[CS3704] Software Engineering

Dr. Chris Brown
Virginia Tech
11/6/2023

Announcements

- **Project Milestone 3 due Friday (11/10) at 11:59pm!**
  - High-level architecture design
  - Low-level architecture design
  - Design sketch
    - Wireframe or storyboard with design decisions
  - Project Check-in Survey
  - Process deliverable
    - ***Must show incremental improvement since PM2!***

Implementation

Software Development Philosophies
Implementation Tools and Practices

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

Warm-Up

Find a partner to discuss the following question:

What is the most interesting software product you have implemented (written some code for)?

Implementation

Goal: translate software design into a concrete system (i.e. write code)

- Can use any programming language, but some languages are better suited to certain types of programs than others
- SE/implementation is a *team* activity!
- *Software Artifacts*: source code, documentation, configuration files, media, executables, bug database, source code repository, issue trackers…

Design in SE Contexts

- Software design is applied regardless of the software process model used.
  - Begins once software requirements have been analyzed and modeled
- Design is the *last software engineering action within the modeling activity* and sets the stage for construction (code generation and testing)
  - “During design, you make decisions that will ultimately affect the success of software construction and, as important, the ease with which software can be maintained” [Pressman]

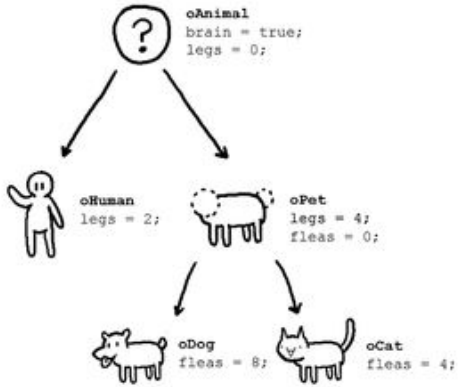
Design Pattern Usage in SE

Do software engineers actually use design patterns?

- Not in terms of modeling (i.e. class diagrams)
- But yes
  - Design patterns promote reuse in software
  - Apply repeatable solutions to common problems that can be transformed directly into code
  - Make code more maintainable, easier to understand (for future you and others),...

Design Patterns

What OOP users claim



Implementation in Practice

- **Tools and processes support productive programming among software development teams.**
  - *Remember, software engineers spend limited time actually coding!*

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

Activity Category	All 100% (N=5928) pct	All min	Typical WD 64% (N=3750) pct	Typical WD min	Atypical WD 36% (N=2099) pct	Atypical WD min	Good WD 61% (N=3028) pct	Good WD min	Bad WD 39% (N=1970) pct	Bad WD 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	

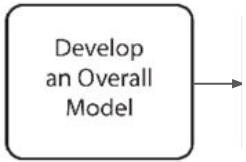
The left number in a cell indicates the average relative time spent (in percent) and the right number in a cell the absolute average time spent (in minutes).

SE Philosophies

- Also known as paradigms, principles, etc.
  - Can be implemented as software processes or combined with existing processes.
    - **Software process: who** does **what, when,** in order to build a piece of software. [Wilson]
- Numerous SE philosophies exist in literature.
  - Feature-driven development
  - Test-driven development
  - Behavior-driven development
  - Secure-by-design
  - Data-driven development
  - …

Feature-Driven Development

- Introduced in 1997
- Model-driven iterative and incremental process that consists of five basic activities:

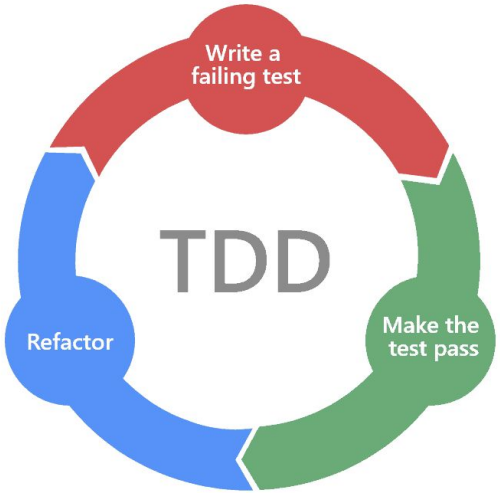


Test-Driven Development

- Introduced in 1999
- Emphasizes tests and software quality over features.

1. Write unit tests based on requirements
2. Run the tests
   a. Expected to fail ✘
3. Write the simplest code to pass the tests
4. Run the tests
   a. Expected to pass ✅
5. Refactor code as needed
6. Repeat

More on testing (11/13)…



Behavior-Driven Development

- Combination of FDD and TDD
- Instead of starting with unit tests (code), start with behavioral specifications
  - Desired behavior, i.e. acceptance tests or scenarios

Stand-Up Bot ser Story Example:
As a *Scrum Master*, I want to *find availability with a list of usernames* so that I can *schedule team daily stand-ups*.

Acceptance criteria:
Given Scrum Master has admin permissions
When Scrum Master types team member usernames
And <User can perform multiple actions here>
Then System finds list of available times between users

Common Implementation Guidelines

Rules of thumb to support software implementation.

- **The First Law**
- **Brooks' Law:** *"Adding manpower to a late software project makes it later"*
- **YAGNI:** ***"****You aren't/ain't gonna need it"* (*XP*)
- **Gall's Law:** *"A complex system that works is invariably found to have evolved from a simple system that worked. A complex system designed from scratch never works and cannot be patched up to make it work. You have to start over with a working simple system."*

Integrated Development Environments

IDEs enable software engineers to complete various activities re am. IDEs often provide a v are developme

- syntax analysis tools, debugg ple language suppor ning, and more...

/poll What IDE(s) do you primarily use?

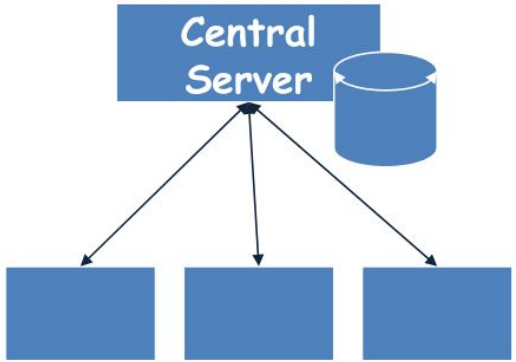
Version Control

A tool to manage changes to documents, programs, and other information. Each set of changes creates a new *commit* to the files.

- What changes have been made?
- Why were the changes made?
- Who made the changes?
- Can we redo/undo some changes?
- Can we branch the project?
- …

Version Control (cont.)

Centralized vs. Distributed



Version Control Tools

- Version control systems allow users to recover old commits reliably, and helps manage conflicting changes made by different users.

Issue Tracking Systems

- Manage and maintain list of issues as needed by the organization.
  - Create, update, and resolve reported issues by customers and developers.
- An ***issue*** is a unit of work to accomplish an improvement to the system. This includes

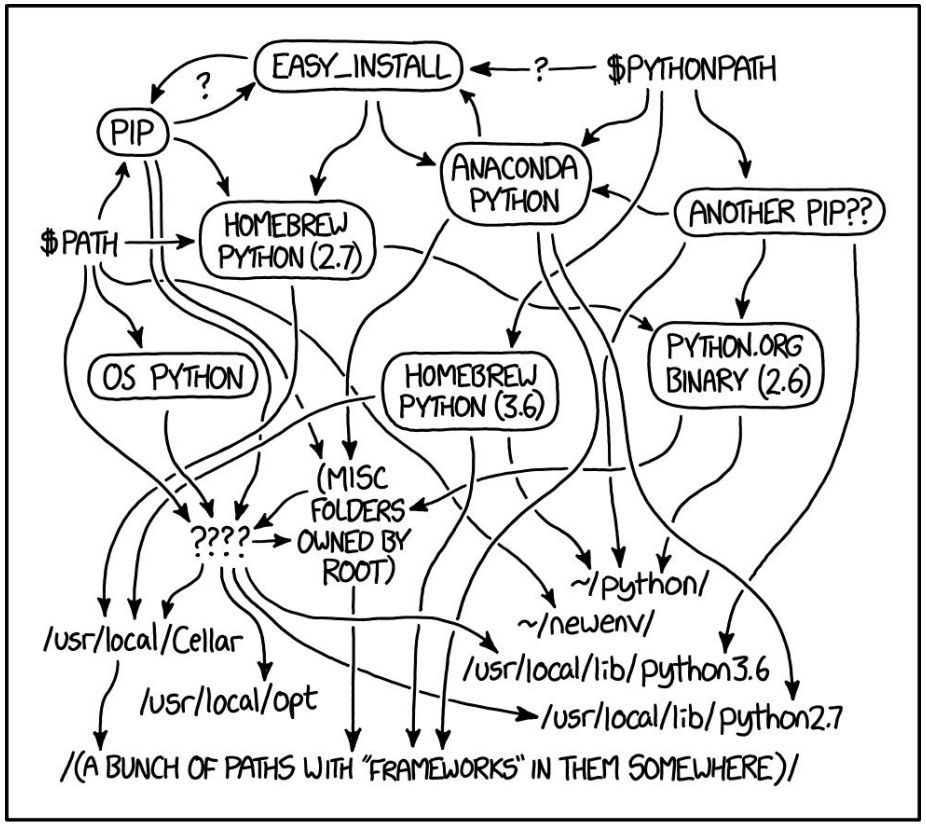
  – a bug
  – a requested feature
  – a patch
  – missing documentation, …

Continuous Integration

More details with deployment (11/27)…

- **Integration:** merging changes with the source code repository
  - Broken integration occurs when code does not build, shared components work in one branch but not another, unit tests fail, etc.
- Manual Integration is expensive
  - Build, test, deploy can take hours or days…
  - Integration problems and bugs detected too late

- **<u>Continuous Integration:</u>** work is integrated frequently, usually multiple integrations daily, and verified by an automated system to detect integration errors as quickly as possible.



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED
THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

More details with deployment (11/27)…

docker

ANSIBLE

puppet by Perforce

CHEF

Code Analysis Tools

- Manual analysis of code is very valuable, but also inefficient.
  - **Ideally, manual analysis should be combined with automated tools.**
- Automated tools can analyze large software systems and support programmers.
  - **Improve code quality, productivity, efficiency, etc…**

TODO: Discuss any development tools you have used to help with programming.

Static Analysis

The process of automatically examining source code without executing the application.

- analysis tools, code metrics, linters, code clone detection, etc.

Why static analysis?

- Improves code quality,
- Finds errors more effectively and efficiently,
- Reduces technical debt and software costs,
- Increases productivity,...

Dynamic Analysis

The investigation of the properties of a *running* software system.

- Loggers, debuggers, etc.

Why dynamic analysis?

- Gap between run-time and code structure
- Collect runtime execution information
- Finds bugs in application
- Allows for program transformation, optimization
- Modify program behaviors on the fly

Development Tools are Useful, but…

Why Don't Software Developers Use Static Analysis Tools to Find Bugs?

Brittany Johnson, Yoonki Song, and Emerson Murphy-Hill
North Carolina State University
Raleigh, NC, U.S.A.
bijohnso,ysong2@ncsu.edu,emerson@csc.ncsu.edu

Robert Bowdidge
Google
Mountain View, CA, U.S.A.
bowdidge@google.com

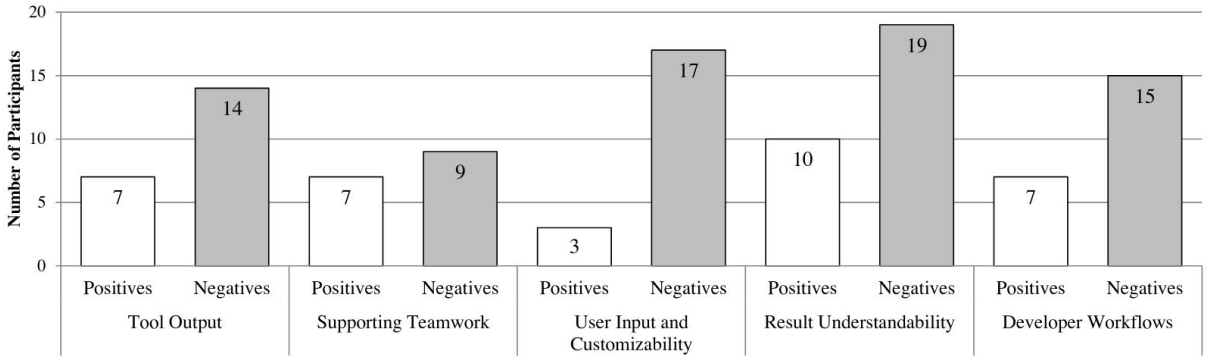
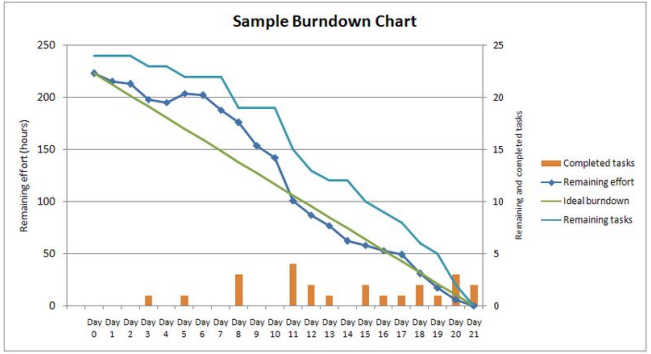


Fig. 1. The number of participants in each category expressing the good and the bad about static analysis tools they have used.

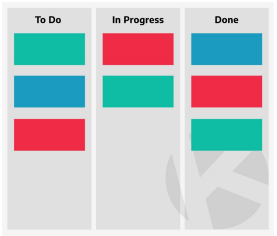
Informative Workspaces and Teams

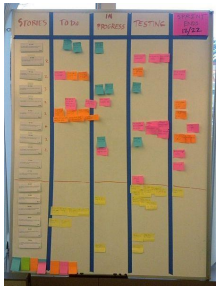
Burndown Chart

Shows how many hours of work was done, and how many remain, over the sprint, or product release.



Kanban Boards





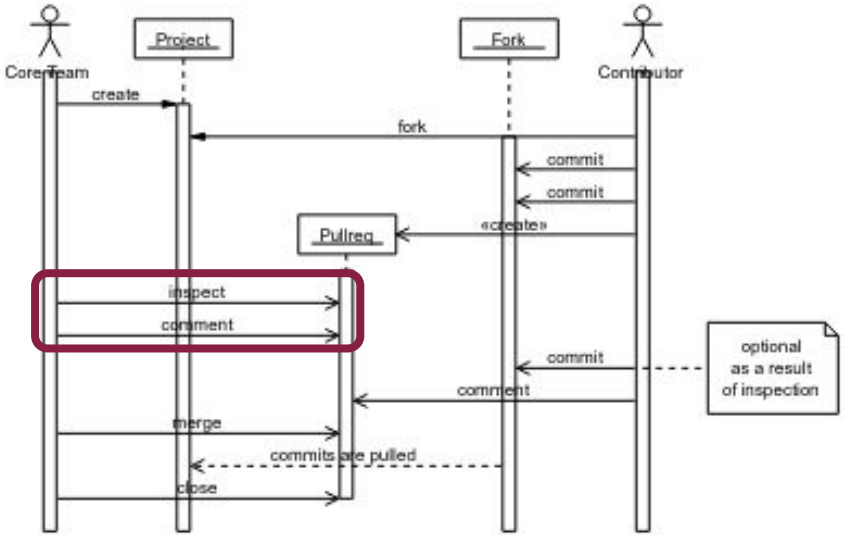
Stand-up Meetings

TODO: Another scrum meeting!

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

Pull-based Software Development

- Standard model for distributed software development and version control systems.



[Gousios]

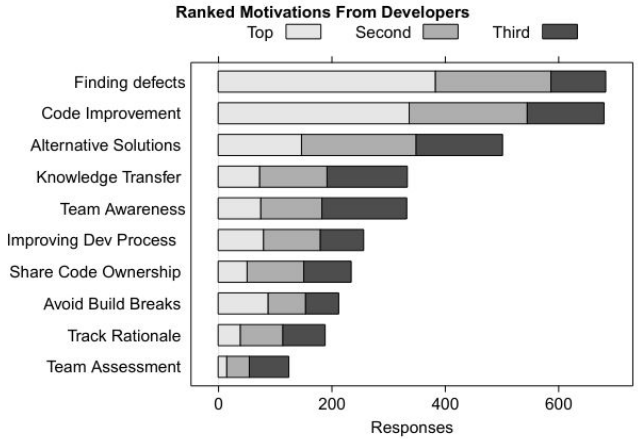
Code Reviews

- 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]

Code Ownership

- **Strong code ownership:** Code is divided into modules (classes, functions, files, etc.) and assigned to one developer. Developers are only allowed to make changes to modules they own.
- **Weak code ownership:** Developers are allowed to make changes to modules owned by other people.
- **Collective code ownership:** abandons any notion of individual code ownership of modules. Code base is owned by the entire team and anyone can make changes anywhere. ⭐

[Fowler]

Pair Programming

- The process of two software engineers working on a task at one computer.
  - One programmer, **the driver**, has control of the keyboard and mouse to create the implementation
  - The other programmer, **the navigator**, *actively* watches the driver's implementation to identify defects and participate in on-demand brainstorming
  - The roles of driver and navigator are periodically rotated between the software engineers.

Pair Programming (cont.)

Benefits:

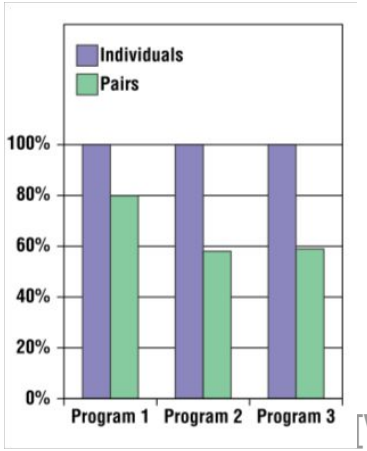
- Higher product quality,
- Improved cycle time,
- Increased programmer satisfaction
- Enhanced learning, team-building, etc.
- And more!

Table 1

Percentage of Test Cases Passed*

	Individuals	Pairs
Program 1	73.4	86.4
Program 2	78.1	88.6
Program 3	70.4	87.1
Program 4	78.1	94.4

*The difference in quality levels is statistically significant to $p < 0.01$; p is the probability that these results could occur by chance.



[Williams]

Pair Programming: Picking Partners

Expert paired with an Expert

Expert paired with a Novice

Novices paired together

Professional Driver Problem

Culture

Pair Programming Activity

Find a partner to complete the following activity:

In-Class Activity

For the following activity, find a partner in class with shared knowledge of a programming language. You must work with only **one** partner unless otherwise specified to receive credit. Each student must complete a designated part of the activity on their own machine, while the partner *actively* participates in the coding.

> ### Roman Numerals
>
> Roman numerals are depicted by using seven different symbols: `I, V, X, L, C, D and M` to represent seven different numerical values: `1, 5, 10, 50, 100, 500, 1000` that can correspond with any other number. For example, the most recent professional American Football championship game was Super Bowl LVII--which was Super Bowl 57. Complete the following for this activity:
>
> - One student will be the driver while the other navigates to write a function that converts an integer to a Roman Numeral.
> - Then, students will switch roles to develop a function that converts Roman numerals to integers.

The driver must complete their portion of the program on their own machine. When driving, ***each student*** should add comments in the program to describe what the function does, who wrote it, and which IDE or text editor was used to write the code. Create your method(s) in a file named Roman.. You are not expected to come up with the most efficient or correct solution---this is an activity to practice pair programming and track attendance, not a job interview. Only the conversion methods are needed.

Each student must complete the following survey to receive credit: https://forms.gle/dgdCG5Tw4wFgRZgG8.

https://github.com/CS3704-VT/Course/blob/main/resources/workshops/PairProgramming.md

Next Time…

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

- Maintenance [11/8]
- Discussion Presentations on *Implementation and Maintenance* [11/10]

References

- RS Pressman. *“Software engineering: a practitioner's approach”.*
- Laurie Williams. *“Strengthening the Case for Pair Programming”.* 2000
- Brad Appleton. *“Streamed Lines: Branching Patterns for Parallel Software Development”.*
- Christian Bird. *“Assessing the Value of Branches with What-if Analysis”*. 2012
- Martin Fowler. *“CodeOwnership”.* 2006
- https://refactoring.guru/design-patterns
- Chris Parnin
- Sarah Heckman
- Na Meng