electronics

MDPI

Article

Application of Model-Based Software Testing in the Health Care Domain

Pragya Jha [1], Madhusmita Sahu [1], Sukant Kishoro Bisoy [1] and Mangal Sain [2,*]

[1] Department of Computer Science and Engineering, C. V. Raman Global University, Bhubaneswar 752054, India; pragya.2k6@gmail.com (P.J.); msahu@cgu-odisha.ac.in (M.S.); sukantabisoyi@cgu-odisha.ac.in (S.K.B.)
[2] Division of Computer and Information Engineering, Dongseo University, Busan 47011, Korea
* Correspondence: mangalsain1@gmail.com

Abstract: The human body's reaction to various therapeutic medications is critical to comprehend since it aids in the appropriate construction of automated decision support systems for healthcare. Healthcare Internet of Things (IoT) solutions are becoming more accessible and trusted, necessitating more testing before they are standardized for commercial usage. We have developed an activity diagram based on the Unified Modeling Language (UML) to represent acceptability testing in IoT systems. The activity flow graph is used to extract all of the necessary information by traversing the activity flow diagram from start to finish, displaying all its properties. In this paper, a test case is generated to compute the type of diabetes using blood sugar test results, estimate the kind of diabetes, and the probability that a person would get diabetes in the future. We have demonstrated how these test cases can function using a telehealth care case study. First, we offer a high-level overview of the topic as well as a design model working diagram. The test case creation method is then outlined using the activity diagram as a guide.

Keywords: unified modeling language; acceptance testing; activity diagram; test case generation

Citation: Jha, P.; Sahu, M.; Bisoy, S.K.; Sain, M. Application of Model-Based Software Testing in the Health Care Domain. *Electronics* **2022**, *11*, 2062. https://doi.org/10.3390/electronics11132062

Academic Editor: Rashid Mehmood

Received: 27 May 2022
Accepted: 28 June 2022
Published: 30 June 2022

1. Introduction

The Internet of Things (IoT) is a network of physical objects and devices that are interconnected and exchange data to a cloud-based central control server. This enables businesses to remotely monitor and manage their equipment and resources. As IoT technology continues to develop, we will see a variety of new applications and systems using IoT technology in many different contexts. To ensure the safety of IoT systems, it is important that they are reliable, secure, and compliant. Testing IoT systems can be difficult because there are a large number of different technologies used to develop them [1]. There is little known about the use of IoT software testing in industry, mainly due to the lack of research into the topic. There is little evidence to suggest that approaches or proposals in this area are effective. This is clear from a review of the relevant scientific literature, which shows that proposals and approaches in this area are uncommon [2].

When it comes to developing and testing software, the healthcare industry is one of the most demanding and distinctive. With wearable technology, hospital indexing systems, and myriad other advancements, product businesses in this field are helping doctors, patients, and other medical professionals redefine what is possible [3–6]. Because of the intricacy of these new goods, rigorous and strict testing is required, as the quality can have a direct impact on a patient's life. The cost and value of the product to the consumer, the preservation of private and confidential patient data, and the safety of all patients or caregivers who interact with the product are all high-stakes aspects to consider.

In [7], a remote health monitoring and data analysis was proposed by combining IoT and deep learning techniques. A unique IoT-based FoG-assisted cloud network architecture has been suggested, which collects real-time health care data from patients via numerous

medical IoT sensor networks and analyzes the data using a deep learning algorithm. The suggested framework not only analyzes healthcare data, but also offers rapid assistance to patients who are in urgent conditions and require immediate medical attention. Such immense improvements in the field of healthcare domain also require an improvement in the software testing techniques.

In truth, the healthcare and insurance sectors have seen significant transformations in a short period of time. As a result, healthcare goods must be precise and accurate, necessitating thorough testing of healthcare applications. Software testing is a term that encompasses all factors that can affect the product's quality. Patients' sensitive data, particularly their health information, need a high level of security. If a healthcare application is not properly secured, it might result in serious data breaches [8]. Security testing is required to make the application secure and fail-safe. It aids in making the application long-lasting and error-free in a variety of demanding scenarios. Expertise and efficiency are critical in healthcare systems in order to provide better and more effective care to patients. Given the complexity of healthcare apps, it is critical to guarantee that they run properly and without glitches. Software testing guarantees that the application runs smoothly and gives users a complete experience.

The healthcare business generates a large amount of data, which includes detailed patient information and their health problems. This information is critical to healthcare organizations because it aids in the development of appropriate strategies and the production of appropriate products. Big data solutions aid in making informed judgments on disease cures, research and development, and a variety of other topics. It is critical that this data be thoroughly tested to ensure that it is implemented correctly and produces the desired results.

However, there has been very little research carried out on the applicability of testing the software used in healthcare domain, and all the more interesting is the lack of acceptance testing in this domain. In [9], the advantages and drawbacks of utilizing model-based testing (MBT) to evaluate healthcare software systems were discussed without any proposal of how the testing could be done. In [10], numerous unique mobile usability evaluation approaches, ranging from the least to the most invasive, as well as their effects on the quality of the usability data obtained, were discussed. The advantages and disadvantages of various methods are also highlighted. In both [10,11], a usability testing procedure was used. In [12,13], model-based and online healthcare testing were studied and their challenges and methods were described in detail. In usability testing, the same scenarios are simulated and tested in both real-time and virtual environments. Automation has made this role easier for testers in recent years by simplifying the entire process. Usability testing also aids in the improvement of the user interface and overall experience of healthcare apps. New monitoring technologies and methodologies, on the other hand, are bringing unforeseen problems and making it increasingly difficult to evaluate healthcare applications.

However, we found that acceptance testing for software tools that used the healthcare domain have not been utilized before. In this regard, we present a scheme for Internet of Healthcare Technology (IoHT) systems' acceptance testing that makes use of the user interface (UI) as that of the primary medium of contact between the system and the user. Using empirical data, our analysis verifies this strategy. Acceptance testing is a black box test that is built on the test cases premise and verified with a UML activity diagram, an algorithm, and the results. The test scenario for the IoT system described in this article was created from an activity diagram. We used a diabetic test scenario with telehealth care to see how a doctor interacts with a patient over the internet. We have put together a comprehensive overview of our approach based on what we learned from our case study, which may be of use to practitioners who are having similar testing challenges. In fact, gathering the system and testing it as a whole is the most reasonable and realistic way to ensure quality. At the same time, for complicated IoT applications, this task might be quite difficult.

1.1. Basic Definitions and Concepts

This section introduces a few definitions and notations that will be used in this paper. UML activity diagrams are briefly described in this section. An activity diagram can be employed to model the interactive characteristics of a group of objects and emphasize on the group of objects' activities and making them suitable for expressing the execution of an operation during the design stage. The series of activities between all participating objects in the control flow throughout operation execution. In the message flow, it also reflects the interaction between activity and object, and also the state change of object in the object flow during activity execution. Activity diagrams are typically incorporated when the use case's control structure includes loops or branching. Using activity diagrams, you may define a coverage criterion to ensure that the test scenarios are as complete as possible. As demonstrated in Figure 1, this diagram can represent all possible scenarios for a single-use case.

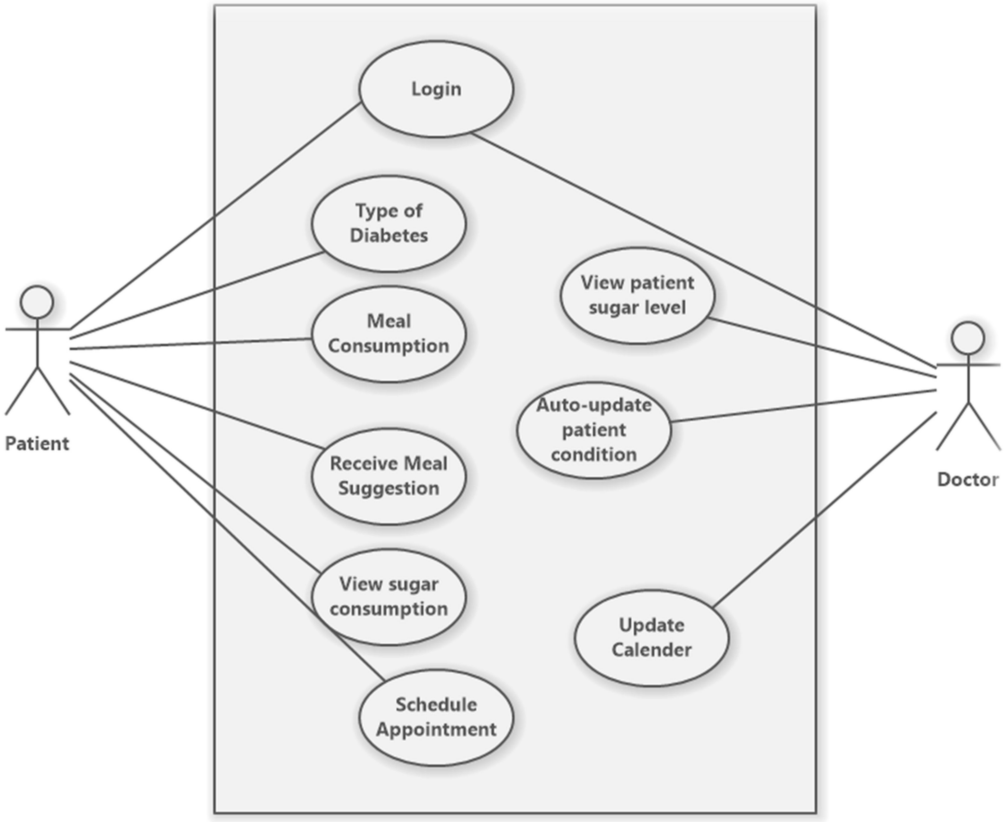

Figure 1. Use case diagram for diabetes telehealth care.

1.1.1. UML Activity Diagram Modeling

An activity diagram like traditional flowcharts enables us to represent a process as an activity composed of nodes connected by edges. It can be linked to any modeling element, such as Use cases, Classes, Interfaces, and Collaborations, to simulate its behavior. It is a directed graph of some sort. The movement of tokens, which represent controlling or information values along the edges from the source node to the sink nodes, is driven by actions and conditions. In an activity diagram, there are two types of modeling elements: *activity nodes* and *activity edges*.

1. *Activity nodes*: There are three different types of nodes in the activity diagrams.
    a. Action nodes (AN): take all input data and control tokens, make new tokens, and transmits them to output activity edges once they are ready.
    b. Control nodes (CN): tokens are routed through the graph by control nodes. The control nodes have components for deciding between different flows (deci-

sion/merge), splitting or merging the flow for parallel processing (fork/join), and so on.

c. Object nodes (ON): these nodes give and take the data tokens. They can also operate as buffers, collecting tokens while waiting to go downstream

2. *Activity edges:* In activity diagrams, there are two different types of edges.

a. Control flow edge: this edge depicts the flow of control throughout the activity

b. Object flow edge: this edge depicts the movement of items during the activity. In this article, we concentrate on the data and control flow of activity diagrams, both of which are critical for test generation.

Definition 1 (Activity Diagram). *An activity diagram* $A_D = (A_s, T_c, C_g, F_r, a_S, a_E)$, *is a 6-tuple where:*

1. $A_s = \{a_{s1}, a_{s2}, \cdots, a_{sm}\}$ *is a set of activity states with a finite number of possibilities;*
2. $T_c = \{t_{c1}, t_{c2}, \cdots, t_{cn}\}$ *is a set of completion transitions with a finite number of possibilities;*
3. $C_g = \{c_{g1}, c_{g2}, \cdots, c_{gn}\}$ *denotes a set of guard conditions, and* C_i *corresponds to the transition* t_i *;*
4. $F \subset (A_s \times T_c \times C_g) \cup (T_c \times C_g \times A_s)$ *denotes the flow relationship between the transitions* t_i *and activity states* a_i*;*
5. $a_S \in A$ *denotes the starting activity state; the end activity state is denoted by* a_E

In addition, there can be only one transition t such that the following is satisfied

$$\{(a_S, t) \text{ and } (t, a_S) \text{ and } (a_E, t)\} \in F \text{ for any } t.$$

Definition 2 (Test Sequence). *A test sequence,* $t_c \in T_s$, *in an activity diagram,* A_D, *can be defined as an execution path from the starting activity state to the end activity state consisting of activities and transitions, i.e.,* $\forall\, t_c \in T_c$, $t_c = a_{s1} \rightarrow t_{c1} \rightarrow a_{s2} \rightarrow t_{c2} \rightarrow \cdots \rightarrow t_{cm} \rightarrow a_{sm}$, *where* $a_i \in A_s$, $t_i \in T_c$, a_{s1} *is the initial state and* a_{sm} *is the final state.* T_s *is the set of test sequences.*

Definition 3 (Basic Paths (BP)). *We ensure that all action states and transitions are covered and that the loops are only run once while utilizing the DFS (depth first search method) to traverse an activity diagram from the initial activity state to the final activity state. As a result, we can obtain all basic paths.*

Definition 4 (Activity Flow Diagram (AFG)). *An activity flow graph is a directed network in which each node represents a construct, such as a starting node, a flow final node, a decision mode, a guard condition, a fork node, a join node, a merging node, and so on, and each edge represents the flow in the activity diagram.*

Definition 5 (Control Flow Activity Mapping Diagram (CFAMT)). *CFAMT is a table that stores information about each activity corresponding to a node in an activity graph.*

We summarize these definitions in a tabular form for easy understanding of the readers in Table 1.

1.1.2. Testing Scenario: A Diabetes Telehealth Care

We chose a diabetes telehealth care IoT system because these systems are extremely difficult to evaluate, and there are no reliable methods available for evaluating these systems. Many software apps for smartphones and IoT systems for patients are already available, with the goal of assisting them in making health-related decisions quickly and easily [14]. The large number of healthcare systems that rely on active human interaction could benefit from a testing technique in which the UI is exercised as a final user would [15]. A diabetes telehealth care IoT system is a system that:

1. Determine whether the patient has diabetes.
2. Determine whether the patient's blood glucose level is larger than or less than 120.
3. Determine whether the glucose level is greater than or less than 120.
4. Determine the kind of diabetes the cloud-based healthcare system can process large amounts of data and convert it into useful information

Table 1. Basic concepts and definitions.

Activity Diagram	An activity diagram $A_D = (A_s,\ T_c,\ C_g, F_r,\ a_S\ ,\ a_E)$, is a 6-tuple where: 1. $A_s = \{a_{s1},\ a_{s2},\ \cdots,\ a_{sm}\}$ is a set of activity states with a finite number of possibilities; 2. $T_c = \{t_{c1},\ t_{c2},\ \cdots,\ t_{cn}\}$ is a set of completion transitions with a finite number of possibilities; 3. $C_g = \{c_{g1},\ c_{g2},\ \cdots,\ c_{gn}\}$ denotes a set of guard conditions, and C_i corresponds to the transition t_i; 4. $F \subset (A_s \times T_c \times C_g) \cup (T_c \times C_g \times A_s)$ denotes the flow relationship between the transi-tions t_i and activity states a_i; 5. $a_S \in A$ denotes the starting activity state, the end activity state is denoted by a_E Also, there can be only one transition t such that the following is satisfied $\{(a_S\ ,\ t)\ and\ (t\ ,\ a_S)\ and\ (a_E\ ,\ t\ )\} \in F\ for\ any\ t.$
Test Sequence	A test sequence, $t_c \in T_s$, in an activity diagram, A_D, can be defined as an execution path from the starting activity state to the end activity state consisting of activi-ties and transitions, i.e., $\forall\ t_c \in T_c,\ t_c = a_{s1} \rightarrow t_{c1} \rightarrow a_{s2} \rightarrow t_{c2} \rightarrow \cdots \rightarrow t_{cm} \rightarrow a_{sm}$, where $a_i \in A_s,\ t_i \in T_c$, a_{s1}, is the initial state and a_{sm} is the final state. T_s is the set of test sequences.
Basic Paths (BP)	We ensure that all action states and transitions are covered and that the loops are only run once while utilizing the DFS (Depth First Search method) to traverse an activity diagram from the initial activity state to the final activity state. As a result, we can obtain all *basic paths*.
Activity Flow Diagram (AFG)	An activity flow graph is a directed network in which each node represents a construct, such as a starting node, a flow final node, a decision mode, a guard condition, a fork node, a join node, a merging node, and so on, and each edge represents the flow in the activity diagram
Control Flow Activity Mapping Diagram (CFAMT)	CFAMT is a table that stores information about each activity corresponds to a node in an activity graph

1.2. Research Gap and Contributions

There is very little research done in the study of the applicability of the testing the software used in of healthcare domain, especially the acceptance testing. This motivated us to develop an activity diagram for this research area and generate the test cases to predict the diabetes type and their symptoms. The main contributions of our paper are:

1. We have proposed an algorithm based on the depth first search technique to generate test cases to detect diabetes.
2. We present how to draw an activity flow graph (AFG) from an activity diagram.
3. We show how acceptance testing can be used to ensure that the software correctly predicts diabetes type from the blood sugar level.

In Section 2, we review the existing literature on the software testing in IoT. Section 3 forms our main contributory section. We first explain our use-case test scenario and then describe how this use case can be converted to an activity diagram. Then an algorithm required to generate test cases using the depth-first search has been proposed. Considering a general working of a mobile telehealth application as a case study, we have described how the activity diagram, test case sequence, and test case can be generated. In Section 4, we discuss our results and conclude our paper.

2. Related Works

Leotta et al., proposed an experimentally proven unique approach for acceptability assessment of IoT systems employing a user interface (UI), which is the way the user interacts with the device [1]. Because many organizations consider acceptance testing, which is a type of black-box testing built on the notion of the test scenario, i.e., a series of actions done on the user interface, to be among the most efficient way to ensure the quality of a fully implemented system, the author focused primarily on acceptance testing in this paper. They present a method for IoT system acceptance testing that involves interacting with graphical user interfaces. The authors used a mobile health IoT system for diabetes monitoring, which includes sensors, mobile phones, and a distant cloud-based system, as a case study. Because the author is concentrating on the acceptability level, a thorough explanation of the expected system behavior is necessary. Similar studies can also be found in [2,16].

Understanding how the human body reacts to medical treatments is challenging, according to Silva et al., which makes automation of decision support systems in the domain of healthcare rather complicated [17]. The authors of this paper describe a framework for medical cybersecurity that will guide researchers to create test cases for their applications by simulating the functioning of medical equipment and patient data using validated models and component models. As per Leotta et al., IoT software and services are becoming more ubiquitous in our lives, and guaranteeing their quality is crucial [18]. Because there are few suggestions in the literature for testing these complex—and frequently safety-critical—systems, testers are left to build their own test cases. The study done by Leotta et al., is one of the first steps toward IoT-based acceptance testing that uses a smartphone as the primary means of user contact with a complicated system that comprises local sensors and actuators as well as a remote cloud-based system. The simple mobile health (m-health) IoT platform for people with diabetes is used as an example to demonstrate the suggested technique.

Chen et al., proposed employing metamorphic testing (MT), a new software testing technique, to evaluate a variety of bioinformatics systems [19]. MT checks whether a pair of test outputs conform to a set of domain-specific properties known as metamorphic relations (MRs), instead of requiring a mechanism to evaluate if an individual test output is valid. Because incorrectly computed findings can lead to incorrect biological conclusions, program accuracy is crucial. MT is easy to set up and use, and it works well in discovering defects in both real-world and intentionally flawed systems. The authors explain how MT may be used to test algorithms from a wide range of bioinformatics areas. The integrated clinical environment (ICE) conceptual functional model can be found in the literature [20]. This is a series of recommendations for safely integrating medical equipment and other devices into a healthcare system; its purpose is to assist medical systems in becoming more error-resistant, hence improving patient safety and treatment efficacy.

According to the author in [21], traditional clinical environments are considered as closed-loop systems in which caregivers act as controllers, medical devices act as sensors and actuators, and patients act as physical plants. MCPS (Medical cyber-physical system) is a sort of CPS that alters this picture by introducing additional computational entities that aid the caregiver in directing the plant, i.e., decision support. Some related works that deal with these issues in tandem have been listed in the paper. A clinical scenario for patient-controlled analgesia has been proposed in [22,23] that can benefit from the closed-loop approach to drug delivery. Both employ the ICE conceptual model. Jiang et al., describe a closed-loop testing environment in which a deployable cardiac pacemaker system is controlled by a patient model, specifically a formal model of the human heart [24]. Similar work has also been done in [25–27]. The goal is to determine how safe and effective the gadget is in relation to the patient's condition.

Some studies show that they primarily concentrate on a few medical situations with models constrained to the components involved (e.g., device models) and variables in those circumstances (e.g., physiological parameters and vital signs) [28,29]. Neither of them pays

attention to the simulation that depicts changes in the human body, such as heart rate, diabetes readings, blood pressure, and body temperature. They don't present the evidence for their indicators; instead, they show a patient model with some of them.

The software industry has grown significantly over the last few decades, owing to recent advances in artificial intelligence. Deep learning (DL) is changing the landscape of software engineering generally, in both research and industry. DL has been discovered to have had a significant impact on the way we approach software testing over the last two decades. Since most organizations have turned to automation testing to bridge the gap that exists between the increasing complexity of deliverable software and the contraction of the delivery cycle, the gap has been widening at an alarming rate, bringing us closer to a tipping point where test automation will fail to deliver quality software on time. DL can help us close this gap and streamline our software delivery process, saving us a significant amount of time and effort. So far, the use of DL in software testing automation has been very successful in some areas [30–33].

The Unified Modeling Language (UML) is a collection of tools for documenting system analysis. Although UML is widely used to describe and evaluate the operation of complex systems, its application to the health care domain has received little attention. Despite the fact that UML models are designed to help reduce problem complexity, as product sizes and complexities grow, UML models themselves become large and complex, involving thousands of interactions across hundreds of objects. In UML, interaction, activity and state machine diagrams can be used to represent the behavior of a use case. Sequence diagrams depict the exchange of messages between objects while a use case is being executed. It is concerned with the order in which the messages are sent. Activity diagrams, on the other hand, concentrate on control flow and object-based relationships. These are very useful for visualizing how several objects work together to complete a task. In the area of healthcare domain, very little work has been done in this aspect, and readers may refer to [34–36].

3. Proposed Work

The goal of this article is to focus on the acceptability level; a detailed explanation of the system's expected behavior is required. Our approach is based on an activity diagram model that describes the system's behaviors and is used to generate code and test cases [37]. Essentially, our concept demonstrated expected system behavior, such as diabetes telehealth care as a tool for remote consultation and an electronic record for self-management. Based on these findings, Figure 1 formalizes the expected behavior of diabetes telehealth care, i.e., recognizes the actual status of the patient and doctor by allowing patients to track diabetes, which type of diabetes, how much sugar they consume in daily meals, insulin dose, glucose sensor, and insulin pump.

The function two users can conduct with the application includes calculating how much food to eat, viewing sugar consumption levels, obtaining meal suggestions, and learning about different types of diabetes. A simplified form of diabetes telehealth care is presented below:

i. The healthcare cloud system is a deterministic system with accurate and predictable behavior, and
ii. We overlook the part of telehealth care dealing with the doctor application because it is the same as the patient application

3.1. Use Case Test Scenario

First, we take a test case as a use case scenario; Figure 1. All possible interactions between the patient and the doctor with the application behavior are displayed.

3.2. Case Study for an Activity Definition of a Scenario and a Test Case

In the unified modeling language (UML), an activity diagram represents each step of the patient's condition. A test case is a set of activities that are done on the IoT system that is being evaluated. In test cases, all possible scenarios indicated by the UML activity

diagram must be tested. A flow chart and its modeling elements, nodes, and edges are used to extract the fundamental notion of a UML activity diagram. The action states, activity states, decisions, objects, and signal senders and receivers are all represented as nodes in the process. Black boxes indicate activity action phases, arrows indicate transitions, and branches are represented by a diamond with an approaching arrow and several departure arrows [38].

The workflow of a complex procedure can be represented using an activity diagram. This research focuses on UML activity diagrams, which represent the operations that are used to construct test cases.

Figure 2 shows a UML activity model for diabetes telehealth care [14], which can be automatically analyzed to extract relevant information and generate test cases.

1. The patient enters his or her login credentials.
2. The existing database of registered users is used to validate these credentials.
3. The user gets forwarded to the dashboard page if the login is successful.
4. The server sends the SMS to the doctor for verification when the patient selects the appointment schedule.
5. The patient then waits for the doctor to send a confirmation text message.
6. Once the confirmation is received, the doctor will review all of the patient's issues as well as the type of diabetes they have. They will then recommend how much food to eat, how to monitor sugar consumption, how to receive meal suggestions, and how to update their app for future use.

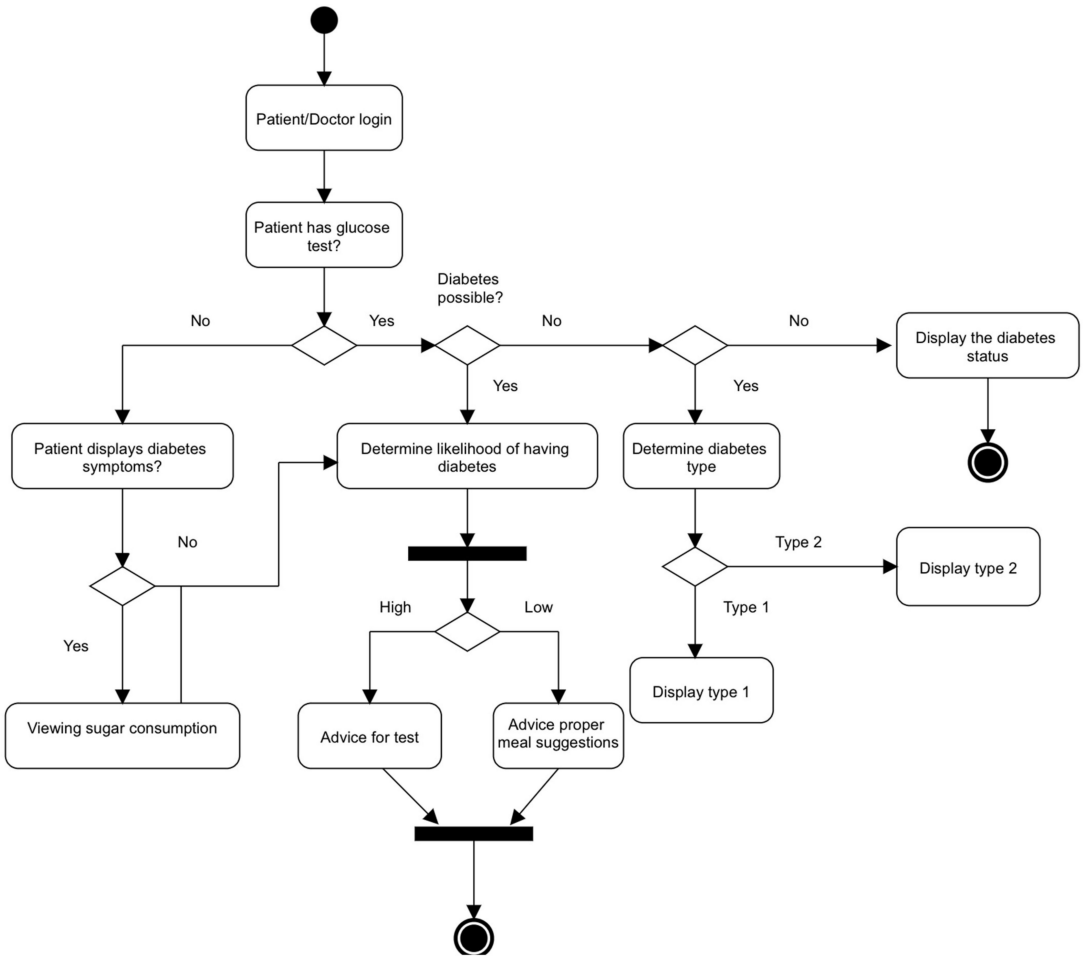

Figure 2. Activity diagram diabetes telehealth care.

3.3. Generation of Test Cases from the Activity Diagram

In this section, we will go over how to make test cases from an activity diagram in this part. Figure 3 shows a schematic representation of our technique for creating test cases using the activity diagram. The phases of recommended approach for creating a test scenario are explained below.

1. Create an activity diagram for the specific use case, complete with the relevant test data.
2. Create an activity flow graph from the activity diagram.
3. Extract the relevant information by traversing the activity flow graph.
4. Create test cases using the activity diagram as a guide.

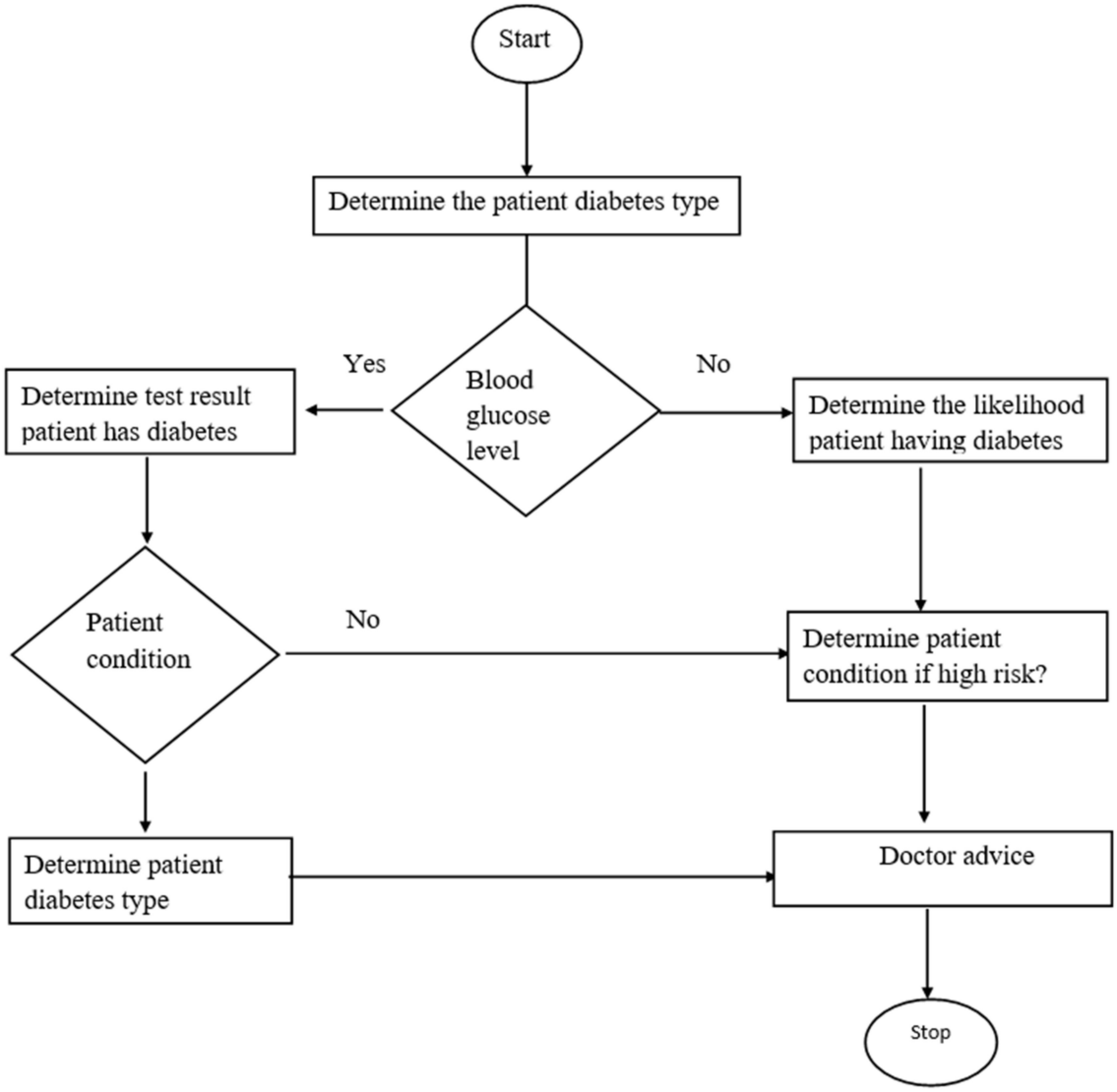

Figure 3. Flow chart diagram diabetes telehealth care.

The rules for modeling the information required for testing in an activity diagram are described below, followed by an example.

A. *Construction of the activity diagram with the required test information:* In order to develop test scenarios, we employ UML models to define a system's need. One or more activity diagrams can be used to depict each use case. A telehealth care scenario is depicted in Figure 1.

B. *The scenario related to a use case (Figure 2) is represented in activity diagrams:* A scenario is a completed "path" across the activity diagram where users of the system have several alternatives for carrying out the functionality indicated in the use case. The main scenario begins at the start node and goes without mistake through all intermediate nodes until it reaches the end node. The guidelines for modeling relevant test data into an activity diagram are outlined in the next paragraph.

C. *Converting activity diagram into activity flow graph:* An activity diagram to activity graph conversion approach is provided. The following steps are used to convert the activity diagram into an activity flow graph:
   a. The activity flow graph is constructed from start to finish node displaying options, conditions, concurrent executions, and loop expressions.
   b. Create a control flow activity entry for each conditional expression. Create nodes in the activity flow graph by traversing the control flow graph.
   c. Conditional statements are created from loop statements.
   d. In each concurrent execution statement, an entry is made in the control flow graph for each execution path and then represented in the activity flow graph by different execution paths.

D. *Extraction of all required information by traversing the activity flow graph:* All necessary data, such as nodes, edges, conditional statements, and so on, are extracted. An activity transition graph can be thought of as a node for each action in an activity flow graph. Multiple control flow sequences are found using the depth-first traversal strategy to traverse the activity flow graph. We seek conditional predicates at each transition during traversal. In this phase, double-check the process to confirm that all required fields have been filled in, such as activity information, input, output, and conditions. Enumerate all feasible paths from the start node to the final node in the activity flow graph and each path visited to generate test cases, to generate test cases that satisfy the activity path criteria. To produce all activity paths, we suggest a pseudo-code of test cases.

E. *Generating the test cases from the activity flow graph:* Using the DFS approach, different flow sequences are detected by traversing the activity flow graph (AFG). The activity path coverage requirement is used to generate the test scripts. To do this, we gather all activity paths from the start node to the end node. To generate test cases that match the activity path criteria, we first enumerate all conceivable paths from the start node to the final node in the activity diagram, Figure 2.

We propose an algorithm, *GenTestCase*, described in Algorithm 1, to generate all the activity paths using the DFS approach described above. The *GenTestCase* algorithm (shown in Algorithm 1) generates all of the test cases. In our method, we explore the activity graph using a depth first search. This algorithm meets the activity path coverage criterion. All of the AFG's basic paths are enumerated by the algorithm. To produce test cases, each path is visited. Let us consider $I(a_1, a_2, \ldots, a_n)$ to be the set of input values for the visited path in the activity flow diagram and $O(d_1, d_2, \ldots, d_m)$ to be the resultant values of the execution of the input values in the visited path.

3.4. Case Study

With the use of a case study, we describe how our approach works in this part. First, we present an overview of the problem as well as the design model's activity diagram. Then, using the activity diagram as a guide, we describe the process of creating test cases.

In this section, we present the telehealth care case study, which illustrates the test generation process. A general step involved in the telehealth care app is described below:

***Step** 1: Start*
***Step** 2: Determine if the patient has diabetes*
***Step** 3: Does the patient have a blood glucose test?*
***Step** 4: If No—Ask the patient to get tested and go to **Step** 6*

***Step** 5: If yes—Go to **Step** 6*
***Step** 6: Check the glucose level*
***Step** 7: If glucose level > 120—declare patient has diabetes and go to **Step** 9*
***Step** 8: If glucose level ≤ 120—declare patient does not have diabetes*
***Step** 9: Determine the patient diabetes types.*
***Step** 10: Is RPG (RPG is random plasma glucose) ≥ 11.1?*
***Step** 11: Is respondent insulin-resistant?*
***Step** 12: If yes, Diabetes symptoms progress slowly?*
***Step** 13: Print "Type 2 diabetes"*
***Step** 14: If no, diabetes symptoms progress fast?*
***Step** 15: Print "Type 1 diabetes"*
***Step** 16: Advice proper treatment to respondent*
***Step** 17: End*

Algorithm 1. GenTestCase

Input: Activity flow diagram
Output: Activity paths
1: Enumerate and store all paths from the start node to the end node in
$$P = P[1], P[2], \ldots, P[n]$$
2: X = f
3: **for** each ($P[i]$), $1 \leq i \leq n$ **do**
4: $N(Node) = StartNode$
5: $E_N = EndNode$
6: $preC_A =$ Pre-Condition of the activity
7: $postC_A =$ Post-Condition of the activity
8: $p_i = f$
9: **while** $N \neq E_N$ **do**
10: **if** Condition = NULL **then**
11: $p_i = preC_A, I(a_1, a_2, \ldots, a_m), O(d_1, d_2, \ldots, d_l), postC_A$
12: **else if** Condition ≠ NULL
13: $C_{value} = C_1, C_2, \ldots, C_k$
14: $p = preC_A, I(a_1, a_2, \ldots, a_m), O(d_1, d_2, \ldots, d_l), C_{value}, postC_A$
15: **end if**
16: $pi = pi \cup p$
17: **end while**
18: $p = \{preC_{EN}, I_{EN}, O_{EN}, postC_{EN}\}$ (the pre-condition, input values, resultant values and the post condition of the End Node)
19: $pi = pi \cup p$
20: $X = X \cup pi$
21: **end for**
22: Return X

3.5. Activity Diagram and CFAMT Table

We can construct the activity diagram for the sample case study described above using Figure 2, and based on this, we construct the Control Flow Activity Mapping Table (CFAMT), as shown in Table 2. The CFAMT (Control Flow Activity Table) is created by analyzing the activity diagram.

The corresponding activity flow graph (AFG) is created by analyzing the CFAMT, as shown in Figure 4. For each label in the CFAMT a node is created in AFG. The sequence of control flow in the CFAMT is maintained in the AFG. During the AFG construction, each activity in the activity diagram is represented by a node in the AFG. The timing ordering of the diagram is maintained in the system. The conditional message in the diagram is represented by a node followed by two outward edges. Whether the condition is true or false, one of the edges is covered.

Table 2. Control flow activity mapping table.

Name of the Node	Name of the Activities	Predicate Conditions
A	Upload the blood report	NULL
B	Check the blood report	NULL
C	Valid report	B1: Valid report
D	Invalid report	B2: Invalid report
E	Check glucose Level	NULL
F	Glucose level $\leq$ 120	E1: No diabetes
G	Glucose level >120	E2: Patient has diabetes
H	Check RPG level	NULL
I	RPG < 11.1	H1: Advice precautionary treatment
J	RPG $\geq$ 11.1	H2: Check diabetes type
K	Check insulin dependency	NULL
L	Insulin independent	K1: Type 1 diabetes
M	Insulin dependent	K2: Type 2 diabetes
N	Advice respondent to proper treatment	NULL

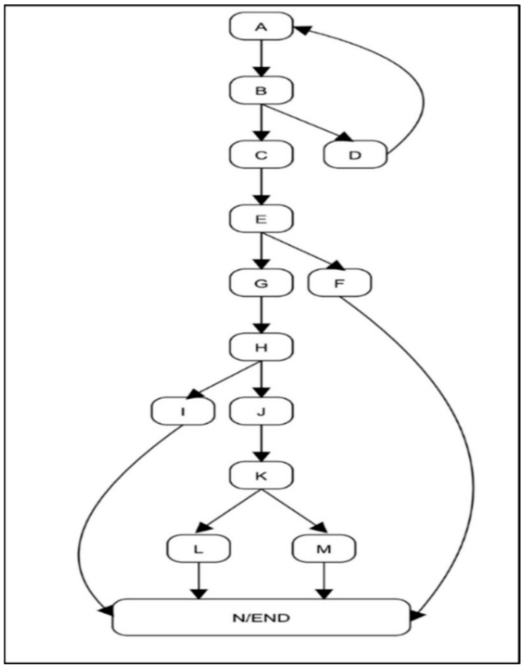

Figure 4. The activity flow graph of the test case.

3.6. Test Sequence Generation

From the AFG described in Figure 4, we can easily identify five control flow sequences by traversing the AFG using our DFS-based *GenTestCase* algorithm as described in Algorithm 1. During this traversal. we look for conditional predicates (Column 3 of Table 2) on each of the transitions, which are based on the conditions described as follows:

1. Pre-condition:
   a. The patient should have a valid blood report.
2. Post-condition:

a. Determines if the patient has diabetes.
b. Determines the type of diabetes.
c. Advise proper treatment.

3. Main scenario
   a. The patient uploads a valid blood test report.
   b. The patient enters the glucose level and RPG value
   c. The user receives information about the type of diabetes and proper treatment.

Applying the algorithm *GenTestCase* on the CFAMT described in Table 1, we obtain the following five activity paths:

1. $P_1 := A- > B- > D- > A$
2. $P_2 := A- > B- > C- > E- > F$
3. $P_3 := A- > B- > C- > E- > G- > H- > I$
4. $P_4 := A- > B- > C- > E- > G- > H- > J- > K- > L- > N$
5. $P_5 := A- > B- > C- > E- > G- > H- > J- > K- > M- > N$

3.7. Test Case Generation

Because there is no subordinate activity network in this example, a combination of activity pathways is not necessary. As a result, we have a total of five activity routes that we process in order to generate test cases. In our technique, a test case has four parts: a branch condition sequence, an activity sequence, object state changes, and an object created. The expected system behavior is made up of activity sequences, object state changes, and object creation. The sequence of branch conditions, on the other hand, is regarded as a source of test input. Each of the branch criteria in the sequence corresponds to some input data supplied in the textual description of the use case whose activity diagram is being examined. We obtain the necessary values of all variables as part of the test case generating process. For this, we use CFAMT constructed in Table 1.

4. Discussion and Conclusions

We provide a technique for IoT system acceptance testing in this research, and we use a realistic diabetic telehealth care test case scenario to determine diabetes type and therapy. We start by creating a test case directly from a UML activity diagram and then using the activity diagram to create an activity flow graph. All relevant data is extracted using the graph. The UML model, which is widely used in the medical field, is the foundation of our work. We created an activity transition graph (AFG) from an activity diagram in this work. The graph provided all of the necessary information. The predicates were then chosen by traversing the graph. Then, using activity path coverage criteria, test cases were created. In the AFG of the activity diagram, we first listed all feasible paths from the start node to the final node. After that, each path was explored in order to generate test cases. We looked for conditional predicates on each of the transitions during our visit in order to execute the corresponding flow and activity. We created test cases for each conditional predicate based on the activity path coverage criteria and the guard condition.

This paper proposed a method for building test cases that is both time and cost-effective.

Author Contributions: P.J. and M.S. (Madhusmita Sahu) contributed to the main idea of the research and wrote the original manuscript. P.J., M.S. (Mangal Sain) and S.K.B., reviewed the manuscript and provided valuable suggestions to further refine the manuscript. S.K.B. and M.S. (Mangal Sain) contributed significantly by supervising and managing to fund for research as well improving the technical and grammatical contents of the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Dongseo University, "Dongseo Cluster Project" Research Fund of 2022 (DSU-20220006).

Conflicts of Interest: The authors declare that there is no conflict of interest regarding the publication of this paper.

References

1. Leotta, M.; Ricca, F.; Clerissi, D.; Ancona, D.; Delzanno, G.; Ribaudo, M.; Franceschini, L. Towards an acceptance testing approach for Internet of Things systems. In Proceedings of the International Conference on Web Engineering, Rome, Italy, 5–8 June 2017; Springer: Cham, Switzerland, 2017; pp. 125–138.
2. Rosenkranz, P.; Wählisch, M.; Baccelli, E.; Ortmann, L. A distributed test system architecture for open-source IoT soft-ware. In Proceedings of the 1st Workshop on IoT Challenges in Mobile and Industrial Systems (IoT-Sys 2015), Florence, Italy, 18 May 2015; ACM: New York, NY, USA, 2015; pp. 43–48.
3. Sharma, S.; Chen, K.; Sheth, A. Toward practical privacy-preserving analytics for IoT and cloud-based healthcare systems. *IEEE Internet Comput.* **2018**, *22*, 42–51. [CrossRef]
4. Huang, H.; Gong, T.; Ye, N.; Wang, R.; Dou, Y. Private and secured medical data transmission and analysis for wireless sensing healthcare system. *IEEE Trans. Ind. Inform.* **2017**, *13*, 1227–1237. [CrossRef]
5. Lin, C.C.; Lin, P.Y.; Lu, P.K.; Hsieh, G.Y.; Lee, W.L.; Lee, R.G. A healthcare integration system for disease assessment and safety monitoring of dementia patients. *IEEE Trans. Inf. Technol. Biomed.* **2008**, *12*, 579–586. [PubMed]
6. Das, P.S.; Park, J.Y. A flexible touch sensor based on conductive elastomer for biopotential monitoring applications. *Biomed. Signal Process. Control* **2017**, *33*, 72–82. [CrossRef]
7. Nagarajan, S.M.; Deverajan, G.G.; Chatterjee, P.; Alnumay, W.; Ghosh, U. Effective task scheduling algorithm with deep learning for Internet of Health Things (IoHT) in sustainable smart cities. *Sustain. Cities Soc.* **2021**, *71*, 102945. [CrossRef]
8. Nagarajan, S.M.; Deverajan, G.G.; Kumaran, U.; Thirunavukkarasan, M.; Alshehri, M.D.; Alkhalaf, S. Secure data transmission in internet of medical things using res-256 algorithm. *IEEE Trans. Ind. Inform.* **2021**. [CrossRef]
9. Vieira, M.; Song, X.; Matos, G.; Storck, S.; Tanikella, R.; Hasling, B. Applying model-based testing to healthcare products: Preliminary experiences. In Proceedings of the 30th International Conference on Software Engineering, Leipzig, Germany, 10–18 May 2008; pp. 669–672.
10. Borycki, E.M.; Monkman, H.; Griffith, J.; Kushniruk, A.W. Mobile usability testing in healthcare: Methodological approaches. In *MEDINFO 2015: eHealth-Enabled Health*; IOS Press: Amsterdam, The Netherlands, 2015; pp. 338–342.
11. Holmes, S.; Moorhead, A.; Bond, R.; Zheng, H.; Coates, V.; McTear, M. Usability testing of a healthcare chatbot: Can we use conventional methods to assess conversational user interfaces? In Proceedings of the 31st European Conference on Cognitive Ergonomics, Belfast, UK, 10–13 September 2019; pp. 207–214.
12. Dey, J. A "new normal" Approach in Post-COVID19 Era: Online Healthcare Testing Strategy. *J. Math. Sci. Comput. Math.* **2020**, *2*, 145–157. [CrossRef]
13. Jabbar, R.; Krichen, M.; Fetais, N.; Barkaoui, K. Adopting formal verification and model-based testing tech-niques for validating a blockchain-based healthcare records sharing system. In Proceedings of the 22nd International Conference on Enterprise In-formation Systems, Prague, Czech Republic, 5–7 May 2020; SCITEPRESS-Science and Technology Publications: Setúbal, Portugal, 2020; pp. 261–268.
14. Klonoff, D.C. The current status of mHealth for diabetes: Will it be the next big thing? *J. Diabet. Sci. Technol.* **2013**, *7*, 749–758. [CrossRef] [PubMed]
15. Islam, S.R.; Kwak, D.; Kabir, M.H.; Hossain, M.; Kwak, K.S. The internet of things for health care: A comprehensive survey. *IEEE Access* **2015**, *3*, 678–708. [CrossRef]
16. Kim, H.; Ahmad, A.; Hwang, J.; Baqa, H.; Le Gall, F.; Ortega, M.A.R.; Song, J. IoT-TaaS: Towards a prospective IoT testing framework. *IEEE Access* **2018**, *6*, 15480–15493. [CrossRef]
17. Silva, L.C.; Perkusich, M.; Bublitz, F.M.; Almeida, H.O.; Perkusich, A. A model-based architecture for test-ing medical cyber-physical systems. In Proceedings of the 29th Annual ACM Symposium on Applied Computing, Gyeongju, Korea, 24–28 March 2014; pp. 25–30.
18. Leotta, M.; Clerissi, D.; Olianas, D.; Ricca, F.; Ancona, D.; Delzanno, G.; Franceschini, L.; Ribaudo, M. An acceptance testing approach for Internet of Things systems. *IET Softw.* **2018**, *12*, 430–436. [CrossRef]
19. Chen, T.Y.; Ho, J.W.; Liu, H.; Xie, X. An innovative approach for testing bioinformatics programs using metamorphic testing. *BMC Bioinf.* **2009**, *10*, 24. [CrossRef] [PubMed]
20. *STAM F2761-2009*; Medical Devices and Medical Systems—Essential Safety Requirements for Equip-ment Comprising the Patient-Centric Integrated Clinical Environment (ICE), Part 1: General Requirements and Conceptual Model. ASTM International: West Conshohocken, PA, USA, 2009.
21. Lee, I.; Sokolsky, O.; Chen, S.; Hatcliff, J.; Jee, E.; Kim, B.; King, A.; Mullen-Fortino, M.; Park, S.; Roederer, A.; et al. Challenges and research directions in medical cyber-physical systems. *Proc. IEEE* **2012**, *100*, 75–90.
22. Lee, I.; Sokolsky, O. Medical cyber physical systems. In Proceedings of the 2010 47th Design Automation Conference (DAC), Anaheim, CA, USA, 13–18 June 2010; ACM/IEEE: New York, NY, USA, 2010; pp. 743–748.
23. Pajic, M.; Mangharam, R.; Sokolsky, O.; Arney, D.; Goldman, J.; Lee, I. Model-driven safety analysis of closed-loop medical systems. *IEEE Trans. Ind. Inform.* **2012**, *10*, 3–16. [CrossRef] [PubMed]

24. Jiang, Z.; Pajic, M.; Mangharam, R. Model-based closed-loop testing of implantable pacemakers. In Proceedings of the 2011 IEEE/ACM Second International Conference on Cyber-Physical Systems, Chicago, IL, USA, 12–14 April 2011; IEEE: New York, NY, USA; pp. 131–140.
25. Arrieta, A.; Sagardui, G.; Etxeberria, L.; Zander, J. Automatic generation of test system instances for configurable cyber-physical systems. *Softw. Qual. J.* **2017**, *25*, 1041–1083. [CrossRef]
26. Hatcliff, J.; King, A.; Lee, I.; Macdonald, A.; Fernando, A.; Robkin, M.; Vasserman, E.; Weininger, S.; Goldman, J.M. Rationale and architecture principles for medical application platforms. In Proceedings of the 2012 IEEE/ACM Third International Conference on Cyber-Physical Systems (ICCPS'12), Beijing, China, 17–19 April 2012; IEEE Computer Society: Washington, DC, USA, 2012; pp. 3–12.
27. Miller, B.; Vahid, F.; Givargis, T. Digital mockups for the testing of a medical ventilator. In Proceedings of the 2nd ACM SIGHIT International Health Informatics Symposium (IHI'12), Miami, FL, USA, 28–30 January 2012; ACM: New York, NY, USA, 2012; pp. 859–862.
28. Lee, E.A. Cyber physical systems: Design challenges. In Proceedings of the 2008 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC'08), Orlando, FL, USA, 5–7 May 2008; IEEE Computer Society: Washington, DC, USA, 2008; pp. 363–369.
29. Lee, E.A.; Seshia, S.A. *Introduction to Embedded Systems, A Cyber-Physical Systems Approach*; MIT Press: Cambridg, MA, USA, 2011; ISBN 978-0-557-70857-4.
30. Dimitri, G.M.; Spasov, S.; Duggento, A.; Passamonti, L.; Toschi, N. Unsupervised stratification in neuroimaging through deep latent embeddings. In Proceedings of the 2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC), Montreal, QC, Canada, 20–24 July 2020; IEEE: New York, NY, USA, 2020; pp. 1568–1571.
31. Esteva, A.; Robicquet, A.; Ramsundar, B.; Kuleshov, V.; DePristo, M.; Chou, K.; Cui, C.; Corrado, G.; Thrun, S.; Dean, J. A guide to deep learning in healthcare. *Nat. Med.* **2019**, *25*, 24–29. [CrossRef] [PubMed]
32. Miotto, R.; Wang, F.; Wang, S.; Jiang, X.; Dudley, J.T. Deep learning for healthcare: Review, opportunities and challenges. *Brief. Bioinform.* **2018**, *19*, 1236–1246. [CrossRef] [PubMed]
33. Vaid, A.; Somani, S.; Russak, A.J.; De Freitas, J.K.; Chaudhry, F.F.; Paranjpe, I.; Johnson, K.W.; Lee, S.J.; Miotto, R.; Richter, F.; et al. Machine learning to predict mortality and critical events in a cohort of patients with COVID-19 in New York City: Model development and validation. *J. Med. Internet Res.* **2020**, *22*, e24018. [CrossRef] [PubMed]
34. Abomhara, M.; Lazrag, M.B. UML/OCL-based modeling of work-based access control policies for collaborative healthcare systems. In Proceedings of the 2016 IEEE 18th International Conference on e-Health Networking, Applications and Services (Healthcom), Munich, Germany, 14–16 September 2016; IEEE: New York, NY, USA, 2016; pp. 1–6.
35. Pişirgen, A.; Peker, S. A UML-Based Conceptual Model for Appointment Booking Systems. In Proceedings of the 2021 6th International Conference on Computer Science and Engineering (UBMK), Ankara, Turkey, 15–17 September 2021; IEEE: New York, NY, USA, 2021; pp. 812–817.
36. Veitaite, I.; Lopata, A. Knowledge-based UML dynamic models generation from enterprise model in hospital information management process example. *Intell. Syst. Sustain. Pers. -Cent. Healthc.* **2022**, *205*, 225–250. [CrossRef]
37. Utting, M.; Legeard, B. *Practical Model-Based Testing: A Tools Approach*; Elsevier: Amsterdam, The Netherlands, 2010.
38. Wang, L.; Yuan, J.; Yu, X.; Hu, J.; Li, X.; Zheng, G. Generating test cases from UML activity diagrambased on gray-box method. In Proceedings of the 11th Asia-Pacific Software Engineering Conference, Busan, Korea, 30 November–3 December 2004; IEEE: New York, NY, USA, 2004; pp. 284–291.