Code Management and Generative AI Best Practices
A Collaborative Review
EE450 Military Robotic Applications

Cadet 1
United States Military Academy
West Point, NY
First.Last@westpoint.edu
Keishun Pickett
United States Military Academy
West Point, NY
keishun.pickett@westpoint.edu
Cadet Amani Haskins
United States Military Academy
West Point, NY
amani.haskins@westpoint.edu
Cadet Taylor Brown
United States Military Academy
West Point, NY
Taylor.Brown@westpoint.edu
Cadet Charlotte Richman
United States Military Academy
West Point, NY
charlotte.richman@westpoint.edu
Cadet Meghan DeClue
United States Military Academy
West Point, NY
meghan.declue@westpoint.edu

***Abstract*—This is a collection of best-practice proposals by the cadets of EE450 Military Robotic Applications. The cadets use this document to share their perspectives, experiences, and recommendations on the use of software IDEs (specifically, Visual Studio Code and the Arduino IDE), robotics project code structure, and the use of generative artificial intelligence in completing robotics projects. This is a "by-cadets-for-cadets" living record that both encourages reflection and guides future cadets in their robotics endeavors.**

I. Instructions to Contributors

This document is divided into four parts:

1) Getting Started with Arduino Programming
2) Code Management and Structure
3) Integrating Generative AI
4) Testimonials and Examples

You may contribute to any or all of these sections as much or as little as you like. It is critical that your contributions remain *high quality* and *easy to understand*. Do your best to place your inputs in the correct sections and subsections, but feel free to create your own sections or subsections if you think you need to. You must not, under any circumstances, provide direct answers to any of the course projects, mini-projects, quizzes, or other assignments — the purpose is to guide other cadets on their own problem-solving, not to solve the problem for them.

Contributions to this document must be made via a *pull request* to the appropriate GitHub repository. This first requires you to *clone* the project repository and create a new *branch*. If you need guidance on how GitHub works, or how to submit a pull request, you can check the provided references. [1] [2]

II. Getting Started with Arduino Programming

A. Visual Studio Code

B. Arduino IDE

1) Installing Libraries: Libraries are a crucial component and starting point towards success in EE450. There are multiple ways to download libraries, which may be necessary to troubleshoot if you run into issues, however this will describe the most common two.

Library Manager:

1) The Library Manager in Arduino IDE looks like stacked books on the left sidebar. Other ways to navigate here are to click `Tools` and then `Manage Libraries` or click on `Sketch`, `Include Library`, and then `Manage Libraries` to navigate to the Library Manager window.
2) Within the Library Manager, you will see a search box. Typing in this search box will search all installed and available libraries within Arduino IDE.
3) Search for the library you need like Pixy2, Servo.h, etc. Once selected click "Install," allow the system to process and show that the library is installed.

Fig. 1. EE450 Best Practices Images

.zip Library: Sometimes during the class you will come across a library that is not included in Arduino IDE and will require you to source the `.zip` file. Here are the easiest steps:

1) Download the non-extracted version of the `.zip` file.
2) In Arduino IDE go to `Sketch`, then `Include Library`, and add the `.zip` Library.
3) Select the `.zip` library and open it. Wait for the program to confirm the library has been installed.

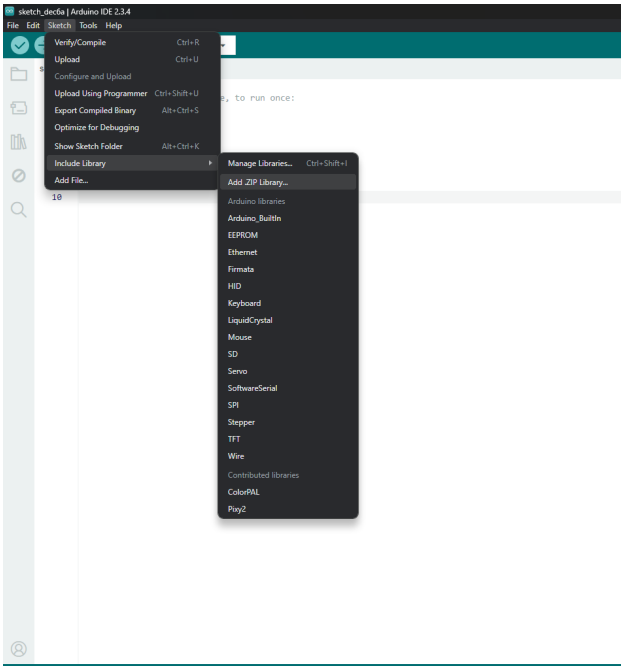

Fig. 2. Additional EE450 Best Practices Image

Note: Some libraries might need some troubleshooting. Extracting all files from the `.zip` and saving them in a folder on your computer, and then moving the library into the Arduino `libraries` folder on your computer, is a way to resolve this issue and try again.

III. Code Management and Structure

A. Incorporating the Sense-Decide-Act Paradigm

The Sense-Decide-Act Paradigm (SDA) is the most important framework to understand and apply for coding in this class. At the start of each project, start with a simple outline of the SDA framework. Typically, I would suggest looking at your block diagram and state diagram to identify what sensors you are using- this is how you know what you are ”sensing”. Your states from your state diagram will indicate your ”decide” criteria. To finish this, you must look at what must occur to move your robot from one state to another. For example, if you are using a PING distance sensor to detect and fire at a target, the distance is within your ”sense” section and choosing to fire at the target is your ”decide” criteria (i.e if object is within x distance of the bot).Lastly , the ”act” section is what your robot does within the state. For example, if the bot senses and then decides to fire, the action is the mechanism that you are firing with. This could be as simple as turning on a buzzer or illuminating an LED or even setting a laser pin high.Ultimately, SDA keeps your code organized and helps you identify what you need to code in each function.

B. Implementing States and Using State Diagrams

I highly recommend creating a state diagram and a block diagram at the start of every project. In my experience in EE450, state diagrams and block diagrams were some of the most useful tools in helping me complete each project. The state diagrams provided a clearly laid out visual representation of what the robot is supposed to be able to do, and how it should transition from one behavior or state to another. By mapping out each state, the conditions for each transition, and the actions associated with each state, I was able to generate my code and final product much more efficiently. State diagrams are extremely helpful at the start of EE450 projects even if they are not required/graded because they give you a clear visual of how your robot should behave. When you breakdown your system into states, it becomes much easier to understand what the robot is supposed to do at any given time and what events cause it to switch behaviors.

In EE450, your robot will almost always need to move between different modes (for example: `SAFE`, `FOLLOW`, `OBJECT DETECTED`, or `ATTACK/DEFEND`). A state diagram lets you map these different mdoes out before you begin to write any code.

For this project with these states, we used a state diagram to plan the overall flow of the robot. Before touching the Arduino code, we mapped out:

- what each state is supposed to do,
- what inputs trigger a transition (like a button press or sensor reading),
- and what the robot should do after switching states.

Having the state diagram made implementation much easier because the code could then be built one state at a time. Each state became its own section of the `loop()` function or its own block. If something was not functioning properly, it was easier to trace which state caused the issue. Especially when working with a new platform like the Traxxas, differentiating between hardware and software issues was eased by using a state machine diagram. Also, if it is demo day and you do not have a fully working robot, having individual states from your state diagram functioning can still earn you points. Breaking your code into clear states allows you to get portions of the robot working even if the whole system is not complete. This means you can demonstrate specific state behaviors and still receive partial credit instead of ending up with no points if the full integration is giving you trouble.

Overall, state diagrams can help you:

- understand the robot’s behavior before coding or asking chatgpt to help you build your code,
- write cleaner, more concise, and organized code,
- help you understand chatgpt generated code more efficiently,
- and troubleshoot faster when things don’t work.

They are a simple but an extremely useful tool, especially when your robot starts getting more complex. For some, in the earlier projects you may feel like the state diagram is unnecessary, however highly recommend getting some repetitions early on, because they are extremely helpful in later projects. Others may benefit from a state diagram from the start of Project 2 to the conclusion of Project 5.

The block diagrams are helpful for visualizing the overall system, including sensors, control logic, servos, buzzers, LEDs, and other components. These diagrams allowed me to break down complex behaviors into manageable parts, making it easier to implement and test each component individually before integrating them into the final system.

C. File Structure and Version Management — Avoid Drowning in Code

To help manage my code and avoid getting lost in what I was doing, I made sure to implement a lot of comments throughout my code. This helped me to remember what each line and section of code was executing and why the code was structured as it was. For example, if I was using multiple sensors within the same project, I would make a header comment at the top of the functions I called for each individual sensor so that the commands did not get confused and merged together, leading to error messages and bugs. Overall, comments helped me stay organized, helped me understand coding better, and made it easier to debug and correct mistakes as I went along in each project.

IV. Integrating Generative AI

The use of generative AI for EE450 projects is very helpful. However, there are use practices that are much more helpful than others. The Sense-Decide-Act framework is a great way to structure your prompts to AI. So, when you begin writing your code, ensure that you have already created a code ”outline” using the SDA framework. For my projects, I wrote a framework that was 50-80 lines of code starting with my equipment setup, pin mapping, and variable defintions. Then, I wrote out my functions with the actions I wanted each of them to perform. Then, I wrote out the decision criteria for each state and transition. Finally, based on any issues I anticipated, I outlined debounce functions, counters, and error handling. Once I had this outline, I used generative AI to help me understand the syntax and options I had for coding my functions. Anytime I used AI, I inserted this code outline to ensure that the generated code was something I understood and could implement. For big projects, I often told generative AI what type of function I wanted to build, and it helped figure out how to build it.

Additionally, generative AI was helpful for group projects. For project 4, my partners and I each coded a separate section or state. By the time we each had the our individual portions completed, we had a lot of code to integrate. By using gen AI, we identified that we needed specific criteria that would allow the STM to move between states based on sensor inputs. We used gen AI to help us with this integration.

Ultimately, generative AI is a great tool to help you understand coding syntax, explore function types and integrate pieces of code into one project. However, it is critical that you understand the components of the code that you are using so that you can effectively debug your project.

A. Registering for Copilot Pro

B. Managing and Integrating Generated Code

Using generated code from sources like ChatGPT, Copilot, etc., can be an extremely useful tool in working as a teammate to assist in fulfilling the project requirements and aiding in understanding the prompt. However, actually getting your robot to fully perform the task requirements will require you to understand the code. It is best to use AI to accelerate the progress, but not replace your understanding of solving the problem. Here are some recommended tips for prompting Copilot and other artificial intelligence platforms and managing the outputs:

1) Copilot is most effective when it knows exactly what you would like it to help you with. Think of components like: What does this code do? What variables are being considered as inputs, and what are the outputs? What are the specifications and constraints of the project, including what hardware is being used in the specific project? How should the states transition? Here is an example of a prompt:

   > “Build code for the DRIVE state that reads ultrasonic sensor distance and drives the servos forward only when distance is less than 30. Print sensor values to the serial monitor.”

2) Give Copilot the current state of your project. This means: What components have you already constructed in the process? By giving Copilot the pin mapping, defined variables, and completed state diagram, the code it generates will be easier to integrate into your own code.
3) What do you do with what it generates? Make sure you read it carefully and ensure there are no functions or variables that conflict with your existing code. Change some components of it, like names or parameters, so that it fits with your intended design. Test the code incrementally or by itself first, this means either test the code in a new sketch or add serial prints throughout the code to ensure the behavior is congruent with your intended design. Recognize that the produced code will not be perfect, but it will help show you different methods or structures of completing tasks.

Generated code can be very helpful in completing the more complicated aspects of a project and it helps you to save time in long projects. However, it is critical that you understand how the code is being implemented in your project to ensure that it is functioning how you want it to. One way to do this is to create an outline for your code first. Before prompting generative AI to write code for you, it is important to first create a test plan for your components and make sure that each piece properly works. For example, if you are using a distance sensor to detect objects, you should start with short test code to make sure the sensor works. This is a great way to use AI to help you save time through the component testing process. You can ask ChatGPT for a short sketch to test the function of each component and individually test them. Once you know that all of the components work, you can start to piece together your final coding sketch. Start by assigning each component in your robot to a pin and defining any variables you need. Make sure you include the proper libraries. Next, define the functions you want to include in your code that you implement from your state diagram. Once you have your structure laid out, you can start to prompt generative AI to help you fill in the functions you need. Generative AI is very helpful for helping you to structure your functions to work how you want. Make sure to provide AI with context on how you plan on using the function and how it relates to the other functions in your code. Lastly, use generative AI to help you debug your code. It can provide helpful explanations for why errors are occurring and provide guidance for how to fix them.

C. Documentation and Citation of Generated Code

When using generative AI to assist you in writing code for this class, it is essential that you properly cite the sources that you used for two primary reasons. The first, and most important, is maintaining academic integrity in your work. The second is to provide a record for yourself to understand what parts of your code were generated from AI and what sections were written by you. Oftentimes you will be able to recycle code from previous projects and modify it to meet the criteria of the project you are currently working on. By ensuring that you cite each section of code, it helps you to understand how the code works and how you can use it to help you better understand coding logic and implementation. It is crucial you do two things to properly document your code. The first thing you will do is create the citation you will include in your references page. You will create a citation in accordance with the DAAW that includes what model of AI

you used (for example: ChatGPT 5.0), the prompt you inputted into that chatbot, and then an explanation of how you implemented the assistance into your code. The second thing you will do is comment in your code exactly which lines were generated by AI. An example of this would be the following: assistance to the editor from ChatGPT 5.0, commented next to the line of code.

V. Testimonials and Examples

Feel free to add any other guidance, examples (good or bad!), or advice to other cadets here.

A. Advice from CDT Charlotte Richman

I highly recommend completing a test integration plan at the onset of each project. This plan should outline how you are going to go about testing each individual component of your robot, to include servos, sensors, LEDs, etc., and then how you will integrate all those separate components into one robot that functions as you want it to. This plan will save you a lot of time and frustration when you get into more complex projects and problems with many moving parts because you will know exactly what is and is not functioning properly before you are too deep in and have to backtrack, sometimes all the way back to the beginning. Having a clear plan for testing and integration will save you a lot of time and energy in the long run, and it will help you to understand what each component of your robot is doing, as well as how it is contributing to your entire system. I think that is pretty cool!

B. Advice from CDT Meghan DeClue

If I had to give one piece of advice, it would be to start with your state diagram for the project. Properly understanding the states that you want your robot to be in and how you want them to transition will make the coding process much easier. From there, I would recommend breaking down each state into a function embedded in your loop() function. This will help you to keep your code organized and easy to understand. By only keeping functions in your loop(), you can simplify your code and make it easier to read and debug in the case of having any errors. Complex projects tend to have hundreds of lines of code and can become hard to understand very quickly. Keeping your code organized by state and function will help to avoid confusion and it will make coding your project more efficient.

C. Advice from Amani Haskins

I have found that Arduino IDE and the libraries we import throughout this class have many tester files. I recommend that when you download a new library, you open ”File” and then ”Examples” to see the files that are included with the library. I would have saved myself a lot of time if I had used these example files earlier in the course. The examples will give you a basic set up for servo controlled robot like the Traxxas and Boebot. They will also demonstrate a basic set up for the Pixycam and other sensors that are commonly used. In my experience, when you are testing code, it also helps to copy and paste the function or section of code you are working on into a different file in Arduino IDE or in a testcode file. I have found that sometimes problems are easier to debug when you isolate the function/ state that the problem is occurring in. Lastly, I highly highly recommend using serial prints throughout your code to debug, especially when you are using multiple states. While states are not hard to build and test on their own, they are hard to track if you do not have an easy mechanism for it. For example, when you integrate a state, just simply write `Serial.print("in state x")` or use an LED and write `digitalWrite(LED_X, HIGH)` so you can keep track of your states during integration.

References

[1] GitHub Docs. About branches. https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-branches, 2025. Accessed 2025-11-14.

[2] GitHub Docs. About pull requests. https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests, 2025. Accessed 2025-11-14.