Chameleo Car: Proof of Concept and Software Demonstration Presentation Transcript

Project Repository: https://github.com/SEPM-Nov-2022/sepm-rc-car
Full Video: https://youtu.be/QjXFkpM8unI

I. **Slide 1: Title Slide**
   a. Hello. The Software Engineering Project Management course, November 2022, is pleased to present our project. Our group consists of Marianne Manaog, Alberto Rossotto, Djordje Savanovic, and I'm Rob Mennell.
   b. Our toy concept is the "Chameleo Car."
   c. This is a smart phone-controlled toy race car capable of changing colour and customising its appearance with the goal of encouraging diversity in racing.

II. **Slide 2: Concept Introduction**
   a. Currently, racing leagues fail to represent the diversity of our population (Mitchell-Malm, 2021; Reid & Lightfoot, 2019; Howe, 2022).
   b. By creating a toy that 1) appeals to all children, and 2) shows that drivers can look like anyone, we hope to grow and recruit drivers as diverse as the world around us.
   c. What we have, today, is software that enables functionality of the car, and a simulation environment to demonstrate these functions.

III. **Slide 3: Requirements: 'Child' Persona**
   a. Before we began any design or development activities, we first generated requirements for the toy's software.
   b. To do so, we used a persona-based framework as per best practices in software engineering (Nielsen & Nielsen, 2019).
   c. After considering the many potential use cases, scenarios, and relationships with this toy, we settled on three personas:
      i. 'Child,' the primary user of the toy, likely between 6 and 12 years of age.
      ii. 'Guardian,' the parent, caretaker, or responsible party monitoring the child's interaction with the toy.
      iii. 'Producer,' the manufacturer of the final physical toy and developer/distributer of the accompanying smart phone app.
   d. Unfortunately, we did not have access to child users, so we researched features of comparable toys on the market.
   e. Due to not having access to these users, there is inherent risk that these requirements do not reflect user wants and needs.
   f. Requirements are prioritized in order of 1) functional dependencies as aligned with our mission and 2) predicted user enjoyment.

IV. **Slide 4: Requirements: 'Guardian' and 'Producer' personas**

a. As with the 'child' persona, we did not have access to Guardian or Producer users, so we conducted additional research to identify the primary equities of these personas.
b. Guardian requirements ensure that the child has a safe and nurturing experience with the toy.
c. Producer requirements enable troubleshooting and future development.

V. **Slide 5: Design and Plan**

a. We applied object-oriented design principles to this application in terms of classes, attributes, methods, and relationships (Phillips, 2018).
b. After researching similar applications, we decided to leverage common game functions from the widely-adopted pygame library (Pygame, 2022). Alexander Svilarov's "Race It!" app inspired our vehicle control, physics, and background implementation (2019).
c. **Alberto** will expound on the design and plan.
d. The three main elements of the simulation are the remote controller, the car, and the analytics mock server.
e. The class Remote models the remote control: it connects to the car, sends the commands, and manages the analytics. It simulates the mobile app responsible for car control and analytics.
f. The class Car models the RC car: it translates inputs into actions moving the car in the simulation area. It encapsulates the data relative to motion and the battery. The battery system has some complexity that deserved a separate class, Battery, to properly separate the dynamics of driving from the logic of consuming energy.
g. To create a more realistic model, the class Game contains all the logic of the simulation, such as most of the pygame functionalities, the game's main loop, and all the elements concerning graphics and audio. The class Game simulates the world where the car lives. For example, the Game's function check walls is a Callable used by Car to detect collisions. Similarly, the functions notify and play audio are callables used by the class Car and the class Remote to interact with the world by sending messages or sounds.
h. The class Car implements the action that can be performed by a car such as steering and accelerating. All the interaction with a car goes through the method command. The only other public methods are get_battery_level to provide info about the battery, and handshake remote to confirm the connection when there is enough charge.
i. The class Remote receives the inputs from the simulation (the class Game) and forwards them to the car. It is also responsible for the storage of analytics.
j. The analytics are managed by the class Analytics which stores the data locally and synchronises with the remote server periodically.
k. The class AnalyticsStorage is a separate component to make it easier to mock the I/O during the tests.
l. An Analytics Mock Server is available for testing. It is a simple Flask-based web application echoing the messages in the shell.
m. Overview of the remaining scripts

   i. The script Constants contains all the constants to simplify the maintenance.
   ii. Logger configures the application's logging system.
   iii. Utils contains a utility method that returns the configuration in environment.yml.
   iv. RC Car Launcher is nothing more than the simulation's entry point.

n. Running the tests
   i. There are the following shell scripts:
      1. run-tests.sh: executes all the unit tests.
      2. run-cucumber.sh: executes all the bdd tests.
      3. run-checks.sh: generates the reports from the linters.
      4. The reports are available in the folder reports.

o. Analytics
   i. The project included a remote Analytics server able to store data from multiple cars, if not multiple types of toys. For the development, the team created a mock server mimicking the real one in order to test the functionalities.
   ii. The Analytics Mock Server is a simple Flask application. Its only function is to expose a REST endpoint and echo the received message in the logs. The class Remote stores locally the analytics and periodically synchronize with the remote server. The class Analytics stores the session's data in the folder analytics creating one file per session. The class Analytics continuously updates the file and deletes it if the synchronization is successful.

VI. Slide 6: Sprint Progress and Project Status

a. This slide shows a sprint-level view of our software development plan.
b. Using Agile principles, we allocated requirements across three fortnightly sprints for this initial release.
c. Due to complications with the parental control menu, the requirements for the Guardian persona were shifted to Release 2.
d. Our progress was enabled by mapping dependencies during the planning phase, adhering to the development plan, and properly testing code before building upon it.

VII. Slide 7: Budgets and Summary

a. As a team of seasoned IT professionals, we used a combination of methods to estimate our labour allocation. These methods included top-down, bottom-up, and expert judgment estimation techniques (Nasir, 2006). We considered our total time and resources available, level of effort per task, and experience with similar projects (Nasir, 2006). The budget reflects standard commercial rates for custom software development.
b. Our team did not limit a member to a specific role, as each person contributed to multiple functions. Thus, we made an effort to track our time according to the tasks performed. Our efforts were largely on track with our initial projection.

VIII. Slide 8: Development of Demo/Simulation

a. **Marianne** will elaborate upon testing throughout the development lifecycle.
b. Testing is a key and iterative component of software engineering projects that adhere to the Agile methodology and spans across the entire software development lifecycle (SDLC) (Gaikwad *et al*., 2017) on the following environments:
    i. development, used to create the functionalities of the application, and related unit tests to verify their correct behaviour,
    ii. testing, leveraged for running automated tests to guarantee the expected behaviour of the functionalities developed against the product requirements,
    iii. user acceptance testing, with focus on the key users outlined in the design document, i.e., 'child', 'guardian', and 'producer',
    iv. production, simulating a live environment where further users may use the application differently, thus covering a broader set of scenarios.
c. Thus, testing involves various phases, such as:
    i. requirements testing (Nielsen & Nielsen, 2019),
    ii. functional testing to ensure the application runs as expected,
    iii. unit testing to verify the behaviour of the individual components of the application,
    iv. security testing to identify any security vulnerabilities in the project's dependencies and mitigate them (if any), and
    v. quality checks to ensure a consistent and production-grade code quality.
d. Impact of testing on coding
    i. The main challenge is testing components performing IO, like those interacting with Pygame. Some components, like the Analytics class, are designed to separate the logic from the IO dependency that is injected separately. The test can inject a mocked object and intercept the calls to verify that the logic components triggers the right IO operations.
e. Key testing technologies
    i. The following technologies were leveraged for testing:
        1. Gherkin for requirements testing,
        2. pytest and pytest-cov for unit testing, including reporting test coverage in percentage and any missing/uncovered lines,
        3. bandit and safety for security checks, assessing vulnerabilities in the dependencies to determine whether adding them to the project, thus enabling security by design (Kreitz, 2019), and
        4. pylint, pycodestyle and pyflakes for linting based on code quality checks.
f. Unit tests
    i. Unit tests, which tests the logic of the key components in the application in a white-box manner (Xie *et al*., 2016), have been added inside the tests folder, which follows the structure of the rc_car directory of the source codes. Once outside the tests folder, i.e., in the top-level directory, unit tests can be run via pytest and pytest-cov and a comprehensive test coverage report with test coverage in percentage and missing/uncovered lines can be obtained.
g. GitHub Actions for CI/CD
    i. GitHub Actions were used for CI/CD (Klotins *et al*., 2022), thus bringing all above-mentioned testing stages together in a single automated pipeline, which also

ensures the visibility of their results on related reports, as it is fully integrated within GitHub, and it is run at every push, including merges into the main branch of the repository following an approval and a merge of any pull requests (PRs).

h. **Djordje** will walk us through some of the user acceptance tests.
    i. My name is Djordje Savanovic and I will be presenting some of the requirements for the Child persona of the Chameleo Car.
    ii. In order to start the game, I will open the terminal window and navigate to the game folder. In this terminal, I will run the 'run.sh' command which will start the game and all of the supporting processes.
    iii. As you may see, once the game started, we have the map and the car. To demonstrate how the game works, I will go through the requirements of the Child persona.
    iv. The first requirement is to control the race car's movement. This is a functional requirement with high priority. We can achieve this requirement by utilizing the arrow up, down, left, and right keys. If I press up, the car will start moving forward. If I push the arrow down key, the car will start going backwards in reverse. By combining the arrow up and left or right, I can steer the car left or right. As you may see, I can control the car's movement and play the game.
    v. The fourth requirement is to change the race car's colour. This is a functional and high priority requirement for the Child persona. To achieve that, I will press the 'C' key on the keyboard. By pressing the 'C' key, the car will change colour. The initial colour is red, if I press the 'C' key, as you can see, the car changes colour to green. If I press it again, the car turns blue, and if I press it again, the car turns to the original red colour.
    vi. The fifth requirement for the child persona is to customise the driver's appearance. It is also a functional high priority requirement, which I will demonstrate now. As you may see, in the top-right corner, this is where the user's appearance or picture is located. In order to change the initial picture, one must push the 'M' key on the keyboard, as I did right now, and this would open a menu window where you can see it says, 'Choose your profile picture.' The game is offering four different possibilities. If I select the first one, I hover the mouse pointer over it and left mouse-click. If I click that, as you may see, it changes the driver's appearance. If I press the 'M' key again, it opens the menu window, and I can select the other three profile pictures. Let's continue with the last one and I can continue using the game. As you can see here, the car is moving.
    vii. The final requirement for the child persona is to send an alert when battery is lower than twenty percent. This is a non-functional, low priority requirement. To demonstrate, I will keep moving the car until the battery drains.
    viii. We just experienced the battery low sound. One thing to notice is that the colour of the battery bar has changed from yellow to green, indicating that the battery needs to charge. Thank you.

IX. Slide 9: Challenges and Solutions

a. Though there were many difficulties along the way, we had three primary challenges to overcome.
b. Having a geographically disbursed team, with six hours of time difference, required flexible and persistent communication. Slack and GitHub integrated our efforts.
c. Developing software remotely, using different operating systems and environments, required additional testing and setup configuration. We implemented a modular structure and a common environment.yml file which standardised setup and minimised the room for configuration errors.
d. Initially, linting checks were run manually, but automating them via GitHub Actions enabled early visibility and correction of errors to increase code quality.

X. Slide 10: References

a. A variety of sources, listed here, helped us with during this assignment.
b. This concludes our presentation. We appreciate the opportunity to brief our project and thank you for the constructive feedback throughout this course.

References

Gaikwad, V., Joeg, P., & Joshi, S. (2017) AgileRE: Agile requirements management tool. In Proceedings of the Computational Methods in Systems and Software (pp. 236-249). Springer, Cham.

Howe, O. R. (2022) Hitting the barriers–Women in Formula 1 and W series racing. European Journal of Women's Studies 13505068221094204.

Klotins, E., Gorschek, T., Sundelin, K., & Falk, E. (2022) Towards cost-benefit evaluation for continuous software engineering activities. Empirical Software Engineering 27(6): 157.

Kreitz, M. (2019) Security by design in software engineering. ACM SIGSOFT Software Engineering Notes 44(3): 23-23.

Mitchell-Malm, Scott (2021) “Hamilton Commission Reveals Stark F1 Diversity Findings”. The Race.com. Available from: https://the-race.com/formula-1/hamilton-commission-reveals-stark-f1-diversity-findings/.) [Accessed 22 Jan. 2023].

Nasir, M. (2006). A Survey of Software Estimation Techniques and Project Planning Practices. [online] IEEE Xplore. doi:10.1109/SNPD-SAWN.2006.11.

Nielsen, L., & Nielsen, L. (2019) Making Your Personas Live. Personas-User Focused Design 161-170.

Phillips, D. (2018). Python 3 Object-Oriented Programming. 3rd Edition. [Insert Publisher Location]: Packt Publishing.

Pygame. (2022). [online] Available at: https://www.pygame.org/.

Reid, M. B., & Lightfoot, J. T. (2019) The physiology of auto racing: a brief review. Medicine and science in sports and exercise 1-15.

Svilarov, A. (2019). Race It! - 2D Racing Game. Available from: https://appoftheday.downloadastro.com/app/race-it-2d-racing-game/ [Accessed 12 Jan. 2023].

Xie, T., Tillmann, N., & Lakshman, P. (2016) Advances in unit testing: theory and practice. In Proceedings of the 38th international conference on software engineering companion, 904-905.