
OUR GOAL IS TO WRITE BUG-FREE SOFTWARE. I'LL PAY A TEN-DOLLAR BONUS FOR EVERY BUG YOU FIND AND FIX.
S. Adams E-mail: SCOTTADAMS@AOL.COM
YAHOO!
WE'RE RICH
YES!!! YES!!! YES!!!
11/13 © 1995 United Feature Syndicate, Inc.(NYC)
I HOPE THIS DRIVES THE RIGHT BEHAVIOR.
I'M GONNA WRITE ME A NEW MINIVAN THIS AFTER-NOON!

Defect Detection and Collaborative Development

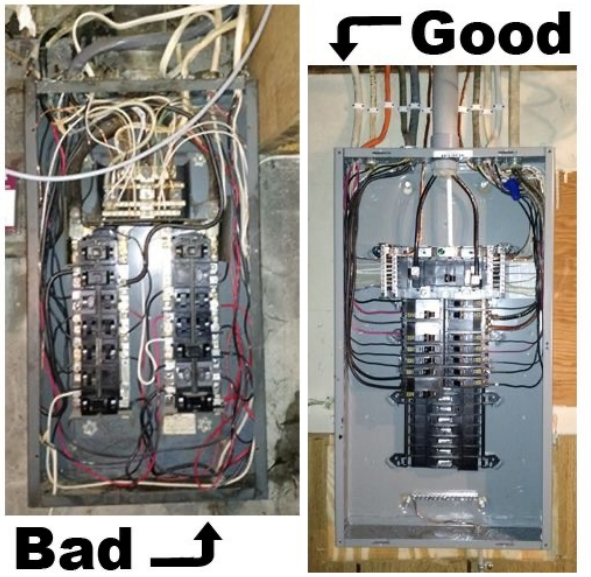
CSCI 2134: Software Development

Agenda

- Lecture Contents
  - Motivation
  - Types of Collaborative Software Development
  - Pair Programming
  - Technical Reviews
  - Brightspace Quiz
- Readings:
  - This Lecture: Chapter 21
  - Next Lecture: Chapter 22

Why Does Software Quality Matter

- User Perspective: It’s harder to sell bad software.



- Developer Perspective: Software needs to be
  - Developed
  - Maintained
  - Fixed
  - Updated
  - Extended
  - Etc.
- **General Principle of Software Quality:**
  Improving software quality reduces development costs

Debugging and Fixing Is Expensive

- **Debugging takes more than 50% of the time! (McConnel, pg 474)**
- **Key Idea:** The earlier a bug/defect is identified (or avoided), the cheaper it is to fix
- Goal: Reduce software defects (good quality software)

Table 3-1 Average Cost of Fixing Defects Based on When They're Introduced and Detected

	Time Detected				
Time Introduced	**Requirements**	**Architecture**	**Construction**	**System Test**	**Post-Release**
Requirements	1	3	5–10	10	10–100
Architecture	—	1	10	15	25–100
Construction	—	—	1	10	10–25

Source: Adapted from "Design and Code Inspections to Reduce Errors in Program Development" (Fagan 1976), *Software Defect Removal* (Dunn 1984), "Software Process Improvement at Hughes Aircraft" (Humphrey, Snyder, and Willis 1991), "Calculating the Return on Investment from More Effective Requirements Management" (Leffingwell 1997), "Hughes Aircraft's Widespread Deployment of a Continuously Improving Software Process" (Willis et al. 1998), "An Economic Release Decision Model: Insights into Software Project Management" (Grady 1999), "What We Have Learned About Fighting Defects" (Shull et al. 2002), and *Balancing Agility and Discipline: A Guide for the Perplexed* (Boehm and Turner 2004).

(McConnell, "Code Complete 2nd Edition, 2004, pg 29)

Techniques to Reduce Software Defects

Avoid software defects

- Set software quality objectives
- Require explicit quality assurance activity
  - Measure the software quality objectives
  - Allocate time to meet the objectives
  - Reward developers for meeting objectives
- Use software engineering guidelines
- Use collaborative development

Detect and fix existing defects

- Use collaborative development
- Perform technical reviews
- Have a testing strategy
  (next week's lectures)
- Have external audits

Set Software Quality Objectives

- All developers must be aiming for the same set of characteristics
  E.g., Maintainability, usability, testability, robustness, correctness
- Developers will focus on the required objectives if they know what they are and that they are being measured!

Table 20-1 Team Ranking on Each Objective

Objective Team Was Told to Optimize	Minimum memory use	Most readable output	Most readable code	Least code	Minimum programming time
Minimum memory	1	4	4	2	5
Output readability	5	1	1	5	3
Program readability	3	2	2	3	4
Least code	2	5	3	1	3
Minimum programming time	4	3	5	4	1

Source: Adapted from "Goals and Performance in Computer Programming" (Weinberg and Schulman 1974).

(McConnell, "Code Complete 2nd Edition, 2004, pg 469)

Setting Software Quality Objectives

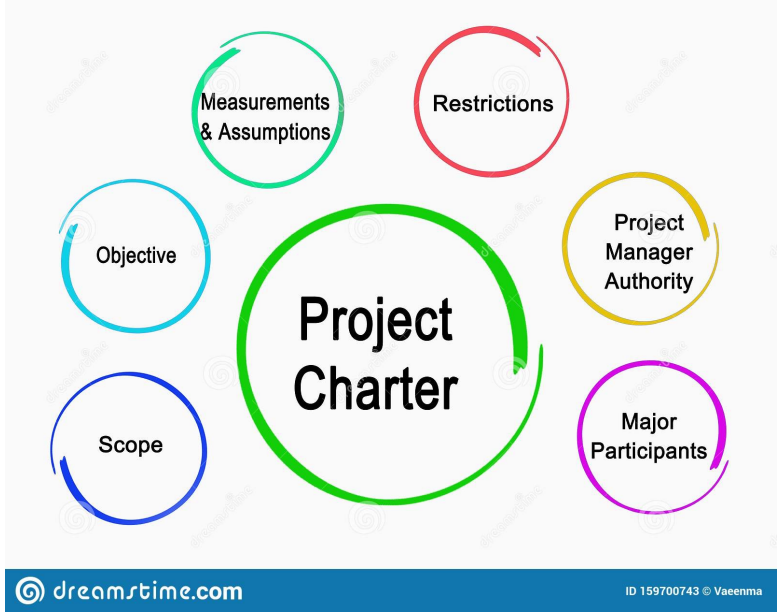
Software quality objectives must be

- Defined and clear
  - There should not be multiple interpretation of the same objective
  - Software quality objectives should be thought of as standards
- Stable
  - Objectives should not change as the project progresses
  - It's hard to meet objectives that are changing
- Consistent
  - Different objectives should not apply to different developers
  - All developers are held to the same standard
- Prioritized
  - If two objectives are in conflict, a clear precedence order must be defined
- Known by all
  - All developers must be onboarded with the same quality objectives
  - It is impossible to be consistent if different developers are working to different objectives.

But how?

Use a Project Charter to Set QA Objectives

- A **Project Charter** is a document that outlines the goals of a project and the responsibility of the team members
- It is the “constitution” of the project team
- Embedding the quality objectives into the project charter is a good way of ensuring that all developers are on the same page
- You will use project charters in
  - CSCI 3130 Software Engineering
  - CSCI 2691, CSCI 3691, CSCI 4691 Project courses
  - Some 4th year courses that have group projects



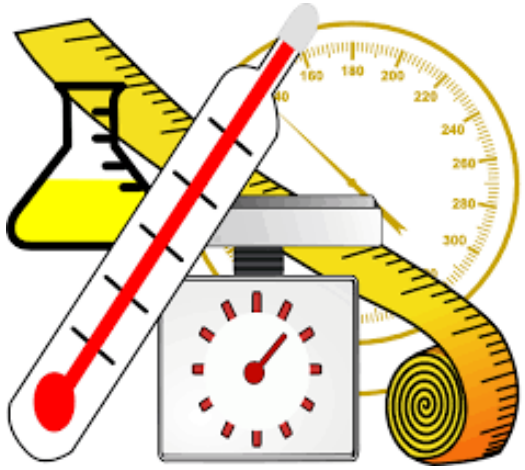
Examples of Software Quality Objectives

- **Readable code**: All code shall follow a specified style guide
- **New code does not break existing code**: All code must compile and pass all tests before being checked in
- **Code must be testable**: All code must be checked in with a full test suite
- **Code is verified**: All code must pass a formal review before being merged
- **Acceptable defect rate**: The number of defects per X lines of code should be less than Y

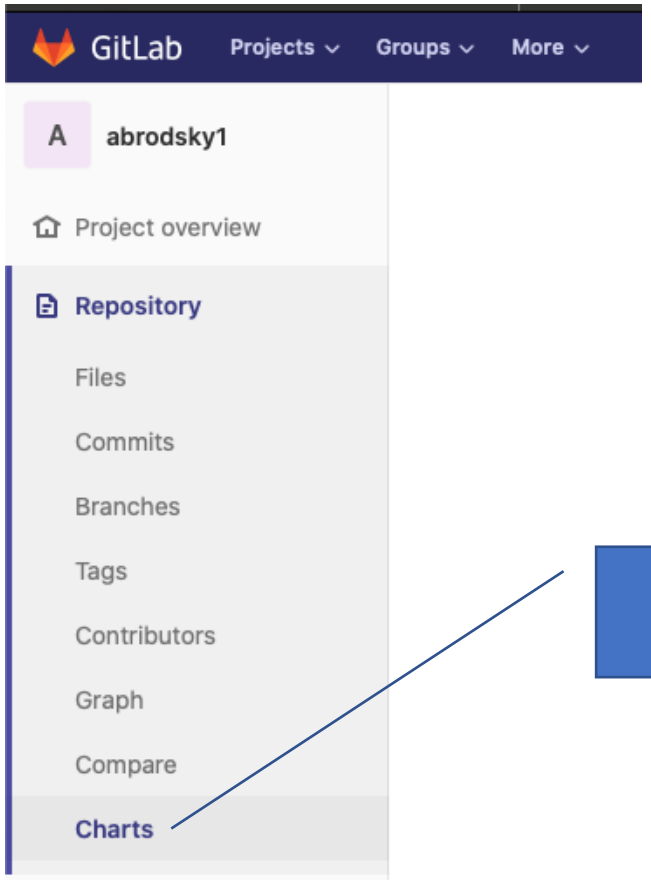
Explicit Quality Assurance Activity

- Quality assurance is not secondary.
  It's a prime part of your job!

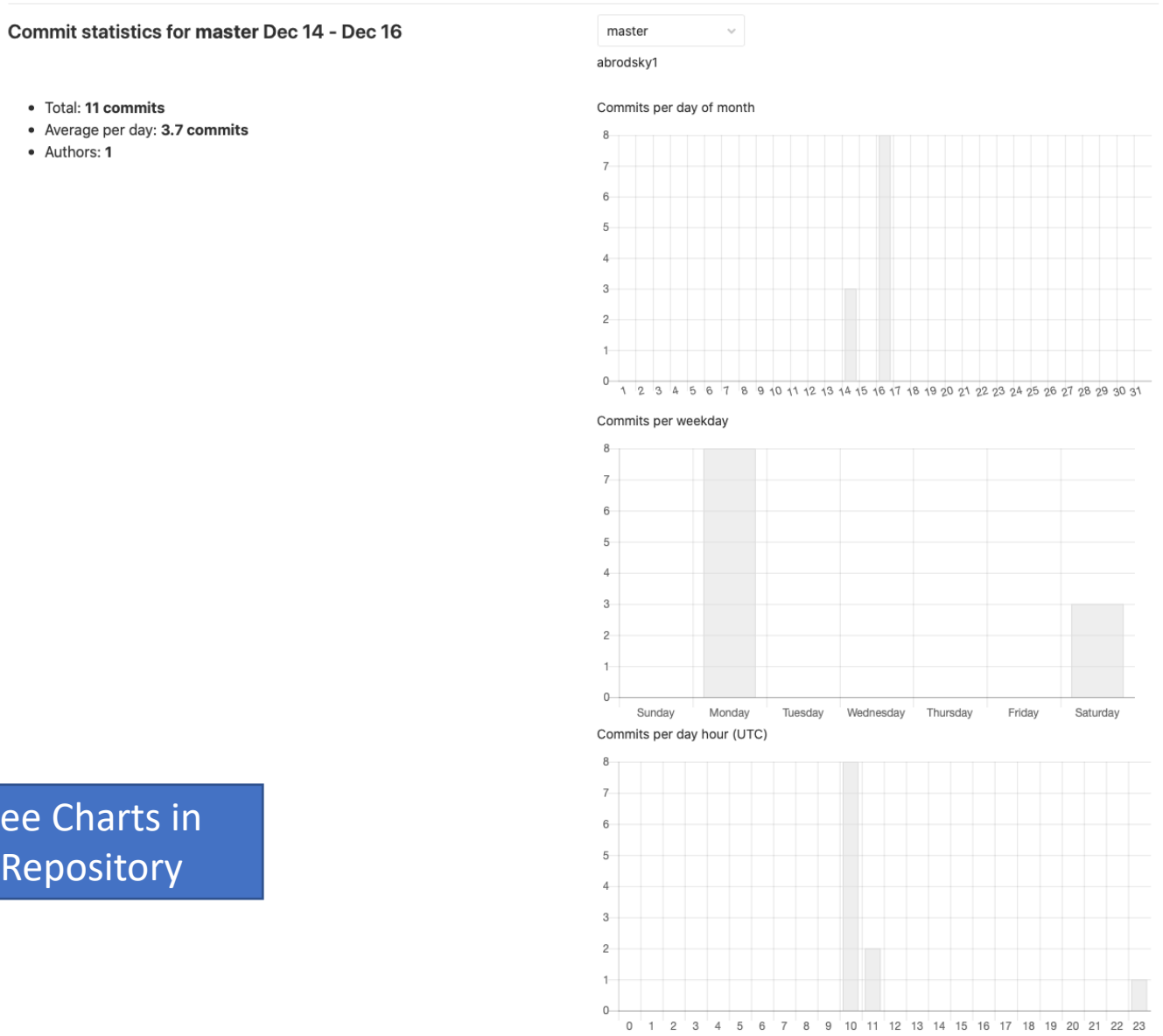
- Measure the software quality objectives
  - If you are not measuring, how do you know if you have achieved it?
  - Determine how and what to measure. E.g.,
    - Number of bugs found in code
    - Amount of time spent debugging
    - Number of corrections needed after a code review
    - Number of merges rejected
  - Take time to gather and review the data and act on it



Example: Use Statistics from GitLab

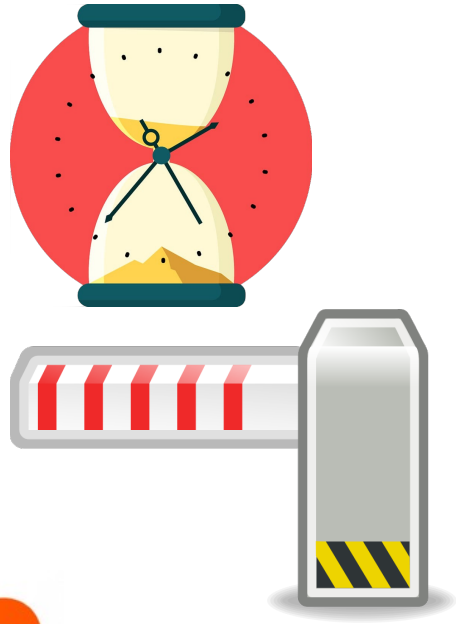


See Charts in Repository



Explicit Quality Assurance Activity

1. Allocate time to meet the objectives
   - If developers are not expected to explicitly do something, they will not!
   - Allocate time for code reviews and audits
   - Plan for time to remediate when objectives are not met
2. Enforce quality objectives
   - Do not allow code to be merged until it meets standards
   - **Gating**: Do not allow a project to proceed until objectives are met
   - Creates more work for supervisors
   - Viewed as a barrier by some
3. Reward developers for meeting objectives
   - Allow project to proceed to next stage
   - Social recognition
   - Monetary rewards



Software Engineering Guidelines

Guidelines are:

- Rules and heuristics that help us make good choices
- Used at each stage of the software development life cycle
- Either:
  - **External:** formulated by a third party or outside organization

    E.g., design patterns, SOLID principles, etc
  - **Internal:** formulated by your organization to be used by its members

    E.g., code style guide
- Not always right, but are right most of the time

  If you are not following the guidelines being used by your team, you should have a very good reason why

Why Use Guidelines?

- **Consistency**

  Similar design and coding scenarios result in the same decisions, leading to similar solutions
- **Correctness**
  - Guidelines are typically based on past experiences and past practices that have resulted in better outcomes
  - Guidelines typically encode “best practices”, which are the current state-of-the-art knowledge for doing things the “right” way
- **Understandability**
  - Guidelines provide standard approaches to problems that are better understood because they have been applied in the past
  - Standard approaches allow the reader to abstract the details, allowing them to focus on the system as a whole
- **Reliability**
  - Guidelines provide standard approaches that have worked well in the past and are more lilely to work in similar instances

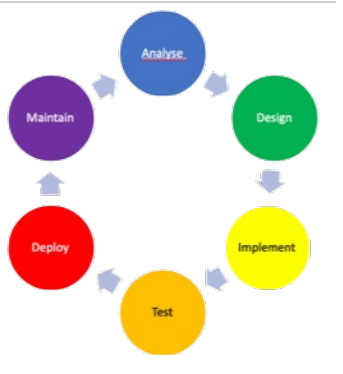
Example: Coding Style Guidelines

- Provide a set of rules for formatting code to make it more readable
- This creates code that is
  - **Consistent**: Code will look as if it is written by the same programmer
  - **Understandable**: Readers of the code will not need to search for information as it will be in the expected places, e.g., comments
  - **More correct and reliable**: Programmers will avoid making common errors, such as closing a code block in the wrong place
- Code readability is an important software quality objective as readable code is much easier to maintain and debug

Intermission

Next: Detecting Defects

Detecting Software Defects (Bugs)



We have many techniques for detecting defects: **Which one do we use?**

- **Informal design reviews** 🔵🟢**:** Informal team discussion of major design decisions
- **Formal design inspections** 🔵🟢: A review of each design decision, where each person has a specific role
- **Informal code reviews** 🟡: Informal team look-over of the one or more pieces of code
- **Formal code inspection** 🟡: A line-by-line review of code, where each person has a specific role
- **Personal desk-checking of code** 🟡: Individual review of personal code
- **Modeling or prototyping** 🟢🟡: Creation of proof-of-concept or simplified versions of the system to verify design decisions
- **Unit test** 🟡🟠🔴🟣: Execution of a complete class, routine, or small program, which is tested in isolation from the more complete system
- **New function (component) test** 🟡🟠🔴🟣: Execution of a class, package, small program, or other program element that involves the work of multiple programmers or programming teams, which is tested in isolation from the more complete system
- **Integration test** 🟡🟠🔴🟣: Combined execution of two or more classes, packages, components, or subsystems that have been created by multiple programmers or programming teams
- **System test** 🟡🟠🔴🟣: Execution of the software in its final configuration, including integration with other software and hardware systems.

Use Combinations of Techniques

- Different techniques are useful for finding different defects in different stages of development
- No technique is 100% effective on its own
- **In combination**, these techniques are **95% effective** (McConnel)

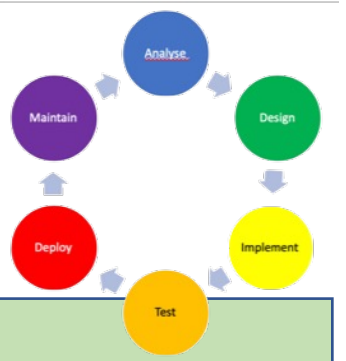
Table 20-2 Defect-Detection Rates

Removal Step	Lowest Rate	Modal Rate	Highest Rate
Informal design reviews	25%	35%	40%
Formal design inspections	45%	55%	65%
Informal code reviews	20%	25%	35%
Formal code inspections	45%	60%	70%
Modeling or prototyping	35%	65%	80%
Personal desk-checking of code	20%	40%	60%
Unit test	15%	30%	50%
New function (component) test	20%	30%	35%
Integration test	25%	35%	40%
Regression test	15%	25%	30%
System test	25%	40%	55%
Low-volume beta test (<10 sites)	25%	35%	40%
High-volume beta test (>1,000 sites)	60%	75%	85%

Source: Adapted from *Programming Productivity* (Jones 1986a), "Software Defect-Removal Efficiency" (Jones 1996), and "What We Have Learned About Fighting Defects" (Shull et al. 2002).

(McConnell, "Code Complete 2nd Edition, 2004, pg 470)

Detecting Software Defects (Bugs)



We have many techniques for detecting defects:

- **Informal design reviews** 🔵🟢**:** Informal team discussion of major design decisions
- **Formal design inspections** 🔵🟢: A review of each design decision, where each person has a specific role
- **Informal code reviews** 🟡: Informal team look-over of the one or more pieces of code
- **Formal code inspection** 🟡: A line-by-line review of code, where each person has a specific role
- **Personal desk-checking of code** 🟡: Individual review of personal code
- **Modeling or prototyping** 🟢🟡: Creation of proof-of-concept or simplified versions of the system to verify design decisions
- **Unit test** 🟡🟠🔴🟣: Execution of a complete class, routine, or small program, which is tested in isolation from the more complete system
- **New function (component) test** 🟡🟠🔴🟣: Execution of a class, package, small program, or other program element that involves the work of multiple programmers or programming teams, which is tested in isolation from the more complete system
- **Integration test** 🟡🟠🔴🟣: Combined execution of two or more classes, packages, components, or subsystems that have been created by multiple programmers or programming teams
- **System test** 🟡🟠🔴🟣: Execution of the software in its final configuration, including integration with other software and hardware systems.

What Is Collaborative Development?

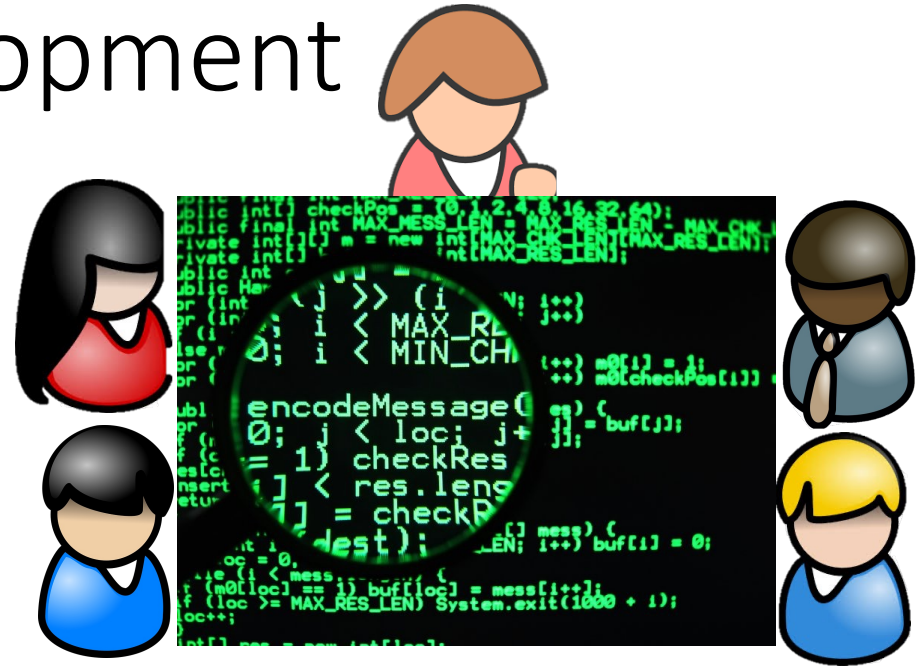
- **Collaborative Development** (Construction) is any technique in which developers share responsibility for creating code and other work products (McConnel, “CC2”, 2004)

- **Assumption**:
  - Every individual has blind-spots, weaknesses, biases that will cause them to create or miss software defects.
  - A team working together will compensate for each other’s blind-spots and make fewer mistakes
  - It is useful for us (developers) to have someone else look at our work

- **African Proverb**:

  “If you want to go quickly, go alone. If you want to go far, go together.”

Collaborative Software Development



- What does collaboration give you?
  - More eyes on the code to find defects
  - More points of view to get to good solutions
  - Shared understanding of the code
  - Shared ownership of the code
- In general, collaborative practices
  - May feel to some like they are wasting time because several people are doing what one person could do alone
  - Typically shorten development time and increase quality because we
    - Find defects earlier, which shortens the later stages of development
    - Make solutions that are more thoughtful

    Principle of software quality

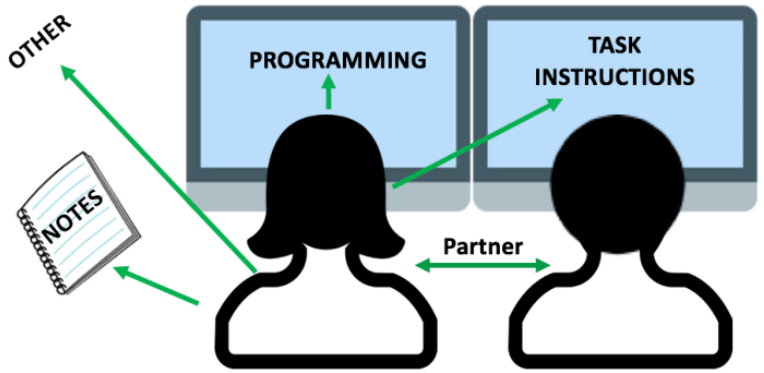
Examples of Collaborative Techniques

Technical Reviews

- Informal reviews
- Code readings
- Code walkthroughs
- Formal code reviews

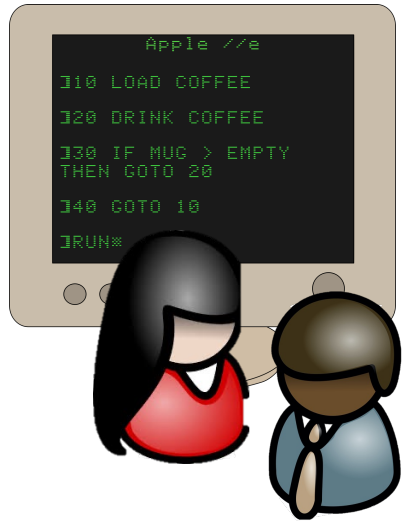
Collaborative Construction

- Extreme programming
- Pair programming



Organizations select the mode(s) that work best for them

Pair Programming



- Develop the code by two people and one keyboard
  - One person is entering the code (driver)
  - The other person (the **navigator**) is “shoulder surfing”
    – actively watching the coder
- Navigator’s Role:
  - Catch bugs as they happen
  - Think ahead of where the design is going so the current code fits well into that plan
  - Look up information on the side to support the coding, like standards, interfaces, data requirements, etc.
- The roles rotate periodically
  - This gives all developers equal opportunity to practice their skills
  - Ensures joint ownership of the work
  - Ensures that both developers are paying attention
- More on this at the end of the course

Technical Reviews

- Reviews must focus on detection of defects

- Reviews should stay away from solving the defects

  We get distracted with solving one defect and:

  - miss other defects
  - run out of time
  - spend the time of many people to solve a defect that takes one person to fix

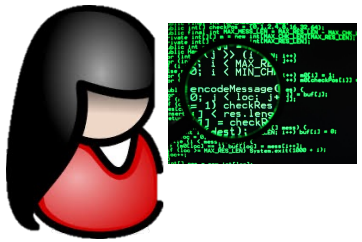
- In almost all reviews, the reviewers must prepare in advance.
  - The reviewers must be familiar with the code being reviewed

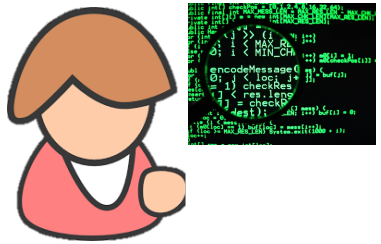
Informal Code Review

- Actions:
  - Ask someone to do a quick check of the code before proceeding in the development process
  - Provide quick feedback
  - Little recording of the defects found
- Advantages:
  - A lightweight (fast) process
  - Typically, good for detecting defects related to
    - layout
    - style
    - common errors
- Disadvantages
  - Only one other person looks at the code
  - Code is not thoroughly reviewed
  - Less effective with finding subtle design flaws

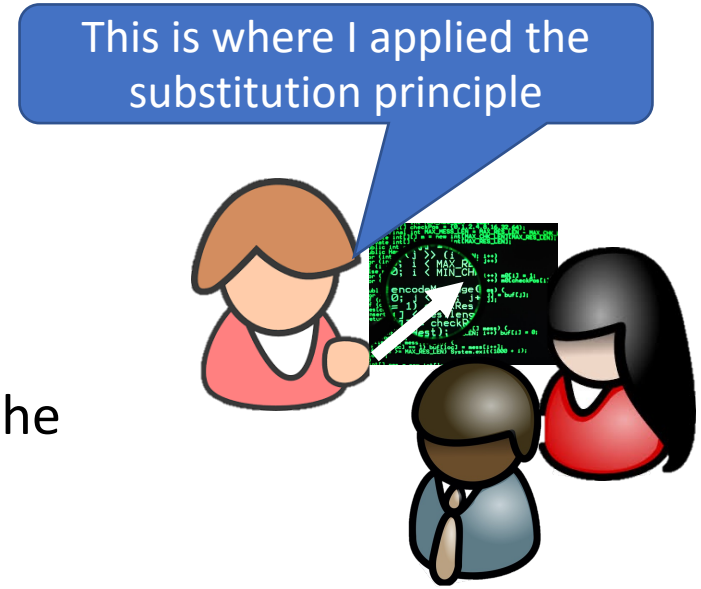
Code Reading

- Action:
  - Provide the code to 1-2 other people to read over
  - Discuss what they found in a small meeting
- Advantages
  - Provides more independent review time
  - More convenient review times (reviewers can read at their own pace)
- Disadvantages
  - Doesn’t ensure that code has been reviewed with the same degree of scrutiny





Code Walkthrough



- Actions:
  - Author-led meeting where the author explains the code or the design to a group of reviewers
- Advantages:
  - Efficient because the author knows the key elements
  - Effective when there are quite a few reviewers with diverse background
- Disadvantages:
  - Unintended bias

    Can lead the reviewers into the author's biases without them knowing
  - Unequal scrutiny of the code

    Author may focus what they are most proud of rather than the where the bugs might lie

Formal Code Review

- Actions:
  - Structured meeting with a clear focus / agenda
  - Not lead by the author of the code
  - Review team of 3-6 people

    Team size depends on the complexity of what you are reviewing
  - Distinct roles are assigned and performed by review members
  - All members of the review have prepared by reading the code
- Advantages:
  - Misses fewer defects than less focused methods
  - Can be done efficiently
- Disadvantages:
  - More time consuming

Formal Code Review Roles

- **Moderator**: Keeps the meeting on track (cannot be the author)
- **Author(s)**: Wrote the code being reviewed
- **Reviewer(s)**: Inspect the code as it is being read
  - Ask questions about it and point out defects
  - Take turns being readers
  - Cannot be the author
  - Different reviewers, may be assigned specific areas to watch. E.g.,
    - Layout and style issues
    - Integrity / security issues
    - Efficiency issues
- **Reader:** Reads the code out load and asks questions about it. (Not the author: unintended bias!)
- **Scribe**: records the location and severity of discovered defects
  - Can be merged with other roles

Checklist-Based Reviews

- It is useful to keep a checklist of common defects
  - Provides consistency in the review process
  - Allows the reviewers to focus on several categories of defects
  - Makes less experienced reviewers more effective
- Defect location(s) and severity are noted down in the appropriate row

CSCI 2134 Code Review Checklist[1]

Reviewers:			
Date:		Project:	
Files:			

Rating: (**A**) Occasional occurrence (**B**) Frequent occurrence (**C**) Pervasive (really bad)

Hygiene and understandability	Rating	Locations if present (File/Line #)
• Commented-out code fragments		
• Magic numbers (instead of constants)		
• Improper or inconsistent naming convention		
Formatting (adheres to code guidelines)	Rating	Locations if present (File/Line #)
• Inconsistent indentation		
• Inconsistent horizontal spacing: between operators, keywords		
• Inconsistent vertical spacing between methods, blocks, classes, etc		
Comments	Rating	Locations if present (File/Line #)
• Needless or redundant comments		
• No method headers to document parameters and functional dependencies		
• Inconsistent in format and level of detail		
Error Handling	Rating	Locations if present (File/Line #)
• Confusing or incomplete error messages		
• Unhandled boundary and edge cases (null, 0, negative)		
• Unvalidated use of parameters or inputs		
Code Decisions	Rating	Locations if present (File/Line #)
• Redundant code present		
• Inappropriate accessibility (public, private, etc.)		
• Unnecessary or redundant classes or objects		
• Variables not in lowest scope		
Logic	Rating	Locations if present (File/Line #)
• Out of bounds array indexes		
• Incorrect conditions in conditional statements and loops		
• Loops may not terminate		
• Division by zero possible		

[1] Based on https://courses.cs.washington.edu/courses/cse403/12wi/sections/12wi_code_review_checklist.pdf

Process of Formal Review

- Moderator arranges the logistics of the review
  - Books a meeting time and location
  - Distributes the code
- **Everyone reads the code before-hand**
- During the review
  - Reviewers take turns reading / paraphrasing the code
  - All reviewers are checking for consistency with desired goal/task
  - Defects are recorded to be fixed later; only discuss solutions which require design changes
    - Resist fixing the defects now
- After the review
  - **Inspection report** is produced and circulated
  - Author(s) are assigned to fix the defects
  - Organization should use reports as part of their software measurement process
  - An additional meeting may be scheduled to discuss solutions
  - Update checklists as needed

Example

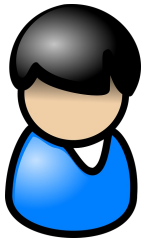
Adding to a linked list

- **add()** takes a key and a value and returns a boolean
- **added** is false, nothing has been added yet
- **found** is true
- Start at head of list, so **current** is **head**
- While key not found and not at end of list
- If current node has the same key as being added
- Set **found** to true

```
public boolean add(String key, int value) {
  boolean added = false;
  boolean found = true;
  Node current = head;

  // See if the key is already in the list.
  while (!found && (current != null)) {
    if (key == current.key) {
      found = true;
    }
    current = current.next;
  }

  // Add it to the list if not found.
  if (!found) {
    current = new Node();
    current.next = head;
    current.key = key;
    current.value = value;
  }
  return added;
}
```



author

reviewer

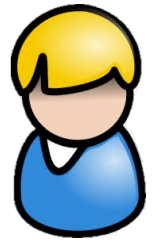
reviewer

- **H**as something been found initially?
- This loop will never execute!!!!

…

- head **is never set to current!**

moderator



scribe

- **found** is not initialized properly
- **head** is not updated

Video of a Formal Code Review

https://www.youtube.com/watch?v=JaJ_nNk3f1o

Key 

Points

- The earlier a software defect is identified (or avoided), the cheaper it is to fix
- Software quality objectives help a team work together to make good code
- Use a combination of techniques to find different defects at different stages of development
- Collaborative Software development gives more eyes, more points of view and a shared understanding but is more time consuming
- Collaborative Software development includes techniques like pair programming, informal code review and formal code review

Image References

Retrieved December 29 - 31, 2019

- https://consiliumeducation.com/itm/wp-content/uploads/sites/4/2017/01/african-2029984_1280-1014x586.png
- https://www.researchgate.net/profile/Mehmet_Celepkolu/publication/329855173/figure/fig2/AS:706489106841600@1545451537633/Pair-programming-setting-Students-look-in-different-directions-during-the-session.png
- https://cdn.clipart.email/d580b23d53775a26d05378f55de405fc_fun-coding-clipart-personal-website-sample-cilpart_800-600.jpeg
- https://www.safedeny.com/wp-content/uploads/2018/01/Secure-Code-Review.png
- http://pengetouristboard.co.uk/vote-best-takeaway-se20/
- https://www.twotwentyone.net/wp-content/uploads/2013/08/USB-receptacle.jpg
- https://i.pinimg.com/originals/b5/22/38/b52238fad11b0a3ecac36fa176041d98.jpg
- https://webstockreview.net/images/clipart-hospital-money-3.png
- https://s3-production.bobvila.com/articles/wp-content/uploads/2018/04/Reasons_Electrical_Outlet_Not_Working.jpg
- https://c7.uihere.com/files/109/173/249/software-quality-assurance-quality-control-quality-management-assurance.jpg
- https://i7.pngguru.com/preview/380/91/790/hourglass-time-clock-clip-art-vector-illustration-time.jpg
- https://1001freedownloads.s3.amazonaws.com/vector/thumb/133270/neoguiri_Barrier.png
- https://thumbs.dreamstime.com/z/six-components-project-charter-components-project-charter-159700743.jpg
- https://thumbs.dreamstime.com/b/compliance-rules-regulations-guidelines-arrow-signs-words-colorful-road-directing-you-to-comply-wih-important-laws-31478130.jpg
- https://www.researchgate.net/profile/Mehmet_Celepkolu/publication/329855173/figure/fig2/AS:706489106841600@1545451537633/Pair-programming-setting-Students-look-in-different-directions-during-the-session.png
- https://www.slideshare.net/ChihyangLi/object-oriented-programming-ch3-srp-dip-isp
- https://www.netclipart.com/pp/m/42-428110_working-clipart-collaboration-forming-stage.png
- Video: https://www.youtube.com/watch?v=JaJ_nNk3f1o

Retrieved September 23, 2020

- https://dilbert.com/strip/1995-11-13