GPIG-C Initial Report

Word count: 3597 *
Friday, 25th October 2013

1 Introduction

This document is intended for both ourselves and the customer. It contains an initial analysis of requirements and a development plan. A proposed solution, based on identified use cases and requirements, is presented herein.

1.1 Single statement of need

We aim to deliver a Health and Usage Monitoring System (HUMS), tailorable to multiple target domains. The target user for the system is any consumer needing to collect, store, analyse and report data from one or more data input clients.

2 Use cases

Three use cases were identified after initial communications with the customer and used to generate the requirements.

ID $UC1$
Name System Monitoring.
Context The consumer develops a system, for which they require a HUMS.
Primary Actor The consumer's technical representative.
Secondary Actors The HUMS, the consumer's system.
Preconditions

- The consumer has a system that requires monitoring.
- The consumer's system and environment must conform to constraint requirements detailed in this document.
- The end user has access to the facilities required to install the HUMS. The end user has basic computing knowledge.

Trigger The technical representative sets up an account and attempts to integrate their system.
Main Success Scenario The end user successfully manages to integrate their system with the HUMS.
Main Success Postcondition Data can be passed between the HUMS and the consumers system.
Exception Scenarios

- The end user fails to integrate their system because their system does not conform to the data input interface.
- The end user fails to integrate their system because their system does not conform to the output interface.
- The end user chooses to abort the process.

Exception Postcondition The user is provided with diagnostics and technical support. If the end user chose to quit the process, they are presented with access to technical support.

ID $UC2$
Name Analysing collected data.
Context The consumer wishes to define how their data should be analysed.
Primary Actor End user.
Secondary Actors The HUMS.
Preconditions

- The user has set up an account.
- The consumer's sensor system and the HUMS have been successfully integrated.

Trigger The user decides how they want to analyse their data abstractly.
Main Success Scenarios

- The user creates a concrete implementation of their abstract analysis system and interfaces it with the HUMS, allowing them to analyse data as per their definition.

Main Success Postcondition User can analyse data stored in the HUMS.
Exception Scenarios User's analysis system does not conform to the analysis interface.
Exception Postcondition HUMS can still collect and store data.

ID $UC3$
Name Reporting and notifications.
Context The consumer's sensor system, analysis system and the HUMS have been successfully integrated, and they now wish to define how their users should be notified of events or receive reports.
Primary Actor End user.
Secondary Actors The HUMS.

* *Using TeXCount, excluding all tables and figures except the Risk Register*

Preconditions

- The user has set up an account.
- The consumer's sensor system, analysis system and the HUMS have been successfully integrated.

Trigger The user decides on the format and communication method used to keep them informed of the state of the system.

Main Success Scenarios

- The user implements a notification and reporting system which correctly integrates with the HUMS.
- The user uses a pre-made notification and reporting system plugin, allowing them to receive updates about the state of the system they are monitoring.

Main Success Postconditions

- The user is notified when the analysis system fires an event.
- The user can request reports.

Exception Scenarios User's notification and reporting system implementation does not conform to the notification and reporting interface.

Exception Postcondition HUMS can still collect, store and analyse data.

3 Requirements

For this project, requirements have been divided into three categories: functional, non-functional and constraint. The requirements have been engineered such that each falls into one of these three categories, which prevents them from becoming too complex. Each requirement has been structured as simply and concisely as possible and been worded in such a way as to avoid ambiguity. The requirements represent the problem to be solved and serve as a contract between us, as the development team, and the customer. All requirements have therefore been checked with the customer to ensure the system described is the system they envisioned.

The functional requirements detail what inputs, behaviour and outputs the system must provide. The non-functional requirements specify the qualities of the system as opposed to its behaviour. Constraint requirements are those that apply to the entire system, including any constraints on the environment the system can be used in and any timing constraints. In order to ensure all requirements are verifiable, appropriate testing procedures have been included.

3.1 Functional Requirements

3.1.1 Sensing Data

ID	Description	Verification
FR.1	Data input clients shall be able to push correctly structured data to the system.	Unit Testing • Send correctly structured data to the system. Assert data is correctly received. • Send incorrectly structured data to the system. Assert data failed structure validation.
FR.2	The system shall allocate a timestamp to new data.	Unit Testing • Assert data is timestamped. • Assert timestamps reflect the order in which data arrives.

3.1.2 Storing Data

ID	Description	Verification
FR.3	The system shall store correctly structured data.	Unit Testing • Attempt to store correctly structured data in the system, assert data can be retrieved. • Attempt to store incorrectly structured data in the system, assert data is rejected.
FR.4	The system shall allow the consumer to define a low storage threshold.	Black Box Testing • Attempt to set a low storage threshold. Assert attempt is successful.

FR.5	The system shall send a notification when the consumers defined low storage threshold is reached.	Unit Testing • Simulate reaching the low storage threshold. Assert method invoking a notification is called.
FR.6	The system shall allow the consumer to set an expiry time on data.	Grey Box Testing • Attempt to set an expiry time on data when adding data to the system, assert data has been stored with the expiry time.
FR.7	The system will delete data when its expiry time is reached.	Integration Testing • Assert no stored data is older than its defined expiry time.
FR.8	The system must store no more data records than the consumers defined storage quota.	Integration Testing • Assert maximum number of data records held by the consumer is less than or equal to their defined quota.
FR.9	The system shall allow the user to define that, upon reaching their defined data storage quota, new data is no longer added.	Unit Testing • Simulate reaching an arbitrary storage quota then attempt to send more data to the system, assert send failed due to insufficient storage.
FR.10	The system shall allow the user to define that, upon reaching their defined data storage limit, old data is deleted to make room for new data.	Unit Testing • Simulate reaching an arbitrary storage quota. Assert next send succeeds adding the new data to storage. Assert previous oldest record in storage is deleted.

3.1.3 Analysing Data

ID	Description	Verification
FR.11	The system shall allow the consumer to specify what patterns of data will produce events.	Black Box Testing • Assert that a specification of a valid pattern within a valid set of data succeeds.
FR.12	Events shall be triggered in response to new data that matches a pattern that the consumer has specified.	Unit Testing • Send the system a pattern of data known to constitute an event, assert an event is produced.

3.1.4 Reporting Events

ID	Description	Verification
FR.13	The system shall dispatch a notification to the data output client after an event has occurred.	Unit Testing • Simulate an event occurring, assert method producing a notification is invoked.
FR.14	The consumer shall be able to assign types to events.	Black Box Testing • Attempt to assign a valid type to a valid event, assert attempt is successful.
FR.15	The consumer shall be able to assign a cool down period for each type of event.	Grey Box Testing • Simulate the customer setting a cool down period, assert attempt is successful.
FR.16	After the sending of a notification for an event of a particular type, no more notifications for an event of that type will be sent during the cool down period.	Black Box Testing • Simulate multiple events of the same type being triggered, assert only one notification during each cool down period is sent.

FR.17	The system must allow data output clients to pull reports from the system.	Unit Testing • Attempt to pull a data report from the system, assert attempt is successful. Assert received data is correct.
FR.18	The system shall allow the consumer to set which data output clients can pull reports.	Unit Testing • Attempt to set which output clients can pull reports. • Attempt to send report to registered output client. Assert it succeeds. • Attempt to send a report to unregistered output client. Assert that it fails.
FR.19	The system shall allow the consumer to set which output clients can be sent notifications.	Unit Testing • Attempt to set which output clients can be sent notifications. • Attempt to send notification to registered output client. Assert it succeeds. • Attempt to send notification to unregistered output client . Assert it fails.

3.2 Non-functional requirements

3.2.1 Maintainability

ID	Description	Verification
NFR.1	The system shall be able to receive hardware changes without loss of previously stored data.	Recovery Testing • Snapshot live system. Change a piece of hardware. Assert after change, data is still consistent with snapshot.

3.2.2 Accessibility

ID	Description	Verification
NFR.2	Users shall be provided with documentation detailing how to use the system.	Acceptance Testing • Require a receipt from the customer verifying the availability and standard.
NFR.3	When running externally on servers, the system must be accessible to end users in multiple geographic locations.	Black Box Testing • Assert networked system can be accessed from multiple geographic locations.

3.2.3 Security

ID	Description	Verification
NFR.4	The system shall only accept data from an input client providing valid credentials.	Unit testing • Check that system both rejects unauthorised data and successfully accepts data from a source with valid credentials.
NFR.5	The system shall store data according to the relevant industry security standards.	Acceptance testing • Have system externally verified against relevant standard setting body guidelines.

3.2.4 Testability

ID	Description	Verification

NFR.6	The system shall be tested to ensure all requirements are met before deployment.	System Testing • Make sure all tests of other requirements have passed.
NFR.7	The customer will complete acceptance testing before the system is deployed.	Acceptance Testing • The customer will be required to sign off the project upon passing acceptance testing.

3.2.5 Scalability

ID	Description	Verification
NFR.8	The system must be able to support at least 5 output clients per HUMS instance.	Black box testing • Set up a HUMS instance and add 5 output clients and verify each can be sent a report.
NFR.9	The system must cope with up to 2000 data input requests per second per HUMS instance.	Grey Box Testing • Set up HUMS instance and send 2000 data valid input requests per second. Verify all data is stored in system.

3.2.6 Reliability

ID	Description	Verification
NFR.10	The system must be available for no less than 99.9% of each month.	Alpha testing • Allow system to be beta tested for one month. Assert it was available for at no less than the required amount of time.
NFR.11	Data must be backed up within 24 hours of having been made available to the system.	Integration testing • Run system for over 24 hours and verify that any data older than 24 hours exists in back-ups. White Box Testing • Verify method exists to automatically back-up data at least every 24 hours.
NFR.12	Timestamps applied by the system must be accurate to within $5ms$ of UTC.	Grey Box Testing • Send system the maximum supported data input requests per second. Assert all timestamps are to within the given accuracy.
NFR.13	The system shall dispatch notifications within $5ms$ of the event being triggered.	Unit Testing • Simulate an event. Assert that the notification is dispatched within the time period specified.

3.3 Constraint Requirements

3.3.1 System Environment

ID	Description
CR.1	A valid data schema defining the form of data that will be entered into the system will be provided.
CR.2	A valid data schema defining the form of data that will be stored the system will be provided.
CR.3	The system will be presented with valid credentials by the consumers data input clients.
CR.4	Valid output clients for the system will be specified by the consumer.
CR.5	The hardware that the system runs on will support the runtime of our software solution.
CR.6	The system will have access to the network to which the data sources are connected.

3.3.2 Time

ID	Description

CR.7	The customer will receive an initial project plan detailing the project requirements no later than 25/10/2013.
CR.8	The customer will receive an interim report detailing project progress no later 14/02/2014.
CR.9	The customer will receive a final report detailing the proposed system no later than 28/05/2014.
CR.10	The customer will be presented with a system prototype on 30/05/2014.

3.3.3 Development

ID	Description
CR.11	The development team will be comprised of seven software engineers.
CR.12	Any requirement changes outside of this document will be negotiated separately.

4 Possible solutions

When examining possible HUMS solutions, we inspected sensing, storing, analysing and reporting data separately. This was done so that the best choice for each could be individually selected, ensuring every sector of the system performs to the highest possible standard.

When examining how the HUMS should sense new data we identified three distinct options:

1. Data input clients are directly linked to the system, with data being pushed or polled depending on client support. This requires the system to implement custom data specifications, which is likely to become infeasible as the system expands across domains.
2. Unmanaged data input clients which store data elsewhere. The HUMS polls for data at the customers request. This option is more inefficient than having data pushed to the system since polls may frequently fail to collect data and therefore waste resources, or data may otherwise be available but uncollected by the HUMS and thereby leave it out-of-date. This approach, however, may integrate better with existing data input clients, reducing workload for the consumer.
3. Provide an input API which data input clients wishing to push data to the system must implement. This option is the most flexible, allowing the system to be integrated with any consumer system, in any domain. It does potentially mean more work for the consumer in migrating existing systems to use the provided API.

When considering options for how to store data there are two core methods, an SQL or NoSQL database. Using an SQL database would provide the consumer with guaranteed atomicity, consistency, isolation and durability (ACID), thereby offering reliability over throughput. Migration for the consumer may be easier when using an SQL database, as SQL has been popular among enterprises for a long time. However, SQL does not easily scale horizontally, making it difficult to expand one system across a large number of consumers. NoSQL solves the problem of scalability at the expense of full ACID compliance. Polyglot persistence offers a solution to all of the problems identified when choosing between SQL and NoSQL, by allowing the use of both. When using polyglot persistence the type and structure of every type of data must be examined to determine which datastore is most suitable, whether SQL or NoSQL.

When designing the data analysis segment of the system there are three feasible implementations:

1. A plugin API, which allows users to implement their own analysis rules. The system would provide multiple layers of abstraction, allowing the consumer to take control of analysis at the level they see fit. For example, using a low level API may be faster and allow greater customisation, but a web API is more generic and easier for the consumer to implement.
2. Providing a rules engine for analysis, allowing the consumer to specify and integrate their analysis requirements at a high level. This option would be useful from the consumers perspective, abstracting from any low level interfacing with the HUMS, but increasing the workload for our team.
3. Create customised single-purpose tools for each domain. This would be massively inefficient from a development perspective, as well as costly and inflexible. However the consumer may prefer to have a system completely designed around their needs.

For the reporting segment of the HUMS there are again multiple options. Consumers may wish to pull reports from the system at specified times, or want the system to notify another system or end user when an event has occurred. When considering the consumer requesting a data report, we could provide them with a single output format such as PDF or XML, however this is not very flexible and is unlikely to provide the required functionality for all consumers. Another option is to allow the user to directly access the HUMS datastore, however this would leave the system open to data inconsistencies and expose implementation details to the consumer. A more flexible solution would be to provide a generic reporting API to which plugins can be added. Each plugin would add a different report format, allowing the consumer to choose which they wish to use in each situation. This solution would be highly flexible, with new plugins being added as the HUMS extends across domains.

When pushing notifications to another system or user, the HUMS could send events encoded with a standard data interchange format, for example JSON or XML. This would allow them to process the output themselves. The HUMS could also provide a set of plugins allowing the consumer to choose what type of notification is triggered when an event of a specific type occurs. For example, the system could send an email for passive events but a mobile push notification or text message for an event that requires urgent action. Allowing a variety of notification channels provides a much more tailorable and maintainable product, but it requires more development time and effort.

4.1 Proposed solution

Our proposed solution, shown in figure 1, uses the concept of a client-server architecture, with the consumer's system acting as the client and the server being the HUMS processing the data. The HUMS could exist within the consumer's data input client, or could be geographically-distant but connected on the same network or though the internet. The solution brings together the best elements from our possible solutions for each section of the system, utilising a plugin architecture for several components.

The HUMS will provide an input API for data input clients to push new data to the system, accepting any correctly formatted sensor data sent on the correct network (**DD.1**). Achieving correctly formatted data may require changes to the consumer's system, or use of middleware. This approach makes our system flexible, supporting a wide range of data input clients, meaning in future the solution could easily extend across domains.

For data storage, the only applicable solution in this scenario is a database (**DD.2**). Using polyglot persistence gives us the option of choosing the database technology most suited to the data, which we will determine during development. To make our choice we will look at all the relevant technologies available and evaluate strengths and weaknesses in relation to other design decisions for the data being stored, and how it will be accessed.

For analysis the system will provide generic plugins, as well as an API for consumers to extend and create their own specialised analysis rules (**DD.4**). Offering an API allows the system to be tailored to different consumers and domains, as generic analysis tools could not hope to meet every consumer's needs by itself. Domain-specific plugins will be provided to simplify the process for those performing common analysis functions. This might include graphical tools or other user-friendly features. The range of plugins offered can be expanded as the system expands into more domains.

We will allow the consumer to customise the types of reports that can be created (**DD.5**). They will be able to request reports in a variety of formats, including human-friendly formats, such as PDF, and common data interchange formats, such as XML. Consumers can use these data formats as a data API to build their own reports or for any other purpose. This will allow the consumer flexibility to produce what they want, whilst also offering them easy to use built-in tools.

Our system will provide plugins to support various types of notifications (**DD.6**) that can be triggered based upon customisable events. Notifications will include emails and mobile push, alongside more generic notifications formatted in standard formats such as XML. The generic notifications could then be processed further by the consumer, allowing them to tailor how notifications are used and delivered.

An administration centre (**DD.3**) that allows the consumer to configure any customisable system settings, including setting up events that trigger notifications and defining data schemas, will also be included. This tool will be offered through an intuitive web based interface which is only accessible by the authorised consumer.

For the prototype we hope to demonstrate at the end of the project, we plan to implement most of the solution described above. The core of the system, including sensing, storing, analysing and reporting data, will be functional, as will a small set of the domain specific components built utilising the plugin architecture. Some additional simple components may be required when demonstrating, such as a tool to simulate data being input to the system.

4.1.1 Design Decision

Design Decision ID	**Requirements Fulfilled**	**Description**
DD.1	$FR.1, FR.2$	Provide an input API which data input clients wishing to push data to the system must conform to.
DD.2	$FR.7, FR.8$	Use a database to store data.
DD.3	$FR.4, FR.5, FR.6, FR.9, FR.10, FR.14, FR.15, FR.18, FR.19$	Provide an admin centre which allows the consumer to define data schemas and any configurable system settings.
DD.4	$FR.11$	Provide an analysis API, which the consumer can choose to implement, and a set of plugins conforming to this API, which perform commonly used analysis functions.
DD.5	$FR.17$	Provide a set of plugins which allow the consumer to pull reports from the system in a variety of common formats, including high level formats such as PDF and low level formats such as XML.

DD.6	$FR.13, FR.13, FR.16$	Provide a set of plugins which push notifications of different formats to end users or data output clients.

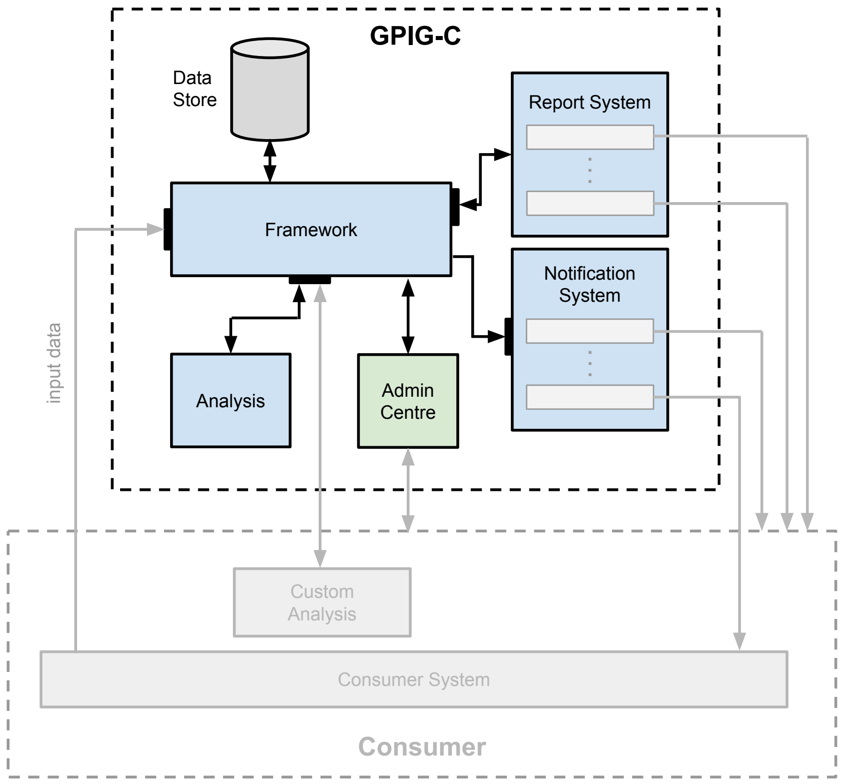

Figure 1: System diagram

5 Team organisation

5.1 Team members

A list of team members is provided below, accompanied by a list of roles and assignments relevant for the course of the project. The responsibilities have been divided in a such a way that effort for all team members is equal and fairly distributed. Assignees have been given roles which, where possible, utilise their identified skills, and minimise exposure to established weaknesses.

AJF-Adam Fahie	**AIF**-Andrew Fairbairn	**TF**-Anthony Free	**TD**-Tom Davies
JM-Joseph Mansfield	**RT**-Rosy Tucker	**MW**-Michael Walker	

5.2 Administrator Roles

Role	**Description**	**Assignees**
Point of Contact	Buffer between customer and team.	RT
Team Coordinator	Manage team communication and organise meetings.	JM
Minutes Secretary	Take minutes during meetings.	AJF,AIF,TF,TD,JM,RT,MW

5.3 Technical Roles

Role	**Description**	**Assignees**
Editors	Review and format draft documentation.	JM,TD,RT
Software Architect	Responsible for ensuring requirements are met, and software is consistent.	AJF
Requirements Analysis	Identification of customer needs and requirements.	RT,AJF,JM,TD,AIF
Developer	Developing software to agreed standards.	AJF,AIF,TF,TD,JM,RT,MW
UX/UI Designer	Accessibility, UI/UX and user stories.	JM,RT,AIF

Security	Advising on relevant security standards and protocols.	TF,MW
Penetration Tester	Test HUMS for exploits and security weaknesses.	AJF,TF,AIF,TD,MW
Software Tester	Creating or assisting in the creation of tests.	AJF,AIF,TF,TD,JM,RT,MW
Risk Manager	Enhancing chance of a successful delivery.	RT.TD
Database Admin	Design of database strategies and schemas.	AJF,AIF,MW
Legal	Avoiding copyright license and patent violations.	JM
Quality Assurance	Ensure cohesion between system modules, analysing system performance and enforce code quality standards.	JM,TD
Product Owner	Representing the views of the customer.	RT
Scrum Master	Ensuring Scrum process is followed by all team members, and ensuring good inter-team communication.	RT

6 Development plan

6.1 Software engineering methodology

When developing the HUMS system, we chose to take an agile approach to development and team management. Our implementation will follow a plugin architecture, with one central codebase providing the core functionality and plugins which provide domain tailorability. If a non-agile approach was to be taken, a plugin architecture would be infeasible as those methodologies require all requirements and documentation to be completed before development commences. For example, the V-model and Waterfall model are linear processes, where requirements are only discussed and designed before implementation commences. This linearity gives no support for changes in requirements during the later phases. Given the time constraints and the need for the system to be able to evolve into various domains, these linear development methodologies are impractical.

Agile software development, however, allows the project to split into smaller sub-projects called ‘iterations’ [1]. Each iteration allows for a different section of the project to be planned, documented, developed and tested. It also allows for the development team to be split into smaller sub-teams working simultaneously on different areas on the project. At the end of each iteration project priorities can be re-evaluated and the team structure altered.

[1] identifies three key perspectives of software development: human, organisational and technological. The scrum methodology harnesses two of those perspectives, providing a platform for both team and project organisation. Unfortunately scrum does not provide a clear approach to development from a technological perspective, providing no clear guidelines on how code should be created or structured. However, the test driven approach does provide a clear method of developing software from a technological perspective. This encourages the development of unit and acceptance tests before the development of code, the code itself is then written to make the tests pass. Using both methodologies side by side appears to be the best approach for this project, allowing for a well managed and organised team to produce well structured and reliable code. Feature driven development could be used in place of test driven and would provide a good platform for plugin generation, allowing each plugin to be designed and implemented independently. However, it conflicts with the project organisation aspects of the scrum methodology, meaning they could not be used together without creating confusion. Since the scrum method provides more guidance for managing team structure and monitoring project progress, feature driven development was discounted in favour of scrum.

6.2 Schedule

After examining the scope of the project we divided the first two phases into a schedule of tasks, with each task assigned to the team members responsible for the corresponding project role. We then organised the tasks into sprints, assigning work periods for each. Once organised the tasks were examined to ensure equal effort from all team members throughout the course of the project. Dependencies between tasks were then extracted, identifying a critical path through the project, ensuring, for every task, all prerequisite tasks are completed prior to the task commencing. The flow of work can be seen in figure 2 and shows an even work schedule throughout.

ID	Task Name	Start	Finish	Dependencies	Assignees
1	Initial problem analysis	11/10	12/10		All
2	Discussions with stakeholders	12/10	17/10	1	RT
3	Requirements analysis	14/10	18/10	1,2	All
4	Identify use cases	19/10	20/10	3	RT,TD
5	Propose prototype solutions	22/10	22/10	4	AJF,AIF,TF, TD,JM,RT
6	Decide on proposed solution	22/10	23/10	5	All
7	Define team structure	23/10	23/10		AJF,JM
8	Identify risks	22/10	23/10		TF
9	Produce initial report	19/10	23/10		All

10	Further requirements engineering	07/11	08/11		All
11	Discussions with stakeholders	07/11	09/11		RT
12	Generate additional use cases	09/11	09/11	10	All
13	Database software decision	08/11	08/11		AJF,AIF
14	Decision on tools and languages	07/11	08/11		AJF
15	Software development infrastructure	09/11	10/11	14	AJF,AIF,JM
16	Sprint planning	10/11	11/11	11-15	All
17	Software development	11/11	25/11	16	All
18	Database development	11/11	25/11	16	AJF,AIF,MW
19	Testing	11/11	25/11	16	AJF,TF,AIF, TD,MW
20	Demo to customer	25/11	26/11	17-19	RT
21	Review customer feedback	25/11	26/11	17-19	All
22	Sprint planning	26/11	27/11	21	All
23	Software development	27/11	11/01	22	All
24	Database development	27/11	11/01	22	AJF,AIF,MW
25	Testing	27/11	11/01	22	AJF,TF,AIF, TD,MW
26	Demo to customer	14/01	14/01	23-25	RT
27	Review customer feedback	14/01	14/01	23-25	All
28	Sprint planning	15/01	15/01	27	All
29	Software development	16/01	29/01	28	All
30	Database development	16/01	29/01	28	AJF,AIF,MW
31	Testing	16/01	29/01	28	AJF,TF,AIF, TD,MW
32	Demo to customer	29/01	30/01	29-31	RT
33	Review customer feedback	29/01	30/01	29-31	All
34	Contingency	31/01	14/02		All
35	Produce interim report	10/11	14/02	12-15	All

7 Risk register

A risk assessment has been performed as part of the initial report to ensure the team is aware of any problems which could later arise, and to provide a guide as to how to react when such problems occur. The hazards (risks) have been identified and classified based upon team members' past experiences in similar projects and group discussion. The areas these hazards impact were then analysed, as well as the probability of occurrence. These are then weighted so that we can identify the risks which are likely to have the greatest detrimental effect on the project. The likelihood score (LS), impact score (IS) and risk matrix score (RS) are listed in the table below.

Risk ID	Risk	LS	IS	Mitigation and Contingency	RS
R.1	Short term loss of team members	6	*Moderate* Deadline failure	The team can then reactively reallocate the team member's work across remaining team members. To aid with this, the team must proactively ensure that no work relating to the project is outside of team version control. Use of the scrum methodology proactively aids work reallocation, ensuring team members are aware of all assigned work.	18
R.2	Long term loss of team members	2	*Catastrophic* Deadline failure and low standard of deliverables	If a team member is unavailable for an extended period, the team will react by notifying the customer and possibly extending deadlines. The proactive procedures mentioned in **R.1** will also be followed to reduce the impact of this scenario.	10
R.3	Short or long term loss of resources	2	*Catastrophic* Deadline failure and loss of code base	Proactive use of a source code repository, meaning code-base and history is decentralised. If the repository is lost, the data can be retrieved from the local repository copies and university backups.	10
R.4	Team member under-performance	3	*Major* Deadline failure and low standard of deliverables	Project plan must be feasible. The skills of the team as a whole, and individual team members must be proactively established early on and taken into account when assigning roles.	12

R.5	Mis-interpretation of requirements	3	*Major* Deliverables that are not valid	Requirements, design and implementation strategy must be proactively verified with the customer. This process is iterative, stopping when both customer and developers are content. Any changes to those requirements must result in re-negotiated deadlines.	12
R.6	Slow response to customer queries	3	*Major* Deadline failure and low standard of deliverables	Customer has assured a two working day response where possible. Further mitigation can be achieved by proactively communicating issues well in advance of deadlines.	12
R.7	Failure to produce required system functionality	2	*Catastrophic* Wasted time and loss of marks	Customer verification of requirements can counteract this risk. System testing, to ensure all agreed upon requirements are met, will also reduce this risk.	10
R.8	Missing internal team deadlines	5	*Moderate* Project falls behind due to missing dependencies	Perform critical path analysis to identify tasks which will take the longest time and which are a prerequisite to others. A greater team effort can then be assigned to these areas if it seems likely to miss a deadline or halt progress elsewhere.	15

The likelihood score defines probability of something occurring. Utilising *Kents Words of Estimative Probability*[2], with ‘certain’ weighted 7 and ‘impossible’ weighted 1.

			Impact Score (Least→Most)				
			1	2	3	4	5
			Negligible	Minor	Moderate	Major	Catastrophic
Likelihood Score	7	Certain	7	14	21	28	35
	6	Almost certain	6	12	18	24	30
	5	Probable	5	10	15	20	25
	4	Chances about even	4	8	12	16	20
	3	Probably not	3	6	9	12	15
	2	Almost certainly not	2	4	6	8	10
	1	Impossible	1	2	3	4	5

Score	**Risk Level**	**Recommended Response**
23-35	HIGH	Mitigation plan is required. Immediate action is required.
11-22	MEDIUM	To be included in the action plan and reviewed.
0-10	LOW	Included in action plan in limited scope. Minimum review.

8 Customer communication

Knowing that the customer may not respond to emails rapidly, we began our correspondence promptly after the project kick off. The customer did respond quickly to all messages, however we feel that erring on the side of caution and starting early was the best strategy. We followed up each response with a team meeting within two days in order to discuss and implement any changes or suggestions mentioned.

Communication so far has focussed on verifying that the outlined requirements meet the expectations of the customer. We used the initial meeting (9/10) to clarify terminology and produce an initial set of requirements, having formatted these we sent them to the customer for feedback (13/10).

Feedback (14/10) suggested that our understanding of system tailorability was inaccurate, with the customer providing clarification. After altering the requirements in response and resending them to the customer (16/10), we received further feedback suggesting that the wording of our requirements was lacking in parts.

We quickly remedied this before forwarding the changes to the customer (19/10), which resulted in helpful feedback allowing us to settle on the outlined list of requirements. In producing and gradually refining the requirements, we began to conceive an idea of how the system should function, and how it should be tailorable. This has directly affected our proposed solution, by leading us towards the outlined client-server architecture, where the HUMS acts as the server. Having decided that we wanted to structure the HUMS as a framework with support for plugins, we pitched this idea to the customer and received positive and succinct feedback.

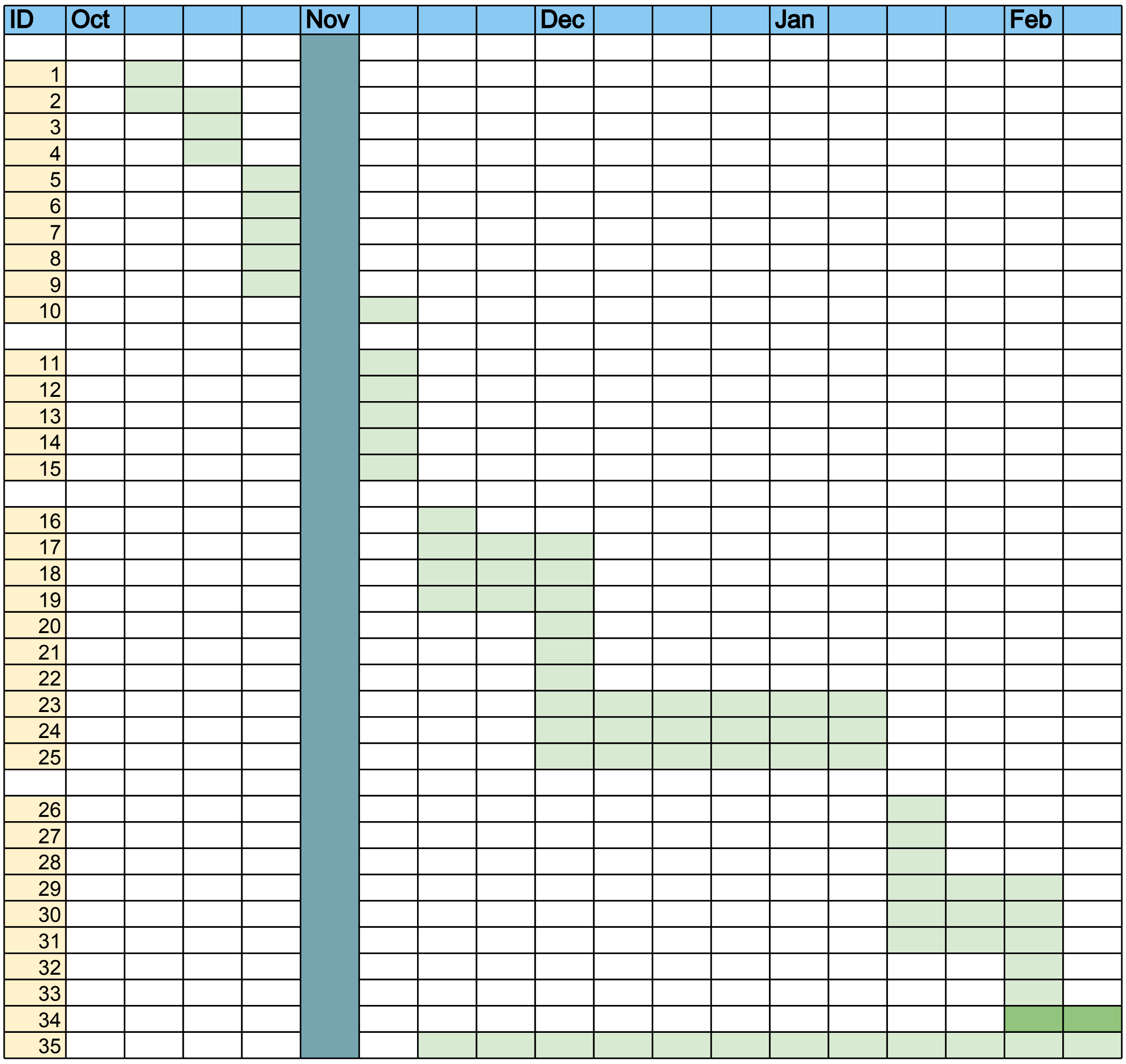

Figure 2: A Gantt chart showing the project flow of work

9 Glossary

HUMS	Health and usage monitoring system(s).
Customer	Thales.
Consumer	The recipient organisation of the system.
(End) User	An individual or organisation using the system.
Client	A piece of computer hardware or software which accesses a service made available by the HUMS system.
The System	The HUMS we are designing and developing.
Event	A point of interest flagged up by an analysis system, which may result in a notification.
Data input client	Anything that provides data to the HUMS through the input interface.
Data output client	Anything that receives data from the HUMS through the reporting or notification interfaces.
Notification	A message sent by the system as a result of an event being fired.
Report	A message sent by the system as a result of a request from a user.
HUMS Instance	One installation or occurrence of the system for a specific consumer.

References

[1] O. Hazzan and Y. Dubinsky, *Agile software engineering.* Springer, 2008.

[2] S. Kent, *Strategic intelligence for American world policy.* Princeton, N.J: Princeton University Press, 1966.