012
- Chris Parnin
- Code Reviews lecture, University of Washington.