  
**System Flow Diagram**

**Literature survey**

Robert Gillman’s report at the World Privacy Forum [1] focuses on privacy issues and legal compliance of sharing data in the cloud. He mentions various legal issues such as the possibility of the cloud being in more than one legal location at the same time with different legal consequences and such legal uncertainty makes it difficult to assess the privacy protection available to users. We cater for this by allowing different legal policies to be configured into our authorization service and by allowing legal policies to stop data being transferred to jurisdictions which do not have proper privacy protection.

Siani Pearson [10] has pointed out the key privacy requirements for clouds and a set of guidelines for designing the cloud service with privacy protection. Siani Pearson et al [11] have proposed to use accountability to enhance privacy protection in the cloud. They have identified the key elements for provisioning accountability within the cloud. Siani Pearson et al [12,13] have proposed a privacy manager which would obfuscate personal data in the client site before sending it to the cloud service provider. This approach would minimize the amount of sensitive data held within the cloud. Only the client will be able to obfuscate or de-obfuscate data with their chosen key. The problem with this approach is that the applications that are able to use the obfuscated data are limited and this will limit the services available to the user. Also the computation overhead of obfuscating and de-obfuscating the data is large and so it imposes constraints on the computing resources available to the user.

Dan et al [14] have proposed a data protection model by which a potential cloud user can find and choose a cloud service provider based on a user based ranking of the service providers. The user can then integrate their privacy policy with that of the service provider and any sub contractors and this combined policy can be coupled with the data, rather like our sticky policies. The work is still at a very preliminary stage and more design and implementation is needed.

Wassim Itani et al [15] have presented a security infrastructure for cloud providers to adopt. Privacy of the data in the cloud is ensured by the use of a secure cryptographic co-processor which provides a

trusted and isolated execution environment in the computing cloud. But the price of the co-processor may not make the solution feasible. A systematic review requires a thorough, objective, and reproducible data search (Cocharane, 2012). The aim of the preliminary search was to identify existing systematic reviews and assess the volume of potentially relevant studies. Cocharane (2012) claimed that data search promotes the producing of reliable effects and relieves preventable bias. Meanwhile, data search distinguishes systematic reviews from traditional narrative reviews. Usually a designated number of databases are searched by using a standardized or customized search filter. After that researchers identify as many relevant studies as possible from the predefined data resources (Cocharane, 2012).

In this study, the data search strategy was developed in consultation with the librarians since the school electronic library (NELLI) is the most credible and available resource. Firstly, the research questions had been broken down by researchers into two main concepts, i.e. mobile Health and chronic disease. Then main concepts’ synonyms, abbreviations, and alternative spellings had been listed as mHealth and Telecare. Precisely, mobile Health (or mHealth) is a term that used for the practice of medicine and public health, supported by mobile devices (WHO, 2011). While, Telecare refers to remote alarm systems and monitoring devices that can help to support vulnerable people to continue living independently in their own homes (May, Finch, Cornford, Exley, Gately, Kirk & Jenkings, 2011). In 2012, definition of chronic disease from WHO was: ‘Diseases of long duration and generally slow progression.’

The initial search for primary studies was undertaken by using school electronic library NELLI. The retrieved databases include EBSCO (CINAHL), EBSCO (Academic Search Elite), Elsevier Science Direct, and Ovid (MEDLINE), using the following keywords and expanded MeSH: ‘mobile health’, ‘mHealth’, and ‘telecare’. These terms were used in conjunction with chronic disease. This research conducted through a systematic literature review to summarize previous mhealth studies in relation to chronic illness care to resolve the research questions. The methodology implementation included four steps: literature search from predefined databases; literature filter by criteria; data extract on research questions; and data analysis in legible diagrams. As the main interest of this study was to describe and discuss mHealth services for chronic patients, this thesis utilized qualitative approach in the synthesis of findings. Statistical figures and trend were not sought for, but the eligible literature core

opinions were intended as the raw data.

The systematic literature review has been widely undertaken in healthcare related researches because it is a practical research method in the field of medicine for searching targeted document (Hemingway & Brereton, 2009). As far as the further application of the mHealth, health-related benefits of mHealth have been covered by innumerable peer-reviewed journal articles, case studies, news articles, and reports (WHO, 2008). However, it is extremely difficult for healthcare professionals and patients to allocate the quality information among enormous publishing with limited time and expertise (Hemingway et al., 2009 & Cochrane Handbook for Systematic Reviews of Interventions, 2012). Systematic review applies predesigned methodologies to identify and access relevant literature, then summarizes conclusions from individual studies to answer specific research question (Hemingway et al., 2009). The applying of the systematic literature review method enables researchers to search information comprehensively. According to Kitchenham (2004), reasons of undertaking systematic review include summarizing the benefits and limitations of empirical evidence, identifying gaps in current research for further investigation, and providing a framework to position new research activities.

Hemingway et al. (2009) defined systematic literature review as a rigorous way of summarizing the research evidence to a particular research question based on a peer-reviewed protocol. Another systematic review’s definition from Cochrane (Cochrane Handbook for Systematic Reviews of Interventions, 2012) was ‘A systematic review attempts to identify, appraise and synthesize all the empirical evidence that meets pre-specified eligibility criteria to answer a given research question.’ Cocharane (2012) also enhanced that researchers should use explicit methods aimed at reducing bias. In this manner, reliable findings will be produced and consequently used to inform decision making.

Kitchenham (2004) indicated that the major advantage of a systematic literature review is to study the research phenomenon across a wide range of settings and empirical methods. The secondary advantage is the usage of meta-analytic technique (Kitchenham 2004). This technique will increase the likelihood of detecting real effects than individual studies. However, this increased power can also be a disadvantage, since more procedures and the workload will be performed aimed in reliable results (Kitchenham, 2004).

A systematic search was conducted by predefining a review protocol to obtain appropriate studies. Kitchenham (2004) stated that a pre-defined protocol is necessary to minimize possible researcher bias. For instance, there are possibilities that literature selection or analysis may be driven by researchers’

expectations if without a protocol (Kitchenham, 2004). The key elements of a protocol include research purpose, research question, research strategy (search resources or databases), study selection criteria, data extraction strategy, and synthesis of the extracted data (Kitchenham, 2004).

CONCLUSION & FUTURE WORK

In this project, we design a cloud-assisted privacy preserving mobile health monitoring system, called CAM, which can effectively protect the privacy of clients and the intellectual property of mobile health service providers. To protect the clients’ privacy, we apply the anonymous Boneh-Franklin identity based encryption (IBE) in medical diagnostic branching programs. To reduce the decryption complexity due to the use of IBE, we apply recently proposed decryption outsourcing with privacy protection to shift clients’ pairing computation to the cloud server. To protect m Heath service providers’ programs, we expand the branching program tree by using the random permutation and randomize the decision thresholds used at the decision branching nodes. Finally, to enable resource constrained small companies to participate in mobile health business.

To achieve the high privacy on the mobile health care system using the cloud assisted privacy preserving mobile health monitoring system where here implements the module. The final model is enhancing the high privacy in health care services.

Here future enhancement is increase the efficiency of information retrieval on cloud. So provides the token with specified and certain time slots for retrieved information on cloud.

World has been receiving reaping benefits from the technological developments that as surfaced in healthcare prospects. Recent advancements in healthcare is most likely beginning to inspire the confidence in the industry. Everyday people envision the future of healthcare, be it in technological advancement or innovative methods, there’s always the thirst to make healthcare more efficient, patient centric and cost effective.

**Motivation**

**CAM** is one aspect of mHealth that is pushing the limits of how to acquire, transport, store, process, and secure the raw and processed data to deliver meaningful results. CAM offers the ability of remote individuals to participate in the health care value matrix, which may not have been possible in the past. Participation does not imply just consumption of health care services. In many cases remote users are valuable contributors to gather data regarding disease and public health concerns such as outdoor pollution, drugs and violence.

The motivation behind the development of the CAM field arises from two factors. The first factor concerns the myriad constraints felt by healthcare systems of developing nations. These constraints include high [population growth](http://en.wikipedia.org/wiki/Population_growth), a high [burden of disease](http://en.wikipedia.org/wiki/Burden_of_disease) prevalence, low health care workforce, large numbers of rural inhabitants, and limited financial resources to support healthcare infrastructure and [health information systems](http://en.wikipedia.org/wiki/Health_information_systems). The second factor is the recent rapid rise in mobile phone penetration in developing countries to large segments of the healthcare workforce, as well as the population of a country as a whole. With greater access to mobile phones to all segments of a country, including rural areas, the potential of lowering information and transaction costs in order to deliver healthcare improves.

The combination of these two factors have motivated much discussion of how greater access to mobile phone technology can be leveraged to mitigate the numerous pressures faced by developing countries' healthcare systems.

**Limitations**

At present, mobile health services have been applied to patients with chronic diseases globally to reduce communication barriers and improve the quality of care (Bielli et al., 2004). In fact, the use of these services for patients with chronic diseases is internationally recognized as an innovation in healthcare. The limitation on language is known as an impediment in assessing all the articles .

**TESTING**

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the

Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

**7.1. Unit testing**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

**Test strategy and approach**

Field testing will be performed manually and functional tests will be written in detail.

**Test objectives**

* All field entries must work properly.
* Pages must be activated from the identified link.
* The entry screen, messages and responses must not be delayed.

**Features to be tested**

* Verify that the entries are of the correct format
* No duplicate entries should be allowed
* All links should take the user to the correct page.

**7.1.1 Unit Testing Using JUnit**

JUnit in version 4.x is a test framework which uses annotations to identify methods that specify a test. Typically these test methods are contained in a class which is only used for testing. It is typically called a Test class.

The JUnit test method can be created via following path Test cases.

*File* → *New* → *JUnit* → *JUnit*.

**7.1.2 Available JUnit Annotations**

|  |  |
| --- | --- |
| **Annotation** | **Description** |
| @Test  public void method() | The @Test annotation identifies a method as a test method. |
| @Test (expected = Exception.class) | Fails if the method does not throw the named exception. |
| @Test(timeout=100) | Fails if the method takes longer than 100 milliseconds. |
| @Before  public void method() | This method is executed before each test. It is used to prepare the test environment (e.g., read input data, initialize the class). |
| @After  public void method() | This method is executed after each test. It is used to clean up the test environment (e.g., delete temporary data, restore defaults). It can also save memory by cleaning up expensive memory structures. |
| @BeforeClass  public static void method() | This method is executed once, before the start of all tests. It is used to perform time intensive activities, for example, to connect to a database. Methods marked with this annotation need to be defined as static to work with JUnit. |
| @AfterClass  public static void method() | This method is executed once, after all tests have been finished. It is used to perform clean-up activities, for example, to disconnect from a database. Methods annotated with this annotation need to be defined as static to work with JUnit. |
| @Ignore | Ignores the test method. This is useful when the underlying code has been changed and the test case has not yet been adapted. Or if the execution time of this test is too long to be included. |

### 7.1.3 Assert statements

JUnit provides static methods in the Assert class to test for certain conditions. These assertion methods typically start with assert and allow you to specify the error message, the expected and the actual result. An assertion methodcompares the actual value returned by a test to the expected value, and throws an AssertionException if the comparison test fails.

The following table gives an overview of these methods. Parameters in brackets are optional.

|  |  |
| --- | --- |
| **Statement** | **Description** |
| fail(String) | Let the method fail. Might be used to check that a certain part of the code is not reached or to have a failing test before the test code is implemented. The String parameter is optional. |
| assertTrue([message], boolean condition) | Checks that the boolean condition is true. |
| assertFalse([message], boolean condition) | Checks that the boolean condition is false. |
| assertEquals([String message], expected, actual) | Tests that two values are the same. Note: for arrays the reference is checked not the content of the arrays. |
| assertEquals([String message], expected, actual, tolerance) | Test that float or double values match. The tolerance is the number of decimals which must be the same. |
| assertNull([message], object) | Checks that the object is null. |
| assertNotNull([message], object) | Checks that the object is not null. |
| assertSame([String], expected, actual) | Checks that both variables refer to the same object. |

### 7.1.4 Create a JUnit test suite

If you have several test classes, you can combine them into a test suite. Running a test suite will execute all test classes in that suite in the specified order.

The following example code shows a test suite which defines that two test classes should be executed. If you want to add another test class you can add it to @Suite.SuiteClasses statement.

**package** com.vogella.junit.first;

**import** org.junit.runner.RunWith;

**import** org.junit.runners.Suite;

**import** org.junit.runners.Suite.SuiteClasses;

*@RunWith(Suite.class)*

*@SuiteClasses({ MyClassTest.class, MySecondClassTest.class })*

**public** **class** AllTests { }

**7.2 Integration testing**

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or one step up software applications at the company level interact without error. The proposed strategy consists in two main phases (figure 1). The first one is related to the generation of the basic testing sequences corresponding to the collaboration between classes. Each generated sequence corresponds to a particular scenario of the collaboration diagram. Those sequences represent the main scenario and its various extensions. We use XML to describe collaboration diagrams and aspects. The proposed strategy consists, in a first step, to generate testing sequences corresponding to all scenarios without aspects integration. This will support the testing process of the collaboration. This step represents an adaptation and extension of some testing sequences generation techniques developed for object-oriented systems. The main goal of this step is to verify the collaboration for the realization of a given task and to eliminate faults that are not related to aspects. Aspects are integrated in a second step, in an iterative way.

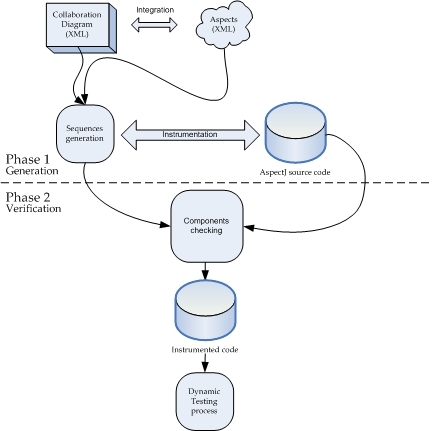


Figure 7.1. Methodology of the strategy

This will reduce the complexity of the testing process. We focus on the impact of aspects integration on the original scenarios of the collaboration. We formally identify the sequences (scenarios) that are affected by aspects integration. Aspects are introduced automatically, in an incremental way, to the original sequences and tested during the verification process.

The second main phase of the strategy consists in a dynamic analysis which verifies the execution of the implementation of each scenario of the collaboration (including aspects). This process is supported by a source code instrumentation of the program under test. The following algorithm represents the important steps of the strategy.

1. Generating control flow graphs corresponding to the methods implied in the collaboration.
2. Generating messages tree of the collaboration.
3. Generating basic sequences (based on the collaboration between objects).
4. Testing the collaboration between classes based on the various scenarios.
5. Integrating aspects: While there is non-integrated aspects
6. Integrating one aspect.
7. Identifying sequences that are affected by this integration (following the aspect's control).
8. Re-testing the affected sequences.
9. If there are no problems, return to step 5.
10. Testing entirely the collaboration including aspects.
11. End

To instrument the software under test we do use an aspect (generated automatically by our tool) for every sequence under test to capture dynamically all invoked methods in the collaboration (aspects and classes). The instrumented code contains the original source code and an aspect to capture the executed methods. This particular aspect verifies dynamically, among others, if the executed sequence is conform to the selected one (sequence of executed methods, conditions).

There are two type of Integration testing

## Top-Down Integration

## Bottom-Up Integration

Top-down testing versus bottom-up testing is really a question of what fits your situation. Bottom-up testing allows for easy use of existing reusable components in the testing approach since drivers can drive those components as the system architecture continues to be developed. Top-down testing must wait for architectural design activities to be finished before beginning. Because top-down testing tests high-level design elements, it can often find critical architectural problems earlier than bottom-up testing.

The following table summarizes the comparison between top-down and bottom-up integration testing.

|  |  |
| --- | --- |
| **Criteria** | **Comparison** |
| Architectural Validation | Top-down testing is better suited than bottom-up testing for early detection of system architecture errors and high-level design errors. Early detection reduces the cost of fixing the errors. |
| System Demonstration | A top-down approach to testing allows the organization to quickly gain confidence in a skeletal system that can then be used for demonstration purposes. A bottom-up approach uses drivers at the highest system levels which would likely be more cumbersome to demonstrate. |
| Test Implementation | Top-down testing will generally place more of a burden on the development team since meaningful stub behavior will be required for the system to be tested. Stubs can become quite complex in order to provide the necessary behavioral characteristics. Reusable components, on the other hand, provide stable behavior and therefore developers do not need to be quite as creative when creating the drivers that drive those low-level components. |
| Test Observation | Top-down and bottom-up testing are about equal on this criteria. High-level components aren’t necessarily meant to generate observable output and must be made to do so using an artificial environment. Likewise, low-level components may need an artificial environment in order to probe their internal behavior. |

The following diagram 7.2 shows graphically the difference between top-down integration testing and bottom-up integration testing.

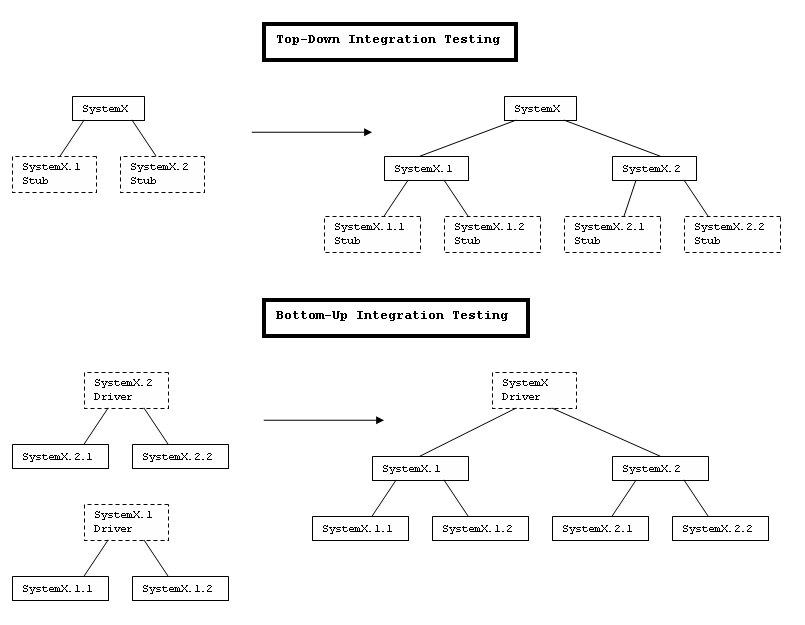


Figure 7.2

**7.3 Functional test**

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

**Valid Input :** identified classes of valid input must be accepted.

**Invalid Input :** identified classes of invalid input must be rejected.

**Functions :** identified functions must be exercised.

**Output :** identified classes of application outputs must be exercised.

**Systems/Procedures:** interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

**7.4 System Testing**

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

Where the system as a whole is tested against the Design Specifications:

* System Testing will be the responsibility of <System Supplier>, and will be managed by the <System Supplier> Project Manager or a delegated <System Supplier> Test Manager;
* System and integration testing shall be performed to ensure that the system works as a whole;
* A System Test Plan will be prepared by the <System Supplier> Development Team;
* System Testing will be performed jointly by <System Supplier> development personnel and <Business Unit> personnel assigned to the project;
* System Testing will be performed in the test environment located on the development server using workstations and printers located at <Location>;
* System Testing will be undertaken against selected live data as well as against special purpose test data;
* System Testing will include volume testing (with a high number of transactions and records processed) and stress testing (with transactions processed at high frequency);
* Results of tests will be recorded and where system components do not perform as expected, a Test Problem Report will be raised and registered.

**7.5 Acceptance Testing**

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Where the system is tested against the Functional Specifications in simulated live operation:

* Acceptance Testing will be the responsibility of <Business Unit>,and will be managed by the <Business Unit> Test Manager;
* Acceptance Testing will be undertaken against an Acceptance Test Plan to be prepared by the <Business Unit>;
* Acceptance Testing will be undertaken on the production server using workstations and printers located at <Location>;
* Standard tests will exist for each program type; (on-line enquiry, on-line update, batch process, standard report, word processing report, etc.);
* A specific set of test cases will exist for each function;
* Tests will also be undertaken on importation of the different types of interface files provided by external organisations;
* Tests for data migration functions will be undertaken;
* Specific tests for security, concurrent data access and database integrity will be undertaken;
* Tests to ensure user group privileges are working will be done;
* The audit trail will be monitored to ensure it is recording all required data changes;
* Specific tests against performance criteria, detailed in <Contract Reference> of the Agreement between <Business Unit> and <System Supplier>, will be undertaken;
* User procedures, batch processes, periodic processes and controls will be tested;
* Backup and recovery procedures will be tested;
* User, technical and operations documentation will be verified;
* The impact of the new system on existing systems will be tested.

**7.6 Test Cases**

A test case in [software engineering](http://en.wikipedia.org/wiki/Software_engineering) is a set of conditions or variables under which a tester will determine whether an [application](http://en.wikipedia.org/wiki/Software_application) or software system is working correctly. The mechanism for determining whether a software program or system has passed or failed such a test is known as a [test oracle](http://en.wikipedia.org/wiki/Oracle_%28software_testing%29). In some settings, an oracle could be a [requirement](http://en.wikipedia.org/wiki/Requirement) or use case, while in others it could be a [heuristic](http://en.wikipedia.org/wiki/Heuristic). It may take many test cases to determine that a software program or system is considered sufficiently scrutinized to be released. Test cases are often referred to as [test scripts](http://en.wikipedia.org/wiki/Test_script), particularly when written. Written test cases are usually collected into [test suites](http://en.wikipedia.org/wiki/Test_suite).

|  |  |  |  |
| --- | --- | --- | --- |
| Test case | Description | Outcome | Status of execution  Pass/fail |
| 1. | New User need to register | Fill in the form to get registered | Pass |
| 2. | Provider registration | Fill in the form to get registered | Pass |
| 3. | STA registration | Fill in the form to get registered | Pass |
| 4. | Validation Check | If correct provider/user/STA is not authorized then ll check for the validation and give a msg box saying Enter correct name and password | Fail |
| 5. | Correct authentication for Provider | Goes to new page where provider can add medical details,view details etc | Pass |
| 6. | Correct authentication for  User | User can request for a query to the STA | Pass |
| 7. | Correct authentication for  STA | STA can either accept the request or discard it | Pass |
| 8. | STA token generation | Discard request | Fails |
| 9. | STA token generation | If accepted a token is generated and send to the user | Pass |
| 10. | User window | User uses the token generated to see the prescription given by provider. | Pass |