

# Empowering Systems Engineering with Large Language Models

From Ideation to Implementation



550 Vandalia St #231  
Saint Paul, MN 55114

[lab651.com](http://lab651.com)  
[recursiveawesome.com](http://recursiveawesome.com)

# About Me

---

- Founder, **Recursive Awesome & Lab651** – AI Consulting and Training & Custom Software Development
- Owner, **Applied AI Weekly** – Publication covering AI
- Host of the “**Conversations on Applied AI Podcast**”
- Adjunct Professor, **University of Saint Thomas** – Teaching graduate courses on IoT & AI/ML
- Co-founder, **AppliedAI.MN** – 501(c)(3) non-profit: Monthly meetups, conferences, videos and podcasts on AI/ML
- Conference on May 10<sup>th</sup>! Visit us at [AppliedAIConf.com](http://AppliedAIConf.com)



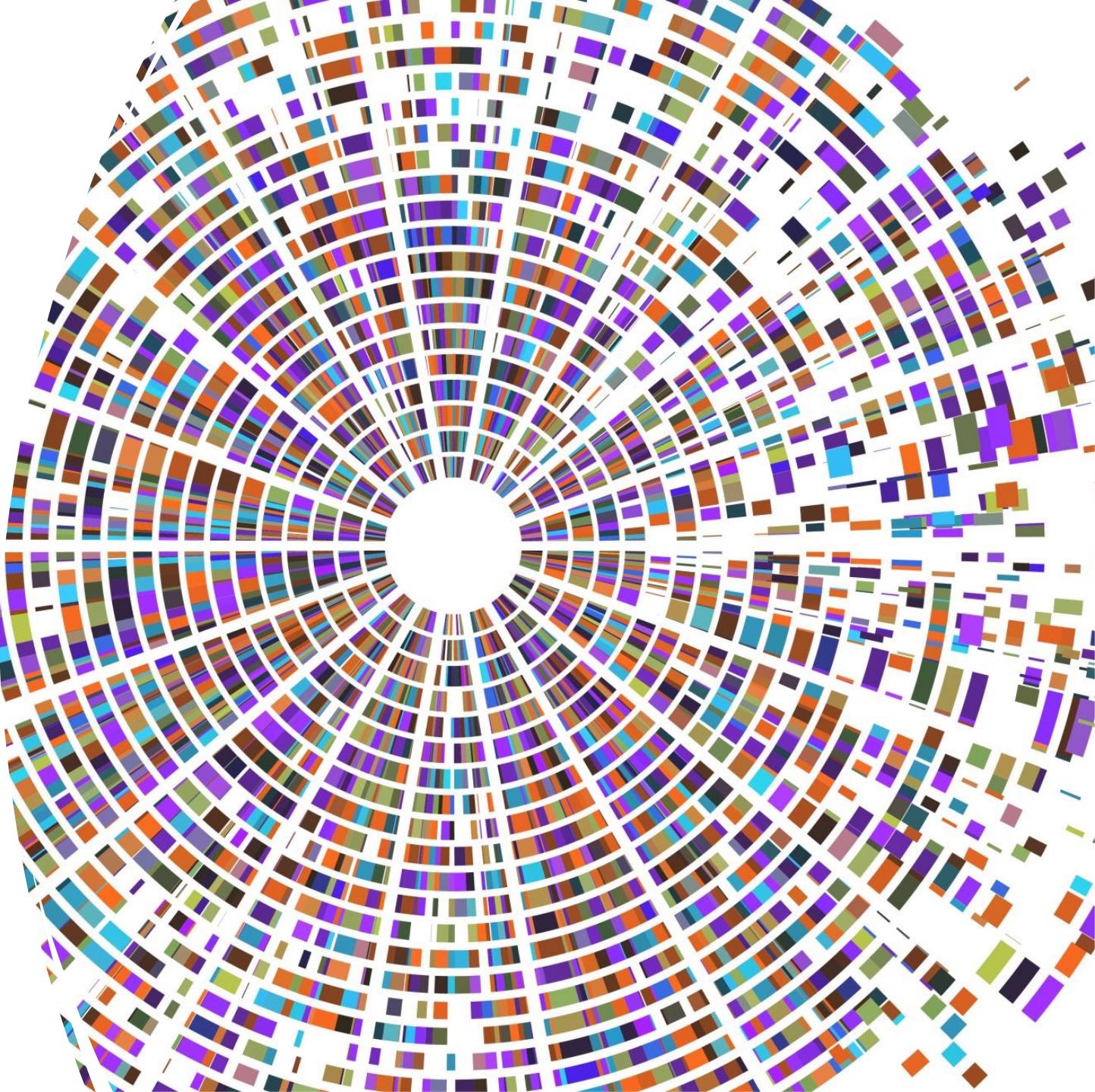


# Agenda

---

- What is an LLM
  - What are they good at?
  - What are they not so good at?
- Story: A Product Journey using generative AI
- Learnings & Key Takeaways

NOTE: All images were generated using AI.



# LLMs and What Are They Good At?

---

- A large language model is a type of artificial neural network that uses deep learning techniques to process, understand, and generate human-like text.
- **Content Generation:** Used for creating articles, reports, stories, and even poetry
- **Language Translation:** High-quality translations between numerous languages
- **Learning and Research:** Assist in educational and research activities by summarizing information
- **Role Playing:** Human-like dialogue makes them useful for chatbots, but also your virtual customer!

# What Are LLMs NOT Good At?

---

- **Fixed Knowledge Base:** Knowledge is static and limited to what was available when it was trained.
  - Systems like Retrieval Augmented Generation (RAG) addressing this
- **Understanding Context Depth:** Lack common sense reasoning, so they struggle with domain specific context. Prone to hallucinations!
  - Smaller, Fine-Tuned Models addressing this
- **Conversational Continuity:** Generally don't maintain continuity over long conversations and lose track of previous conversations.
  - Larger context windows and frameworks are helping with this
- **Dependency on User Input:** Relies heavily on the quality and clarity of user queries ( prompts ).
  - Experience in giving good prompts are helping

# The Product: Remote Patient Monitoring

---

- Takes data from an ECG
- Passes that data via Bluetooth to a smart phone
- Data is sent to a system hosted in the cloud
- Critical Information can be reviewed by doctor.

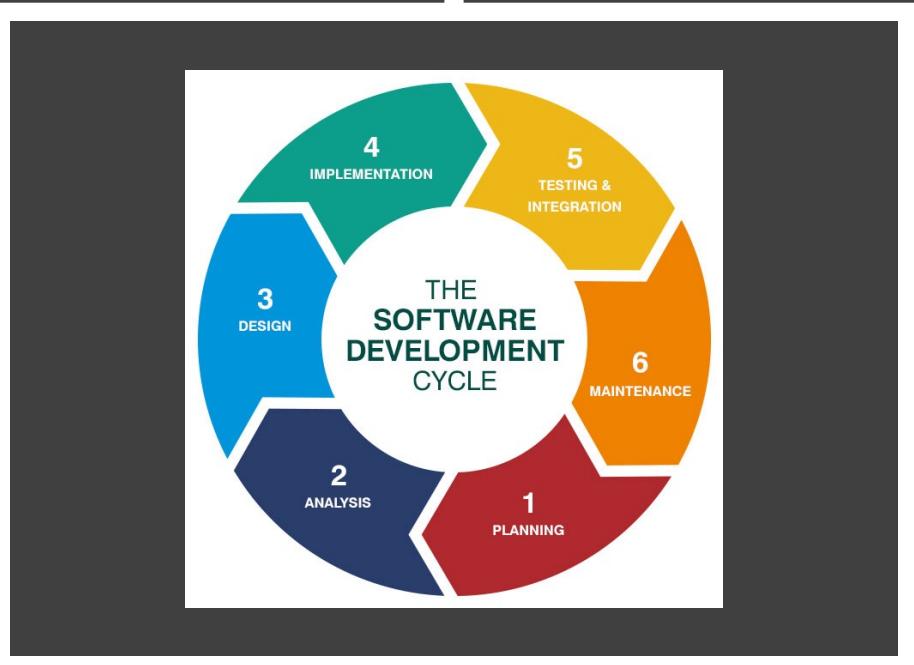




Eliana – Software Engineer



Taylor –Systems Engineer



Lucas –Quality Assurance



Alex - Product Manager

# Product Manager (Alex)

---

- **Challenge**
  - Has a vision for the product
  - Translating requirements can be daunting
- **Where LLMs help**
  - Market research & competitive analysis
  - Draft surveys and interview questions - GenAI to the rescue!
  - Review unstructured notes and data from stakeholders
- **Output:** Product Requirements Doc



# Prompt: What are the competitive landscape and market trends?

---



## ChatGPT

Understanding the competitive landscape in the ECG device market is crucial because it helps identify market trends, technological innovations, and gaps in current offerings. This insight enables companies to differentiate their products, meet unaddressed customer needs, and strategically position themselves in the market.

### Clinical-Grade ECG Monitors

1. **GE Healthcare:** Advanced diagnostic features and robust integration with health systems.
2. **Philips Healthcare:** Wide range, focusing on innovation and user-friendly interfaces.
3. **Schiller:** Compact, portable systems emphasizing cost-effectiveness and usability.

### Wearable and Portable ECG Devices

1. **AliveCor (KardiaMobile):** User-friendly, smartphone-compatible devices for home monitoring.
2. **Apple Watch (Series 4 and later):** Smartwatch with ECG function, part of a health and fitness ecosystem.
3. **Withings Move ECG:** Hybrid smartwatch that combines traditional aesthetics with



## Market Trends and Analysis

- **Wearable Electrocardiography (ECG) Devices:** There's a significant trend towards wearable ECG devices, with the market expected to grow substantially. Smartwatch and pocket ECG monitor segments are projected to have the highest growth rates. The U.S. and China are key markets for these products, driven by technological advancements and increasing demand [11].
- **Adoption of AI and Advanced Technologies:** The integration of AI for ECG interpretation and the incorporation of wearable sensor technologies are major trends. These innovations aim to enhance patient monitoring and care quality, with an emphasis on personal health monitoring through wearables and mobile applications [12] [13].
- **Increased Focus on Speed to Market:** There's a push towards reducing time-to-market for medical devices without compromising on safety or quality. This is achieved through strategies like technology reuse, strategic partnerships, and efficient resource planning [14].

## Similar Products and Innovations

- Companies like Apple, Samsung, and Fitbit are leading in the wearable ECG devices market, with products that incorporate ECG functions into consumer wearables. This indicates a trend towards integrating medical-grade features into everyday technology [15] [16].
- The emphasis on inclusivity and access is shaping product development, with a greater focus on creating devices that cater to a diverse range of users and conditions, especially in developing countries [17].

# Play the role of...



You

Play the role of the patient. What are some of the most important features you would like for this device to have?



ChatGPT

As a patient using an ECG medical device, several key features would significantly enhance my experience and satisfaction with the product, ensuring it meets my health monitoring needs effectively:

1. **Comfort and Wearability:** The device should be lightweight, non-intrusive, and comfortable to wear for extended periods. Ideally, it would have a low profile so it doesn't interfere with clothing or daily activities. The ability to wear the device without noticing it much would make continuous heart monitoring much less burdensome.
2. **Ease of Use:** Simplicity is crucial. The device should be easy to put on, activate, and sync with other devices. Clear instructions and a straightforward interface on both the device and the accompanying app would minimize confusion and ensure I can manage my health without needing constant assistance.
3. **Real-time Feedback:** It would be beneficial to receive immediate, understandable feedback on

# Play the role of...

---



## ChatGPT

As a physician, leveraging insights from previous discussions with practitioners, the following features are paramount for an ECG device system to effectively support patient care and clinical decision-making:

1. **Real-Time Alerts and Notifications:** The system should provide real-time alerts for abnormal ECG readings, enabling immediate intervention when necessary. This is particularly crucial for patients with high-risk cardiac conditions. Practitioners emphasized the need for customizable thresholds for alerts to tailor monitoring to individual patient needs and avoid alert fatigue.
2. **Interoperability with Health Systems:** Seamless integration with existing Electronic Health Records (EHRs) and other hospital management systems is essential. Physicians need to access ECG data within the broader context of patient health records, ensuring that care decisions are informed by a comprehensive view of the patient's health status.
3. **Data Analysis and Reporting Tools:** Advanced data analysis capabilities, including AI-powered diagnostic support, can greatly enhance the physician's ability to interpret ECG data accurately and efficiently. Features facilitating the easy generation of reports and summaries of the ECG data for clinical reviews were highlighted as particularly valuable. Practitioners also expressed a desire for the system to support longitudinal data analysis to track patient health trends over time.
4. **Patient Data Security and Privacy:** In line with regulatory compliance and ethical considerations, the system must ensure robust security measures to protect patient data. This includes encryption of data at rest and in transit, secure access controls, and adherence to relevant privacy laws like HIPAA or GDPR.

# Product Requirements Document



You

Thanks! This is a great summary of the main concerns both for doctors and patients. Based on the output that you have defined, please create a Product Requirements document that has all of the industry standard sections and details in each section that are required for our Systems Engineers and UI/UX professionals to begin designing this system. From the device to the mobile application to the cloud services that will be ingesting the data.



ChatGPT

The Product Requirements Document for the ECG Medical Device has been created successfully. This document includes all the necessary sections and details required for Systems Engineers and UI/UX professionals to begin the design and development process for the device, mobile application, and cloud services.

You can download the document using the link below:

[Download ECG Medical Device Product Requirements Document \[→\]](#)



# And while we are at it...



You

I need some help coming up with a fun name for the system that we will be building. Can you give me 5 possible product names as we work through the development of this product? They should try to embody the fact that we are trying to help both patients have an easy way to get their ECG data from their body to the physician, but also that this data will provide a new level of care that doctors will be able to give to their patients.



ChatGPT

Here are five possible names for your ECG Medical Device system, aiming to reflect the ease of use for patients and the enhanced care quality for physicians:

1. **HeartLink**: Emphasizing the direct connection between patients' heart data and healthcare providers.
2. **PulsePath**: Highlighting the journey of heart rate data from the individual to the medical professional.
3. **CardioCareConnect**: Focusing on the comprehensive cardiac care facilitated by seamless data connectivity.
4. **EchoFlow**: Suggesting the smooth flow of vital ECG data from the wearable device to healthcare experts.
5. **VitalBeat**: Capturing the essential nature of heartbeats in patient care and data transmission.

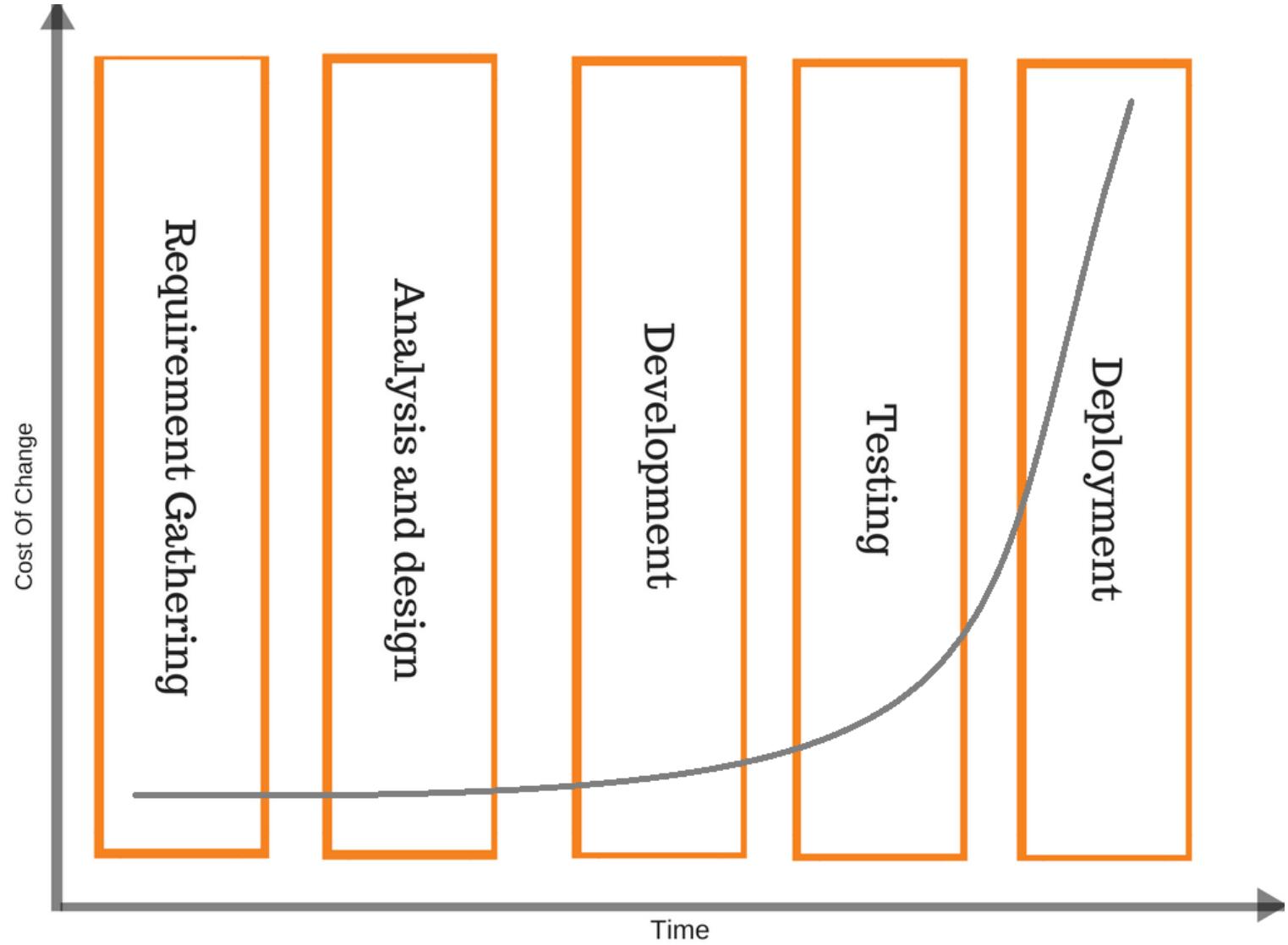


These names are designed to encapsulate the core mission of providing an easy and efficient way



Exponential increasing cost of changes to software as development progresses

# Boehm's Curve



# Prompt: Where is there ambiguity in the product requirements document?

---

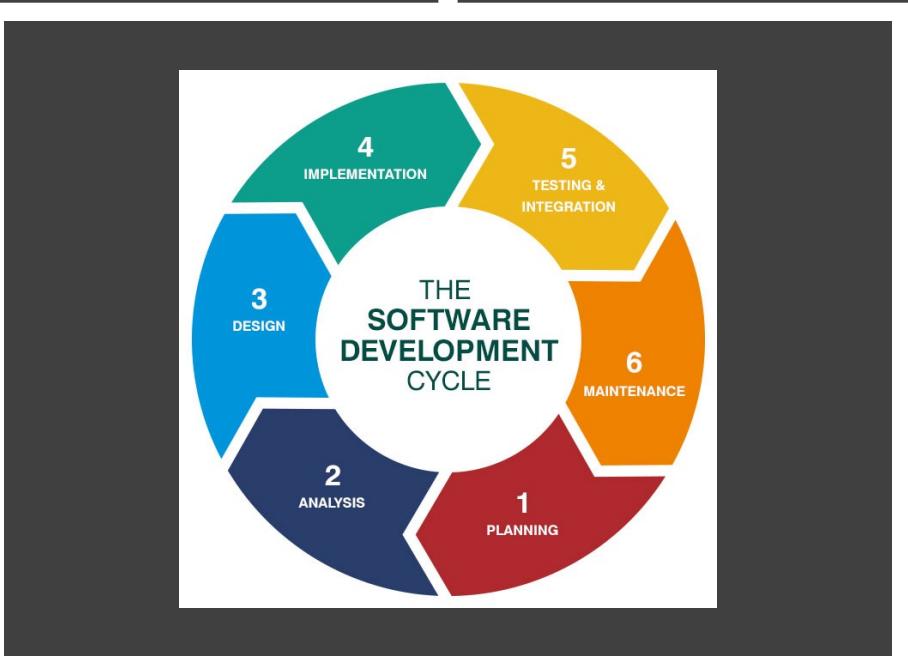


1. **Data Transmission Security:** The requirements mention secure data transmission, but specifics aren't clear.
  - "What encryption standards and protocols should we implement for Bluetooth and Internet data transmission?"
2. **User Authentication:** The document specifies a mobile application but doesn't detail user authentication.
  - "Do we need multi-factor authentication for user access, and are there any specific regulatory requirements for patient data access?"
3. **Data Storage and Retention:** While cloud storage is mentioned, there are no details on data retention policies.
  - "What are the data retention policies we need to follow, and are there differences in requirements across regions?"
4. **Mobile App Functionality:** The scope of functionality within the mobile app is broad.
  - "Can we break down the mobile app's features into priority tiers, and are there any must-have features for the initial release?"
5. **Error Handling:** The system must handle errors gracefully, but guidelines are not provided.
  - "What are the requirements for error logging and reporting, especially for data transmission failures?"
6. **Real-Time Data Processing:** The requirements mention real-time data processing without specifics

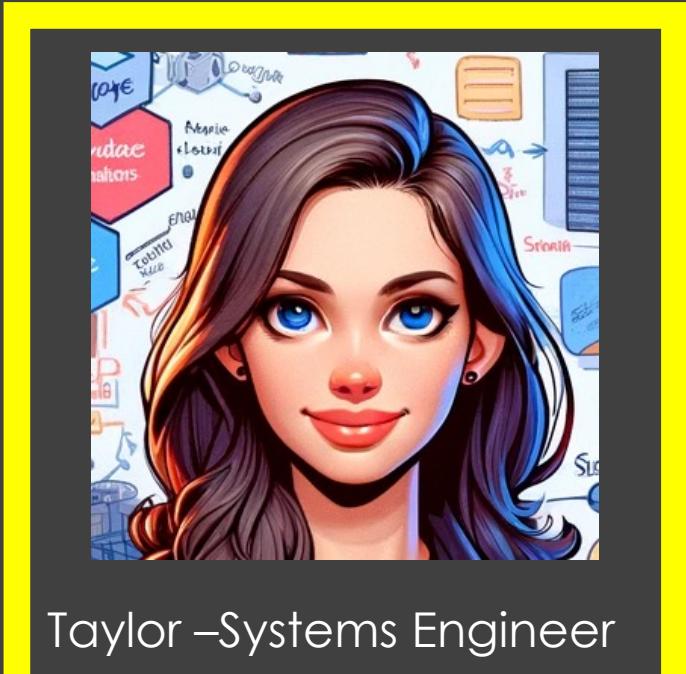




Eliana – Software Engineer



Lucas – Quality Assurance



Taylor – Systems Engineer



Alex - Product Manager

# Systems Engineer (Taylor)

- **Challenge**
    - Interpreting and implementing requirements
    - Navigating healthcare regulations
  - **Solution with LLMs**
    - Dissecting the PRD
    - Identify risks where other projects have failed
    - Recommending tools and methodologies
    - Draft regulatory submissions



# Sample Specification...

## Initial Implantable Cardiac Monitor System Requirements

The Vector Implantable Cardiac Monitor (VICM) System consists of a multi-sensor implantable device, an Android phone application, and a cloud server backend. The implantable device has sensors for electrocardiograms (ECG), phonocardiograms (PCG), seismocardiography (SCG), and an accelerometer to measure patient activity levels. The implantable device will use Bluetooth to transmit the acquired data to an Android phone. The phone in turn uses WiFi or cellular communications to send data to the cloud server. The implant should be able to transmit data to the phone at distances up to 2.4 meters.

The implant shall use a primary cell battery with a 2-year operational life. The battery shall be capable of supplying 3 Volts.

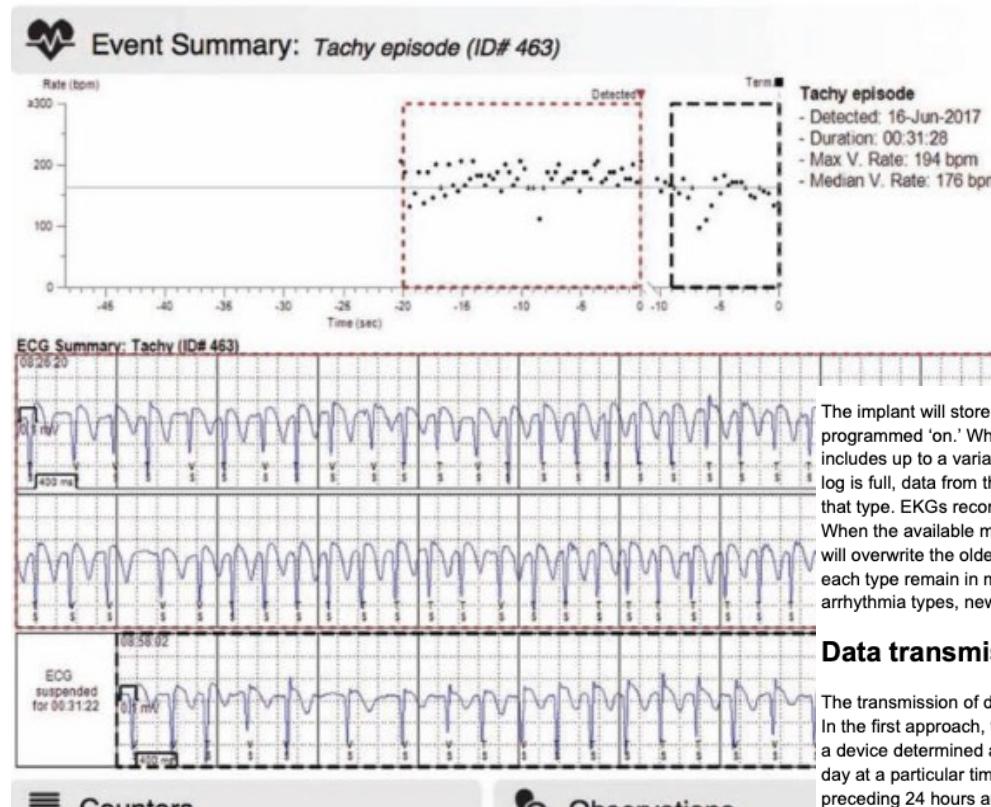
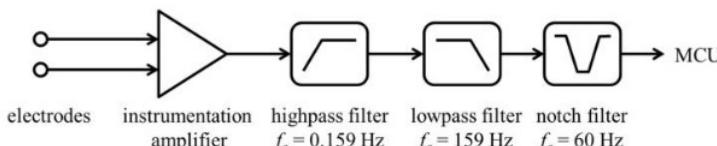
The fully burdened cost of manufacturing the implantable device shall be no more than \$1200.

The implant size should not exceed 19.4 mm × 55.4 mm × 9 mm.

The implant should consist of:

- a PCB containing a microprocessor, real-time clock, non-volatile RAM, Flash EEPROM, radio-frequency transceiver for telemetry, ECG sensor, and associated circuitry.
- An antenna
- A battery
- At least 2 electrodes for sensing
- A hermetically sealed package containing the above items.
- A set of feedthroughs allowing the sensed electrical signal of the heart to be transmitted to the PCB containing the microprocessor.

The ECG sensor should have the following structure:



The implant will store episode data and ECGs for each episode type where detection is programmed 'on.' When it is 'on,' arrhythmia episode data is stored in an episode log. This includes up to a variable number of episodes, specified by the Android application. When the log is full, data from the most recent episode may overwrite the oldest stored episode data of that type. EKGs recorded before and during an episode can also be stored in device memory. When the available memory for automatically detected episodes is full, a new ECG recording will overwrite the oldest stored ECG recording, provided that a minimum number of episodes of each type remain in memory. After the device has reached the storage limit for individual arrhythmia types, new episodes will not be stored.

## Data transmission

The transmission of data from the patient to the physician is accomplished via two approaches. In the first approach, the data is transmitted as alert notifications. An alert notification consists of a device determined abnormal rhythm and is transmitted to the web-based central server once a day at a particular time. The alert notification consists of a total count of all episodes in the preceding 24 hours and consists of a snapshot of the arrhythmia electrogram of 30 seconds at the onset of an episode from the longest episode. This approach is based on the assumption that the electrograms from the transmitted episode are representative of all the episodes. The second approach is the manual download of the data that is prompted by the patient via a home monitoring system or cellphone-based application. Often, these downloads are performed at scheduled intervals (every 31 or 90 days). During a manual transmission of data, all stored

# Read the specification for me...



You

Initial Implantable Cardiac Monitor Syst...  
PDF

Our team has created the attached system requirements. Please review and I'll have some questions for you.

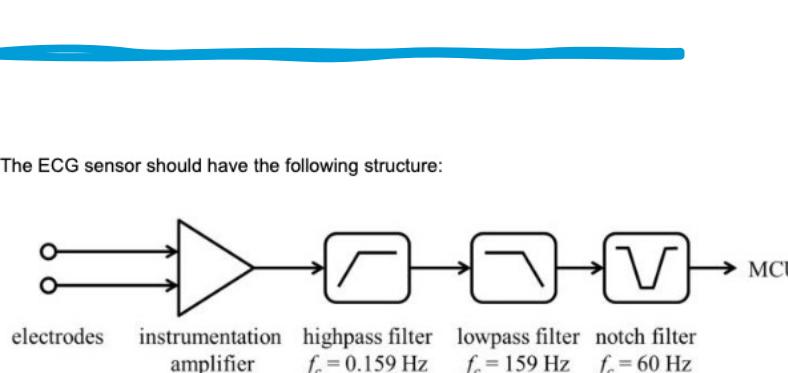
ChatGPT

The Vector Implantable Cardiac Monitor (VICM) System, as detailed in the document, encompasses three major components that need to be developed by a Systems Engineer:

1. **Multi-Sensor Implantable Device:** Incorporates sensors for ECG, PCG, SCG, and an accelerometer for measuring patient activity levels. This device communicates data to an Android phone app via Bluetooth.
2. **Android Phone Application:** Receives data from the implantable device using Bluetooth and sends it to a cloud server via Wi-Fi or cellular communication. The app allows adjustment of various thresholds and settings based on the patient's requirements.
3. **Cloud Server Backend:** Manages data storage from multiple patients (up to 100,000) and presents this information to clinicians through a web server. The server is responsible for handling alert notifications and manual data downloads initiated by the patient or automatically scheduled.

These components collectively form the foundation of the VICM System, focusing on capturing and analyzing critical cardiac data for patient monitoring and physician review.

# Prompt: What components do you suggest we use in the structure of the ECG?



ChatGPT

The ECG sensor structure depicted in your document includes three filters arranged in series to condition the signal before it reaches the microcontroller unit (MCU). The purpose of these filters is to ensure that the signal fed into the MCU is free of noise and artifacts that could distort the readings. Here are the three phases:

1. **Instrumentation Amplifier:** This component is the first stage of the ECG sensor. It amplifies the bioelectric potential that is picked up by the electrodes from the heart. Its primary role is to provide a high input impedance and to amplify the very small signals while rejecting noise and interference that might be common to both inputs (common-mode noise).  
*Suggested Components:* Precision instrumentation amplifiers like the Analog Devices AD620 or Texas Instruments INA128.
2. **High-pass Filter (HPF):** This filter allows signals with a frequency higher than the cut-off frequency ( $f_c = 0.159$  Hz) to pass through while attenuating frequencies below this threshold. This helps to eliminate the direct current (DC) components and slow-moving artifacts like the wandering baseline caused by respiration or body movements.  
*Suggested Components:* A passive high-pass filter using a resistor and a capacitor could suffice for this application. The values of the resistor and capacitor would be calculated based on the desired cut-off frequency using the formula  $f_c = \frac{1}{2\pi RC}$ .
3. **Low-pass Filter (LPF):** The low-pass filter allows signals with a frequency lower than the cut-off frequency ( $f_c = 159$  Hz) to pass and attenuates the frequencies above it. This filter would be essential to eliminate high-frequency noise such as power line interference, muscle tremors, or electronic device interference.  
*Suggested Components:* A passive low-pass filter could also be designed using a resistor and a capacitor, or an active low-pass filter using operational amplifiers (op-amps) for better performance.
4. **Notch Filter:** The notch filter with a center frequency ( $f_c$ ) of 60 Hz is designed to remove the

**Prompt:** Can you provide a list of recommended microprocessors for medical devices requiring long battery life and robust data processing capabilities?



Here's a list of microprocessors and microcontroller units (MCUs) that are often recommended for such applications:

1. **ARM Cortex-M Series:** These MCUs are known for their energy efficiency and are widely used in medical device applications. The Cortex-M4 and Cortex-M7, for instance, offer floating-point processing which is beneficial for data-intensive tasks.
2. **Texas Instruments MSP430:** This is another family of ultra-low-power MCUs designed for battery-powered and low-power consumption devices.
3. **STMicroelectronics STM32:** The STM32 family has various lines like STM32L that are optimized for low power consumption while still providing high performance and are suitable for complex medical devices.
4. **Atmel AVR and SAM:** AVR microcontrollers are known for their simple architecture and low power consumption, while the Atmel SMART SAM series offers higher processing capabilities with an ARM Cortex core.
5. **NXP's Kinetis:** These are ARM Cortex-M core-based MCUs that focus on energy efficiency and are suitable for portable health monitoring devices.
6. **Microchip PIC Microcontrollers:** With features like XLP (Extreme Low Power), these MCUs are designed for applications where power consumption is critical.
7. **Analog Devices ADuCM Series:** These ultra-low-power MCUs integrated with RF transceivers are ideal for portable or wearable health devices.
8. **Silicon Labs EFM32 Gecko Family:** This family of MCUs is designed for energy-

# Prompt: What areas of this specification document are missing when we implement this solution?

---



**ChatGPT**

Your specification document for the implantable cardiac monitor system appears to cover many critical aspects, but there are several areas that could be elaborated upon or added to ensure a comprehensive and clear guide for successful implementation:

## 1. Regulatory Compliance:

- Specific requirements for compliance with relevant medical device regulations (e.g., FDA in the USA, EMA in Europe, etc.) should be outlined.
- Details about adherence to standards such as ISO 13485 for medical devices and IEC 62304 for medical device software.

## 2. Safety Requirements:

- Explicit mention of biocompatibility for all materials that will come into contact with human tissue.
- Description of safeguards against thermal and electrical hazards to the patient.
- Procedures for failsafe operation in the event of a component failure.

## 3. Security Protocols:

- Detailed security protocols for data protection, beyond just encryption, such as access controls and methods for secure software updates.
- Strategies for secure end-of-life device data handling and disposal.

## 4. Quality Assurance:

- Testing procedures for both hardware & software components.
- Protocols for in-vivo testing, including animal trials and eventual human trials.

# Prompt: What are some questions that I should be asking you to aid me in designing this system?

---



1. **Component Recommendations:** "Can you provide a list of recommended microprocessors that are suitable for medical devices requiring long battery life and robust data processing capabilities?"
2. **Risk Mitigation:** "Based on industry standards, what are common risks associated with implantable medical devices, and what best practices can be employed to mitigate these risks?"
3. **Regulatory Compliance:** "What are the key FDA or CE regulatory compliance points that I need to consider when designing an implantable ECG monitor?"
4. **Data Security:** "How can we implement end-to-end encryption in a medical device that involves Bluetooth transmission of data?"
5. **Signal Processing:** "What signal processing techniques are most effective for filtering ECG data in real-time on a low-power device?"
6. **Power Consumption Optimization:** "What strategies can be employed to optimize power consumption in a wearable ECG device?"
7. **Design Validation:** "What are the best methods for validating the design and functionality of an implantable cardiac monitor before beginning clinical trials?"
8. **Cloud Integration:** "Can you describe a secure and scalable architecture for integrating an implantable cardiac monitor with cloud services?"
9. **Testing Protocols:** "What testing protocols should be established to ensure the reliability and safety of the implantable cardiac monitor?"

# Prompt: Create a sequence diagram showing the entire process of dataflow from ECG to EMR to physician

---



Lucid

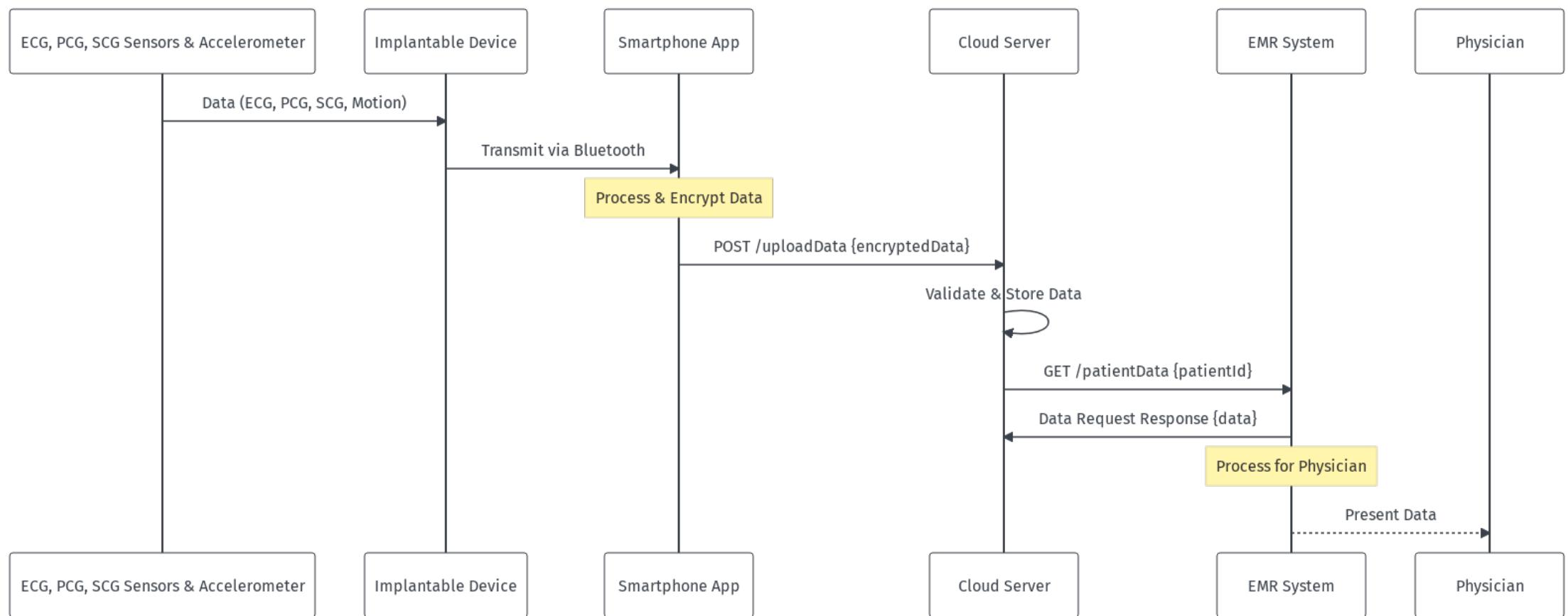
- ✓ Talked to mlai.lucid.app

I've created a sequence diagram to illustrate the end-to-end process of data flow from ECG capture to being presented to a physician via an EMR system. This diagram includes API endpoints and key information exchanges that could be valuable for software engineering implementation.

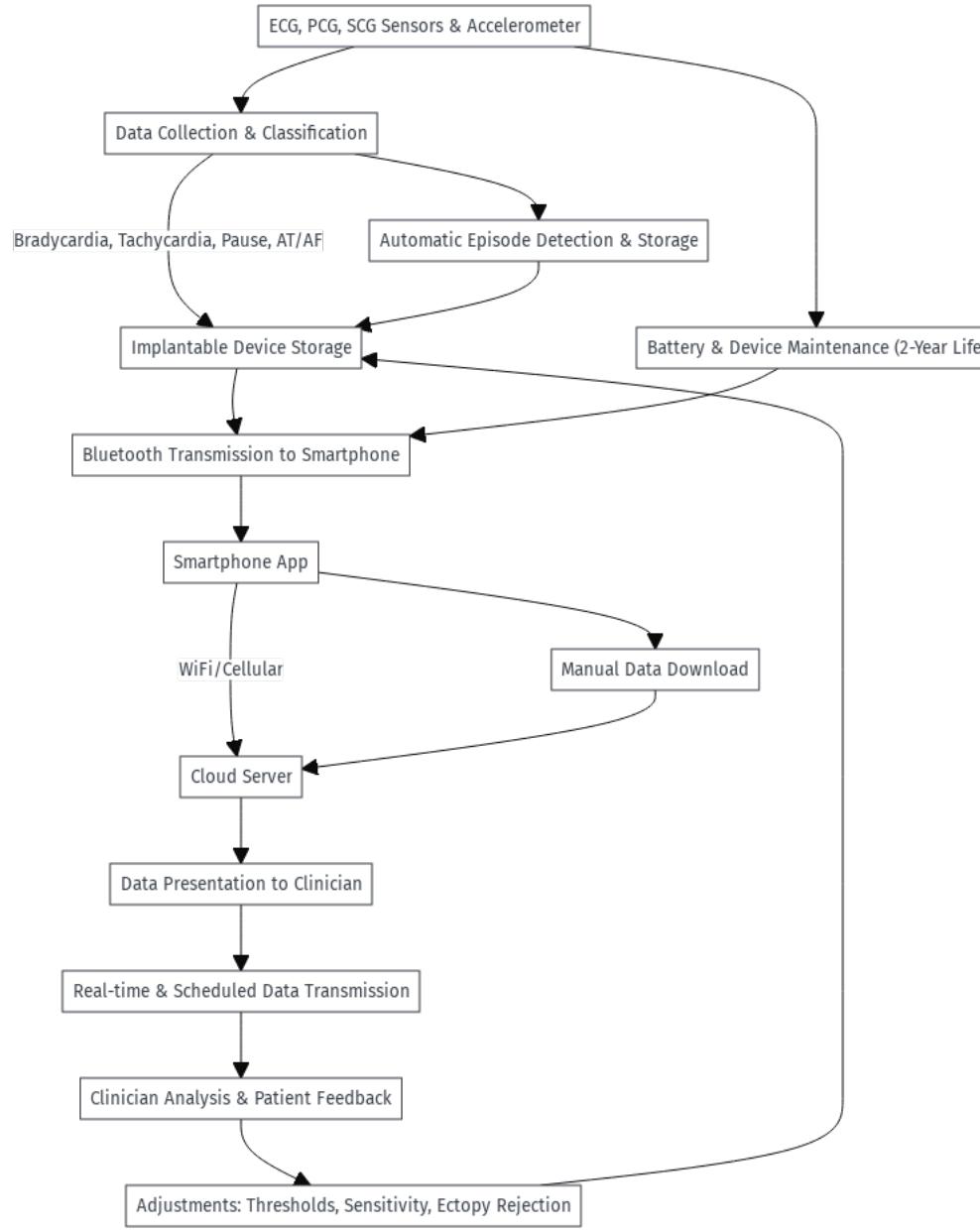
- **Sequence Diagram Image:** [View Sequence Diagram](#)

This visualization is aimed at aiding software engineers in understanding the flow of data through the system, including interactions between components, data processing, and the final presentation of data to a physician. Unfortunately, this type of diagram cannot be directly edited in Lucidchart through a link provided here. If you need adjustments or more detailed inclusion of API endpoints and data structures, feel free to ask!

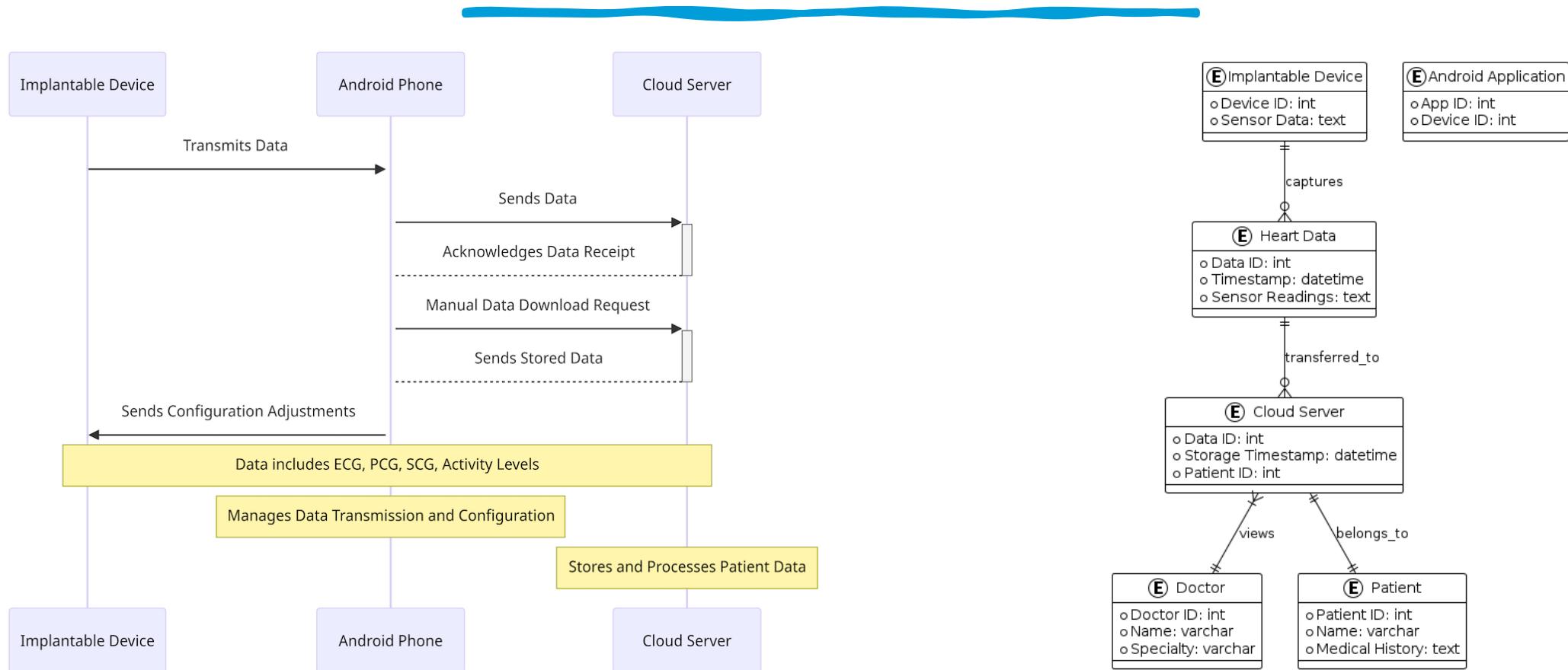
# Lucid App Results



**Prompt:** Cool.  
How about we  
do a flowchart  
too?



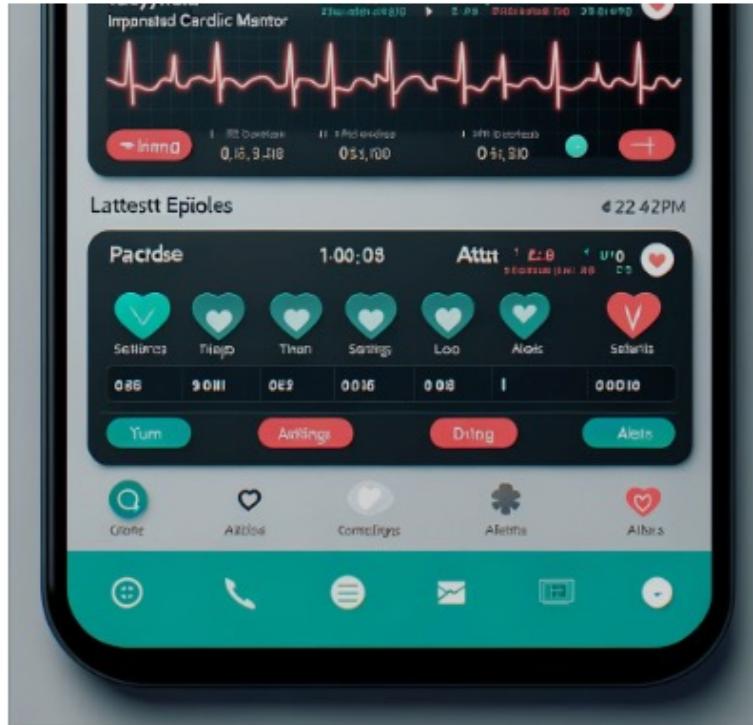
# Other Tools Can Be Used As Well!



Mermaid

PlantUML

# Prompt: How about a mock mobile application interface too?



Here's a mockup for the Android application interface, designed to display real-time ECG data from an implantable cardiac monitor. This visualization focuses on delivering an intuitive user experience, with a clear, real-time ECG graph at the top for immediate visibility. Below the graph, you'll find sections for the latest episodes, categorized by bradycardia, tachycardia, pause, and atrial tachycardia/fibrillation, each with a timestamp and a brief description for quick reference. The navigation bar at the bottom allows easy switching between the Dashboard, Episode Log, Settings, and Alerts, ensuring that all critical features are accessible within a few



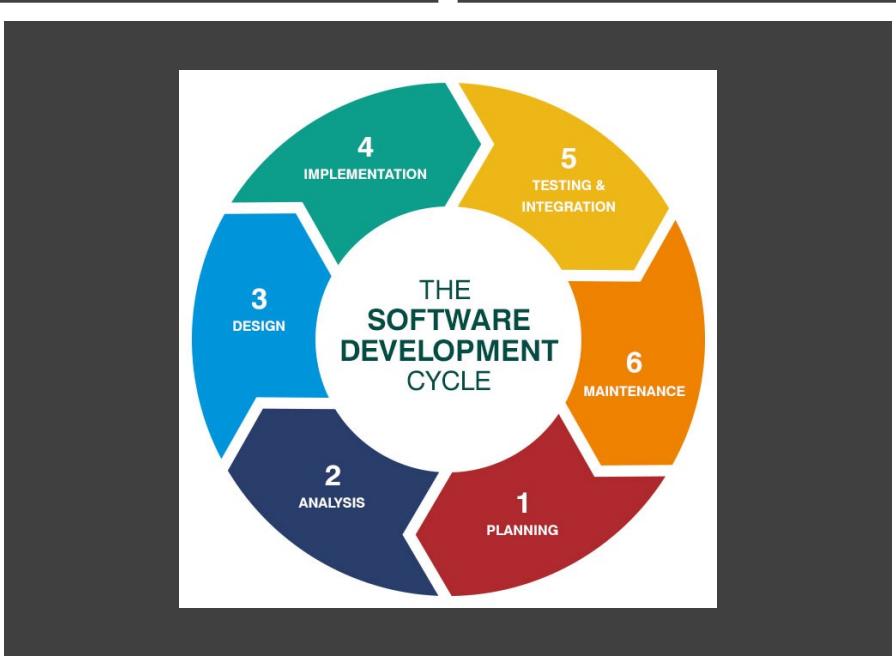
Eliana – Software Engineer



Lucas – Quality Assurance



Taylor – Systems Engineer



Alex - Product Manager

# Software Engineer Engineer (Eliana)

---

- **Challenge**
  - Implementing features and functionality
  - Adjusting to changing requirements
  - Questions and clarifications on specific points in the requirements
- **Solution with LLMs**
  - Code generation
  - Asking questions of stakeholders
  - Exploring alternative solutions



# Prompt:

Generate me code that I would use to access an ECG signal on an ARM Cortex-M Series processor



ChatGPT

To access an ECG signal on an ARM Cortex-M series processor, you would typically use an Analog-to-Digital Converter (ADC) peripheral available on the microcontroller to read the analog ECG signal and convert it to a digital value for further processing. Below is a simplified example of how you might set up the ADC and read a value from it in C, which is commonly used for ARM Cortex-M processors.

Note that this is a generic code template; the actual implementation details will depend on the specific Cortex-M series MCU and development environment you're using:

c

 Copy code

```
#include "stm32f4xx_hal.h" // Change this include to match your specific

// Initialize the ADC
void ADC_Init(void)
{
    ADC_HandleTypeDef hadc;

    // Configuration for the ADC peripheral, including setting the clock,
    hadc.Instance = ADC1; // Use the correct ADC instance for your MCU
    hadc.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV2;
    hadc.Init.Resolution = A ↓ RESOLUTION_12B;
    hadc.Init.ScanConvMode = DISABLE;
```

# Prompt: Please generate code in Kotlin to take data from a BLE based wireless device and send it to REST based APIs in the cloud.



## Prerequisites:

1. Add Bluetooth permissions to your Android manifest.
2. Add Internet permission to your Android manifest for REST API communication.
3. Ensure you have a BLE library added to your build.gradle, such as RxAndroidBle if you want reactive extensions.
4. Have the REST API endpoints and specifications ready for use.

## Bluetooth Data Collection:

First, you need to see how you could implement

```
kotlin\n\n// Assuming you\nimport com.polidea.rxandroidble.RxBle\nimport com.polidea.rxandroidble.RxBleClient\n\n// ... Inside your class\n\nprivate lateinit var client: RxBleClient\n\n// Call this in onCreate()\nfun initializeRxBLE()
```

## Sending Data to REST API:

After collecting the data, you can send it to your REST API using a library like Retrofit:

First, define your REST API interface:

```
kotlin\n\nimport retrofit2.http.Body\nimport retrofit2.http.POST\n\ninterface ApiService {\n    @POST("your_api_endpoint")\n    fun sendData(@Body data: EcgData)\n}\n\n// Assuming EcgData is your data class\n\nclass EcgData /* ... properties */\n\nclass ResponseData /* ... properties */
```

Next, you'd use Retrofit to send the data:

```
kotlin\n\nCopy code\n\nimport retrofit2.Retrofit\nimport retrofit2.converter.gson.GsonConverterFactory\n\n// Initialize Retrofit\nval retrofit = Retrofit.Builder()\n    .baseUrl("https://your_api_base_url")\n    .addConverterFactory(GsonConverterFactory.create())\n    .build()\n\nval service = retrofit.create(ApiService::class.java)\n\n// Send the data\nval ecgData = EcgData(/* initialize with actual data */)\nval call = service.sendData(ecgData)\n\ncall.enqueue(object : Callback<ResponseData> {\n    override fun onResponse(call: Call<ResponseData>, response: Response<\n        ResponseData>)\n        if (response.isSuccessful) {\n            // Data sent successfully\n        }\n    }\n})
```

# Prompt:

## Create automated unit test cases for the supplied source code.



ChatGPT

To create test cases for the provided code snippets, we'll focus on unit testing each component separately. We'll use frameworks like JUnit and Mockito to write and mock the tests.

### Test Cases for Bluetooth Connectivity (RxAndroidBle)

Test Case 1: Test Bluetooth Client Init

**Objective:** Ensure that the Bluetooth

- **Test Steps:**

1. Call `initializeBleClient()`
2. Verify that the `rxBleClient` object is not null.

- **Expected Result:** `rxBleClient` object is not null.

Test Case 2: Test Bluetooth Device Connection

**Objective:** Verify that the app can scan for devices and connect to a device.

- **Test Steps:**

1. Mock the BLE scan result.
2. Call `connectToDevice()` function.

### Example Test Code (using Mockito and JUnit)

kotlin

```
import com.polidea.rxandroidble2.RxBleClient
import com.polidea.rxandroidble2.scan.ScanResult
import org.junit.Assert.assertNotNull
import org.junit.Before
import org.junit.Test
import org.mockito.Mockito
import org.mockito.Mockito.`when`
import org.mockito.MockitoAnnotations
import retrofit2.Call
import retrofit2.Callback
import retrofit2.Response

class BleApiTest {

    @Mock
    lateinit var rxBleClient: RxBleClient

    @Before
    fun setup() {
        MockitoAnnotations.initMocks(this)
    }

    @Test
    fun testScanForDevices() {
        // Arrange
        val scanResult = ScanResult("device1", "12:34:56:78:90:AB", -50)
        `when`(rxBleClient.scanForDevices()).thenReturn(scanResult)

        // Act
        val result = rxBleClient.scanForDevices()

        // Assert
        assertEquals(scanResult, result)
    }
}
```

Copy code

# Prompt: Taylor sent me a mockup of the mobile app. Generate the code for that image.



## UX/UI Designer

To help kickstart the development of the Android application for displaying real-time ECG data, I'll provide a simple example that includes the main components: an activity to display the ECG graph and setup for receiving real-time data updates. This example will use Kotlin, a popular language for Android development.

### MainActivity.kt

This is the main activity for the ECG graph in real-time.

```
kotlin\n\npackage com.example.ecg\n\nimport androidx.appcompat.app.AppCompatActivity\nimport android.os.Bundle\nimport com.jjoe64.graphview.GraphView\nimport com.jjoe64.graphview.Plot\nimport com.jjoe64.graphview.series.DataPoint\n\n\nclass MainActivity : AppCompatActivity()\n{\n    \n    \n}
```

### activity\_main.xml

This XML file defines the layout for your main activity, including the GraphView for the ECG data.

```
xml\n\n<?xml version="1.0" encoding="utf-8"?>\n<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"\n    xmlns:tools="http://schemas.android.com/tools"\n    xmlns:app="http://schemas.android.com/apk/res-auto"\n    android:layout_width="match_parent"\n    android:layout_height="match_parent"\n    tools:context=".MainActivity">\n\n    <com.jjoe64.graphview.GraphView\n        android:id="@+id/ecgGraph"\n        android:layout_width="match_parent"\n        android:layout_height="wrap_content"\n        android:layout_alignParentTop="true"\n        app:layout_constraintTop_toTopOf="parent" />\n\n    <!-- Add other UI elements here -->\n\n</RelativeLayout>
```

Copy code



# Prompt: Create a service in Python on Google Cloud that will ingest the ECG data from the Android device.



## 1. Define Your API

You would start by defining the API that the Android device will communicate with. This is typically done using an OpenAPI specification or gRPC, which Cloud Endpoints supports.

## 2. Implement the Service

Implement a service using a framework like Flask for Python or Spring Boot for Java.

```
python
from flask import Flask, request
app = Flask(__name__)

@app.route('/upload_ecg')
def upload_ecg():
    ecg_data = request.json
    # Here you would add logic to store the data
    return jsonify(success=True)

if __name__ == '__main__':
    app.run(host='127.0.0.1', port=5000)
```

## 3. Deploy to App Engine

You'll need to deploy this service to Google App Engine, which will automatically scale your service according to demand.

## 4. Store the Data

For the actual data storage, you could use Firestore for a NoSQL approach or Cloud SQL if you prefer a relational database. The choice depends on the structure of your data and the queries you'll need to run.

## 5. Secure the API

You'll want to make sure that your API is secure. This can involve setting up authentication using Identity-Aware Proxy (IAP) or Firebase Authentication, and ensuring that all data is sent over HTTPS.

## 6. Test Your API

Before going live, thoroughly test your API using tools like Postman or directly from your Android application to ensure that it can accept and process ECG data correctly.

## 7. Monitor and Log

Make use of Google Cloud's monitoring and logging tools like Cloud Monitoring and Cloud Logging.



**Prompt:** We are using Firestore. Show me how I would create and query the database with this information.



## Step 1: Set up Firestore in your Google Cloud Project

1. Go to the [Google Cloud Console](#).
2. Select or create a new Google Cloud project.
3. Navigate to 'Firestore' in the left menu.
4. Click 'Create database'.
5. Choose 'Start in production mode' if it fits your needs.
6. Select a location for your Cloud Firestore database to be stored.
7. Click 'Create Database'.

## Step 2: Define Your Data Model

For ECG data, you might define a data model with documents that contain fields for the ECG data, timestamp, patient ID, etc. Here's an example structure:

- Collection: `ecg\_readings`
  - Document: `ecg\_reading\_<UNIQUE\_ID>`
    - Fields:
      - `patient\_id`: string
      - `timestamp`: timestamp
      - `heart\_rate`: number
      - `rhythm`: string

## Step 3: Add Data to Firestore

From your server code (assuming you're using Node.js), you'd use the Firestore SDK like so:

```
javascript
const { Firestore } = require('@google-cloud/firestore');

// Instantiate a new Firestore instance
const firestore = new Firestore();

// Reference to the collection
const ecgReadingsCol = firestore.collection('ecg_readings');

// Data from your ECG device
const ecgData = {
  patient_id: 'patient123',
  timestamp: Firestore.Timestamp.now(),
  heart_rate: 60,
  rhythm: 'Normal Sinus Rhythm',
  timestamp: new Date()
};

// Write the data to the collection
ecgReadingsCol.add(ecgData)
```

# Prompt: Please review the following code. I'm getting a ClassCastException, but I don't understand why?



ChatGPT

The `ClassCastException` you're encountering is due to a mistake in how you're attempting to retrieve the `BluetoothGattCharacteristic` from the `BluetoothGatt` object. Specifically, the issue arises in these lines:

kotlin

Copy code

```
// Incorrectly using getService instead of getCharacteristic  
val ecgCharacteristic: BluetoothGattService? = gatt.getService(ecgCharacteristicUUID)  
if (ecgCharacteristic == null) {  
    println("ECG characteristic not found.")  
    return  
}
```

In this snippet, you are calling `gatt.getService(ecgCharacteristicUUID)` with the intention of getting a `BluetoothGattCharacteristic`. However, `getService()` returns a `BluetoothGattService`, not a `BluetoothGattCharacteristic`. This misunderstanding leads to trying to cast a `BluetoothGattService` to a `BluetoothGattCharacteristic`, which is incorrect and causes the `ClassCastException`.

Here's how to fix the code:



```
// This will cause a class cast exception since we're treating a service as a characteris
val ecgData: ByteArray? = (ecgCharacteristic as BluetoothGattCharacteristic).value
if (ecgData != null) {
    processEcgData(ecgData)
} else {
    println("No ECG data received.")
}
```

```
fun processEcgData(data: ByteArray) {
    println("Received ECG data: ${data.joinToString(", ")}")
```

I'm getting a ClassCastException, but I don't understand why?

Using `/fix` to generate response (re-run without)

To fix the ClassCastException, you need to change the line where `ecgCharacteristic` is assigned to  
`gatt.getCharacteristic(ecgCharacteristicUUID)` instead of  
`gatt.getService(ecgCharacteristicUUID)`.

Accept

Discard



1 change



Copilot For The Win! Fix it inline..

src/modules/noble/noble.service.ts

Comments on lines 47 to 51

```
47      +  @Cron('0 3 * * *', { name: 'old-job-cleanup', timeZone: 'Europe/London' })
48      +  async oldJobCron() {
49      +    console.log('Cleaning up old jobs');
50      +    await this.cleanupOldJobs();
51      }
```

 **CodeRabbit** Bot 1 min ago

Contributor ...

The cron expression '0 3 \* \* \*' schedules the job to run every second of the 3rd minute of every hour, which does not align with the intention to run it daily. Consider using '0 0 3 \* \*' to schedule the job to run daily at 3 AM.

```
-  @Cron('0 3 * * *', { name: 'old-job-cleanup', timeZone: 'Europe/London' })
+  @Cron('0 0 3 * *', { name: 'old-job-cleanup', timeZone: 'Europe/London' })
```

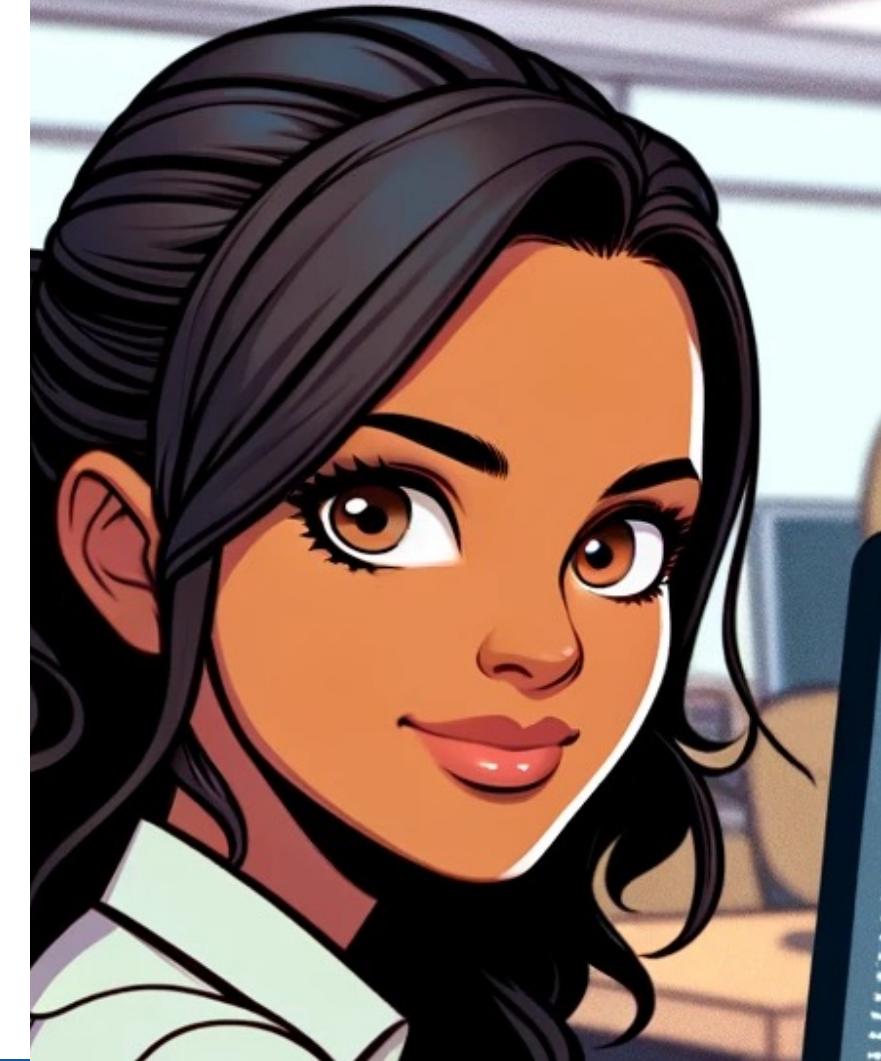


► Committable suggestion

 **johnsmith** Now

Collaborator Author ...

@coderabbitai Great catch! Just fixed it.



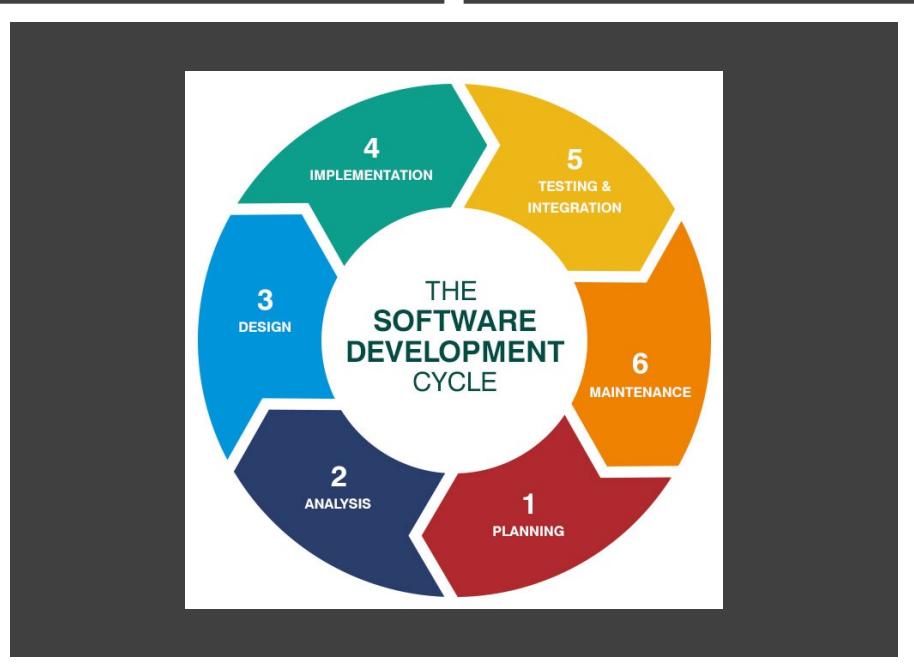
# AI Code Reviews



Eliana – Software Engineer



Taylor –Systems Engineer



Lucas –Quality Assurance



Alex - Product Manager

# Quality Assurance (Lucas)

---

- **Challenge**
  - Ensuring that tests cover all possible use cases
  - Keeping the testing framework and test cases up to date is challenging
- **Solution with LLMs**
  - Automatic generation and updating of testing documentation
  - Parse user-submitted bug reports
  - Knowledge and tutorials on new testing methodologies and technologies (Mocks)



# Prompt: Create a test plan based on the functionality in the specification

---

## Performance Test Cases

### 1. TP1: Connection Timeliness

- **Objective:** Measure the time taken to establish a BLE connection.
- **Expected Result:** Connection occurs within an acceptable timeframe.

### 2. TP2: Data Upload Latency

- **Objective:** Determine the response time for data uploads to the cloud.
- **Expected Result:** Data is uploaded within the required performance benchmarks.

## Security Test Cases

### 1. TS1: Data Encryption Verification

- **Objective:** Confirm that data transmission uses encryption.
- **Steps:** Analyze network packets during data transmission.
- **Expected Result:** Data packets are encrypted.

### 2. TS2: Unauthorized Access Attempt

- **Objective:** Test for proper handling of unauthorized access attempts.
- **Steps:** Attempt to access the cloud API without proper credentials.
- **Expected Result:** Access is denied.

## Functional Test Cases

### 1. TC1: Device Scan and Discovery

- **Objective:** Verify the application can discover BLE devices.
- **Steps:** Initiate a scan for BLE devices within range.
- **Expected Result:** The application lists all available BLE devices.

### 2. TC2: Device Connection

- **Objective:** Verify the application can connect to a BLE device.
- **Steps:** Select a BLE device from the list and attempt to connect.
- **Expected Result:** The application successfully establishes a connection.

### 3. TC3: Data Reception from BLE Device

- **Objective:** Verify the application correctly receives data from the BLE device.
- **Steps:** Check if the incoming data stream from the BLE device is captured.
- **Expected Result:** ECG data is received and displayed correctly.

### 4. TC4: Data Transmission to Cloud

- **Objective:** Ensure the application can send data to the cloud via REST API.
- **Steps:** Trigger the data upload function after receiving data from the BLE device.
- **Expected Result:** Data is sent to the cloud, and a success confirmation is received.

# Prompt:

## Create acceptance test criteria and functional test cases and code



### Acceptance Test Criteria

**Feature:** BLE Device Connectivity

**Scenario:** Connecting to a BLE ECG Device

- Given the BLE is enabled on the smartphone,
  - And the ECG device is powered on and discoverable.
  - When the user initiates a scan for nearby devices.
  - Then the app should discover the ECG device.
  - And allow the user to connect to the device.
  - And successfully establish a connection.
1. **Mock the BLE environment:** Use Mockito to mock the `RxBleClient` and simulate the discovery of a BLE ECG device.
  2. **UI interactions:** Use Espresso to simulate user actions that trigger BLE device scanning and connection within the app.
  3. **Validation:** Verify that the app's UI updates to reflect a successful connection and that data retrieval begins.

### Functional Test Case and Code Example

**Objective:** Validate the app's ability to connect to and interact with a BLE ECG device.

**Tools:** Assuming the use of Espresso for UI interactions and Mockito for mocking operations within the Android framework.

Test Code Example

```
kotlin
@RunWith(AndroidJUnit4::class)
class BLEConnectivityTest {

    @Mock
    private lateinit var mockRxBleClient: RxBleClient

    @Mock
    private lateinit var mockRxBleDevice: RxBleDevice

    @Mock
    private lateinit var mockRxBleConnection: RxBleConnection

    @Before
    fun setup() {
        MockitoAnnotations.initMocks(this)
    }

    // Test cases...
}
```

# Prompt: A user of our system can't see anything on the screen



Here's a guide for Lucas to assist the customer:

## Troubleshooting Steps for the Customer:

1. **Restart the App:** Ask the customer to restart the app to see if the issue persists.
2. **Restart the Device:** Suggest restarting the customer's device to rule out any temporary system glitches.
3. **Check Bluetooth Permissions:** Ensure that the app has necessary permissions to access Bluetooth. In `AndroidManifest.xml`, verify that the `<uses-permission android:name="android.permission.BLUETOOTH"/>` and `<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>` permissions are declared.
4. **Check Location Permissions (if app is running on API level 23 or higher):** Verify that the app has location permissions declared in `AndroidManifest.xml`. The `<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>` and `<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>` permissions must be declared in `AndroidManifest.xml`. Ensure that the app has granted location permissions to the application.
5. **Verify BLE Device Connectivity:** Ensure that the app can successfully connect to BLE devices.

In the provided code, there are several areas where a software engineer should focus when troubleshooting issues related to a blank screen or lack of data flow in the system. Here are some specific areas to examine:

### 1. Bluetooth Connectivity (RxAndroidBle):

- **Initialization:** Check if the Bluetooth client (`rxBleClient`) is initialized properly in the `initializeBleClient()` function. Ensure that any required context or permissions are provided correctly during initialization.
- **Scanning and Connection:** Review the `connectToDevice()` function to verify that BLE device scanning and connection establishment are functioning as expected. Check for any potential errors in scanning BLE devices or establishing connections.
- **Error Handling:** Ensure that proper error handling mechanisms are implemented, especially in the event of failed BLE device scanning or connection attempts. The `subscribe()` function should handle both successful and error cases appropriately.

### 2. REST API Integration (Retrofit):

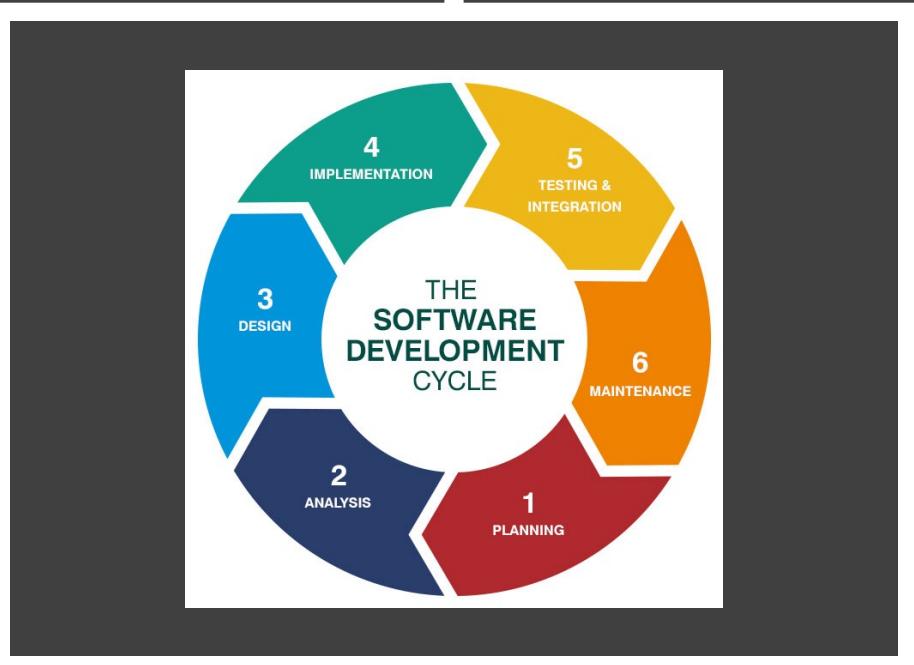
- **Initialization:** Confirm that Retrofit is initialized correctly with the base URL and Gson converter factory. Ensure that the service interface (`AniService`) is defined and used correctly in the code.



Eliana – Software Engineer



Taylor –Systems Engineer



Lucas –Quality Assurance



Alex - Product Manager

# How can I use LLMs?



Alex – Project Manager

- Voice of the customer(s)
  - Doctors
  - Patients
  - Nurses
- Generate surveys & questions
- Summarize unstructured data
  - Survey responses
  - Interviews
- Generate and/or interrogate the PRD
- Create & ideate on the name of the product
- What am I missing or not asking?

# How can I use LLMs?



Taylor –Systems Engineer

- Product Requirements Document
  - Dissecting
  - Summarizing
- Domain Specifications
  - Healthcare certifications
  - Technology selection
- Architecture diagrams & flowcharts
- User Interface
- What am I missing or not asking?

# How can I use LLMs?



Eliana – Software Engineer

- Product Requirements Document
  - Summarize
  - Find areas of ambiguity to ask stakeholders
  - Suggest implementation alternatives
- Generate software code
- Generate test cases
- Perform code reviews
- Tracking down bugs

# How can I use LLMs?



Lucas – Quality Assurance

- QA
  - Generation of acceptance tests
  - Generating documentation & FAQs
  - Create release notes
- Support
  - Categorize & prioritize support tickets
  - Sift through prior solutions and suggest fixes
- Training and onboarding new members

# What About My Data?

---

- Enterprise Privacy at OpenAI
  - <https://openai.com/enterprise-privacy>

## Ownership: You own and control your data

- ✓ We do not train on your business data (data from ChatGPT Team, ChatGPT Enterprise, or our API Platform)
- ✓ You own your inputs and outputs (where allowed by law)
- ✓ You control how long your data is retained (ChatGPT Enterprise)

## Control: You decide who has access

- ✓ Enterprise-level authentication through SAML SSO (ChatGPT Enterprise and API)
- ✓ Fine-grained control over access and available features
- ✓ Custom models are yours alone to use and are not shared with anyone else

## Security: Comprehensive compliance

- ✓ We've been audited for SOC 2 compliance (ChatGPT Enterprise and API)
- ✓ Data encryption at rest (AES-256) and in transit (TLS 1.2+)
- ✓ Visit our Trust Portal to understand more about our security measures

# Learnings & Key Takeaways

- **If your team is not leveraging LLMs you are falling behind by lost productivity.**
- Companies that once were too small are competitors.
- “Still needs to get better before I’ll use it”
  - Only 18 months since ChatGPT
  - Imagine 18 months from now?
- “AI can’t do creative tasks”. FALSE!
- Good Prompting, Custom GPTs & Specific Models



Need Help  
Assessing  
Your  
Options?

Talk with me!



# Thank you!

**Justin Grammens**

Founder + CEO

[justin@lab651.com](mailto:justin@lab651.com)

<https://lab651.com>

<https://recursiveawesome.com>

Join us at an upcoming  
Applied AI event!

<https://appliedai.mn>

