Appified AV Platform
- **Appified** autonomous vehicle (AV): self-driving functionalities are developed in a crowd-sourcing manner and installed as modules
- Traditional vehicle architecture:
  - Outsourcing
  - Hardware suppliers
  - Heavy testing work
  - Usually takes over 24 months

Appified AV architecture:
- Crowd-sourcing
- Software suppliers
- Software testing
- High requirement on the functionality

Example: Path-Following AV App on PolySync
```java
class PathFollowingNode {
    //Subscribe on the vehicle report
    registerListener("VEHICLE");
    //Load the trajectory data
    map = loadMap("map.dst");
    void MessageHandle(Message msg) {
        Position pos = msg.position;
        for (i = map.size(); i--;
            //Calculate distance and heading errors
            distError = getError(pos, map[i]);
            headingError = getError(pos, map[i]);
            Message message = new Message();
            //Set the adjusted steering angle
            if (distError > MAX_DIST_ERR &&
                headingError > MAX_HEADING_ERR) {
                message.setStoringAngle(_angel); message.publish();
            }
            //If vehicle is not in path, perform
            //emergency stop with maximum throttle
            else {
                message.setThrottleGain(180); message.publish();
            }
        }
    }
}
```

Overview: AVGuard Approach
- **Standard App Development**
  - Source code
  - Allowable circumstances
  - Description

- **Static App Vetting**
  - Checking with Principles
    - Traffic Rule
    - AV Standard
    - Cybersecurity Policy

- **Dynamic App Vetting**
  - Trace-driven Simulation
    - Naturalistic Trace
    - Worst-case Scenario
  - Accelerated Evaluation

Step One: Standard App Development
- Facilitate app vetting by requiring developers to provide
  - Purpose statements
    - Matching user expectation with app behavior; Justifying access control decisions
  - Required resources
    - E.g., exclusive usage of throttle pedals, radar, or required network bandwidth
  - Usage constraints
    - Required by the DoT standard for autonomous driving functionalities
    - E.g., high-way only, sunny day only (with clear camera vision)

Step Two: Static App Vetting
- Facilitate app vetting by requiring developers to provide
  - Purpose statements
  - Matching user expectation with app behavior; Justifying access control decisions
  - Required resources
    - E.g., exclusive usage of throttle pedals, radar, or required network bandwidth
  - Usage constraints
    - Required by the DoT standard for autonomous driving functionalities
    - E.g., high-way only, sunny day only (with clear camera vision)

Step Three: Dynamic App Vetting
- Accelerated Evaluation Approach
  - Based on importance sampling techniques
  - Quantifies potential risks of AV apps, and runs off-the-vehicle.
  - Reduces the required test mileages for each analysis by a factor of 10,000 to 100,000.

Case Study & Discussion
- Dynamic analysis revealing potential risks of the path-following AV app
  - Current appification status in AV field
    - Industry: pioneering companies to open critical access of vehicles to researchers
      - e.g., Ford XC, PolySync
    - Academia: U-M to provide OpenAV to researchers; Udacity courses to targets for developing library of open source AV functionalities.

- Ongoing Work
  - Complete the implementation of AVGuard to be used for students taking Automated Vehicle Control course in 2017 provided by U-M
  - Collecte rules and principles to be added to the static app vetting process using crowd-sourcing.
  - Build database for common faults made by developers