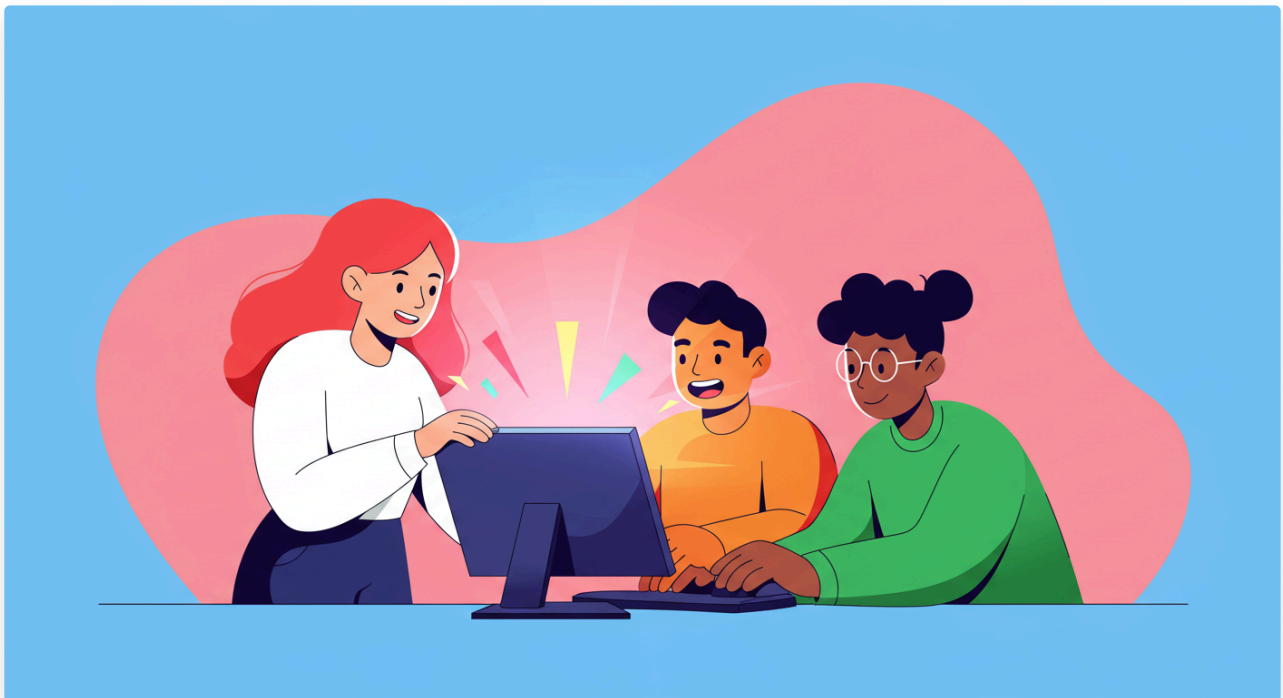


# Team Project and the Software Development Life Cycle

## Textbook

# Team Project and the Software Development Life Cycle



Get ready to work together on an exciting team project! You'll be building a program that does some interesting calculations with heights. This project will also help you understand how real-world software is made using something called the **Software Development Life Cycle (SDLC)**.

## Project Requirements and Timeline

You'll work in teams of **3 students** to create a program that takes the height (in inches) of 10 different people. To make it fair and easy to check your answers, your teacher might give your whole class the same list of 10 heights to use.

Here are the specific tasks for your team:

1. **Average Height:** One team member will write the code to find the *average height* of all 10 people. The final answer should be shown in **feet and inches** (e.g., "5 feet 8 inches").
2. **Tallest & Shortest:** Another team member will write the code to find the *tallest* person and the *shortest* person from the list. These answers should also be printed in **feet and inches**.
3. **Total Stacked Height:** The third team member will write the code to find the *total height* if all 10 people were stacked one on top of the other. This answer should also be printed in **feet and inches**.

**Bonus Challenge:** If you finish early, you could try to expand your program! For example, convert the answers to centimeters, meters, or yards.

## The Software Development Life Cycle (SDLC)

Building software isn't just about writing code; it's a process. The Software Development Life Cycle (SDLC) is a step-by-step method that helps teams create high-quality programs. Following these steps will guide your group project from start to finish:

1. **Planning:** This is where your team decides what your program will do, its main goals, and the general approach to achieve them.
2. **Define Requirements:** Get super clear on exactly what your program needs to do. For this project, you already have clear requirements: calculate average height, find tallest/shortest, and calculate total stacked height, all in feet+inches.
3. **Design the Program:** Before you write any code, brainstorm ideas! How will your program work? You might draw diagrams, create flowcharts, or even sketch out what the output will look like.
4. **Software Development:** This is the fun part – building the program! Each team member will write their assigned part of the code.
5. **Testing:** Once your program is built, you'll need to test it thoroughly. Does it work correctly with all the heights? Does it give the right answers? Try different combinations to make sure it's robust.
6. **Operations and Maintenance:** After a program is launched, it needs to be kept updated and any "bugs" (errors) that pop up need to be fixed. For your project, this might involve reviewing your code and making sure it's easy to understand for future updates.

By following these SDLC steps, your team will not only complete the project but also gain valuable experience in how software is developed professionally!

Okay, let's integrate instructions for a student peer review, including the basics of programming style, into the "Reviewing Your Computer Program" section.

Here's how that updated section could look:

## Reviewing Your Computer Program

After writing a program, reviewing it is vital for quality. This means checking your code, often with others, to ensure it works correctly, is understandable, and follows good practices—like proofreading an essay.

### Checking Functionality and Usability

First, check **program functionality**: Does it do what it's supposed to do? Test it thoroughly with various inputs, including unusual ones, to ensure accurate output.

Then, assess **program usability**: Is it easy for others to use? Are instructions clear, and is the output easy to read? A perfectly working program isn't useful if it's hard to understand.

### Following Programming Styles and Standards

A good program is also well-written. **Programming style** is how your code looks and is organized (clear names, comments, neat indentation). Consistent style makes code easier to read and fix.

Many languages have **style guides** (like Python's PEP 8) with rules for formatting. Following these is crucial for teamwork, preventing errors, and easier maintenance, ensuring everyone "writes" the same way.

## The Value of Peer Review: How to Do It

**Peer review** involves teammates checking each other's code. It's about helping improve, not just finding faults. A fresh perspective often spots mistakes or better approaches. This catches problems early, offers learning opportunities, and strengthens the team. Always give constructive feedback focused on the code.

## Instructions for Your Peer Review:

For your project, you will swap code with a classmate for a peer review. Here's what to do:

1. **Understand the Goal:** Before reviewing, make sure you understand exactly what your classmate's program is supposed to do based on the project requirements.
2. **Run the Program:** First, run your classmate's code.
  - Does it run without errors?
  - Does it produce the correct output for the project's requirements? Try testing it with some different inputs than what they might have used.
  - Is it easy to use? Are the questions clear? Is the output easy to read?
3. **Read the Code:** Now, read through their code line by line.
  - **Functionality Check:** Does the code logically follow the steps needed to solve the problem? Are there any parts that seem missing or unnecessary?
  - **Programming Style Basics:**
    - **Clear Names:** Are variable and function names easy to understand (e.g., `user_age` instead of `x`)?
    - **Comments:** Does the code have comments explaining tricky parts or key decisions?
    - **Indentation:** Is the code properly indented (usually 4 spaces per level in Python)? This is crucial for readability and correctness.
    - **Spacing:** Is there appropriate spacing around operators (like `a = b + c` instead of `a=b+c`)?
    - **Consistency:** Is the style consistent throughout the entire program?
  - **Efficiency (Optional but good to note):** Can you think of a simpler or faster way to achieve a certain part of the code?
4. **Provide Feedback:**
  - **Be Kind and Respectful:** Remember you are helping, not judging.
  - **Be Specific:** Instead of "this is bad," say "Line 15: The variable name `x` could be `user_input` to be clearer."
  - **Offer Solutions (if possible):** If you find an issue, suggest a way to fix it.
  - **Focus on the Code:** Keep comments about the code, not the person who wrote it.
5. **Discuss and Learn:** After reviewing, share your feedback directly with your classmate. Listen to their explanations and be open to discussing different approaches. Both of you will learn a lot from this process!

## A Plan

For your group project, your team needs to develop a comprehensive plan that outlines every critical aspect of your work. This plan should clearly define the project requirements, detailing what needs to be accomplished and the specific criteria for success.

Next, establish the structural design of your project, breaking down the main components and how they will fit together. You'll also need to provide realistic time estimates for each phase and task, ensuring efficient project management.

Finally, incorporate robust testing elements into your plan, outlining how you will verify that your project meets all defined requirements and functions as intended.

## Presentation

Present your team project for the class. Use appropriate voice and tone when speaking or writing. Show what your project does and discuss the process you took to get there. Share what you learned and what you would do differently next time. Share what worked and what didn't.

## Questions (5)

**1. In your team project, one person is tasked with writing code to find the total height of 10 elephants stacked on top of each other. This task falls under which part of the Software Development Life Cycle (SDLC)?**

MULTIPLE CHOICE

**Choose the correct answer:**

- A. Planning
- B. Defining Requirements
- C. Software Development (Coding)
- D. Operations and Maintenance

**2. Before your team starts writing any code for the height calculator, you all sit down to brainstorm ideas, draw diagrams, and sketch what the final program output might look like. Which SDLC step are you performing?**

MULTIPLE CHOICE

**Choose the correct answer:**

- A. Testing
- B. Design the Program
- C. Deployment
- D. Maintenance

**3. After your team has written the code for the height calculator, you try it out with different sets of heights to make sure it gives the correct answers every time and doesn't crash. Which SDLC step are you performing?**

MULTIPLE CHOICE

**Choose the correct answer:**

- A. Planning
- B. Software Development
- C. Testing
- D. Deployment

**4. When you review a classmate's code during a peer review, you notice that their variable names are confusing (e.g., "x" instead of "user\_height"). What aspect of good programming practice are they not following well?**

MULTIPLE CHOICE

**Choose the correct answer:**

- A. Functionality
- B. Usability
- C. Programming Style
- D. Debugging

**5. During a peer review, your classmate's program runs without errors, but the instructions for the user are confusing, and the final results are hard to read. What aspect of the program needs improvement?**

MULTIPLE CHOICE

**Choose the correct answer:**

- A. Functionality
- B. Usability
- C. Programming Style (Indentation)
- D. Code Comments

## Answer Keys & Solutions

### Questions

1. In your team project, one person is tasked with writing code to find the total height of 10 elephants stacked on top of each other. This task falls under which part of the Software Development Life Cycle (SDLC)?

MULTIPLE CHOICE

**Correct Answer:**

- |                                  |             |
|----------------------------------|-------------|
| A. Planning                      | ✗ Incorrect |
| B. Defining Requirements         | ✗ Incorrect |
| C. Software Development (Coding) | ✓ Correct   |
| D. Operations and Maintenance    | ✗ Incorrect |

**Explanation:**

Think about the phase where the actual program is built.

2. Before your team starts writing any code for the height calculator, you all sit down to brainstorm ideas, draw diagrams, and sketch what the final program output might look like. Which SDLC step are you performing?

MULTIPLE CHOICE

**Correct Answer:**

- |                       |             |
|-----------------------|-------------|
| A. Testing            | ✗ Incorrect |
| B. Design the Program | ✓ Correct   |
| C. Deployment         | ✗ Incorrect |
| D. Maintenance        | ✗ Incorrect |

**Explanation:**

Consider the step that involves planning how the software will work before coding.

3. After your team has written the code for the height calculator, you try it out with different sets of heights to make sure it gives the correct answers every time and doesn't crash. Which SDLC step are you performing?

MULTIPLE CHOICE

Correct Answer:

- A. Planning ✗ Incorrect
- B. Software Development ✗ Incorrect
- C. Testing ✓ Correct
- D. Deployment ✗ Incorrect

**Explanation:**

Think about the phase where you check if the program works correctly.

4. When you review a classmate's code during a peer review, you notice that their variable names are confusing (e.g., "x" instead of "user\_height"). What aspect of good programming practice are they not following well?

MULTIPLE CHOICE

Correct Answer:

- A. Functionality ✗ Incorrect
- B. Usability ✗ Incorrect
- C. Programming Style ✓ Correct
- D. Debugging ✗ Incorrect

**Explanation:**

Think about how code is organized and looks, specifically names.

5. During a peer review, your classmate's program runs without errors, but the instructions for the user are confusing, and the final results are hard to read. What aspect of the program needs improvement?

MULTIPLE CHOICE

Correct Answer:

- A. Functionality ✗ Incorrect

B. Usability

✓ Correct

C. Programming Style (Indentation)

✗ Incorrect

D. Code Comments

✗ Incorrect

**Explanation:**

Think about how easy or difficult the program is for someone else to interact with.