Protocol

FRC Team 3461 Programming Committee - 2023

Note: this is the first iteration of this protocol and is still under development, so may not be followed exactly as revisions are made.

1. Committee organization:
    a. Mentors
        i. Provide suggestions and insights, but do not push for any one idea
        ii. Provide help when requested
    b. Head
        i. In charge of mentoring newer students
        ii. In charge of reviewing and testing code
        iii. Has final say on ideas and disputes
        iv. In charge of maintaining organization and communication
    c. Programming Committee students
        i. To be a member of the committee, students must contribute at least 1 commit per two weeks of build season
        ii. Committee members must be involved in the actual writing of code, not just planning
    d. Other team members
        i. team members not on the committee should still attend planning meetings
            1. Committee members get to make decisions, but other team members' input is helpful
2. Github organization
    a. The repo will be named accordingly: frc-<year>
    b. Team members can request to collaborate via a google form containing
        i. Student name
        ii. Github username
        iii. Programming experience
    c. Branches

i. Will be used to organize features/subsystems
   1. Each new feature gets a new branch
   2. When the feature is ready for testing, students will submit a pull request
   3. The request will be reviewed and tested at specified times as per section 6.c by the committee head or others directed by the head
   4. If up to standard, the request will be accepted
   5. Otherwise, the student will receive feedback and an opportunity to work on issues with the head of committee or a mentor during the meeting

d. Releases
   i. Will keep track of stable versions and stripped down competition versions
   ii. Before each competition:
      1. Consult drive team and set up controls as they want
      2. Create a release from the master branch - single integer versions as follows: v<number>_complete
      3. Create a new branch called competition, and remove all unused code
      4. Create another release from this branch, call it v<number>_competition

e. Commits
   i. Use 3-letter commit tags at the start of commits (to quickly tell why a commit was made)
      1. “fet” - new feature
      2. “oiu” - operator interface update
      3. “eom” - end of meeting commit (to transfer changes to home computer)
      4. “crp” - competition fixes
      5. “quf” - quick fix
   ii. Feature commits should be made for every new method students write

                1. Commit messages will contain the file the change was made in, the name of the method that was written, whether it uses any other incomplete methods, and the files those incomplete methods are in.
                    a. For example: “fet Shoot execute, using incomplete Shooter”
                2. If students forget to commit after completing a method and write another one, they should combine the commit messages for both.
            iii. For changes involving user interface:
                1. Commit messages will list each binding that was changed, removed, or added, unless it exceeds the character limit
            iv. For changes involving drafting code as per 6.a
                1. A commit should be made for every file drafted.
                2. Commit messages will state the file changed, and list each method added, within the character limit
    f. Readme
        i. The github readme file will contain the following:
            1. Technical information about the code
            2. A link to the mission statement
            3. A link to this document
            4. A link to the form to sign up to collaborate
            5. Directions on contributing
            6. The most important information from this document, presented less formally
            7. A list of subsystems, and information about them
            8. Information about the OI of the robot

3. Meeting structure
    a. code testing schedule as follows (when circumstances allow), beginning when the drivetrain is functional
        i. Programming committee has access to the robot at the start of meetings, to test new code
        ii. During the meeting, changes and reviews can be made

        iii. At the end of the meeting, Programming Committee gets to test changes made during the meeting

    b. In addition to daily testing, the programming committee gets access to the robot for at least an hour on long saturday meetings, for major debugging.
        i. Additional time to be negotiated on a case by case basis.
    c. Committee planning
        i. The committee will confer for at least 15 minutes per team meeting, to maintain awareness about activity, and plan tasks
        ii. On Saturdays, longer conferences will take place, at least 30 minutes, to plan broader goals and assess progress.
        iii. The committee will keep track of all tasks and what is being worked on using airtable

4. Organization
    a. Use airtable to keep track of potential subsystems as follows:
        i. During meetings, discuss with the mechanical committee on what plans are being made
        ii. Write down potential subsystems in airtable as per 4.d.ii, with their components (motor controllers, etc.), and give each subsystem, and each component a percentage of how likely we think it is to end up on the robot.
            1. If we feel that the subsystem is at least 80% likely to end up on the robot, begin the coding process for that component
            2. Begin coding for each component of a subsystem that past this threshold once we are also 80% certain the component will be a part of the subsystem
    b. To plan the code needed for accepted subsystems
        i. In conference, make a list of all possible uses for the accepted components of the subsystem
            1. Organize them into must have, should have, and nice to have priorities
            2. Add to airtable as per 4.d.ii

        ii. Figure out what methods the subsystem will need to achieve as many use cases as possible - keep in mind extensibility
        iii. The committee head or someone delegated by the committee head will then draft the subsystem as per 6.a
    c. To plan commands
        i. List as many potential commands as possible
            1. Rank them relatively, in the order they should be completed
        ii. Draft each command as per 6.a
    d. Airtable use
        i. The programming committee should have it's own airtable base, to not clutter the main team one
        ii. There should be distinct tables as follows:
            1. A table for potential components, with their rank, and whether they meet the threshold as per 4.a.ii
            2. A table for each accepted subsystem and command, listing
                a. Every potential method for each one, with priority, as per 4.b.i and 4.c.i.1
                b. Each method's status:
                    i. Not started
                    ii. In progress
                    iii. Waiting for dependencies
                    iv. Finished
                c. What each method's dependencies are
        iii. A table for tasks related to planning
            1. Conference agendas
5. Code structure
    a. Naming
        i. Variables
            1. should be named in camelCase, except for constants (final)
            2. Do not sacrifice descriptiveness to try to achieve a shorter name

        3. Constants, using the final keyword, should be named with ALL_CAPS_UNDERSCORE_CASE
        4. Don’t use “m_” on local variables
    ii. Classes
        1. Should have descriptive names
        2. Don’t include whether the class is a subsystem or command in the name, since they are already separate
        3. Subsystem class names should be nouns
        4. Command class names should be verbs
    iii. Methods
        1. Always named in camelCase
        2. In subsystems
            a. Method names should either start with “get” or “set”
        3. Names should be as accurate as possible
        4. Try to be consistent with other names in the project where possible
    iv. Parameters
        1. Parameter names in constructors should match local variable names. Use the “this” keyword to distinguish between them, instead of adding “m_” to the class variable
b. general practices
    i. Always use private variables with getter methods to maximize safety
    ii. Make variables final wherever possible
    iii. All variables are initialized at the top of their respective block of code
    iv. The constructor should be at the top of the file
    v. Be mindful of ugly whitespace
c. File structure
    i. We will follow the default command based layout, with the following exceptions:
        1. Custom OI binding files in the main directory
        2. Additional folder in the commands folder for autonomous commands

6. Code writing process
    a. Code will first be "drafted" by either the head of committee or someone chosen by the head, as follows:
        i. Each method's body will be created, returning a garbage value
        ii. A javadoc comment will be written above it, detailing exactly what the function will do, so that anyone could write it.
    b. After most drafting is complete, anyone who wants can write a feature using the process described in 2.c
    c. Code reviews:
        i. Prior to drive team being established, the committee head or someone chosen by the committee head can review and accept pull requests to the main branch
        ii. Once the drive team is established:
            1. Code reviews require
                a. The programmer responsible for the changes
                b. Another member of the programming committee
                c. The driver and operator
            2. The programmer who made the changes explains the purpose of the changes, the other member will assess the technical correctness of the code, while the drive team will ensure the code meets their expectations for functionality
            3. When all members of the code review agree, the request will be accepted.
7. System testing
    a. In conferences, maintain a list of tasks to run with the robot to ensure all code is functioning as expected
8. Events
    a. Get firmware on all hardware up to date, and all wpilib software up to date.
    b. Create code release, as per 2.d
    c. Create list of tasks that need to be done at the event for setup (e.g. camera calibration)

            i. Make sure people who are attending the event are aware of potential problems

    d. Have safe code backup on two separate s.d. cards
    e. Have any objects needed for system testing, e.g. vision targets
    f. When code needs to be modified during an event
        i. Modify code between matches, upload code
        ii. Run system tests
            1. If tests fail, revert to backup s.d. Card or old version of code for the next match

9. priorities:
    a. Speed - Last year's code was slower than it should have been.
        i. Minimize loop times
        ii. Minimize code running in the main loop
    b. Reliability - This was pretty good last year, with the exception of vision calibration.
        i. Think about redundancy, failure cases
        ii. Plan a routine for events, to make sure setup goes smoothly
    c. Automation - The less drivers have to think about, the better… Make sure the robot is assisting the drivers in every way possible.
        i. Just make sure to prioritize tasks well, so more important things are done first.
    d. Simple controls - The Operator Interface should make as much sense as possible (of course, it’s whatever the drivers want, but the more options we can give them the better)
    e. Extensibility - Ideally, code should be able to be modified quickly under pressure, and reverted easily to its previous state, while maintaining readability - keep this in mind when planning.
    f. Consistency - make sure everything is laid out consistently, so files and variables are easy to find. This was not so great last year.
    g. FUN - don’t take things too seriously. Failure is fine, as long as you learn from it.

10. All programmers must wear a striped, blue, hand-knit beanie or face instantaneous expulsion from the committee (not really)