PYTCH

PYTCH Fitness Application Project
Software Architecture Document (SAD)

CONTENT OWNERS : Kieran Jaggernauth(620138494)
Patrick Witter(620139423)

DOCUMENT NUMBER:	**RELEASE/REVISION:**	**RELEASE/REVISION DATE:**
● 1	● 1	● 12/1/22

BACKGROUND

This template is based on the Software Engineering Institute's "View and Beyond" method for documenting software architectures, as described in Clements, et al., *Documenting Software Architecture: Views and Beyond* (Addison Wesley, 2002). The current version is available for free download from the SEI's architecture web site.

TIPS FOR USING THIS TEMPLATE

To create an instance of this document:

- Insert relevant information on cover sheet and in placeholders throughout.
- Insert relevant information in page header: Move to a page of the body of the report, select *View > Header and Footer* from the main menu, and then replace relevant information in the header box at the top of the page.

To update the contents and page numbers in the Table of Contents, List of Figures, and List of Tables:

- Position the cursor anywhere in the table to be updated.
- Click the *F9* function key.
- Answer "Update entire table".

To insert a figure or table caption:

- From the main menu, choose *Insert > Reference > Caption* and then either *Figure* or *Table* as needed.
- Click the OK button.
- Add a colon and a tab stop after the figure number in the caption itself.
- The caption should use the *Caption* style.
- Add a colon and a tab stop after the table/figure number in the caption itself.

TIPS FOR MAKING YOUR DOCUMENT MORE READABLE

- A gray box containing *CONTENTS OF THIS SECTION* is provided at the beginning of most sections and subsections. After determining what specific information will be included in your document, you can remove this gray box or leave it to serve as a quick-reference section overview for your readers. In the case that text has been provided in the template, inspect it for relevance and revised as necessary.
- Consider hyperlinking key words used in the document with their entries in the Glossary or other location in which they are defined. Choose *Insert > Hyperlink*.
- Don't leave blank sections in the document. Mark them "To be determined" (ideally with a promise of a date or release number by which the information will be provided) or "Not applicable."
- Consider packaging your SAD as a multi-volume set of documentation. It is often helpful to break your documentation into more than one volume so that the document does not become unwieldy. There are many ways that this can be accomplished. The structuring of the document must support the needs of the intended audience and must be determined in the context of the project. Each document that you produce should include the date of issue and status; draft, baseline, version number, name of issuing organization; change history; and a summary. A few decomposition options are:
  - *A 2-Volume approach:* Separate the documentation into two volumes; one that contains the views of the software architecture and one that contains everything else. A common variant of this approach has one volume per view, and one volume for everything else.
  - *A 3-Volume approach:* Document organizational policies, procedures, and the directory in one volume, system specific overview material in a second, and view documentation in a third.
  - *A 4-Volume approach:* Create one volume for each viewtype [Clements 2002] (module, component-and-connector, allocation) that contains the documentation for the relevant views. Include all of the other information in the fourth volume.
  - Software interfaces are often documented in a separate volume.

In *any* case, the information should be arranged so that readers begin with the volume containing the Documentation Roadmap (Section 1 in this template).

Table of Contents

List of Tables

Documentation Roadmap

The Documentation Roadmap should be the first place a new reader of the SAD begins. But for new and returning readers, it is intended to describe how the SAD is organized so that a reader with specific interests who does not wish to read the SAD cover-to-cover can find desired information quickly and directly.

Sub-sections of Section 1 include the following.

- Section 1.1 ("Document Management and Configuration Control Information") explains revision history. This tells you if you're looking at the correct version of the SAD.
- Section 1.2 ("Purpose and Scope of the SAD") explains the purpose and scope of the SAD, and indicates what information is and is not included. This tells you if the information you're seeking is likely to be in this document.
- Section 1.3 ("How the SAD Is Organized") explains the information that is found in each section of the SAD. This tells you what section(s) in this SAD are most likely to contain the information you seek.
- Section 1.4 ("Stakeholder Representation") explains the stakeholders for which the SAD has been particularly aimed. This tells you how you might use the SAD to do your job.
- Section 1.5 ("Viewpoint Definitions") explains the *viewpoints* (as defined by IEEE Standard 1471-2000) used in this SAD. For each viewpoint defined in Section 1.5, there is a corresponding view defined in Section 3 ("Views"). This tells you how the architectural information has been partitioned, and what views are most likely to contain the information you seek.
- Section 1.6 ("How a View is Documented") explains the standard organization used to document architectural views in this SAD. This tells you what section within a view you should read in order to find the information you seek.

1.1 Document Management and Configuration Control Information

CONTENTS OF THIS SECTION: This section identifies the version, release date, and other relevant management and configuration control information associated with the current version of the document. Optional items for this section include: change history and an overview of significant changes from version to version.

- Revision Number: 1
- Revision Release Date: December 7, 2022
- Purpose of Revision: To document the initial architecture of the PYTCH system
- Scope of Revision: Not applicable

1.2 Purpose and Scope of the SAD

CONTENTS OF THIS SECTION: This section explains the SAD's overall purpose and scope, the criteria for deciding which design decisions are architectural (and therefore documented in the SAD), and which design decisions are non-architectural (and therefore documented elsewhere).

This SAD specifies the software architecture for **the Personally Training for Caribbean Health Project (PYTCH).** All information regarding the software architecture may be found in this document, although much information is incorporated by reference to other documents.

What is software architecture? The software architecture for a system[1] is the structure or structures of that system, which comprise software elements, the externally-visible properties of those elements, and the relationships among them [Bass 2003]. "Externally visible" properties refers to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on. This definition provides the basic litmus test for what information is included in this SAD, and what information is relegated to downstream documentation.

Elements and relationships. The software architecture first and foremost embodies information about how the elements relate to each other. This means that architecture specifically omits certain information about elements that does not pertain to their interaction. Thus, a software architecture is an *abstraction* of a system that suppresses details of elements that do not affect how they use, are used by, relate to, or interact with other elements. Elements interact with each other by means of interfaces that partition details about an element into public and private parts. Software architecture is concerned with the public side of this division, and that will be documented in this SAD accordingly. On the other hand, private details of elements—details having to do solely with internal implementation—are not architectural and will not be documented in a SAD.

Multiple structures. The definition of software architecture makes it clear that systems can and do comprise more than one structure and that no one structure holds the irrefutable claim to being the architecture. The neurologist, the orthopedist, the hematologist, and the dermatologist all take a different perspective on the structure of a human body. Ophthalmologists, cardiologists, and podiatrists concentrate on subsystems. And the kinesiologist and psychiatrist are concerned with different aspects of the entire arrangement's behavior. Although these perspectives are pictured differently and have very different properties, all are inherently related; together they describe the architecture of the human body. So it is with software. Modern systems are more than complex enough to make it difficult to grasp them all at once. Instead, we restrict our attention at any one moment to one (or a small number) of the software system's structures. To communicate meaningfully about an architecture, we must make clear which structure or structures we are discussing at the moment—which *view* we are taking of the architecture. Thus, this SAD follows the principle that documenting a software architecture is a matter of documenting the relevant views and then documenting information that applies to more than one view.

For example, all non-trivial software systems are partitioned into implementation units; these units are given specific responsibilities, and are the basis of work assignments for programming teams. This kind of element will comprise programs and data that software in other implementation units can call or access, and programs and data that are private. In large projects, the elements will almost certainly be subdivided for assignment to sub-teams. This is one kind of structure often used to describe a system. It is a very static structure, in that it focuses on the way the system's functionality is divided up and assigned to implementation teams.

Other structures are much more focused on the way the elements interact with each other at runtime to carry out the system's function. Suppose the system is to be built as a set of parallel processes. The set of processes that will exist at runtime, the programs in the various implementation units described previously that are strung together sequentially to form each process, and the synchronization relations among the processes form another kind of structure often used to describe a system.

None of these structures alone is *the* architecture, although they all convey architectural information. The architecture consists of these structures as well as many others. This example shows that since

[1] Here, a system may refer to a system of systems.

architecture can comprise more than one kind of structure, there is more than one kind of element (e.g., implementation unit and processes), more than one kind of interaction among elements (e.g., subdivision and synchronization), and even more than one context (e.g., development time versus runtime). By intention, the definition does not specify what the architectural elements and relationships are. Is a software element an object? A process? A library? A database? A commercial product? It can be any of these things and more.

These structures will be represented in the views of the software architecture that are provided in Section 3.

Behavior. Although software architecture tends to focus on structural information, *behavior of each element is part of the software architecture* insofar as that behavior can be observed or discerned from the point of view of another element. This behavior is what allows elements to interact with each other, which is clearly part of the software architecture and will be documented in the SAD as such. Behavior is documented in the element catalog of each view.

1.3 How the SAD Is Organized

CONTENTS OF THIS SECTION: This section provides a narrative description of the major sections of the SAD and the overall contents of each. Readers seeking specific information can use this section to help them locate it more quickly.

This SAD is organized into the following sections:

- **Section 1 ("Documentation Roadmap") provides information about this document and its intended audience**. It provides the roadmap and document overview. Every reader who wishes to find information relevant to the software architecture described in this document should begin by reading Section 1, which describes how the document is organized, which stakeholder viewpoints are represented, how stakeholders are expected to use it, and where information may be found. Section 1 also provides information about the views that are used by this SAD to communicate the software architecture.
- **Section 2 ("Architecture Background") explains why the architecture is what it is.** It provides a system overview, establishing the context and goals for the development. It describes the background and rationale for the software architecture. It explains the constraints and influences that led to the current architecture, and it describes the major architectural approaches that have been utilized in the architecture. It includes information about evaluation or validation performed on the architecture to provide assurance it meets its goals.
- **Section 3 (Views") and Section 4 ("Relations Among Views") specify the software architecture**. Views specify elements of software and the relationships between them. A view corresponds to a viewpoint (see Section 1.5), and is a representation of one or more structures present in the software (see Section 1.2).
- **Sections 5 ("Referenced Materials") and 6 ("Directory") provide reference information for the reader.** Section 5 provides look-up information for documents that are cited elsewhere in this SAD. Section 6 is a *directory*, which is an index of architectural elements and relations telling where each one is defined and used in this SAD. The section also includes a glossary and acronym list.

1.4 Stakeholder Representation

This section provides a list of the stakeholder roles considered in the development of the architecture described by this SAD. For each, the section lists the concerns that the stakeholder has that can be addressed by the information in this SAD.

Each stakeholder of a software system—customer, user, project manager, coder, analyst, tester, and so on—is concerned with different characteristics of the system that are affected by its software architecture. For example, the user is concerned that the system is reliable and available when needed; the customer is concerned that the architecture can be implemented on schedule and to budget; the manager is worried (in addition to cost and schedule) that the architecture will allow teams to work largely independently, interacting in disciplined and controlled ways. The developer is worried about strategies to achieve all of those goals. The security analyst is concerned that the system will meet its information assurance requirements, and the performance analyst is similarly concerned with it satisfying real-time deadlines.

This information is represented as a matrix, where the rows list stakeholder roles, the columns list concerns, and a cell in the matrix contains an indication of how serious the concern is to a stakeholder in that role. This information is used to motivate the choice of viewpoints chosen in Section 1.5.

CONTENTS OF THIS SECTION: The list of stakeholders will be unique for each organization that is developing a SAD. ANSI/IEEE 1471-2000 requires that at least the following stakeholders be considered:

- Users
  - Trainers
  - Trainees
- Acquirers
  - PYTCH Organization
- Developers
- Maintainers.
- Project Manager

You may wish to consider the following additional stakeholders.

● Customer ● Application software developers ● Infrastructure software developers ● End users ● Application system engineers ● Application hardware engineers	● Project manager ● Communications engineers ● Chief Engineer/Chief Scientist ● Program management ● System and software integration and test engineers ● Safety engineers and certifiers	● External organizations ● Operational system managers ● Trainers ● Maintainers ● Auditors ● Security engineers and certifiers

1.5 Viewpoint Definitions

CONTENTS OF THIS SECTION: This section provides a short textual definition of a viewpoint and how the concept is used in this SAD. The section describes viewpoints that may be used in the SAD. The specific viewpoints will be tailored by the organization.

The SAD employs a stakeholder-focused, multiple view approach to architecture documentation, as required by ANSI/IEEE 1471-2000, the recommended best practice for documenting the architecture of software-intensive systems [IEEE 1471].

As described in Section 1.2, a software architecture comprises more than one software structure, each of which provides an engineering handle on different system qualities. A *view* is the specification of one or more of these structures, and documenting a software architecture, then, is a matter of documenting the relevant views and then documenting information that applies to more than one view [Clements 2002].

ANSI/IEEE 1471-2000 provides guidance for choosing the best set of views to document, by bringing stakeholder interests to bear. It prescribes defining a set of viewpoints to satisfy the stakeholder community. A viewpoint identifies the set of concerns to be addressed, and identifies the modeling techniques, evaluation techniques, consistency checking techniques, etc., used by any conforming view. A view, then, is a viewpoint applied to a system. It is a representation of a set of software elements, their properties, and the relationships among them that conform to a defining viewpoint. Together, the chosen set of views show the entire architecture and all of its relevant properties. A SAD contains the viewpoints, relevant views, and information that applies to more than one view to give a holistic description of the system.

The remainder of Section 1.5 defines the viewpoints used in this SAD. The following table summarizes the stakeholders in this project and the viewpoints that have been included to address their concerns.

Table 1: Stakeholders and Relevant Viewpoints

Stakeholder	**Viewpoint(s) that apply to that class of stakeholder's concerns**
Trainees	Deployment Diagram , Package Diagram
Trainers	Deployment Diagram , Package Diagram
PYTCH	Deployment Diagram , Package Diagram
Software Developer	Deployment Diagram , Package Diagram
Project Manager	Deployment Diagram , Package Diagram

1.5.1 Package Diagram Viewpoint Definition

There will be one of these subsections for each viewpoint defined. The subsections are as follows:

- Abstract: A brief overview of the viewpoint
- Stakeholders and their concerns addressed: This section describes the stakeholders and their concerns that this viewpoint is intended to address. Listed are questions that can be answered by consulting views that conform to this viewpoint. Optionally, the section includes significant questions that cannot be answered by consulting views conforming to this viewpoint.
- Elements, relations, properties, and constraints: This section defines the types of elements, the relations among them, the significant properties they exhibit, and the constraints they obey for views conforming to this viewpoint.
- Language(s) to model/represent conforming views: This section lists the language or languages that will be used to model or represent views conforming to this viewpoint, and cite a definition document for each.
- Applicable evaluation/analysis techniques and consistency/completeness criteria: This section describes rules for consistency and completeness that apply to views in this viewpoint, as well as any analysis of evaluation techniques that apply to the view that can be used to predict qualities of the system whose architecture is being specified.
- Viewpoint source: This section provides a citation for the source of this viewpoint definition, if any.

Following is an example of a viewpoint definition.

1.5.1 **Module decomposition viewpoint definition**

1.5.1.1 Abstract. Views conforming to the module decomposition viewpoint partition the system into a unique non-overlapping set of hierarchically decomposable implementation units (*modules*).

1.5.1.2 Stakeholders and Their Concerns Addressed. Stakeholders and their concerns addressed by this viewpoint include

- project managers, who must define work assignments, form teams, and formulate project plans and budgets and schedules;
- COTS specialists, who need to have software elements defined as units of functionality, so they can search the marketplace and perform trade studies to find suitable COTS candidates;
- testers and integrators who use the modules as their unit of work;
- configuration management specialists who are in charge of maintaining current and past versions of the elements;
- system build engineers who use the elements to produce a running version of the system;
- maintainers, who are tasked with modifying the software elements;
- implementers, who are required to implement the elements;
- software architects for those software elements sufficiently large or complex enough to warrant their own software architectures;
- the customer, who is concerned that projected changes to the system over its lifetime can be made economically by confining the effects of each change to a small number of elements.

1.5.1.3 Elements, Relations, Properties, and Constraints. Elements of the module decomposition viewpoint are modules, which are units of implementation that provide defined functionality. Modules are hierarchically decomposable; hence, the relation is “is-part-of.” Properties of elements include their names, the functionality assigned to them (including a statement of the quality attributes associated with that functionality), and their software-to-software interfaces. The module properties may include requirements allocation, supporting requirements traceability.

1.5.1.4 Language(s) to Model/Represent Conforming Views. Views conforming to the module decomposition viewpoint may be represented by (a) plain text using indentation or outline form [Clements 2002]; (b) UML, using subsystems or classes to represent elements and “is part of” or nesting to represent the decomposition relation.

1.5.1.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria.
Completeness/consistency criteria include (a) no element has more than one parent; (b) major functionality is provided for by exactly one element; (c) the union of all elements’ functionality covers the requirements for the system; (d) every piece of source code can be mapped to an element in the module decomposition view (if not, the view is not complete); (e) the selection of module aligns with current and proposed procurement decisions. Additional consistency/completeness criteria apply to the specifications of the elements’ interfaces. Applicable evaluation/analysis techniques include (a) scenario-based evaluation techniques such as ATAM [Clements 2001] to assure that projected changes are supported economically by the decomposition; (b) disciplined and detailed mapping to requirements to assure coverage and non-overlapping functionality; (c) cost-based techniques that determine the number and composition of modules for efficient procurement.

1.5.1.6 Viewpoint Source. [Clements 2002, Section 2.1] describes the module decomposition style, which corresponds in large measure to this viewpoint.

1.5.1.1 Abstract

Views conforming to the package diagram module decomposition viewpoint partition the system into a unique non-overlapping set of hierarchically decomposable implementation units (modules). These modules are arranged in to logically related groupings and also show dependencies among members in each grouping.

1.5.1.2 Stakeholders and Their Concerns Addressed

Stakeholders and their concerns addressed by this viewpoint include:

1. Project managers, who must define work assignments, form teams, and formulate project plans and budgets and schedules;
2. COTS specialists (Software Developers), who need to have software elements defined as units of functionality, so they can search the marketplace and perform trade studies to find suitable COTS candidates;
3. Testers and integrators (Software Developers) who use the modules as their unit of work; configuration management specialists who are in charge of maintaining current and past versions of the elements;
4. System build engineers (Software Developers) who use the elements to produce a running version of the system;
5. Maintainers (Software Developers), who are tasked with modifying the software elements;
6. Implementers (Software Developers), who are required to implement the elements; software architects for those software elements sufficiently large or complex enough to warrant their own software architectures;
7. PYTCH,Trainers and Trainees (Customers) who is concerned that projected changes to the system over its lifetime can be made economically by confining the effects of each change to a small number of elements.

1.5.1.3 Elements, Relations, Properties, and Constraints

Elements of the package diagram module decomposition viewpoint are modules, which are units of implementation that provide defined functionality. Related modules are placed into packages that are named to define their purpose within the system. Properties of elements include their names,and their software-to-software interfaces (which other modules they access and/or import).

1.5.1.4 Language(s) to Model/Represent Conforming Views

UML

1.5.1.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria

Completeness/consistency criteria include:
(a) no element is in more than one package;
(b) major functionality is provided for by exactly one element;
(c) the union of all elements' functionality covers the requirements for the system;
(d) every piece of source code can be mapped to an element in the package diagram module decomposition view (if not, the view is not complete);
(e) the selection of module aligns with current and proposed procurement decisions. Additional consistency/completeness criteria apply to the specifications of the elements' interfaces.
Applicable evaluation/analysis techniques include:
(a) scenario-based evaluation techniques such as ATAM [Clements 2001] to assure that projected changes are supported economically by the decomposition;
(b) disciplined and detailed mapping to requirements to assure coverage and non-overlapping functionality;
(c) cost-based techniques that determine the number and composition of modules for efficient procurement.

1.5.1.6 Viewpoint Source

[Clements 2002, Section 2.1] describes the module decomposition style, which corresponds in large measure to this viewpoint.

1.5.2 Deployment Diagram Viewpoint Definition

1.5.2.1 Abstract

A deployment diagram is used to model the deployment of software.What are the artefacts to be created and the various physical parts of the software, where (in terms of physical devices) will they be located and how they will communicate with each other.

1.5.2.2 Stakeholders and Their Concerns Addressed

Stakeholders and their concerns addressed by this viewpoint include:
1. Project managers, formulate project plans (hosting platforms may take time to configure) and budgets (platforms may incur additional costs) and schedules;
2. Testers and integrators (Software Developers) who are concerned with how these platforms will be integrated within the software system and how to conduct tests on these platforms to ensure proper functionalit. They are also concerned writing tests on dependents of these platforms to ensure that they run properly incase these platforms fail ;
3. System build engineers (Software Developers) who use the elements to produce a running version of the system;

Maintainers (Software Developers), who are tasked with modifying the software elements;
Implementers (Software Developers), who are required to implement the elements;
software architects for those software elements sufficiently large or complex enough to warrant their own software architectures;
PYTCH,Trainers and Trainees (Customers) what platforms the software will be deployed on and the operating system that is supported.

1.5.2.3 Elements, Relations, Properties, and Constraints

Elements of the package diagram module decomposition viewpoint are modules, which are units of implementation that provide defined functionality. Related modules are placed into packages that are named to define their purpose within the system. Properties of elements include their names,and their software-to-software interfaces (which other modules they access and/or import).

1.5.2.4 Language(s) to Model/Represent Conforming Views

UML

1.5.2.4 Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria

Completeness/consistency criteria include:
(a) no element is in more than one package;
(b) major functionality is provided for by exactly one element;
(c) the union of all elements' functionality covers the requirements for the system;
(d) every piece of source code can be mapped to an element in the package diagram module decomposition view (if not, the view is not complete);
(e) the selection of module aligns with current and proposed procurement decisions. Additional consistency/completeness criteria apply to the specifications of the elements' interfaces.
Applicable evaluation/analysis techniques include:
(a) scenario-based evaluation techniques such as ATAM [Clements 2001] to assure that projected changes are supported economically by the decomposition;
(b) disciplined and detailed mapping to requirements to assure coverage and non-overlapping functionality;
(c) cost-based techniques that determine the number and composition of modules for efficient procurement.

Viewpoint Source

[Clements 2002, Section 2.1] describes the module decomposition style, which corresponds in large measure to this viewpoint.

1.6 How a View is Documented

CONTENTS OF THIS SECTION: This section describes how the documentation for a view is structured and organized. If you change the *organization* of information in Section 3, then you should also change its description in here. Otherwise, this section is all boilerplate.

If you choose to document all information in a view in a single presentation, then you will not need view packets. In that case, the template is as follows:

- Section 3.i: Name of view
- Section 3.i.1: View description
- Section 3.i.2: Primary presentation. This section presents the elements and the relations among them that populate this view packet, using an appropriate language, languages, notation, or tool-based representation.
- Section 3.i.3: Element catalog. Whereas the primary presentation shows the important elements and relations of the view packet, this section provides additional information needed to complete the architectural picture. It consists of subsections for (respectively) elements, relations, interfaces, behavior, and constraints.
- Section 3.i.4: Context diagram. This section provides a context diagram showing the context of the part of the system represented by this view packet. It also designates the view packet's scope with a distinguished symbol, and shows interactions with external entities in the vocabulary of the view.
- Section 3.i.5: Variability mechanisms. This section describes any variabilities that are available in the portion of the system shown in the view packet, along with how and when those mechanisms may be exercised.
- Section 3.i.6: Architecture background. This section provides rationale for any significant design decisions whose scope is limited to this view packet.

Section 3 of this SAD contains one view for each viewpoint listed in Section 1.5. Each view is documented as a set of view packets. A view packet is the smallest bundle of architectural documentation that might be given to an individual stakeholder.

Each view is documented as follows, where the letter *i* stands for the number of the view: 1, 2, etc.:

- Section 3.i: Name of view.
- Section 3.i.1: View description. This section describes the purpose and contents of the view. It should refer to (and match) the viewpoint description in Section 1.5 to which this view conforms.
- Section 3.i.2: View packet overview. This section shows the set of view packets in this view, and provides rationale that explains why the chosen set is complete and non-duplicative. The set of view packets may be listed textually, or shown graphically in terms of how they partition the entire architecture being shown in the view.
- Section 3.i.3: Architecture background. Whereas the architecture background of Section 2 pertains to those constraints and decisions whose scope is the entire architecture, this section provides any architecture background (including significant driving requirements, design approaches, patterns, analysis results, and requirements coverage) that applies to this view.
- Section 3.i.4: Variability mechanisms. This section describes any architectural variability mechanisms (e.g., adaptation data, compile-time parameters, variable replication, and so forth) described by this view, including a description of how and when those mechanisms may be exercised and any constraints on their use.
- Section 3.i.5: View packets. This section presents all of the view packets given for this view. Each view packet is described using the following outline, where the letter *j* stands for the number of the view packet being described: 1, 2, etc.

− Section 3.i.5.j: View packet #j.

- Section 3.i.5.j.1: Primary presentation. This section presents the elements and the relations among them that populate this view packet, using an appropriate language, languages, notation, or tool-based representation.
- Section 3.i.5.j.2: Element catalog. Whereas the primary presentation shows the important elements and relations of the view packet, this section provides additional information needed to complete the architectural picture. It consists of the following subsections:
    - Section 3.i.5.j.2.1: Elements. This section describes each element shown in the primary presentation, details its responsibilities of each element, and specifies values of the elements' relevant *properties*, which are defined in the viewpoint to which this view conforms.
    - Section 3.i.5.j.2.2: Relations. This section describes any additional relations among elements shown in the primary presentation, or specializations or restrictions on the relations shown in the primary presentation.
    - Section 3.i.5.j.2.3: Interfaces. This section specifies the software interfaces to any elements shown in the primary presentation that must be visible to other elements.
    - Section 3.i.5.j.2.4: Behavior. This section specifies any significant behavior of elements or groups of interacting elements shown in the primary presentation.
    - Section 3.i.5.j.2.5: Constraints: This section lists any constraints on elements or relations not otherwise described.
- Section 3.i.5.j.3: Context diagram. This section provides a context diagram showing the context of the part of the system represented by this view packet. It also designates the view packet's scope with a distinguished symbol, and shows interactions with external entities in the vocabulary of the view.
- Section 3.i.5.j.4: Variability mechanisms. This section describes any variabilities that are available in the portion of the system shown in the view packet, along with how and when those mechanisms may be exercised.
- Section 3.i.5.j.5: Architecture background. This section provides rationale for any significant design decisions whose scope is limited to this view packet.
- Section 3.i.5.j.6: Relation to other view packets. This section provides references for related view packets, including the parent, children, and siblings of this view packet. Related view packets may be in the same view or in different views.

1.7 Relationship to Other SADs

CONTENTS OF THIS SECTION: This section describes the relationship between this SAD and other architecture documents, both system and software. For example, a large project may choose to have one SAD that defines the system-of-systems architecture, and other SADs to define the architecture of systems or subsystems. An embedded system may well have a *system* architecture document, in which case this section would explain how the information in here traces to information there.

If none, say "Not applicable."

Not Applicable

1.8 Process for Updating this SAD

CONTENTS OF THIS SECTION: This section describes the process a reader should follow to report discrepancies, errors, inconsistencies, or omissions from this SAD. The section also includes necessary contact information for submitting the report. If a form is required, either a copy of the blank form that may be photocopied is included, or a reference to an online version is provided. This section also describes how error reports are handled, and how and when a submitter will be notified of the issue's disposition.

Peer Review

Architecture Background

1.9 Problem Background

CONTENTS OF THIS SECTION: The sub-parts of Section 2.1 explain the constraints that provided the significant influence over the architecture.

Based on interviews and QAWs done with stakeholders of PYTCH, the most significant constraints that were derived stressed that the users wanted high availability, mobility, security, modifiability and scalability. This provided the most significant constraint of having the system's backend be an entirely cloud-based serverless service. The mobility need imposed the constraint of having the system be primarily designed with mobile devices in mind as this would be the easiest way to interface with the wearable devices they would like to receive exercise data.

1.9.1 System Overview

CONTENTS OF THIS SECTION: This section describes the general function and purpose for the system or subsystem whose architecture is described in this SAD.

The system will have the following functionality:

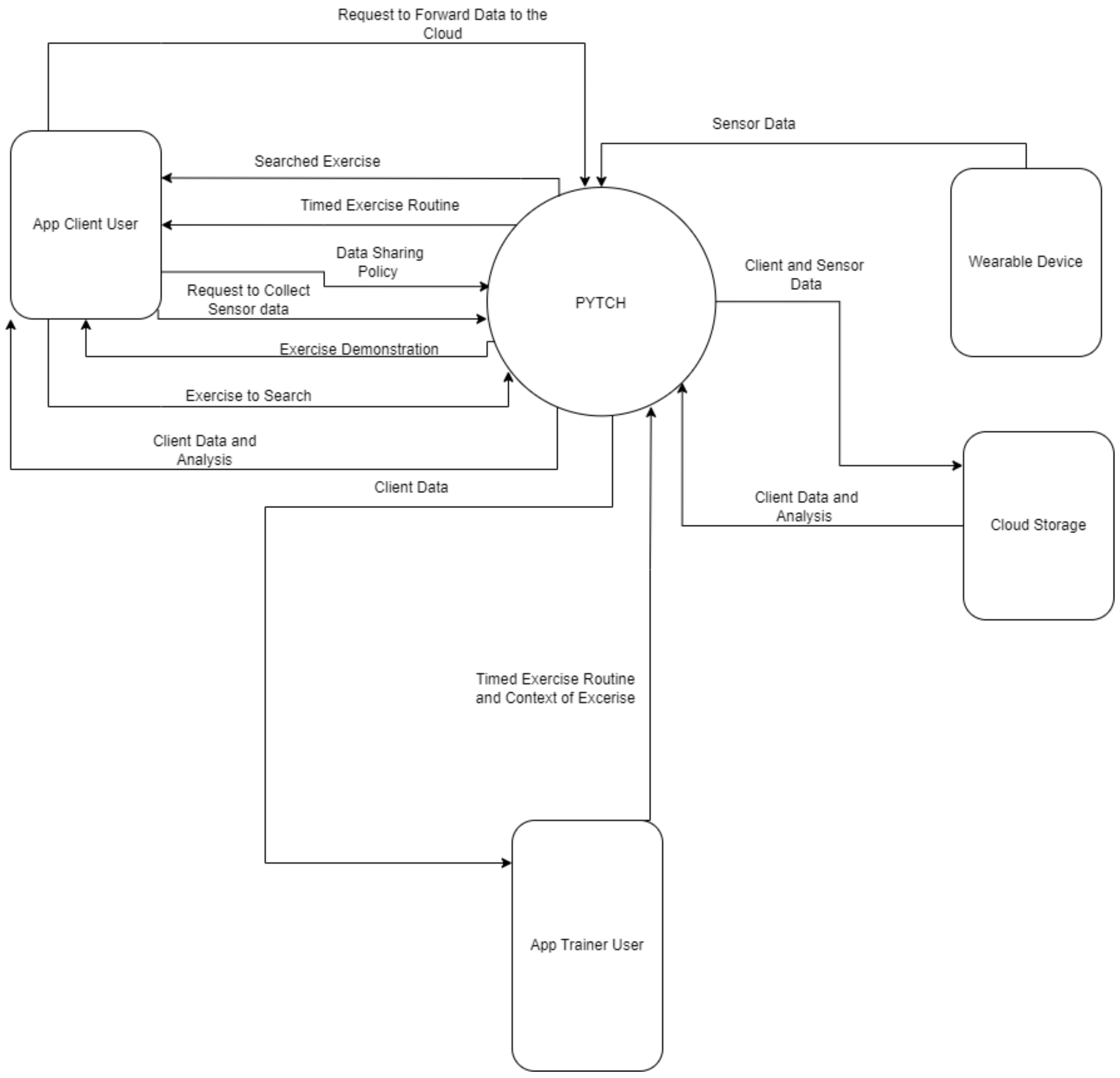
- Allow trainers to create exercises
- Allow clients to view exercises
- Collect sensor data from the wearable
- Send data to cloud storage
- Realtime updates to cloud storage
- Allow clients to download past data and cloud analytics from cloud storage
- Allow clients to set data sharing policy
- Allow trainer to retrieve client data

1.9.2 Goals and Context

CONTENTS OF THIS SECTION: This section describes the goals and major contextual factors for the software architecture. The section includes a description of the role software architecture plays in the life cycle, the relationship to system engineering results and artifacts, and any other relevant factors.

Context

- Figure showing the flow of data to and from the PYTCH app



Goals

- Implement a secure cloud-based solution with real-time database updates.
- Achieve a high application performance for all users.
- Ensure a high level of modifiability with no issues from additional components being added

1.9.3 Significant Driving Requirements

CONTENTS OF THIS SECTION: This section describes behavioral and quality attribute requirements (original or derived) that shaped the software architecture. Included are any scenarios that express driving behavioral and quality attribute goals, such as those crafted during a Quality Attribute Workshop (QAW) [Barbacci 2003] or software architecture evaluation using the Architecture Tradeoff Analysis Method[2SM] (ATAM[SM]) [Bass 2003].

Availability

- When the mobile app desynchronizes with the firebase firestore database as a result of being offline the data should be resynchronized upstream and downstream with no data loss.

Deployability

- A new component release to the google or apple play store should be deployed to the production environment with no loss of data from previous versions of the application.

Performance

- A user starts up the application under normal operation. The application presents the home screen within 2 seconds.
- The data collection component requests sensor data from a degraded wearable device. The data upload component under normal operation switches to degraded operation with a 5ms loss of upload speed

Scalability

- A deployment of a new component to the google or apple play store production environment should cause no issues within the rest of the application.

Energy Efficiency

- The battery drain of the mobile device should stay at a maximum of 1% every 10 minutes during the application runtime.

Portability

- The application's sensor data format should be standardized to a single format to be compatible with different wearable devices
- The application should be deployable to the google and apple play store with no difference in functionality.

Maintainability

- A defect arises during the normal operation of the application. The defect is isolated to the component it is related to and the other components continue to function normally

Usability

- The application shall provide notifications to the user regarding fitness updates during its runtime.

[2SM] Quality Attribute Workshop and QAW and Architecture Tradeoff Analysis Method and ATAM are service marks of Carnegie Mellon University.

- The user should be able to read the application's interface clearly while using the exercise feature.
- The user should be able to customise the application's colour scheme during the normal application runtime to their satisfaction.

Security

- Data should be transferred using a secure private key encryption scheme to and from firebase when data is synced from the application to the database by the user during normal app usage.
- The application should not connect to other wearable devices except for the owner's during normal runtime while reading the wearable device sensor data.
- Personal data such as a user's name, age and date of birth and fitness data such as a user's weight and location should not be shared according to the privacy policy that was set during the application runtime in a data upload upstream or downstream.

1.10 Solution Background

CONTENTS OF THIS SECTION: The sub-parts of Section 2.2 provide a description of why the architecture is the way that it is, and a convincing argument that the architecture is the right one to satisfy the behavioral and quality attribute goals levied upon it.

- The system was constrained to being mobile which led to the mobile device design and the system also needed to be able to facilitate realtime updates be secure which firebase satisfies

1.10.1 Architectural Approaches

CONTENTS OF THIS SECTION: This section provides a rationale for the major design decisions embodied by the software architecture. It describes any design approaches applied to the software architecture, including the use of architectural styles or design patterns, when the scope of those approaches transcends any single architectural view. The section also provides a rationale for the selection of those approaches. It also describes any significant alternatives that were seriously considered and why they were ultimately rejected. The section describes any relevant COTS issues, including any associated trade studies.

- To satisfy the recurring requirement of no data loss a redundant database was added as an extra layer of data redundancy
    - The risk here is that there is only one backup layer and it's safety depends solely on the PYTCH organization.
- The layered architecture was used. Modules were arranged into layers based on functionality and had a unidirectional dependencies inorder to increase the maintainability of the software. This is because layers can be tested and modified in isolation without affecting the other modules. This also allows us to achieve several of our quality attributes goals namely modifiability.

1.10.2 Analysis Results

CONTENTS OF THIS SECTION: This section describes the results of any quantitative or qualitative analyses that have been performed that provide evidence that the software architecture is fit for purpose. If an Architecture Tradeoff Analysis Method evaluation has been performed, it is included in the analysis sections of its final report. This section refers to the results of any other relevant trade studies, quantitative modeling, or other analysis results.

- Analyzing results from the quality attribute workshop showed that stakeholder needs for Scalability and Modifiability were the same. Therefore these requirements were merged under scalability

- Energy Efficiency and Performance needs clashed however focus groups held with the developers determined that both needs can be managed independently of each other. However, energy efficiency was determined to be of more importance to monitor as this was more out of the control of the developers.

- The deployment and modifiability requirement to lose no data upon adding a new component to the deployment environment were the same and therefore the requirements were merged under deployment

- There is a security risk with the bluetooth connection of the wearable device to the mobile device since a decent amount of security measures will need to be implemented to ensure a unwanted wearable devices does not connect.

1.10.3 Requirements Coverage

CONTENTS OF THIS SECTION: This section describes the requirements (original or derived) addressed by the software architecture, with a short statement about where in the architecture each requirement is addressed.

- Allow trainers to create exercises
  - This requirement is satisfied in the package diagram view
    - CreateTimedExercise Package

- Allow clients to view exercises
  - This requirement is satisfied in the package diagram view
    - View_Do_Exercise Package

- Collect sensor data from the wearable
  - This requirement is satisfied in the package diagram view
    - sensorData_service Package

- Send data to cloud storage
  - This requirement is satisfied in the package and deployment view
    - firebase_service Package
    - Firebase <<cloud service>>

- Realtime updates to cloud storage
  - This requirement is satisfied in the package and deployment view
    - firebase_service, update_viewmodel Package
    - Firebase <<cloud service>>

- Allow clients to download past data and cloud analytics from cloud storage
  - This requirement is satisfied in the package and deployment view
    - firebase_service, trainee_viewmodel Package
    - Firebase <<cloud service>>

- Allow clients to set data sharing policy
  - This requirement is satisfied in the package view
    - UpdateAndSettings Package
- Allow trainer to retrieve client data
  - This requirement is satisfied in the package view
    - trainer_viewmodel Package

1.10.4 Summary of Background Changes Reflected in Current Version

CONTENTS OF THIS SECTION: For versions of the SAD after the original release, this section summarizes the actions, decisions, decision drivers, analysis and trade study results that became decision drivers, requirements changes that became decision drivers, and how these decisions have caused the architecture to evolve or change.

Not Applicable

1.11 Product Line Reuse Considerations

CONTENTS OF THIS SECTION: When a software product line is being developed, this section details how the software covered by this SAD is planned or expected to be reused in order to support the product line vision. In particular, this section includes a complete list of the variations that are planned to be produced and supported. "Variation" refers to a variant of the software produced through the use of pre-planned variation mechanisms made available in the software architecture. It may refer to a variant of one of the modules identified in this SAD, or a collection of modules, or the entire system or subsystem covered by this SAD. For each variation, the section identifies the increment(s) of the software build in which (a) the variation will be available; and (b) the variation will be used. Finally, this section describes any additional potential that exists to reuse one or more of the modules or their identified variations, even if this reuse is not currently planned for any increment.

Not Applicable

Views

CONTENTS OF THIS SECTION: The sub-parts of Section 3 specify the views corresponding to the viewpoints listed in Section 1.5.

This section contains the views of the software architecture. A view is a representation of a whole system from the perspective of a related set of concerns [IEEE 1471]. Concretely, a view shows a particular type of software architectural elements that occur in a system, their properties, and the relations among them. A view conforms to a defining viewpoint.

Architectural views can be divided into three groups, depending on the broad nature of the elements they show. These are:

- Module views. Here, the elements are modules, which are units of implementation. Modules represent a code-based way of considering the system. Modules are assigned areas of functional responsibility, and are assigned to teams for implementation. There is less emphasis on how the resulting software manifests itself at runtime. Module structures allow us to answer questions such as: What is the primary functional responsibility assigned to each module? What other software elements is a module allowed to use? What other software does it actually use? What modules are related to other modules by generalization or specialization (i.e., inheritance) relationships?
- Component-and-connector views. Here, the elements are runtime components (which are principal units of computation) and connectors (which are the communication vehicles among components). Component and connector structures help answer questions such as: What are the major executing components and how do they interact? What are the major shared data stores? Which parts of the system are replicated? How does data progress through the system? What parts of the system can run in parallel? How can the system's structure change as it executes?
- Allocation views. These views show the relationship between the software elements and elements in one or more external environments in which the software is created and executed. Allocation structures answer questions such as: What processor does each software element execute on? In what files is each element stored during development, testing, and system building? What is the assignment of the software element to development teams?

These three kinds of structures correspond to the three broad kinds of decisions that architectural design involves:

- How is the system to be structured as a set of code units (modules)
- How is the system to be structured as a set of elements that have run-time behavior (components) and interactions (connectors) ?
- How is the system to relate to non-software structures in its environment (such as CPUs, file systems, networks, development teams, etc.)?

Often, a view shows information from more than one of these categories. However, unless chosen carefully, the information in such a hybrid view can be confusing and not well understood.

The views presented in this SAD are the following:

Name of view	Viewtype that defines this view	Is this a module view?	Is this a component-and-connector view?	Is this an allocation view?
Deployment	Allocation	No	No	Yes
Package	Module	Yes	No	No

1.12 Deployment View

CONTENTS OF THIS SECTION: For each view documented in this SAD, the sub-parts of Section 3.1 specify it using the outline given in Section 1.6. This part of the template assumes you are using view packets to divide up a view into management chunks. If not, then see the note in Section 1.6 as to what outline to use for each view.

1.12.1 View Description

- The deployment view shows how the physical elements of the architecture are accounted for.

1.12.2 View Packet Overview

This view has the following view packet for convenience of presentation:

- UML Deployment Diagram

1.12.3 Architecture Background

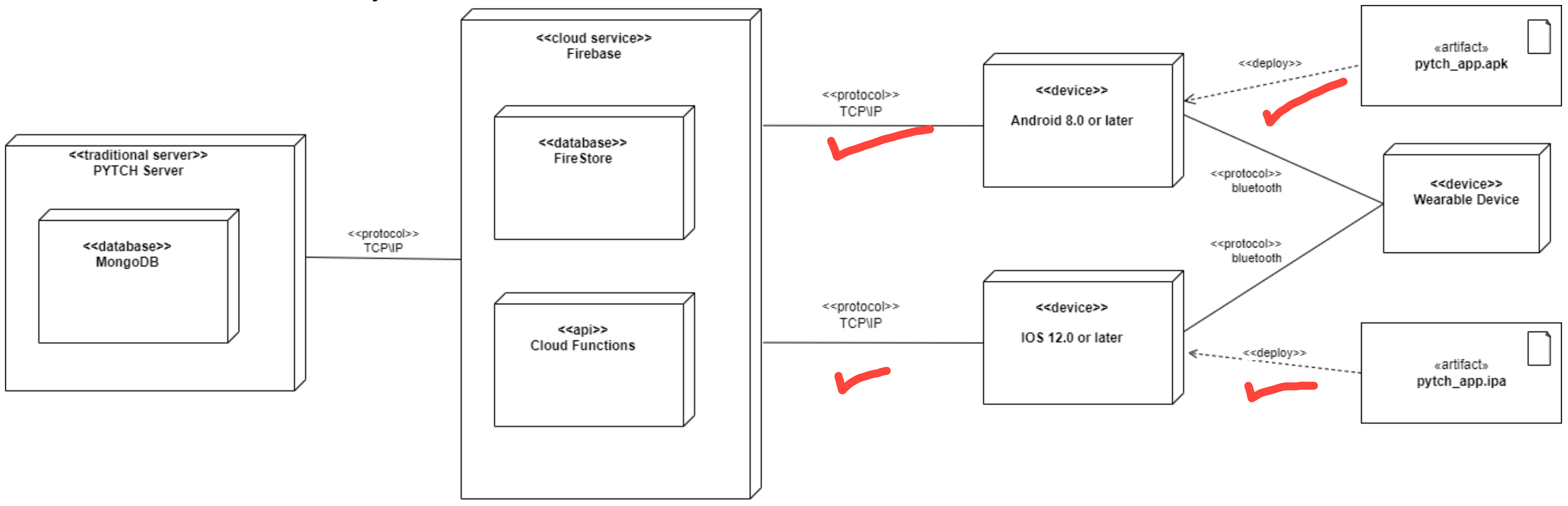
- PYTCH Stakeholders wanted a mobile application to interface with their wearable devices therefore the deployment architecture originates at the mobile phones as the client interaction point.
- They also wanted real-time updates, cloud storage and no data loss according to the extracted quality attributes.
  - The real-time updates and cloud storage is facilitated by the cloud service firebase which allows us to implement both these stakeholder critical features easily
  - To facilitate no data loss the firebase database will have to sync with a local PYTCH database. Further discussion will need to be done with the PYTCH team to evaluate the practicality and scale of this decision and but it is an important need to address this stakeholder concern. We will facilitate this by having a local MongoDB database on-site that communicates with firebase to sync the database at a set interval.

1.12.4 View Packets

CONTENTS OF THIS SECTION: For each view packet in the view, this section describes it using the outline given in Section 1.6.

1.12.4.1 View packet 1

1.12.4.1.1 Primary Presentation



1.12.4.1.2 Element Catalog

1.12.4.1.2.1 Elements

- Device - These elements are the ones the end user will primarily interact with
- Cloud Service - This contains the layout of the cloud services used
- Traditional Server - This is a physical on-site server.
- Database - This is a point within a system that employes a database filesystem
- API - This is an intermediary point that the system sends requests through that interact with the overall system.
- Artefact - This is a file or software element that exists on a device.

1.12.4.1.2.2 Relations

- Protocol - This specifies the communication protocol that will be used to communicate between devices
- Deploy - This shows where arefacts are deployed to a device

1.12.4.1.2.3 Constraints

- The end user must interact with the system through a mobile device

1.12.4.1.3 Variability Mechanisms

- The system uses both Andriod & IOS operating systems to interact with the system in the same way

1.12.4.1.4 Related View Packets

- 1.13.5.1 - Package View

1.13 Package View

CONTENTS OF THIS SECTION: For each view documented in this SAD, the sub-parts of Section 3.1 specify it using the outline given in Section 1.6. This part of the template assumes you are using view packets to divide up a view into management chunks. If not, then see the note in Section 1.6 as to what outline to use for each view.

1.13.1 View Description

- The package view shows how the physical elements of the architecture are accounted for.

1.13.2 View Packet Overview

This view has been divided into the following view packets for convenience of presentation:

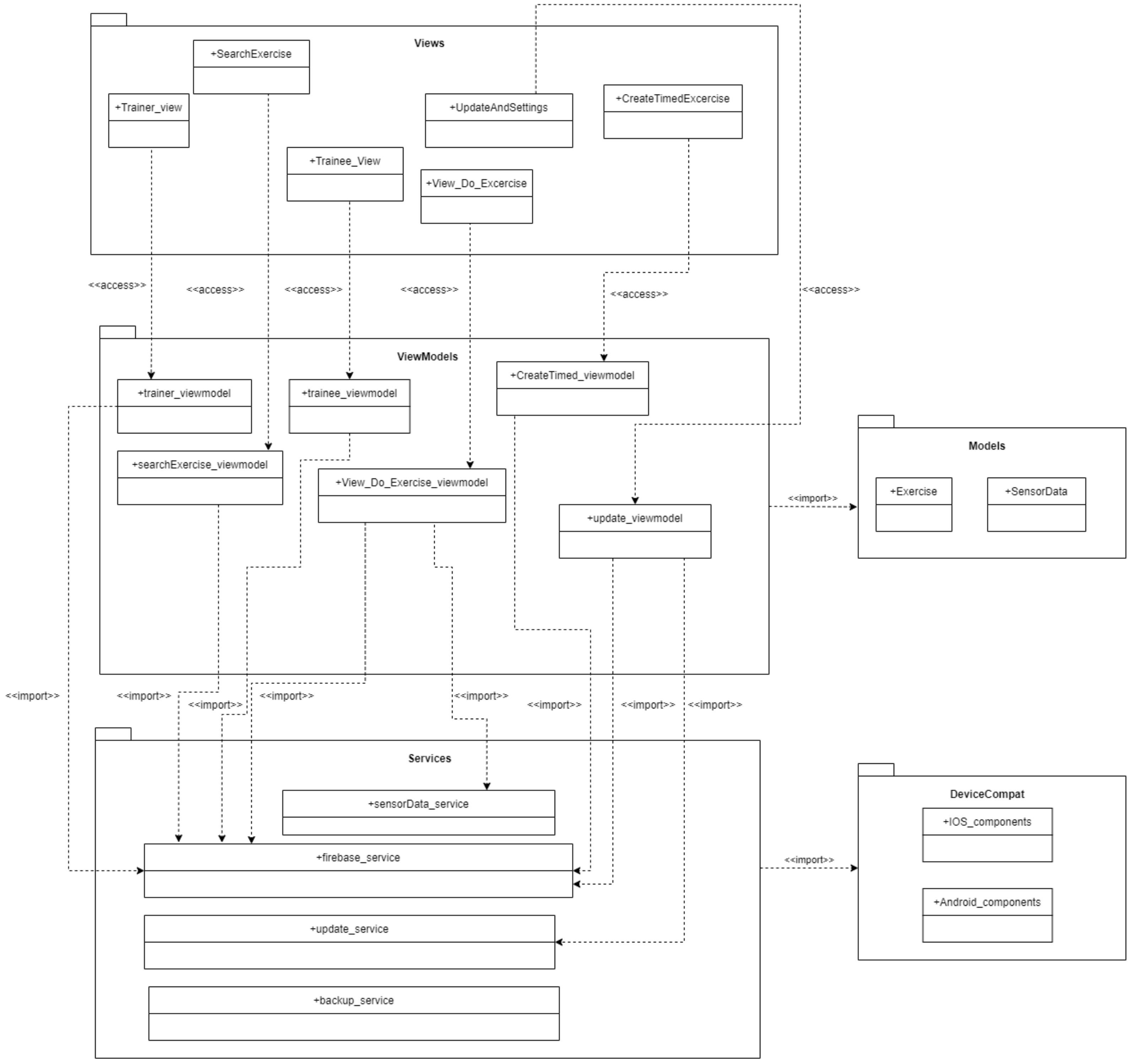
- UML Package Diagram

1.13.3 View Packets

CONTENTS OF THIS SECTION: For each view packet in the view, this section describes it using the outline given in Section 1.6.

1.13.3.1 View packet 2

1.13.3.1.1 Primary Presentation



1.13.3.1.2 Element Catalog

1.13.3.1.2.1 Elements

Views Package: This element contains the modules that are responsible for generating the user interface for the app and recording events from the user into the app (eg. a button press)
ViewModel Package: This element contains the modules that are responsible for responding to the events by the user (i.e the logic of our app) by performing some action or by using services
Services Package: This element contains the modules that are responsible for doing providing a set of cohesive services.
Modules Package: This element contains the modules that represent the data types within our app.
DeviceCompat Package: This element contains the modules that allow the app to be cross-platform

1.13.3.1.2.2 Relations

Access- This allows modules in X to only access modules in Y only. If Y imports or accesses any other module, X will not have access
Import - This allows modules in X to access modules in Y and any module that is imported or accessed by Y.

1.13.3.1.2.3 Constraints

Relations must be uni-directional

1.13.3.1.3 Architecture Background

The relationships in the package diagram reflet a layered architectural background. This approach is to separate the system into modules grouped by a single purpose and only allow unidirectional dependencies to achieve easy modifiabilty of modules.

1.13.3.1.4 Related View Packets

- 1.12.4.1 - Deployment View

Relations Among Views

Each of the views specified in Section 3 provides a different perspective and design handle on a system, and each is valid and useful in its own right. Although the views give different system perspectives, they are not independent. Elements of one view will be related to elements of other views, and we need to reason about these relations. For example, a module in a decomposition view may be manifested as one, part of one, or several components in one of the component-and-connector views, reflecting its runtime alter-ego. In general, mappings between views are many to many. Section 4 describes the relations that exist among the views given in Section 3. As required by ANSI/IEEE 1471-2000, it also describes any known inconsistencies among the views.

1.14 General Relations Among Views

CONTENTS OF THIS SECTION: This section describes the general relationship among the views chosen to represent the architecture. Also in this section, consistency among those views is discussed and any known inconsistencies are identified.

The deployment view shows the physical layout of the system and the package view shows the code structure of the system. Each package maps to a file or device that it specifically interacts with. All packages will be compiled into the relative mobile device application files.

1.15 View-to-View Relations

CONTENTS OF THIS SECTION: For each set of views related to each other, this section shows how the elements in one view are related to elements in another.

Deployment to Package View Relations

Deployment	Package	Relation
pytch_app.apk pytch_app.ipa	(ALL PACKAGES)	- Each package is compiled into their android and ios devices as the relative file format
pytch_app.ipa	IOS_Components	- The IOS_Components package contains the elements needed to make the ipa file on an IOSdevice
pytch_app.apk	Android_Components	- The Android_Components package contains the elements needed to make the apk file on an Android device
<<device>> Wearable Device	sensorData_service	- The sensorData_service package contains all the elements need to interpret sensor data from the wearable device.
<<cloud service>> Firebase	firebase_service	- The firebase_service package contains elements responsible for interacting with the firebase cloud service.
<<traditional server>> PYTCH Server	backup_service	- The backup_service package contains the elements responsible for interacting with the local PYTCH server to sync the database from firebase to the local copy

✓

Good

1.16 Referenced Materials

CONTENTS OF THIS SECTION: This section provides citations for each reference document. Provide enough information so that a reader of the SAD can be reasonably expected to locate the document.

Barbacci 2003	Barbacci, M.; Ellison, R.; Lattanze, A.; Stafford, J.; Weinstock, C.; & Wood, W. *Quality Attribute Workshops (QAWs)*, Third Edition (CMU/SEI-2003-TR-016). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. <http://www.sei.cmu.edu/publications/documents/03.reports/03tr016.html>.
Bass 2003	Bass, Clements, Kazman, *Software Architecture in Practice,* second edition, Addison Wesley Longman, 2003.
Clements 2001	Clements, Kazman, Klein, *Evaluating Software Architectures: Methods and Case Studies,* Addison Wesley Longman, 2001.
Clements 2002	Clements, Bachmann, Bass, Garlan, Ivers, Little, Nord, Stafford, *Documenting Software Architectures: Views and Beyond*, Addison Wesley Longman, 2002.
IEEE 1471	ANSI/IEEE-1471-2000, *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, 21 September 2000.

Directory

1.17 Index

CONTENTS OF THIS SECTION: This section provides an index of all element names, relation names, and property names. For each entry, the following are identified:

- the location in the SAD where it was defined
- each place it was used

Ideally, each entry will be a hyperlink so a reader can instantly navigate to the indicated location.

1.18 Glossary

CONTENTS OF THIS SECTION: This section provides a list of definitions of special terms and acronyms used in the SAD. If terms are used in the SAD that are also used in a parent SAD and the definition is different, this section explains why.

Term	Definition
software architecture	The structure or structures of that system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [Bass 2003]. "Externally visible" properties refer to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on.
view	A representation of a whole system from the perspective of a related set of concerns [IEEE 1471]. A representation of a particular type of software architectural elements that occur in a system, their properties, and the relations among them. A view conforms to a defining viewpoint.
view packet	The smallest package of architectural documentation that could usefully be given to a stakeholder. The documentation of a view is composed of one or more view packets.
viewpoint	A specification of the conventions for constructing and using a view; a pattern or template from which to develop individual views by establishing the purposes and

	audience for a view, and the techniques for its creation and analysis [IEEE 1471]. Identifies the set of concerns to be addressed, and identifies the modeling techniques, evaluation techniques, consistency checking techniques, etc., used by any conforming view.

1.19 Acronym List

API	Application Programming Interface; Application Program Interface; Application Programmer Interface
ATAM	Architecture Tradeoff Analysis Method
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
CORBA	Common object request broker architecture
COTS	Commercial-Off-The-Shelf
EPIC	Evolutionary Process for Integrating COTS-Based Systems
IEEE	Institute of Electrical and Electronics Engineers
KPA	Key Process Area
OO	Object Oriented
ORB	Object Request Broker
OS	Operating System
QAW	Quality Attribute Workshop
RUP	Rational Unified Process
SAD	Software Architecture Document
SDE	Software Development Environment
SEE	Software Engineering Environment
SEI	Software Engineering Institute Systems Engineering & Integration Software End Item
SEPG	Software Engineering Process Group
SLOC	Source Lines of Code
SW-CMM	Capability Maturity Model for Software
CMMI-SW	Capability Maturity Model Integrated - includes Software Engineering
UML	Unified Modeling Language

Sample Figures & Tables

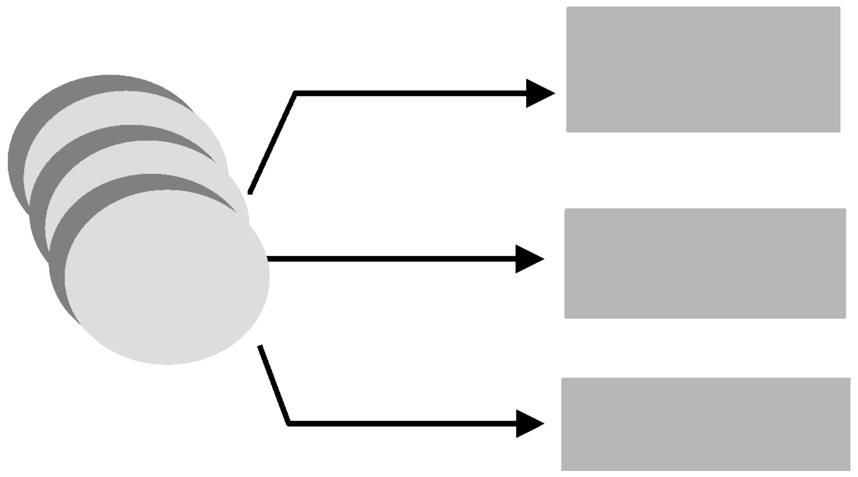


Figure 1: Sample Figure

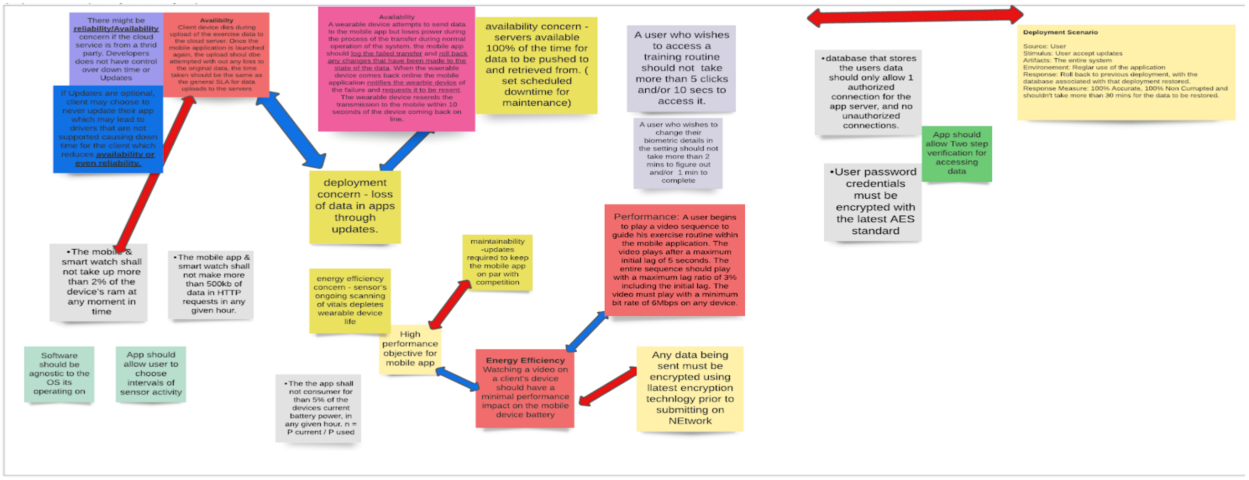
Table 2: Sample Table

Table Heading	**Table Heading**	**Table Heading**	**Table Heading**
Table Body	Table Body	Table Body	Table Body
Table Body	Table Body	Table Body	Table Body
Table Body	Table Body	Table Body	Table Body
Table Body	Table Body	Table Body	Table Body

Appendix A Appendices

CONTENTS OF THIS SECTION: Appendices may be used to provide information published separately for convenience in document maintenance (e.g., charts, classified data, API specification). As applicable, each appendix is referenced in the main body of the document where the data would normally have been provided. Appendices may be bound as separate documents for ease in handling. If your SAD has no appendices, delete this page.

A.1 QAW Appendix



There might be **reliability/Availability** concern if the cloud service is from a thrid party. Developers does not have control over down time or Updates

Availibilty
Client device dies during upload of the exercise data to the cloud server. Once the mobile application is launched again, the upload shoul dbe attempted with out any loss to the original data, the time taken should be the same as the general SLA for data uploads to the servers

If Updates are optional, client may choose to never update their app which may lead to drivers that are not supported causing down time for the client which reduces **availability or even reliability.**

Availability
A wearable device attempts to send data to the mobile app but loses power during the process of the transfer during normal operation of the system. the mobile app should log the failed transfer and roll back any changes that have been made to the state of the data. When the waerable device comes back online the mobile application notifies the wearble device of the failure and requests it to be resent. The wearable device resends the transmission to the mobile within 10 seconds of the device coming back on line.

availability concern - servers available 100% of the time for data to be pushed to and retrieved from. (set scheduled downtime for maintenance)

A user who wishes to access a training routine should not take more than 5 clicks and/or 10 secs to access it.

A user who wishes to change their biometric details in the setting should not take more than 2 mins to figure out and/or 1 min to complete

•database that stores the users data should only allow 1 authorized connection for the app server, and no unauthorized connections.

App should allow Two step verification for accessing data

•User password credentials must be encrypted with the latest AES standard

Deployment Scenario

Source: User
Stimulus: User accept updates
Artifacts: The entire system
Environement: Reglar use of the application
Response: Roll back to previous deployment, with the database associated with that deployment restored.
Response Measure: 100% Accurate, 100% Non Currupted and shouldn't take more than 30 mins for the data to be restored.

deployment concern - loss of data in apps through updates.

•The mobile & smart watch shall not take up more than 2% of the device's ram at any moment in time

•The mobile app & smart watch shall not make more than 500kb of data in HTTP requests in any given hour.

energy efficiency concern - sensor's ongoing scanning of vitals depletes wearable device life

maintainability -updates required to keep the mobile app on par with competition

High performance objective for mobile app

Performance: A user begins to play a video sequence to guide his exercise routine within the mobile application. The video plays after a maximum initial lag of 5 seconds. The entire sequence should play with a maximum lag ratio of 3% including the initial lag. The video must play with a minimum bit rate of 6Mbps on any device.

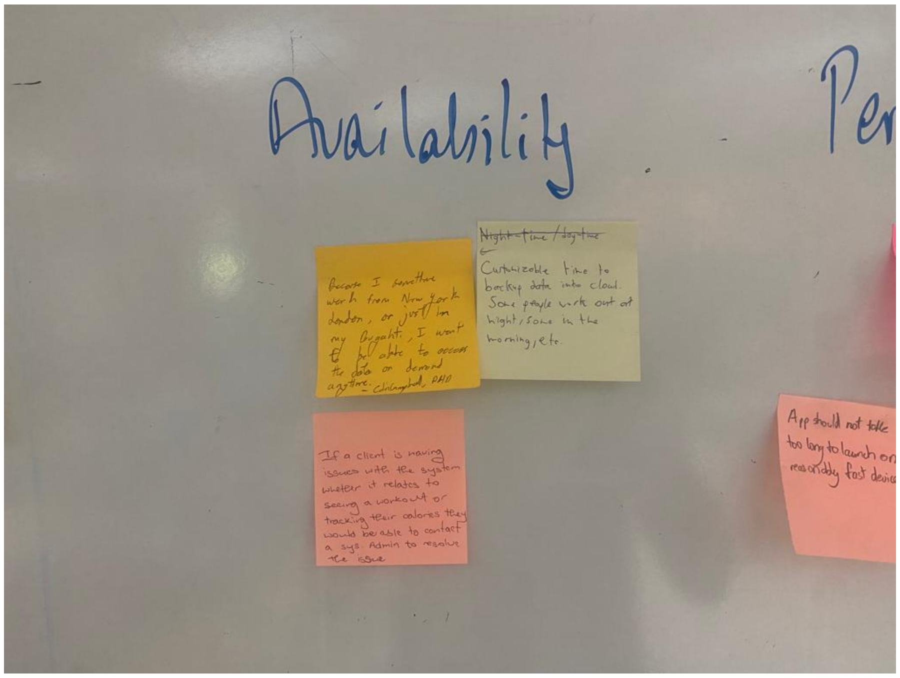
Software should be agnostic to the OS its operating on

App should allow user to choose intervals of sensor activity

•The the app shall not consumer for than 5% of the devices current battery power, in any given hour. n = P current / P used

Energy Efficiency
Watching a video on a client's device should have a minimal performance impact on the mobile device battery

Any data being sent must be encrypted using llatest encryption technlogy prior to submitting on NEtwork

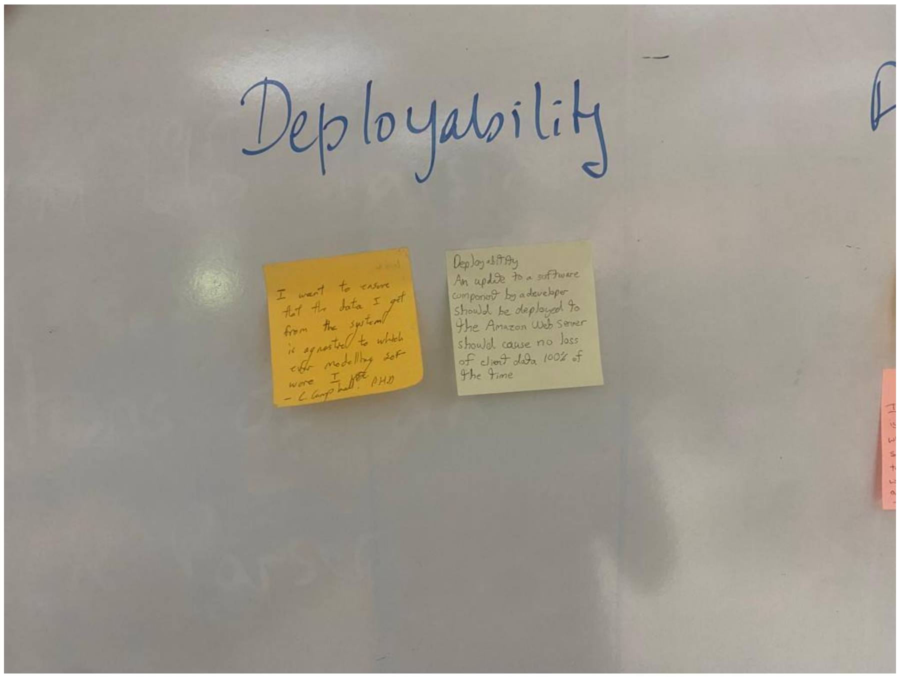


Because I sometime work from New York London, or just I'm my Bugatti, I want to be able to access the data on demand anytime. - [illegible], PHD

~~Night-time / day-time~~
Customizable time to backup data into cloud. Some people work out at night, some in the morning, etc.

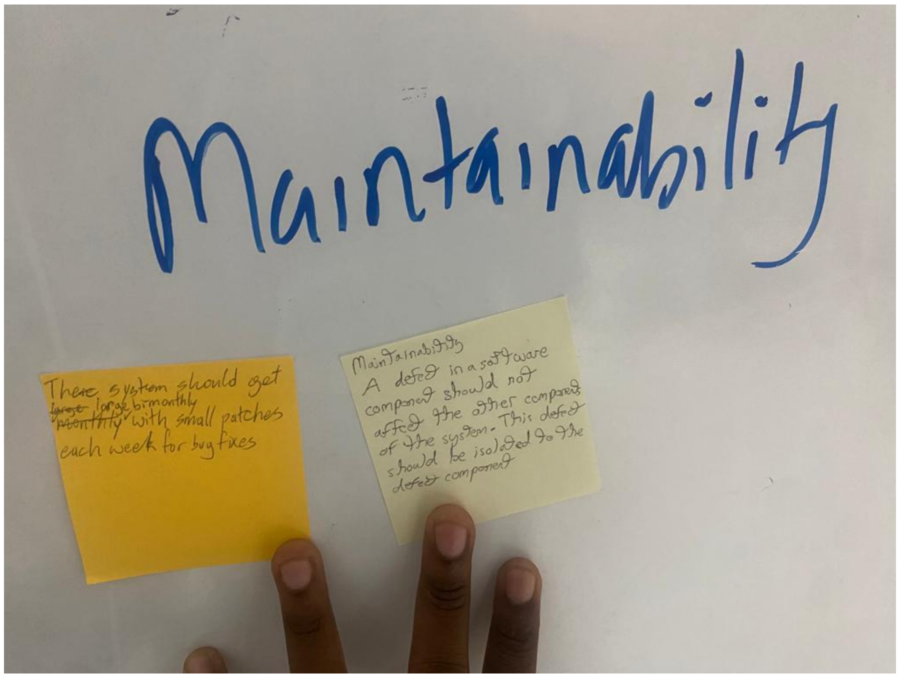
If a client is having issues with the system wheteer it relates to seeing a workout or tracking their calories they would be able to contact a sys Admin to resolve the issue

App should not take too long to launch on reasonably fast devic



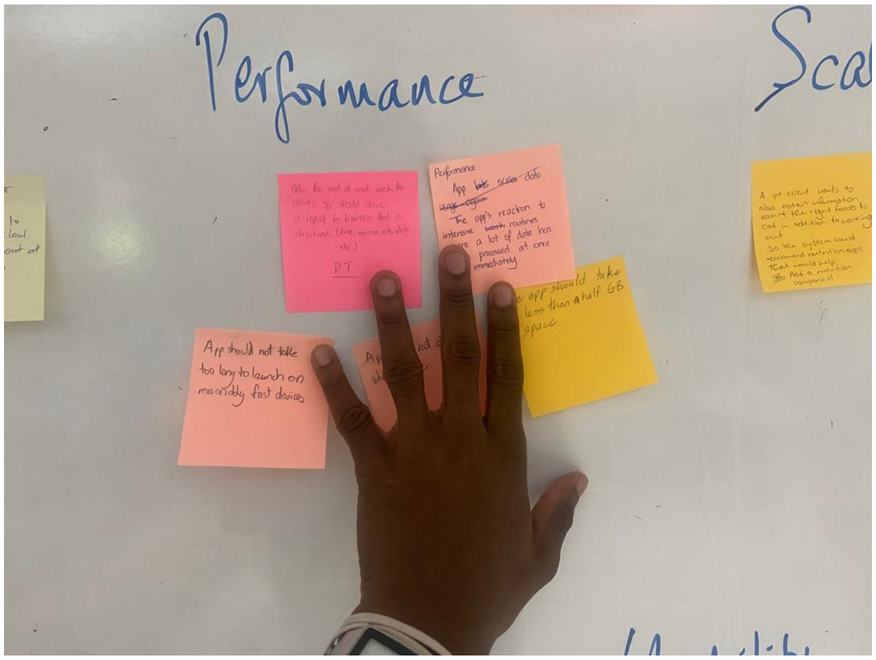
I want to ensure that the data I get from the system is agnostic to which ever modelling software I use - C. Campbell PHD

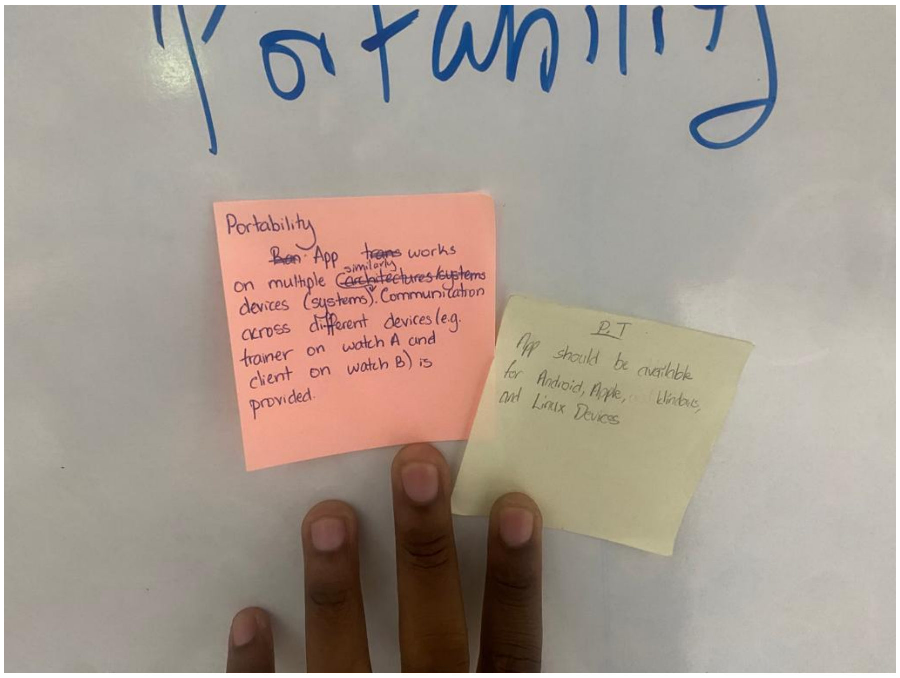
Deployability
An update to a software component by a developer should be deployed to the Amazon Web Server should cause no loss of client data 100% of the time

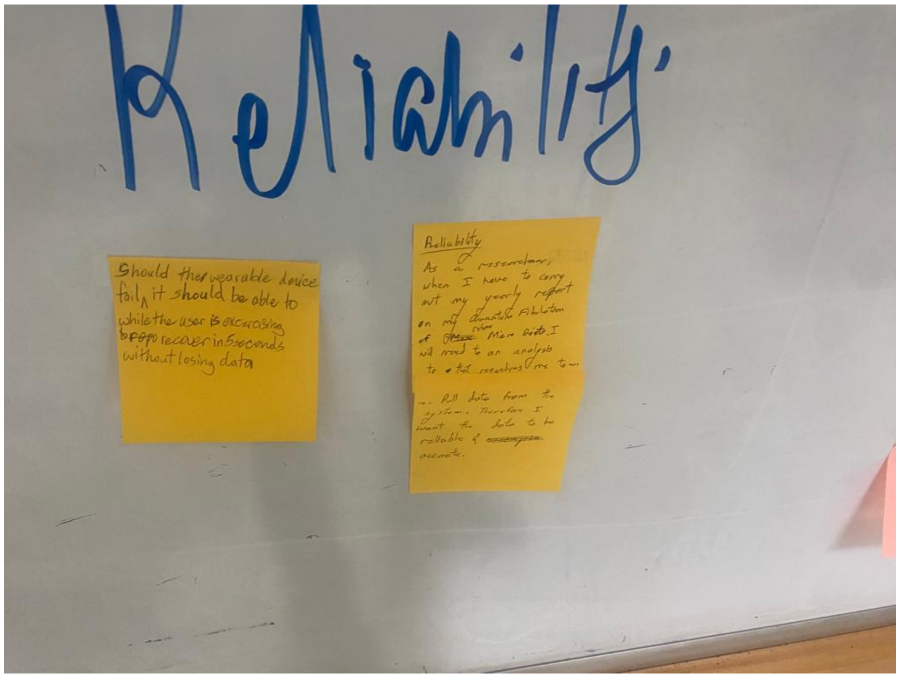


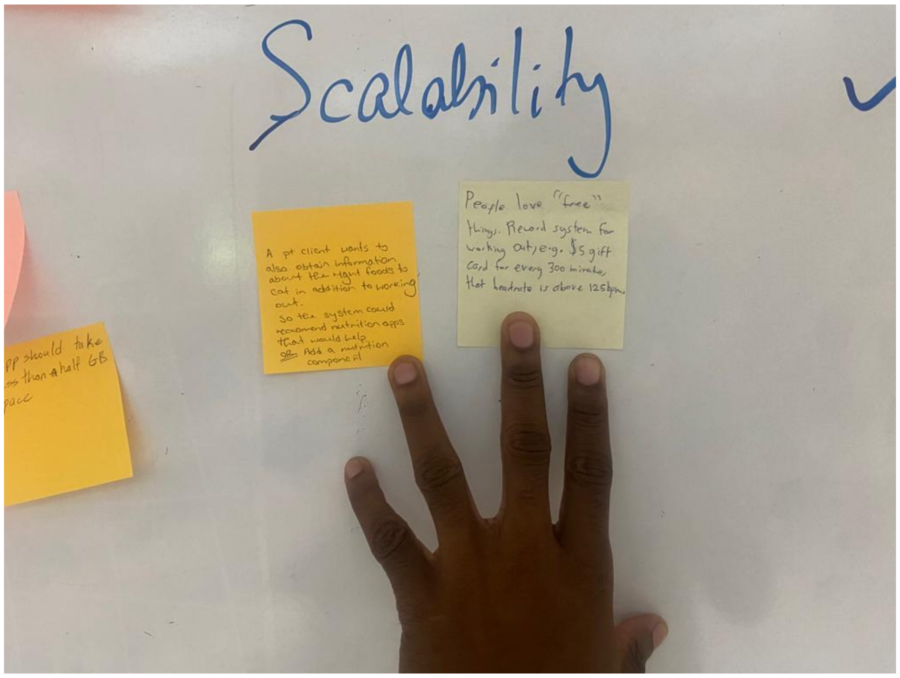
There system should get ~~large~~ large bimonthly ~~monthly~~ with small patches each week for bug fixes

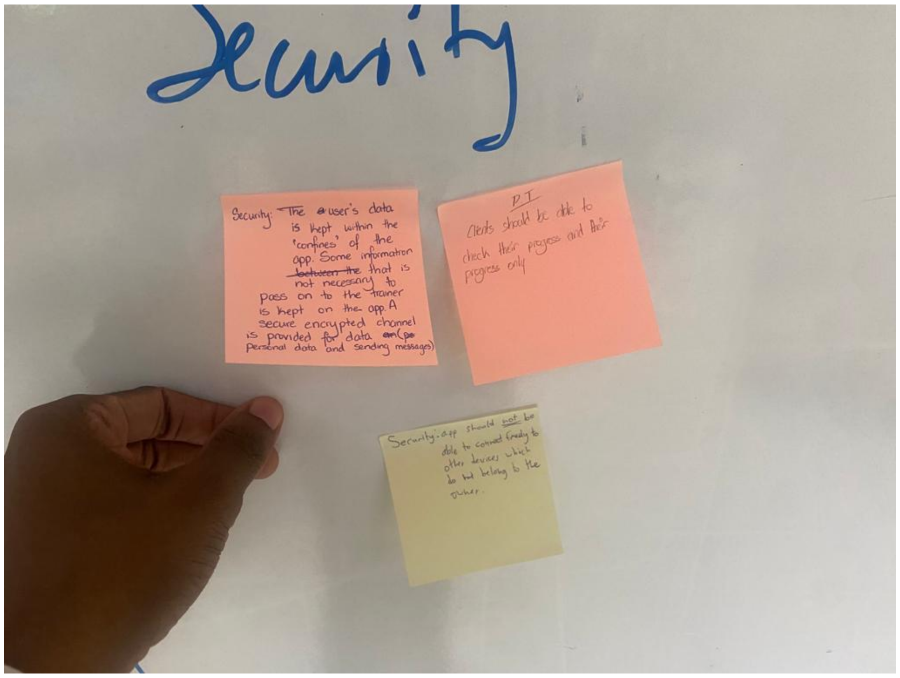
Maintainability
A defect in a software component should not affect the other components of the system. This defect should be isolated to the defect component


Performance
Sca
After the end of each week, the client's app should show a report to trainers that is structured. (time, exercise sets, white etc.)
P.T.
Performance
App ~~lets scales~~ data ~~usage again~~
The app's reaction to intensive ~~work~~ routines where a lot of data has to be processed at once immediately
A pt client wants to also obtain information about the right foods to eat in addition to working out.
So the system could recommend nutrition apps that would help ~~OR~~ Add a nutrition component
App should not take too long to launch on reasonably fast devices
The app should take less than a half GB space


Portability
Portability
~~Ban~~ App ~~trans~~ works similarly on multiple ~~(architectures/systems~~ devices (systems). Communication across different devices (e.g. trainer on watch A and client on watch B) is provided.
P.T
App should be available for Android, Apple, Windows, and Linux Devices


Reliability
Should the wearable device fail while the user is excercising, it should be able to recover in 5 seconds without losing data
Reliability
As a researcher, when I have to carry out my yearly report on my Quantum Fibrilation of other Micro Dots. I will need to an analysis to that requires me to …
… Pull data from the system. Therefore I want the data to be reliable & ~~[illegible]~~ accurate.

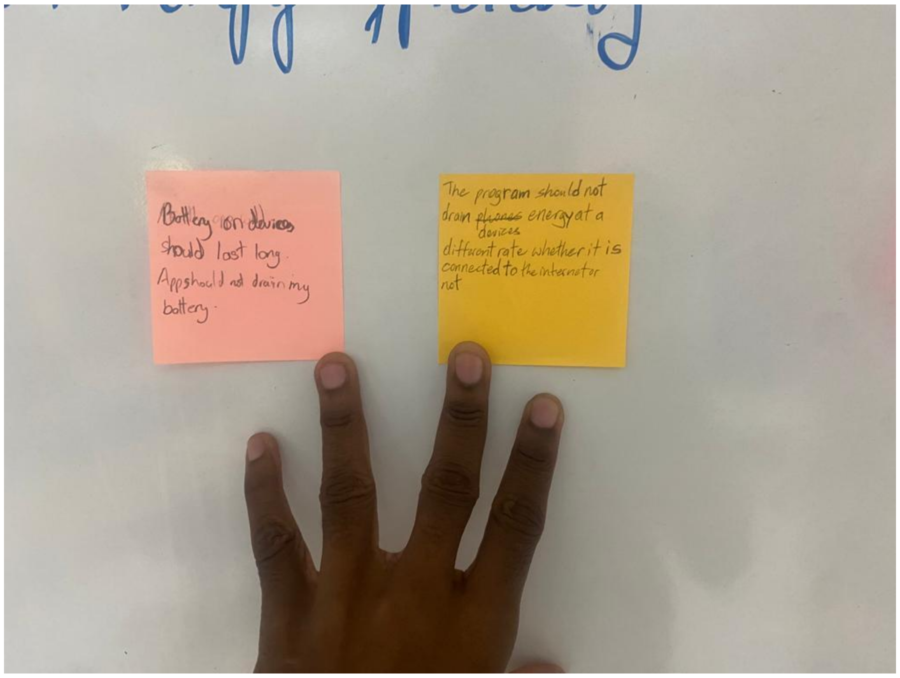

Scalability
pp should take ss than a half GB pace
A pt client wants to also obtain information about the right foods to eat in addition to working out.
So the system could recommend nutrition apps that would help
OR Add a nutrition component
People love "free" things. Reword system for working out, e.g. $5 gift card for every 300 minutes that heartrate is above 125bpm.



Security: The user's data is kept within the 'confines' of the app. Some information ~~between the~~ that is not necessary to pass on to the trainer is kept on the app. A secure encrypted channel is provided for data (personal data and sending messages)

P.T
Clients should be able to check their progress and their progress only

Security: app should not be able to connect freely to other devices which do not belong to the owner.

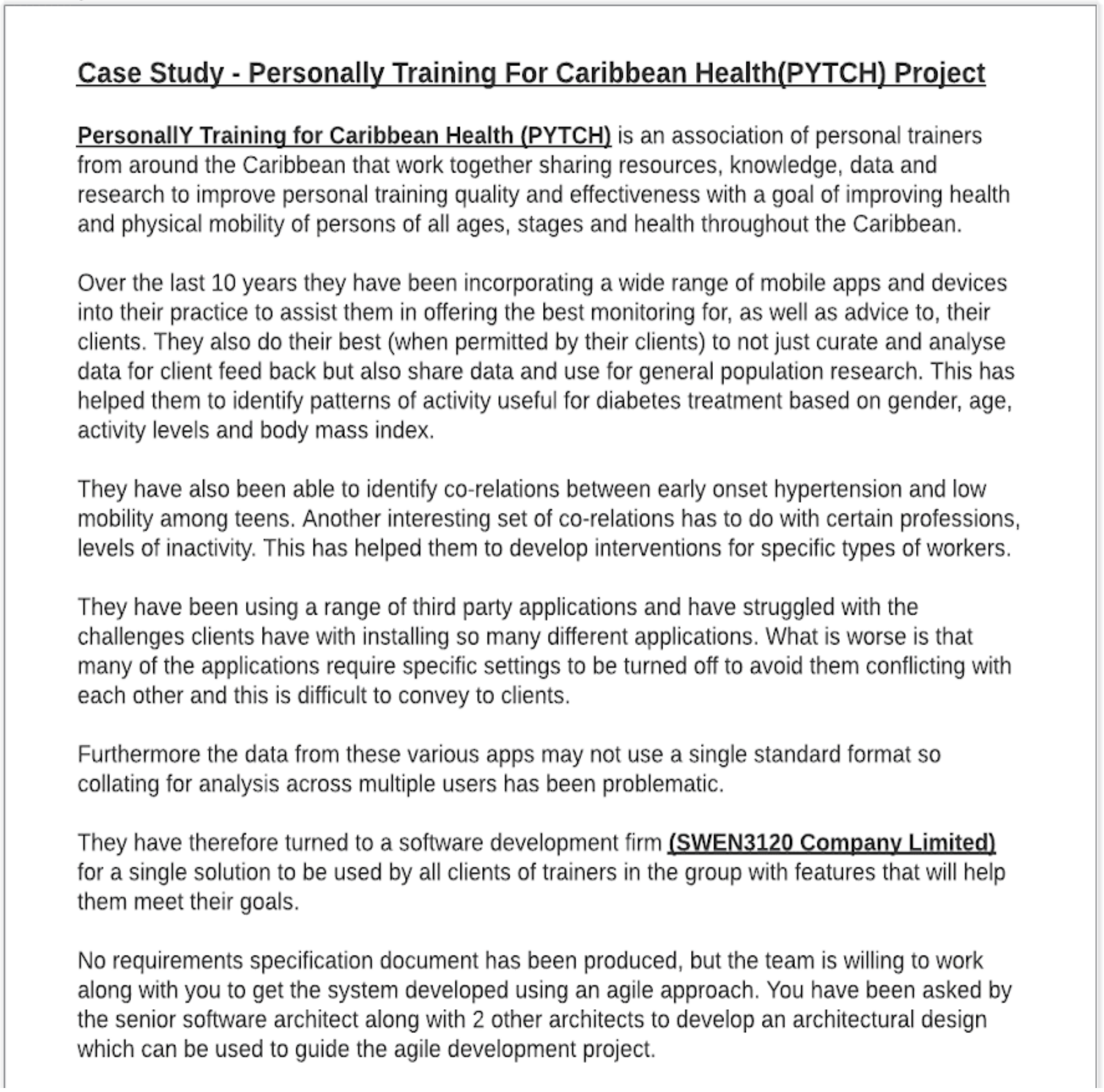


Battery on devices should last long. App should not drain my battery.

The program should not drain ~~phones~~ devices energy at a different rate whether it is connected to the internet or not

A.2 Case Study Appendix

Case Study



Case Study - Personally Training For Caribbean Health(PYTCH) Project

PersonallY Training for Caribbean Health (PYTCH) is an association of personal trainers from around the Caribbean that work together sharing resources, knowledge, data and research to improve personal training quality and effectiveness with a goal of improving health and physical mobility of persons of all ages, stages and health throughout the Caribbean.

Over the last 10 years they have been incorporating a wide range of mobile apps and devices into their practice to assist them in offering the best monitoring for, as well as advice to, their clients. They also do their best (when permitted by their clients) to not just curate and analyse data for client feed back but also share data and use for general population research. This has helped them to identify patterns of activity useful for diabetes treatment based on gender, age, activity levels and body mass index.

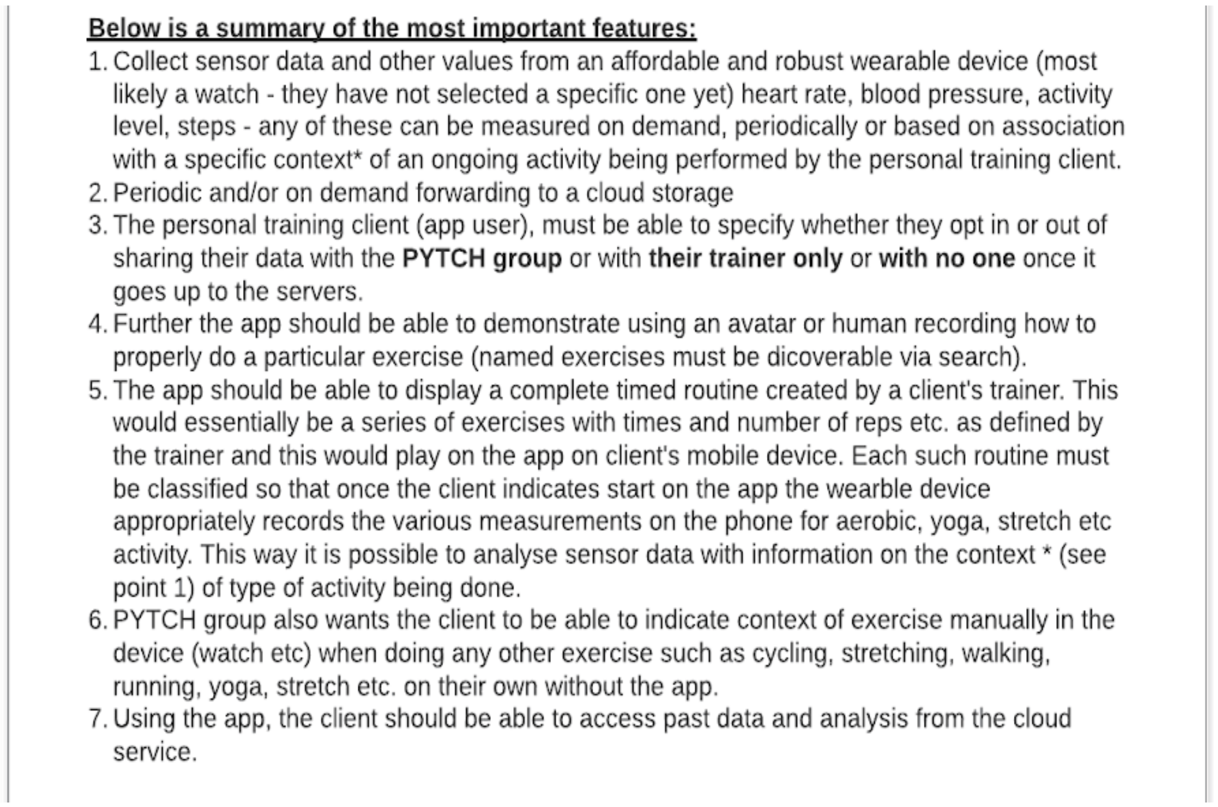
They have also been able to identify co-relations between early onset hypertension and low mobility among teens. Another interesting set of co-relations has to do with certain professions, levels of inactivity. This has helped them to develop interventions for specific types of workers.

They have been using a range of third party applications and have struggled with the challenges clients have with installing so many different applications. What is worse is that many of the applications require specific settings to be turned off to avoid them conflicting with each other and this is difficult to convey to clients.

Furthermore the data from these various apps may not use a single standard format so collating for analysis across multiple users has been problematic.

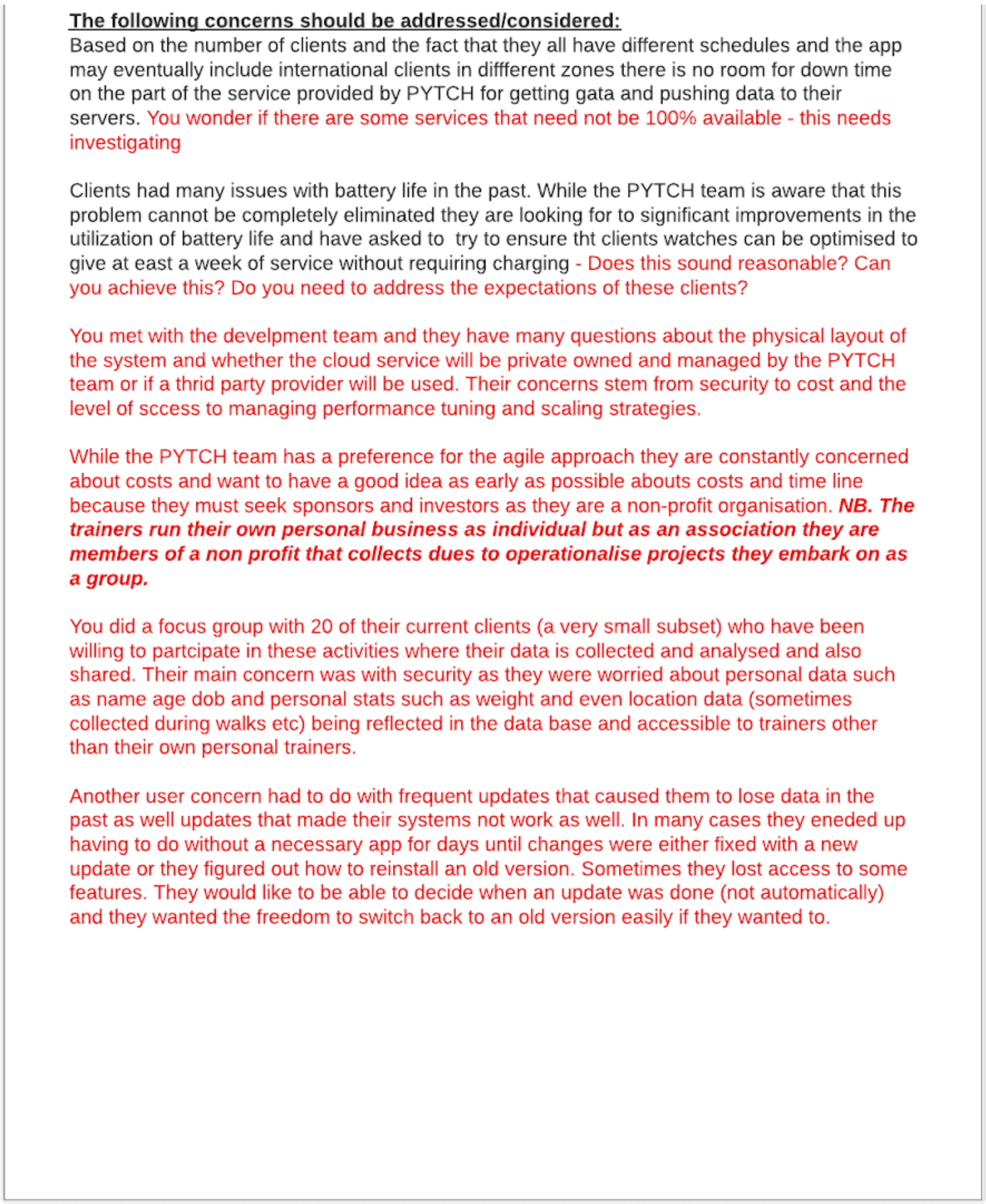
They have therefore turned to a software development firm **(SWEN3120 Company Limited)** for a single solution to be used by all clients of trainers in the group with features that will help them meet their goals.

No requirements specification document has been produced, but the team is willing to work along with you to get the system developed using an agile approach. You have been asked by the senior software architect along with 2 other architects to develop an architectural design which can be used to guide the agile development project.



Below is a summary of the most important features:

1. Collect sensor data and other values from an affordable and robust wearable device (most likely a watch - they have not selected a specific one yet) heart rate, blood pressure, activity level, steps - any of these can be measured on demand, periodically or based on association with a specific context* of an ongoing activity being performed by the personal training client.
2. Periodic and/or on demand forwarding to a cloud storage
3. The personal training client (app user), must be able to specify whether they opt in or out of sharing their data with the **PYTCH group** or with **their trainer only** or **with no one** once it goes up to the servers.
4. Further the app should be able to demonstrate using an avatar or human recording how to properly do a particular exercise (named exercises must be dicoverable via search).
5. The app should be able to display a complete timed routine created by a client's trainer. This would essentially be a series of exercises with times and number of reps etc. as defined by the trainer and this would play on the app on client's mobile device. Each such routine must be classified so that once the client indicates start on the app the wearble device appropriately records the various measurements on the phone for aerobic, yoga, stretch etc activity. This way it is possible to analyse sensor data with information on the context * (see point 1) of type of activity being done.
6. PYTCH group also wants the client to be able to indicate context of exercise manually in the device (watch etc) when doing any other exercise such as cycling, stretching, walking, running, yoga, stretch etc. on their own without the app.
7. Using the app, the client should be able to access past data and analysis from the cloud service.



<u>The following concerns should be addressed/considered:</u>
Based on the number of clients and the fact that they all have different schedules and the app may eventually include international clients in diffferent zones there is no room for down time on the part of the service provided by PYTCH for getting gata and pushing data to their servers. You wonder if there are some services that need not be 100% available - this needs investigating

Clients had many issues with battery life in the past. While the PYTCH team is aware that this problem cannot be completely eliminated they are looking for to significant improvements in the utilization of battery life and have asked to try to ensure tht clients watches can be optimised to give at east a week of service without requiring charging - Does this sound reasonable? Can you achieve this? Do you need to address the expectations of these clients?

You met with the develpment team and they have many questions about the physical layout of the system and whether the cloud service will be private owned and managed by the PYTCH team or if a thrid party provider will be used. Their concerns stem from security to cost and the level of sccess to managing performance tuning and scaling strategies.

While the PYTCH team has a preference for the agile approach they are constantly concerned about costs and want to have a good idea as early as possible abouts costs and time line because they must seek sponsors and investors as they are a non-profit organisation. ***NB. The trainers run their own personal business as individual but as an association they are members of a non profit that collects dues to operationalise projects they embark on as a group.***

You did a focus group with 20 of their current clients (a very small subset) who have been willing to partcipate in these activities where their data is collected and analysed and also shared. Their main concern was with security as they were worried about personal data such as name age dob and personal stats such as weight and even location data (sometimes collected during walks etc) being reflected in the data base and accessible to trainers other than their own personal trainers.

Another user concern had to do with frequent updates that caused them to lose data in the past as well updates that made their systems not work as well. In many cases they eneded up having to do without a necessary app for days until changes were either fixed with a new update or they figured out how to reinstall an old version. Sometimes they lost access to some features. They would like to be able to decide when an update was done (not automatically) and they wanted the freedom to switch back to an old version easily if they wanted to.

1. Documentation Road Map (10%) 4
2. Architecture Background (20%) 16
3. Views (40%)
    1. Provide 1 Deployment view
    2. Provide at least 1 module view (this may be somewhat of a high level view) 35
4. Relations Among Views (10%) 7
5. Other (5% - e.g. risk identification and issue tracking) 3
6. Presentation of work - well organised, grammar etc. (5%) 5

70