STEVENS
INSTITUTE OF TECHNOLOGY
1870

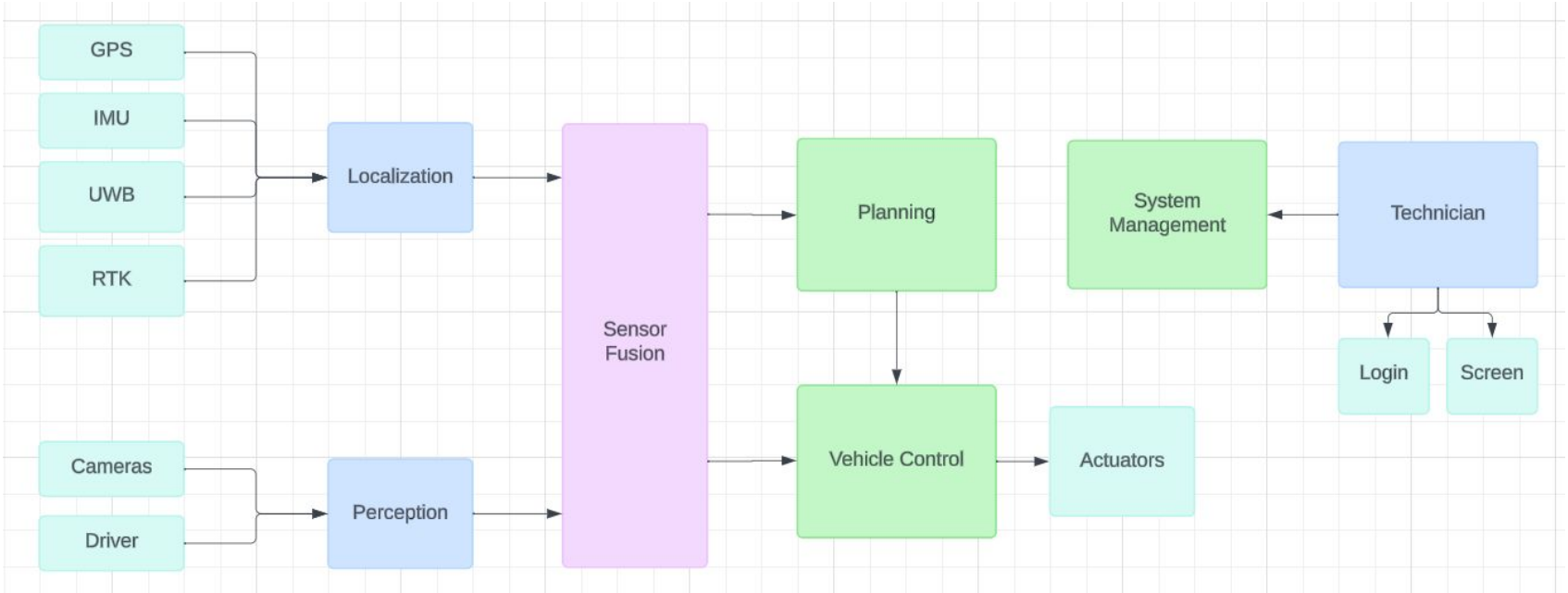
ALSET IoT Hug The Lanes

Haig Emirzian, Michael Buzzeta, Christos Zervas, Nicolas Buendia

Our Drive

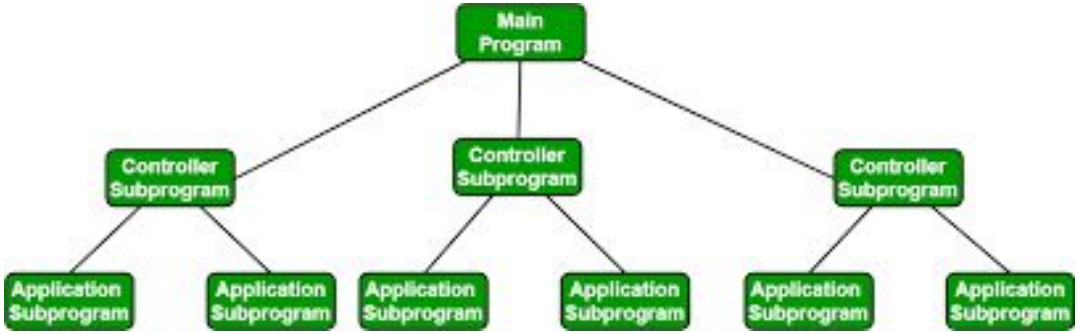
- **Mission:** Find a path to human assistance driving and Level-5 autonomy
- **Use Cases:**
  - The disabled
  - Older adults
  - Unlicensed individuals
  - Anyone else regardless of background
- **Value:**
  - Higher utilization
  - Cheaper ride fares
  - Unrealized business opportunities

Software Architecture

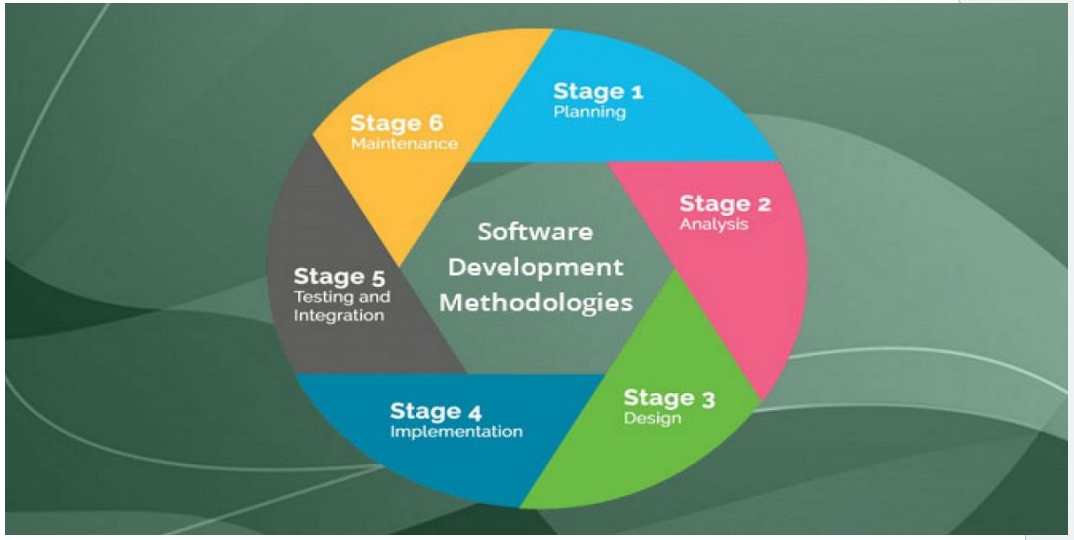


Software Development Process

- Importance of Choosing the Right Architecture
- Considered Architectures:
  - Data Centered
  - Data Flow
  - Call Return (Chosen)
  - Object-Oriented
  - Layered
  - Model View Controller
  - Finite State Machine
- Rationale for Choosing Call Return Architecture
  - Hierarchical Structure
  - Subprogram Organization
- Benefits for Our Project



Component-level Design



- Breakdown of Functionalities:
  - Automatic Parking
  - Acceleration Launch
  - Obstacle Avoidance
  - Traffic Sign Detection
  - Cruise Control
- Each Component's Responsibilities:
  - Scanning, Planning, and Executing
  - Analyzing Road Conditions and Launch Control
  - Detecting and Evading Obstacles
  - Recognizing Traffic Signs and Adjusting Behavior
  - Maintaining Vehicle Speed
- Independence and Collaboration Within Architecture

Reflection on Project Ohm: How did we do?

Challenges we faced

Despite our strong finish, Project Ohm had challenges along the way:

1. **Integration Complexity:** One challenges was integrating various modules and ensuring they work seamlessly together, without having unneeded redundancy within our modules. Each module, such as CrashDetection, CruiseControl, etc., has different dependencies, requirements, and scopes that needed to be managed.
2. **Testing Complexity:** Testing possible scenarios and edge cases for each module was time-consuming and required significant effort.
3. **Resource Management:** Allocating resources efficiently, including time, manpower, and budget, was a challenge. Ensuring the project stays on track and within budget constraints was difficult when having to make many charts, diagrams, tests, and most other deliverables from scratch.

Excellencies

Challenges breed innovation and creativity, and so the project exceeded expectations in several areas:

1. **Modularity:** The project is well-structured with modular components (classes) for different functionalities, which promotes code reusability and maintainability.
2. **Clear Requirements:** The requirements for each module are well-defined, making it easier to develop and test each component independently.
3. **Testable Code:** The use of unit testing in our test cases demonstrates a commitment to quality assurance and ensures that individual components perform as expected.
4. **Functionality:** The project covers a range of functionalities related to vehicle systems, from navigation to crash detection, which is comprehensive and crucial for a self driving vehicle.
5. **Innovation and Creativity:** Encouraging a culture of innovation and creativity within the team lead to the generation of novel ideas, solutions, and approaches to address the challenges faced and enhance project outcomes. (i.e. sequence diagram tools discovered, architecture decisions)

Improvements for the next version:

We can improve on or add:

1. **Code Reusability:** Identify common functionalities across modules and refactor code to promote reusability. For example, common methods like check_preconditions could be abstracted into a utility class.
2. **Automated Testing:** Implement automated testing frameworks (e.g., pytest, unittest) for more extensive test coverage and faster regression testing, also reducing cost.
3. **Continuous Integration/Continuous Deployment (CI/CD):** Set up CI/CD pipelines to automate the build, testing, and deployment processes, reducing manual effort and ensuring consistent quality thus reducing cost.
4. **Documentation:** Enhance documentation, including code comments, API documentation, and user guides, to improve code understanding and facilitate onboarding for new developers cutting down on training times translating to lower developer costs.
5. **Performance Optimization:** Identify performance bottlenecks through profiling and optimize critical sections of code to improve overall system performance.
6. **Hardware Emulation/Simulation:** Use hardware emulation or simulation tools where applicable to simulate real-world scenarios.

Demo

Automatic Parking

```
class AutomaticParking:
    def __init__(self, parking_button):
        self.parking_button = False


    # Activate automatic parking feature
    def AutomaticParkingSystem(self):
        self.parking_button = False
        self.motor_output = 10
        self.traction_control = 100


    # Verify car state
    def ParkedModule(self, parking_button):
        self.AutomaticParkingSystem(parking_button)
        if self.parking_button:
            print("Car is parked successfully")
        else:
            print("Car could not park successfully")


    # Detect parking spaces and obstacles
    def Cameras(self):
        return [0, 1, 0, 0, 1]

    # Find parking spaces to use
    def Planning(self):
        if 0 in self.Cameras():
            print("Parking space detected")
        else:
            print("No parking space detected")

    # Control vehicle movement for parking
    def VCS(self, motor_output, traction_control):
        self.AutomaticParkingSystem(motor_output, traction_control)
        if self.motor_output and self.traction_control:
            print("Vehicle is moving")
        else:
            print("Vehicle is not moving")
```

Obstacle Avoidance

```
class ObstacleAvoidance:
    def __init__(self):
        self.obstacle_data = []
        self.evasive_maneuver = 0


    # Detect obstacles
    # Analyze data for evasive maneuver
    def ObstacleAvoidanceSystem(self, obstacle_data, evasive_maneuver):
        self.obstacle_data = obstacle_data
        self.evasive_maneuver = evasive_maneuver


    # Receive obstacle data
    def SensorFusion(self, obstacle_data):
        self.obstacle_data = [0, 1, 0, 0, 0, 1, 1]
        return self.obstacle_data


    # Calculate evasive maneuver
    # 5 represents a level 5 evasive maneuver
    def Planning(self, evasive_maneuver):
        self.evasive_maneuver = 5
        return self.evasive_maneuver


    # Control steering, throttle, and brake
    def VCS(self, steering, throttle, brake):
        if steering and throttle and brake:
            print("Vehicle is avoiding obstacle")
        else:
            print("Vehicle is not avoiding obstacle")
```

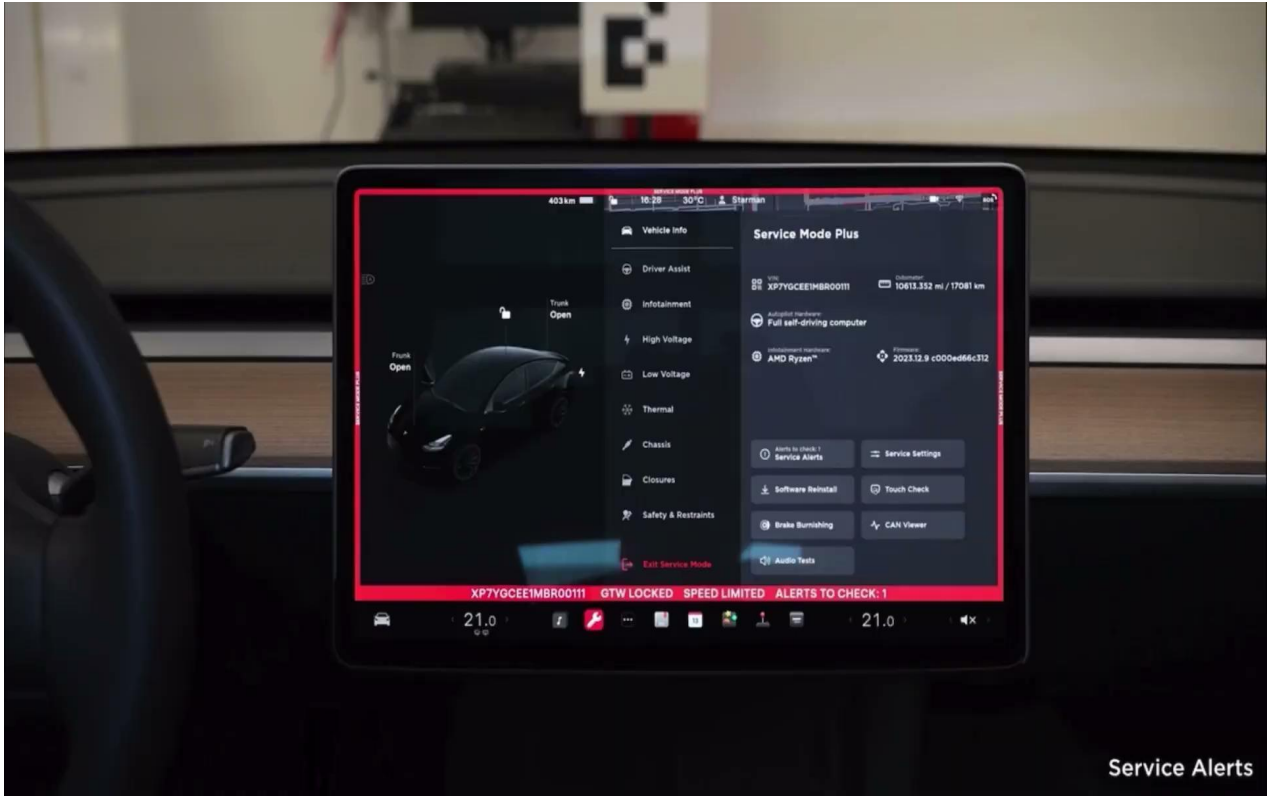
Forms of testing conducted:

- Integration Testing
- Functional Testing
- Performance Testing
- End-to-End Testing

Demo

Demo

Demo


Service Mode Plus
Vehicle Info
Driver Assist
Infotainment
High Voltage
Low Voltage
Thermal
Chassis
Closures
Safety & Restraints
Exit Service Mode
Trunk
Open
Frunk
Open
XP7YGCEE1MBR00111
10613.352 mi / 17081 km
Full self-driving computer
AMD Ryzen™
2023.12.9 c000ed66c312
Service Alerts
Service Settings
Software Reinstall
Touch Check
Brake Burnishing
CAN Viewer
Audio Tests
XP7YGCEE1MBR00111 GTW LOCKED SPEED LIMITED ALERTS TO CHECK: 1
21.0
21.0
Service Alerts

STEVENS
INSTITUTE OF TECHNOLOGY
1870

THANK YOU

Stevens Institute of Technology
1 Castle Point Terrace, Hoboken, NJ 07030