Fault-Tolerant Quadcopter

ECE 453 Project Proposal (Fall 2018)
University of Wisconsin-Madison

Vaughn Kottler, Mayank Katwal, Cooper Green

1 Abstract

This document serves as the formal, semester-project proposal to be vetted by the course instructor: Joe Krachey. We have additional online documentation that we plan to keep in sync with our project's scope and current progress.

2 Executive Summary

2.1 Problem Statement

Seemingly no "multi-vehicle fleet" or autonomous-flight capable drone technology is available to interact with in the consumer market. Similarly, open-source hardware and software in the "DIY drone" ecosystem cannot be easily converted or augmented to solve that problem.

We identified this problem while exploring learning objectives related to developing skills for working in aerospace avionics and software. We believe that gaining relevant experience with this problem space requires designing, testing and building a custom flying machine.

2.2 Proposal

We intend to build a **quadcopter** plus **ground station** and **web-based user interface**. We have chosen to call this project the **fault-tolerant quadcopter**. This name reveals one of our design goals that will be covered in a future section.

With respect to the problem statement (and single-semester timeline), we do not anticipate delivering a polished technical solution to any of the specific deficiencies we identify in the "free and open-source" ecosystem. We have tentatively planned to continue working on this project until our source repository on GitHub represents completed work to a well-defined scope that may differ from this course project's scope.

2.3 Concept of Operation

The essence of what we would like to create is best captured by **Figure 1**.

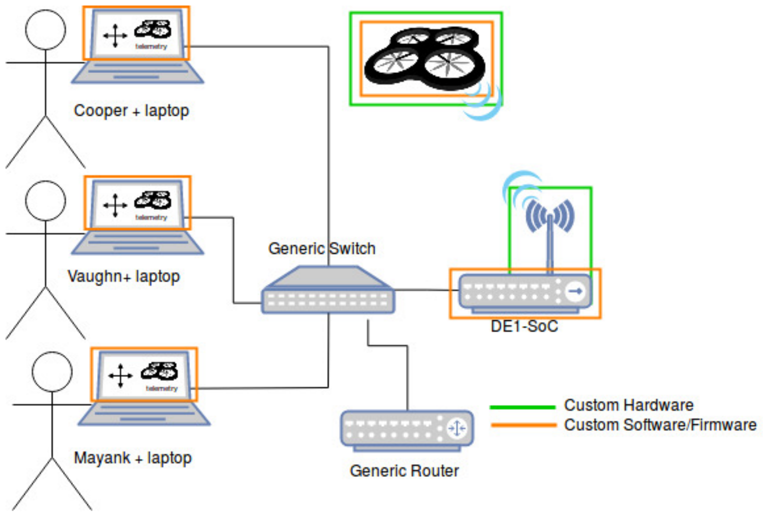


Figure 1: The intended physical configuration.

Our presentation content goes into more detail about how this architecture fundamentally differs from what is available in the consumer market. We believe the strength of this topology is fully realized during final integration stages and the development of actual control algorithms for flight. A graphical interface displaying diagnostic information is essential to vehicle development of any kind and exposing this interface directly to our workstations opens up opportunities for extremely tight development loops (imagine if you could change configurations or even upload new code mid-flight via this interface). We emphasize that the user interface is not just a convenient or heavily styled view of data but a tool we will use throughout the lifecycle of this project.

2.4 Overview

This project is designed for three major bodies of work that were mentioned above but are better captured by **Figure 2**:

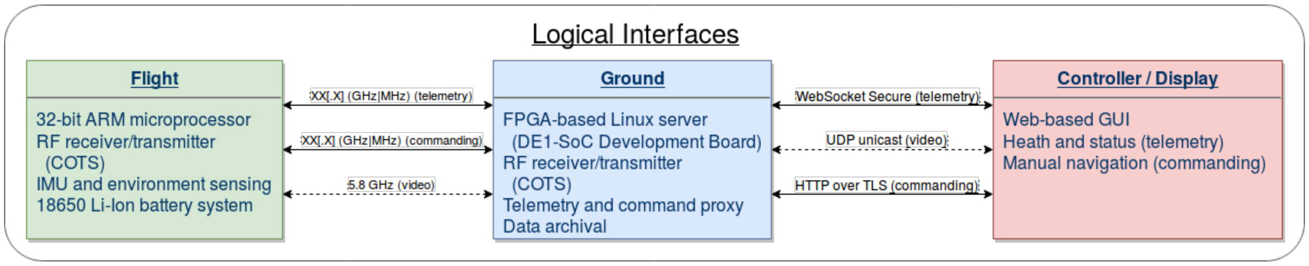

Logical Interfaces
Flight
32-bit ARM microprocessor
RF receiver/transmitter (COTS)
IMU and environment sensing
18650 Li-Ion battery system
XX[.X] (GHz|MHz) (telemetry)
XX[.X] (GHz|MHz) (commanding)
5.8 GHz (video)
Ground
FPGA-based Linux server (DE1-SoC Development Board)
RF receiver/transmitter (COTS)
Telemetry and command proxy
Data archival
WebSocket Secure (telemetry)
UDP unicast (video)
HTTP over TLS (commanding)
Controller / Display
Web-based GUI
Heath and status (telemetry)
Manual navigation (commanding)

Figure 2: High-level overview of the major components and their interfaces.

This high-level architecture is inspired by existing aerospace avionics and software systems that we have done research on and have some first-hand experience with.

3 Technical Features and Goals

Note that we focus on system-level goals over specific technical requirements of individual pieces. Our limited experience in this problem space makes an attempt at the latter not valuable (i.e. battery life, thrust and payload capability, etc.).

3.1 Telemetry Viewing

Initially, a list of channel names and corresponding values (with staleness) in a table format. This can grow in complexity to incorporate more interesting visualizations, potentially incorporating an annotated model of the vehicle itself.

3.2 Manual Commanding

A set of user inputs that can dispatch requests to the vehicle, with feedback indicating the result of the request (accepted, rejected, couldn't send, etc.).

3.3 Holding-Pattern Stability

Support a flight algorithm that enables high-fidelity control over motion of the vehicle.

3.4 Fault Tolerance

A vehicle that is *single-fault tolerant* is capable of continuing nominal operation after experiencing any arbitrary failure in a well-defined *fault space*. We aim to implement single-fault tolerance by:

1. Limiting the initial fault space to a "loss of ground station heartbeat" event
2. Executing a "landing maneuver" upon fault detection
3. Iteratively hardening our design to a broader fault space, time permitting

4 Milestones

We recognize some intermediate milestones that will need to be reached before an *automatic flight-termination system* described above can be expected to function. Note that we expect to parallelize completion of these milestones when possible.

4.1 Establish wireless communication between the vehicle and ground station

Estimated hours of work: **40**

Involves custom firmware/software for the DE1-SoC and an ARM microcontroller development board. This comes with the initial burden of establishing a build system and efficient workflow for development on these two platforms.

4.2 Establish percentage-based throttle control over each motor

Estimated hours of work: **8**

Requires interfacing with an ESC (electronic speed controller) via PWM signal. this won't take as long as other firmware tasks because a base is established at this point. This control will be done from a command-line interface on the DE1-SoC, which will require implementing an "application layer protocol" on top of the wire-level protocol used by the radios.

4.3 Establish manual-commanding capability from a web-based user interface

Estimated hours of work: **25**

Requires completing the initial user interface as well as the server-side software capable of dispatching hardware tasks to the previously completed lower layers.

4.4 View live telemetry from a web-based user interface

Estimated hours of work: **15**

This involves defining a data structure for telemetry packets that will be easily serializable for our wire-level protocol mediums and composable visual elements to display data.

4.5 Sense angular velocity via gyroscope and force experienced via IMU

Estimated hours of work: **20**

Previous experience interfacing with gyroscopes and intertial-measurement units allows us to proceed intelligently with respect to these data paths. Choosing the right sensors and architecting firmware to allow the full data rate (some digital filtering applied) to propagate into a control algorithm is the essence of the task.

4.6 Develop a control algorithm to fly in a stable hover or holding pattern

Estimated hours of work: **8**

With the data pipeline implemented all the way through to the user interface, this should only require implementing and tuning PID control loops that drive PWM signals to the ESC based on gyroscopic and inertial inputs. The user interface is a key component that allows this development task to be expedited.

4.7 Extend control algorithm to control for velocity in three axes

Estimated hours of work: **4**

This should only require injecting new "setpoints" into the functional control algorithm developed previously. It is not known at this time whether or not it will be possible to use setpoints associated with actual units of velocity, it may end up boolean or throttle-based.

4.8 Sense relative altitude

Estimated hours of work: **10**

An altimeter or other sensing technology has not been chosen, we can't accurately characterize the difficulty of this task yet.

4.9 Extend control algorithm to control for a specific *delta-z*

Estimated hours of work: **2**

If altitude can be measured to some degree of accuracy, injecting a new setpoint into the control algorithm was demonstrated previously which should make this straightforward.

5 Roles and Responsibilities

"A problem well stated is a problem half solved." - Charles Kettering

Distribution of work and responsibilities has been discussed and the following summaries represent a consensus reached on initial roles and task delegation.

5.1 Vaughn Kottler

Flight Hardware Final design and assembly of the quadcopter. Validation of component selection to ensure flight is feasible.

Flight Software The firmware image flashed to the flight computer. Includes necessary control algorithms and runtime configurability.

Project Oversight Track project progress and maintain documentation.

5.2 Mayank Katwal

Telemetry Display A web-based dashboard that summarizes the health and status of the quadcopter during flight.

Vehicle Control Interface A web-based user interface that allows manual control of the quadcopter during flight.

Data and Command APIs User programs that expose flight data and command capability to client browsers via HTTP over TLS requests.

5.3 Cooper Green

Ground Station Hardware asbstraction layer source code and mezzanine board implementation. A coherent and well-defined interface to the hardware from user programs.

Radio Frequency Communication Choice of frequency bands and protocols used to send and receive data with radio modules.

Telemetry Storage A data archival system for post-flight analysis.

6 Detailed Block Diagrams

6.1 Quadcopter

For the flight vehicle, we intend to pursue an implmentation depicted in **Figure 3**:

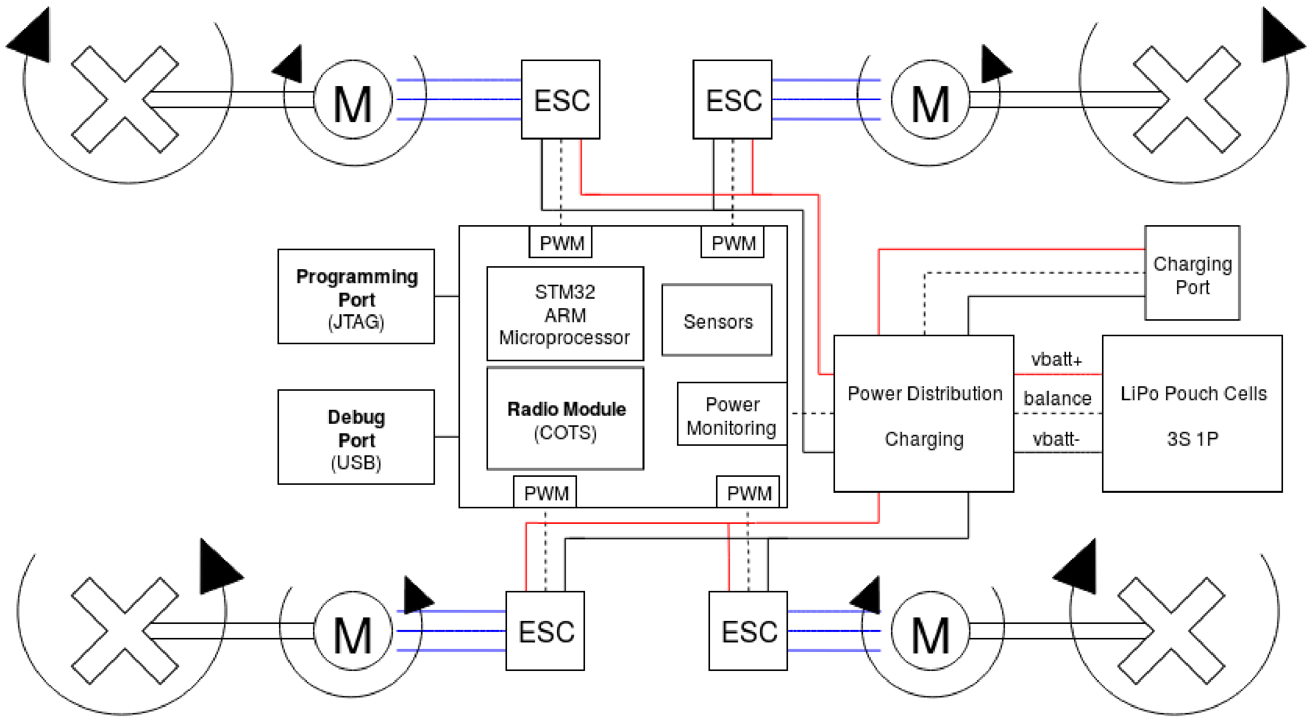


Figure 3: Block diagram view of the quadcopter.

Responsible Engineer: **Vaughn**

6.2 Ground Station

For the ground station, we intend to pursue an implmentation depicted in **Figure 4**:

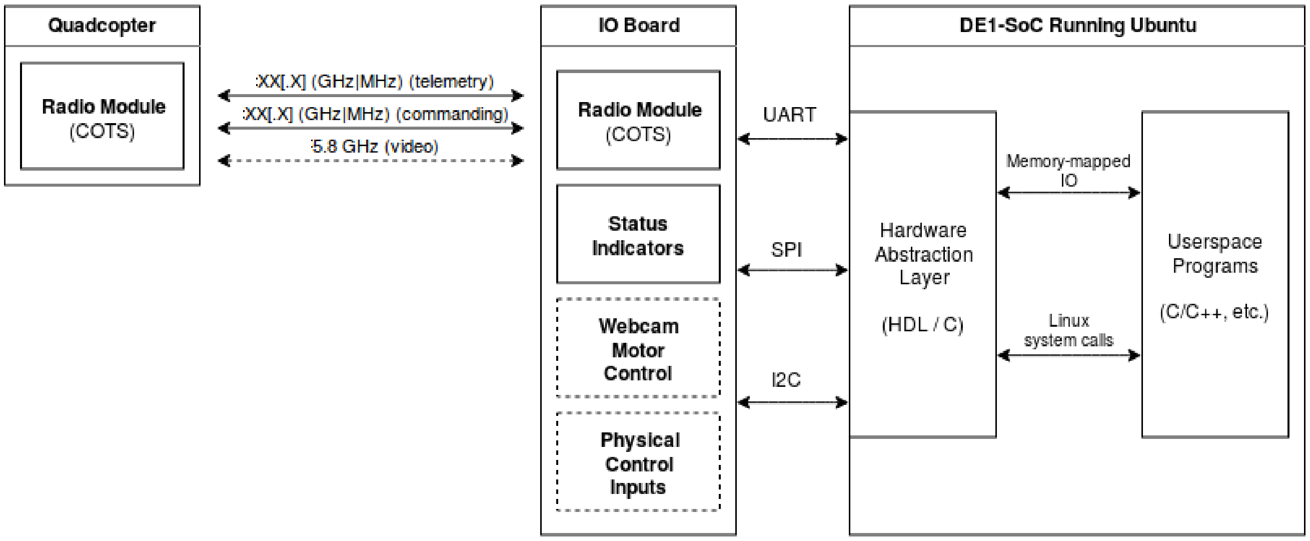
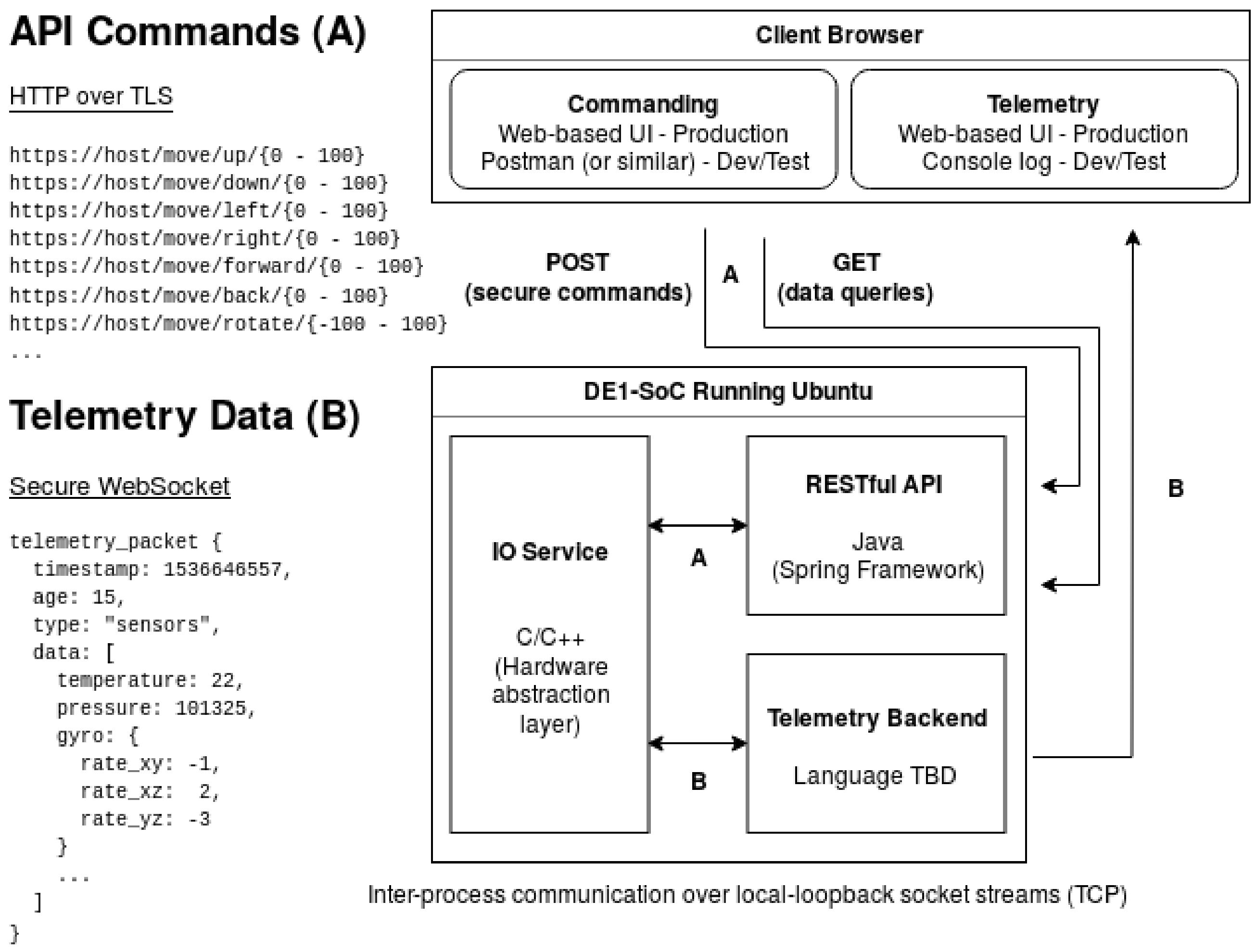


Figure 4: Block diagram view of the ground station.

Responsible Engineer: **Cooper**

6.3 Display and Controller

For the display and control interface, we intend to pursue an implmentation depicted in **Figure 5**:



```
https://host/move/up/{0 - 100}
https://host/move/down/{0 - 100}
https://host/move/left/{0 - 100}
https://host/move/right/{0 - 100}
https://host/move/forward/{0 - 100}
https://host/move/back/{0 - 100}
https://host/move/rotate/{-100 - 100}
...
```

```
telemetry_packet {
  timestamp: 1536646557,
  age: 15,
  type: "sensors",
  data: [
    temperature: 22,
    pressure: 101325,
    gyro: {
      rate_xy: -1,
      rate_xz:  2,
      rate_yz: -3
    }
    ...
  ]
}
```

Figure 5: Block diagram view of the display and control user interface.

Responsible Engineer: **Mayank**

7 Cost Estimation

This is our estimate of what items are required to complete this project. Items listed as having zero cost were provided to us or are things that we already previously owned.

Quadcopter			
Item (as hyperlink)	Purpose	Quantity	Total Cost
Chassis	Holds the vehicle components together	1	$19
Motors + ESCs	Components necessary for turning the propellers to create lift	4	$60
Propellers	Physical component that moves the air to create lift	4	$11
3S LiPo Battery	Power source for all vehicle electronics	1	$14
LiPo Charger	Charger for the battery	1	$50
STM32 Development Boards	Development platform for testing before production flight controllers are available	4	$56
ST-Link V2 Programmer	USB device required for flashing firmware onto STM32 microcontrollers	2	$20
Custom Flight Controller	PCB and components cost	1	$75
Ground Station			
Item (as hyperlink)	Purpose	Quantity	Total Cost
Radio Candidate 1 (Pair)	Wireless communication with the vehicle (SPI, 2300M range)	1	$16
Radio Candidate 2 (Pair)	Wireless communication with the vehicle (SPI, 1100M range)	1	$12
Radio Candidate 3 (Pair)	Wireless communication with the vehicle (UART, 3000M range)	1	$30
DE1-SoC Development Board	Compute platform, interfaces with the mezzannine board	1	$0
DE1-SoC Custom Mezzanine	Custom PCB, hosts the radio and other ground-side IO	1	$50

A number of tools have been provided to us by the instructor and we have set up some of our own equipment in the lab space. There are some development tools and consumables we anticipate needing to make purchases for, though. A brief list may include:

- Breadboarding supplies (jumper wire, through-hole passive components, etc.)
- Various gauges (and colors) of stranded or solid-core wire
- Tape (electrical, kapton), fasteners
- Crimp-terminal, lug and wire ferrule kits
- Networking equipment (router, switch, patch cables)

Estimated additional development cost: **$200** (contingent on available funding)

Total estimated cost: **$305 + $108 + $200 = $613**

8 Project Management

"If I had an hour to solve a problem I'd spend 55 minutes thinking about the problem and 5 minutes thinking about solutions." - Albert Einstein

The value of detailed (but *precise*!) planning for a project of this scope and timeline cannot be overstated. Prototyping and experimenting is inevitable but must be guided to avoid allocating disproportionate effort towards minor goals that put higher-level goals at risk. **Figure 6** introduces our approach and **Figure 7** establishes a proposed process for handling integration and final stages in project realization.

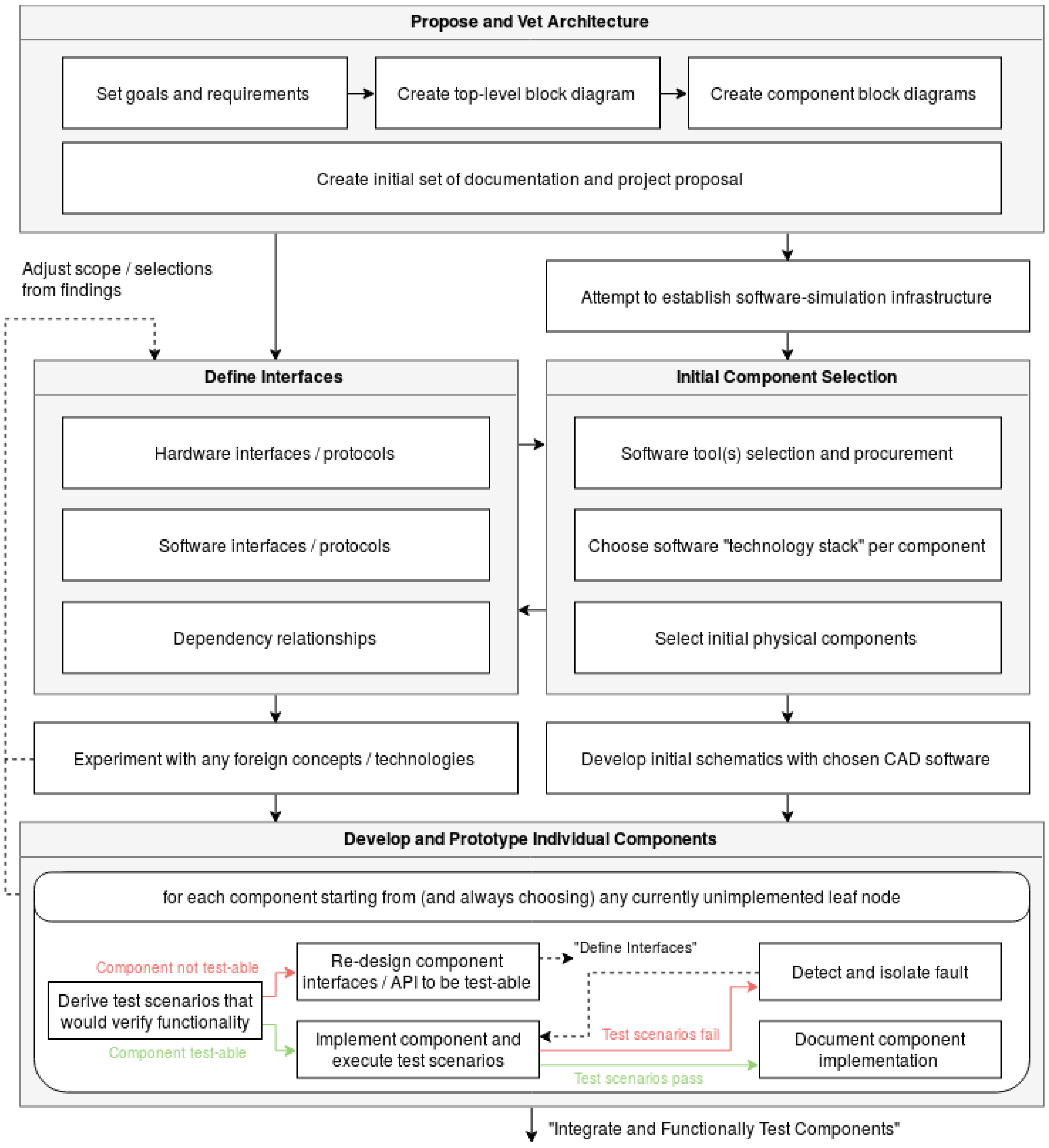

Propose and Vet Architecture
Set goals and requirements
Create top-level block diagram
Create component block diagrams
Create initial set of documentation and project proposal
Adjust scope / selections from findings
Attempt to establish software-simulation infrastructure
Define Interfaces
Hardware interfaces / protocols
Software interfaces / protocols
Dependency relationships
Initial Component Selection
Software tool(s) selection and procurement
Choose software "technology stack" per component
Select initial physical components
Experiment with any foreign concepts / technologies
Develop initial schematics with chosen CAD software
Develop and Prototype Individual Components
for each component starting from (and always choosing) any currently unimplemented leaf node
Component not test-able
Derive test scenarios that would verify functionality
Component test-able
Re-design component interfaces / API to be test-able
"Define Interfaces"
Implement component and execute test scenarios
Test scenarios fail
Test scenarios pass
Detect and isolate fault
Document component implementation
"Integrate and Functionally Test Components"

Figure 6: A flexible workflow diagram that attempts to capture our ideal process for initial prototype and development work.

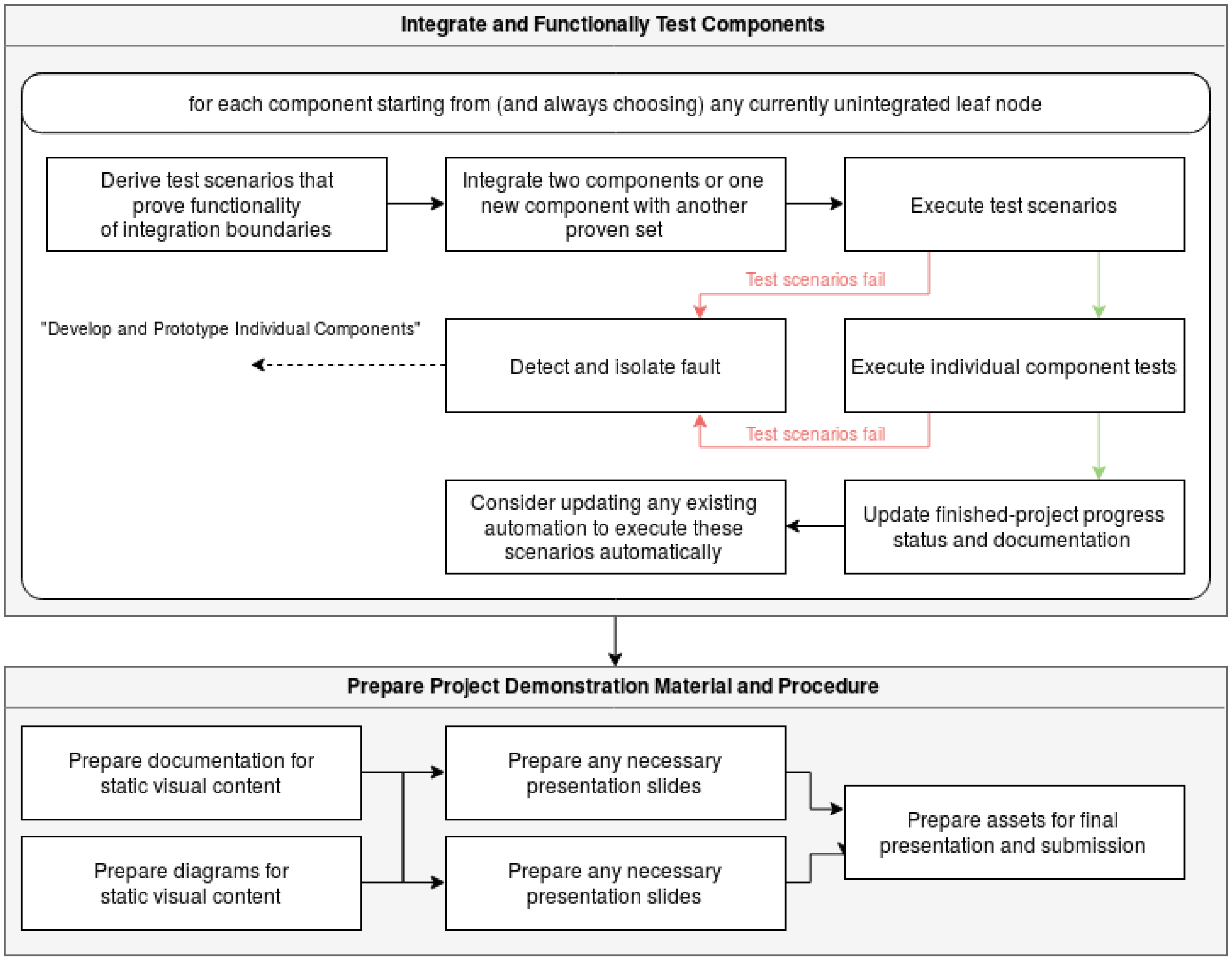

Integrate and Functionally Test Components
for each component starting from (and always choosing) any currently unintegrated leaf node
Derive test scenarios that prove functionality of integration boundaries
Integrate two components or one new component with another proven set
Execute test scenarios
Test scenarios fail
"Develop and Prototype Individual Components"
Detect and isolate fault
Execute individual component tests
Test scenarios fail
Consider updating any existing automation to execute these scenarios automatically
Update finished-project progress status and documentation
Prepare Project Demonstration Material and Procedure
Prepare documentation for static visual content
Prepare diagrams for static visual content
Prepare any necessary presentation slides
Prepare any necessary presentation slides
Prepare assets for final presentation and submission

Figure 7: The remaining stages of the workflow diagram that captures our intended process for bringing the project to completion.