Public Health Complaint Software Product Line

Leonardo Pondian Tizzei* Cecília Mary Fischer Rubira†

January 11, 2012

*Institute of Computing - University of Campinas - e-mail: tizzei@ic.unicamp.br
†Institute of Computing - University of Campinas - e-mail: cmrubira@ic.unicamp.br

Contents

Abstract

1 Introduction

Software product line (SPL) engineering is a paradigm to develop software applications using core assets and mass customization [39]. The extractive adoption of a SPL capitalizes on existing system [33], and focuses on reusing software assets of existing products in order to reduce costs. This approach is very appropriate when the collection of products has a significant amount of commonality and differences among them [31]. The Feature-oriented reengineering process is an example of extractive approach, which provides guidelines to build a SPL from legacy software assets by using feature model to support the creation of other development assets, such as the product line architecture [29]. Feature model is usually applied to represent the commonalities and variabilities of a SPL.

Aspect-oriented Programming (AOP) can support extractive SPL adoption by inserting variability on existing components and by wrapping crosscutting concerns [50]. Crosscutting concerns are concerns that cut across other concerns and are responsible for producing tangled representations that are difficult to understand and maintain [40]. AOP can facilitate the integration of existing components by implementing glue-code [32]. Furthermore, there are evidences that the use of aspects supports the design of stable PLA [22, 47, 49]. However, existing feature-oriented reengineering approaches does not provide support for aspect-oriented techniques.

We propose a feature-oriented product line approach, which uses aspect-oriented techniques to support separation of concerns throughout the adoption process. Instead of creating the entire approach from scratch, as feature-oriented software development is a solid discipline we combined existing feature-oriented approaches to achieve our goal. When necessary, we also extended feature-oriented techniques to support reasoning about crosscutting concerns.

As integrating existing techniques is a difficult task, we performed an exploratory case study to assess the feasibility of our approach. Three legacy applications from the same domain have been chosen and we have executed the approach to produce a software product line from which three products can be derived, one for each corresponding legacy application. Another objective of the case study is to promote empirical studies with product lines, since there is lack of empirical studies in this area. Award-winning product lines are usually commercial systems which are not publicly available for conducting research [17]. Therefore, we made available all artifacts (e.g. feature diagrams, architecture, source codes) produced for the product line adoption.

2 Reverse engineering

The reverse engineering phase is important to capitalize on existing artifacts [33]. Models such as architectures and component specification support developers and domain analysts to understand legacy applications and understanding is a key factor for reuse [24]. Figure 1 shows the main activities of reverse engineering phase, which is based on Lee et al. [33].

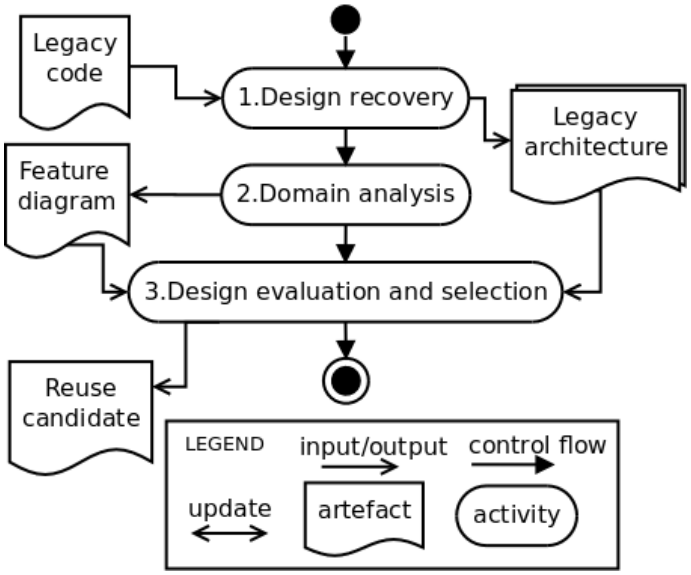

Figure 1: Reverse engineering activities

2.1 Design Recovery

The *Design recovery* activity can be supported by CASE tools such as Enterprise Architect [1]. The result of this activity is a set of legacy architectures for each corresponding application.

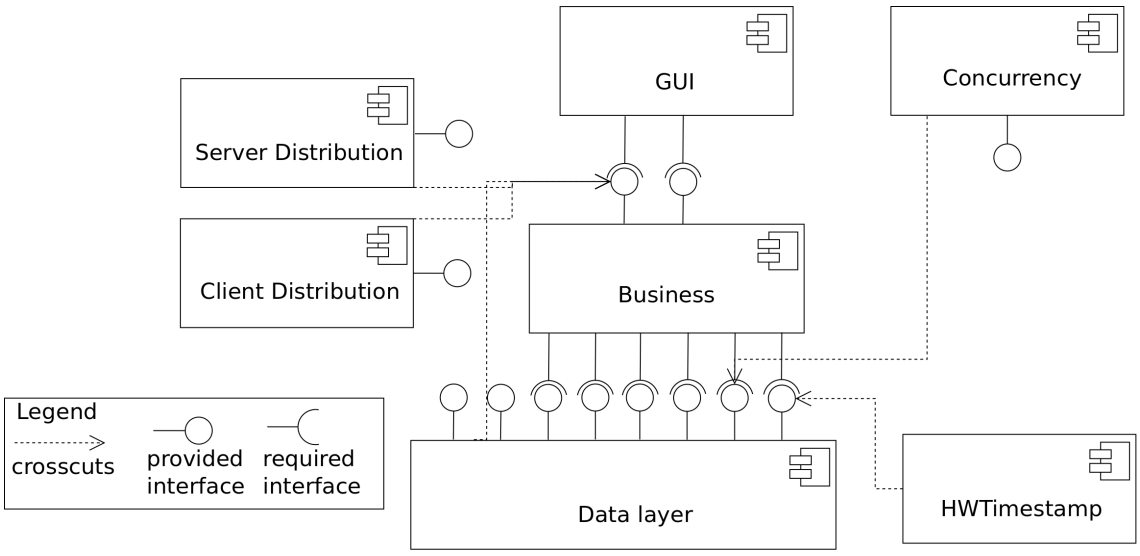

Figure 2: Component-based Architecture of Legacy Healthwatcher (based on Healthwatcher legacy architecture [3])

[1] http://www.sparxsystems.com/products/ea/downloads.html

2.2 Domain Analysis

Domain analysis defines product line features based on legacy application features and market needs [29], that is, legacy application features can be removed or changed and new features can be added due to market needs. Thus, both features that have been implemented and features that will be implemented must be represented in the feature diagram. In order to create the feature diagram, it is important to define an initial set of intended products for product line as well as their intended commonalities and variabilities [39]. The feature diagram has been created according to the guidelines proposed by Lee et al. [35]. Previous models and documents (e.g. requirements, use cases specification), and existing artifacts (e.g. users manual, the application itself) can also be useful to identify features [28]. Figure 3 shows the feature diagram of the Public Health Complaint SPL. We identified the features based on use case specification (e.g. Healthwatcher use cases [7]) and on the use of the legacy applications (i.e. Healthwatcher, Medwatch [10], and DPH-LA [1]). Feature variability should be consistent with use case variability. The relationship among features is further specified in Section 2.2.2.

2.2.1 Product Features

It should be possible to derive three software products from the core assets of the Public Health Complaint SPL: Healthwatcher [2], Medwatch [10], and DPH-LA [1].

Table 1 describes the feature of each product. The creation of this table was based on use case specification (Section 3.2). Some features were also extracted by using these software products (i.e. applications).

2.2.2 Feature Groups

Features can be grouped in order to place a constraint on how the features are used by a certain product [25]. Establishing these relationships among features also support building a modular product line architecture (PLA) [34]. According to PLUS method [25, Chapter 5.5], there are four types of feature groups, namely *exactly-one-of*, *zero-or-one-of*, *at-least-one-of*, *zero-or-more-of*. Table 2 describes feature groups and provides additional information about the relationship among features.

3 Analysis

Figure 4 shows that in this approach Analysis consists of three activities: *Use case specification*, *Croscutting concerns identification and composition*, and *Aspect-oriented feature view*.

3.1 Nonfunctional requirements

Based on the documentation and use of the applications, we identified the nonfunctional requirements (i.e. quality attributes) described below. The descriptions of nonfunctional requirements were either copied or adapted from Healthwatcher Requirements [7], Aspect-oriented Requirements Engineering (AORE) models [4], AORE viewpoints [44], and Multi-dimensional Separation of Concerns (MDSOC) [5].

Features	Products		
	Healthwatcher	**Medwatcher**	**Environmentwatcher**
Public Health Complaint SPL	✓	✓	✓
Infrastructure Management	✓		✓
Register tables	✓		✓
Update employee	✓		✓
Register new employee	✓		✓
Change logged employee	✓		✓
Login/Logout	✓		✓
Update Health Unit	✓		✓
Complaint Management	✓	✓	✓
Update Complaint	✓		✓
Complaint Specification	✓	✓	✓
Food Complaint Specification	✓	✓	✓
Animal Complaint Specification	✓		✓
Drug Complaint Specification		✓	
Special Complaint Specification	✓		✓
Support services for users	✓	✓	✓
Query Information	✓		
RSS feeds		✓	✓
Publish RSS feeds		✓	✓
Receive alerts via feeds		✓	✓
Exception Handling	✓	✓	✓
Security	✓	✓	✓
Captcha			✓
Encryption	✓	✓	✓
Computer infrastructure	✓	✓	✓
Hardware	✓	✓	✓
2.0 GHz, 1GB RAM, netCard 3Com	✓	✓	✓
Software	✓	✓	✓
Ubuntu 10.4	✓	✓	✓
Database	✓	✓	✓
MySQL	✓	✓	
Oracle			✓
Java Servlets	✓	✓	✓
Java RMI	✓	✓	✓
Distribution	✓	✓	✓
Usability	✓	✓	✓
User interface	✓	✓	✓
Compatibility	✓	✓	✓
Standards	✓	✓	✓
Performance	✓	✓	✓
Response time	✓	✓	✓
Persistence	✓	✓	✓
Concurrency	✓	✓	✓

Table 1: Features of each product

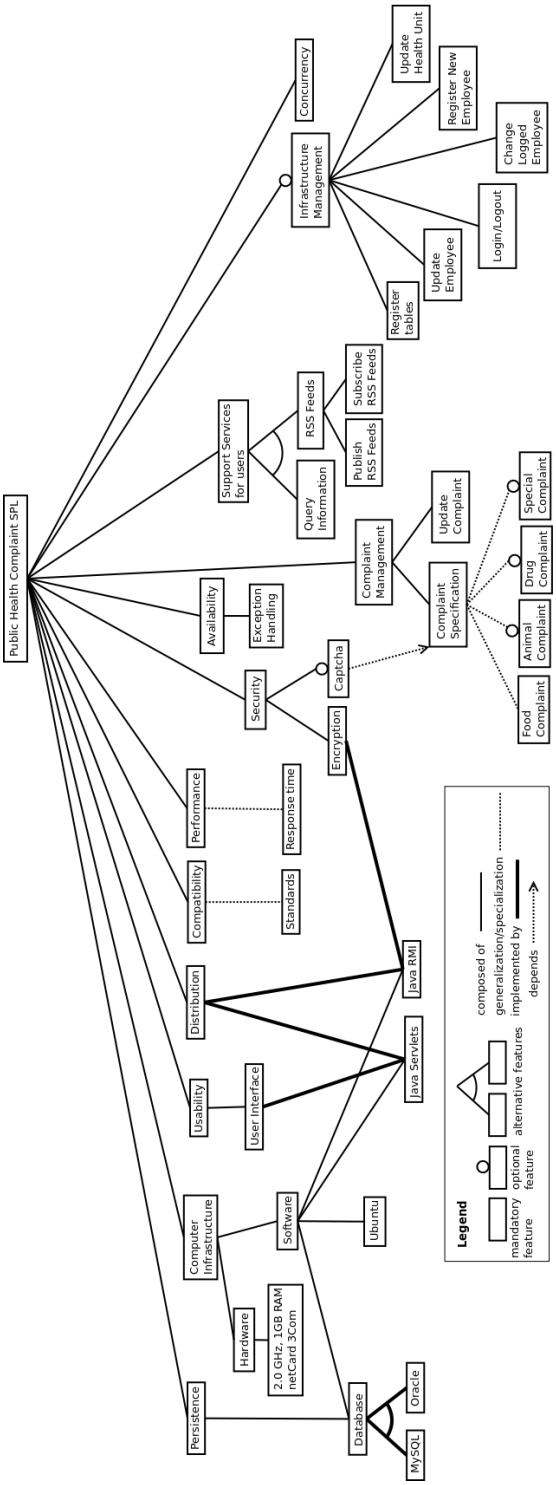

Figure 3: Public Health Complaint Software Product Line feature diagram

Feature Group Name	**Feature Group Category**	**Features in Feature Group**	**Feature Category**
Complaint Specification	at-least-one-of	Animal Complaint	optional
		Food Complaint	mandatory
		Drug Complaint	optional
		Special Complaint	optional
Database	exactly-one-of	MySQL	alternative
		Oracle	alternative
Support services for users	at-least-one-of	Query information	alternative
		RSS feeds	alternative

Table 2: Feature groups

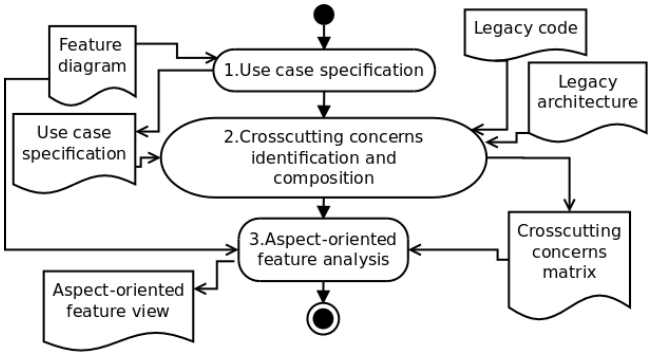

Figure 4: Analysis activities

3.1.1 Usability [NFR01]

Priority: Important

The system should have an easy to use GUI, as any person who has access to the internet should be able to use the system [7]. The user interface must be implemented using Servlets [7].

3.1.2 Availability/Exception Handling [NFR02]

Priority: Essential

The system should be available 24 hours a day, 7 days a week. The nature of the system not being a critical system, the system might stay off until any fault is fixed [7]. Several functionalities might raise errors while the user interacts with the system and require different handling techniques. General errors that apply to most cases are due to missing information (e.g. users do not fill in the required fields in an entry form) and the system signals the error and show which fields need to be provided. Other error might be related to entering invalid data and the error handling mechanism should try either to avoid that or to raise the error and suggest the correction [4]. According to Bass et al. [13, Chapter 4.1], availability is closely related to reliability and one technique to improve reliability is exception handling.

3.1.3 Performance/Response time [NFR03]

Priority: Essential

The system must be capable to handle 20 simultaneous users. The response time must not exceed 5 seconds [6, 7].

3.1.4 Encryption/Security [NFR04]

Priority: Important

The system should use a security protocol when sending data over the internet. To have access to the complaint registration features, access must be allowed by the access control sub-system [4, 7].

3.1.5 Standards/Compatibility [NFR05]

Priority: Important

The system must be developed according to the standards established by X[2], responsible for the norms and standardization of systems for the City Hall [6, 7].

3.1.6 Harware and Software/Operational environment [NFR06]

This section lists the hardware and software to be used for the system to operate in a desirable fashion [7].

Software: Ubuntu 10.04 LTS for the workstation.

Hardware: One computer with: 2.0 GHz processor, 1 GB of RAM memory, net card 3Com 10/100. This equipment shall be used by the attendant as a workstation.

3.1.7 Distribution [NFR07]

Priority: Essential

The system should be capable of running on separate machines. For example, the system core could be running on one machine and the Servlets on another [7].

3.1.8 (Flexible) Storage medium/Persistence [NFR08]

Priority: Essential

The persistence mechanism should store data about the complaints, employees, health units, deceases, specialities and citizens that complaint. The system must be capable of extension on the storage matter, making possible to use, arrays or different databases (MySQL, Oracle, etc.) [4, 7].

3.1.9 Concurrency [NFR09]

Priority: Essential

The system must be capable to handle 20 simultaneous users [4].

[2]The company name is confidential due to commercial reasons

3.2 Use cases specification

The use case specification has three main goals: (i) further detail the requirements based on legacy documentation (ii) specify use case variability (iii) identify crosscutting and non-crosscutting use cases. In order to achieve the first goal, we elicited the use cases based on existing documents of legacy applications. Fantechi et al. proposed an approach to use legacy requirements document to specify product line variability [21]. Even the use of legacy applications can be useful to understand them. Use case variability is determined based on features variability, which has already been specified in the feature diagram. PLUS method [25] represents the use case variability with stereotypes (kernel, alternative, and optional). Use cases can also represent crosscutting concerns as proposed Jacobson and Ng [27, Chapter 6.2]. Crosscutting concerns are represented by use case extensions, which add new behavior to the existing use cases. In this way, both variability and crosscutting concerns are represented.

3.2.1 Functional Use Cases

Figure 5 shows the use case diagram for the Public Health Complaint SPL. All use cases were adapted from *Healthwatcher - Requirements* document [7], but the UC05, UC15, UC16 use cases, that were included based on the domain analysis (Section 2.2).

Query information [UC01]
Use Case Name: Query information
Description: This use case allows a citizen to perform queries.

Query Health Guide. The citizen might query:

- Which health units take care of a specific specialty.
- What are the specialties of a particular health unit.

Query Speciality Information. The citizen might query:

- Information about a complaint made by a citizen:
  - Complaint details.
  - Situation (OPENED, SUSPENDED, or CLOSED).
  - Technical analysis.
  - Analysis date.
  - Employee that made the analysis.
- Information about diseases:
  - Description.
  - Symptoms.
  - Duration.

Priority: Important
Category: Optional
Inputs and pre-conditions: The data to be queried must be registered on the system

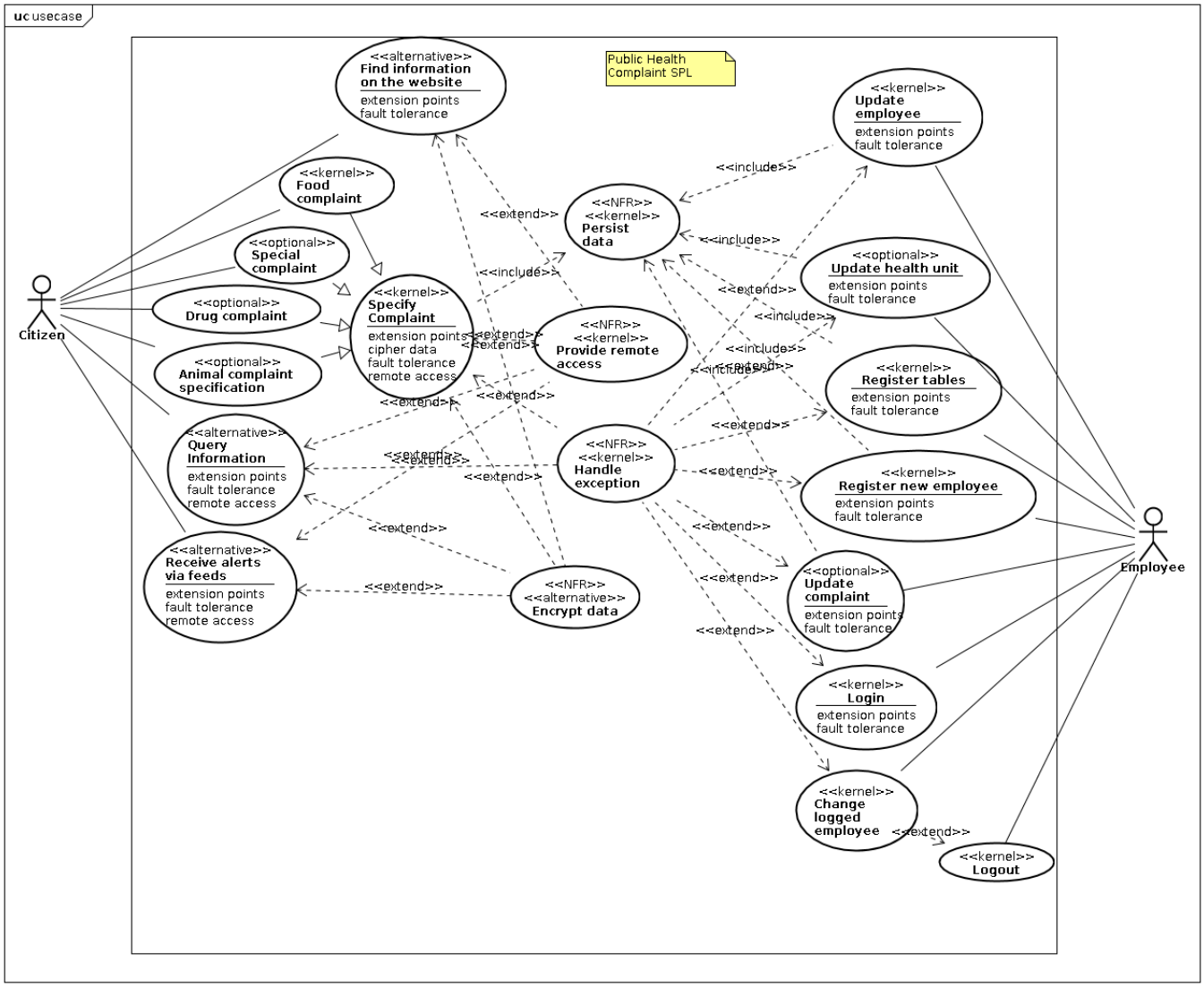

Figure 5: Public Health Complaint SPL use case diagram

Outputs and post-conditions: The query result to the citizen
Main flow of events:

1. The citizen chooses the type of query

   (a) In the case of query on specialties grouped by health units, the system retrieves the list of health units stored.

      i. The system retrieves the details of each health unit such as its description and specialties.

      ii. The list of health units is presented to the user on their local display.

(b) In the case of a query on health units grouped by specialties, the system retrieves the list of registered specialties.
   i. The system retrieves the details of each specialty such as its unique identifier and name.
   ii. The list of specialties is presented to the user on their local display.

(c) In the case of a query on diseases, the system retrieves the list of diseases.
   i. The system retrieves the details of each disease type such as its unique identifier and name.
   ii. The list of disease is presented to the user on their local display.

2. The citizen provides the data for the query

(a) In the case of a query on specialties grouped by health units, the citizen selects the health unit to be queried.
   i. A unique identifier representing the selected health unit is sent to the server.
   ii. The system ensures the health unit information is consistent.
   iii. The unique identifier is used by the system to search the repository for the selected health unit.
   iv. The details of the selected health unit are retrieved including its specialties.
   v. The specialties for the selected health unit are returned to the user.

(b) In the case of a query on health units grouped by specialties, the citizen selects the specialty to be queried.
   i. A unique identifier representing the selected specialty is sent to the server.
   ii. The system ensures the health unit information is consistent.
   iii. The unique identifier is used to retrieve the list of health units which are associated with the selected specialty.
   iv. The details of the health units and specialties are retrieved.
   v. The retrieved health units are returned to the user.

(c) In the case of a query on complaints, the citizen provides the complaint code.
   i. The unique identifier representing the complaint to be retrieved is sent to the server.
   ii. The system ensures the complaint information is consistent.
   iii. The unique identifier is used to retrieve the complaint entry.
   iv. The system must determine the complaint type as to retrieve the appropriate information.
      A. If the complaint is a special complaint the complainer's age, education level and occupation are retrieved (in addition to the standard complaint information).
      B. If the complaint is a food complaint the meal which was consumed, the number of people who ate the meal, the number of sick people, etc. are retrieved (in addition to the standard complaint information).
      C. If the complaint is an animal complaint the animal species and the number of animals affected (in addition to the standard complaint information).

v. The complaint with all the appropriate information is returned to the user.

(d) In the case of a query on diseases, the citizen selects the disease to be queried.

i. The unique identifier is used to retrieve the list of health units which are associated with the selected specialty.

ii. The details of the health units and specialties are retrieved.

iii. The retrieved health units are returned to the user.

(e) In the case of a query on complaints, the citizen provides the complaint code.

i. The unique identifier representing the complaint to be retrieved is sent to the server.

ii. The system ensures the complaint information is consistent.

iii. The unique identifier is used to retrieve the complaint entry.

iv. The system must determine the complaint type as to retrieve the appropriate information.

A. If the complaint is a special complaint the complainer's age, education level and occupation are retrieved (in addition to the standard complaint information).

B. If the complaint is a food complaint the meal which was consumed, the number of people who ate the meal, the number of sick people, etc. are retrieved (in addition to the standard complaint information).

C. If the complaint is an animal complaint the animal species and the number of animals affected (in addition to the standard complaint information).

v. The complaint with all the appropriate information is returned to the user.

(f) In the case of a query on diseases, the citizen selects the disease to be queried.

i. The unique identifier representing the disease type to be retrieved is sent to the server.

ii. The system ensures the disease type information is consistent.

iii. The unique identifier is used to retrieve the disease type to query.

iv. The symptoms for the selected disease type are retrieved.

v. The complete disease information is returned to the user.

3. The query results are formatted and presented to the user on their local display.

Specify complaint [UC02]
Use Case Name: Specify complaint
Description: This use case allows a citizen to register complaints. Complaints can be related to Food, Animal, Drugs, or Special. The four kinds of complaints have the following information in common: Complaint data: description (mandatory) and observations (optional);
Priority: Essential
Category: Optional
Inputs and pre-conditions: None
Outputs and post-conditions: The complaint saved on the system
Main flow of events:

1. The citizen selects the kind of complaint.

2. The system shows the specific screen for each type of complaint.
3. The system registers the kind, date and time of the complaints.
4. The citizen provides the complaint specific data.
5. The system saves the complaint.
   (a) The information entered by the user is sent to the server.
   (b) The system parses the data entered by the user.
   (c) The system creates a new instance of the appropriate complaint type.
   (d) The system generates a unique identifier and assigns this to the new complaint.
   (e) The complainers address is parsed and saved.
   (f) The common complaint information is parsed and stored with the OPENED state.
   (g) The specific complaint data is then extracted and stored accordingly.
   (h) The system ensures the data is left in a consistent state.
6. The unique identifier is returned and presented to the user on their local display.

Extension Points:
E1. Send data over the internet
The *Send data over the internet* extension point occurs before step 5

Food complaint specification [UC03]
Use Case Name: Food complaint specification
Description: Food Complaint - DVISA

- Cases where there is a suspicion infected food being eaten.

Priority: Essential
Category: Mandatory
Inputs and pre-conditions: none
Outputs and post-conditions: The food complaint is saved on the system
Main flow of events:

1. The citizen chooses *Food* as the kind of complaint.
2. The system shows the *Food* complaint screen.
3. The system registers the kind, date and time of the complaints.
4. The citizen provides the complaint specific data.
5. The system saves the complaint.
   (a) The information entered by the user is sent to the server.
   (b) The system parses the data entered by the user.
   (c) The system creates a new instance of the appropriate complaint type.

(d) The system generates a unique identifier and assigns this to the new complaint.
(e) The complainers address is parsed and saved.
(f) The common complaint information is parsed and stored with the OPENED state.
(g) The specific complaint data is then extracted and stored accordingly.
(h) The system ensures the data is left in a consistent state.

6. The unique identifier is returned and presented to the user on their local display.

Animal complaint specification [UC04]
Use Case Name: Animal complaint specification
Description: This use case allows a citizen to register food complaints
Animal Complaint - DVA

- Sick animals.
- Infestations (rodents, scorpions, bats, etc.)
- Diseases related to mosquitoes (dengue, filariose).
- Animal maltreatment

Priority: Essential
Category: Optional
Inputs and pre-conditions: none
Outputs and post-conditions: The animal complaint is saved on the system
Main flow of events:

1. The citizen chooses *Animal* as the kind of complaint.
2. The system shows the *Animal* complaint screen.
3. The system registers the kind, date and time of the complaints.
4. The citizen provides the complaint specific data.
5. The system saves the complaint.
   (a) The information entered by the user is sent to the server.
   (b) The system parses the data entered by the user.
   (c) The system creates a new instance of the appropriate complaint type.
   (d) The system generates a unique identifier and assigns this to the new complaint.
   (e) The complainers address is parsed and saved.
   (f) The common complaint information is parsed and stored with the OPENED state.
   (g) The specific complaint data is then extracted and stored accordingly.
   (h) The system ensures the data is left in a consistent state.

6. The unique identifier is returned and presented to the user on their local display.

Drug complaint specification [UC05]
Use Case Name: Drug complaint specification
Description: Required information for a drug complaint:

- Patient information: Patient identifier; patient age; patient weight in kilograms or pounds
- Type of problem (check at least one of the following): Adverse event; Product use error; Product problem (e.g. defects, malfunction); Problem with different manufacturer of same medicine
- Outcomes attributed to adverse event (check at least one of the following): Death (specify the date); Life-threatening; Hospitalization - initial or prolonged; Disability or permanent damage; Congenital anomaly/Birth defect; Required intervention to prevent permanent impairment/damage (devices); Other serious (important medical events)
- Date of event
- Date of this complaint (automatically filled)
- Describe the event, problem or use error textually (up to a total of 6400 characters allowed.)
- Relevant tests/Laboratory data, including dates (textual description up to a total of 2000 characters allowed.)
- Other Relevant History, Including Preexisting Medical Conditions (e.g. allergies, race, pregnancy, smoking and alcohol use, liver/kidney problems,etc.) Textual description up to a total of 2000 characters allowed.
- Product Available for Evaluation (check one of the following): yes, no, returned to the manufacturer on (specify the date)
- Suspected products information: Product name; Label strength; manufacturer/labeler; dose/amount; frequency; route; dates of use (If unknown, give duration) from/to (or best estimate); Diagnosis or Reason for Use (Indication); Event Abated After Use Stopped or Dose Reduced? (check one of the following): yes, no, doesn't apply; lot number; expiration date; Event Reappeared After Reintroduction? (check one of the following): yes, no, doesn't apply; NDC number or Unique ID.
- Suspected medical device: brand name; common device name; manufacturer name, city, and state; model number; catalog number; serial number; lot number; expiration date; other number; operator device (check one of the following): health professional, lay user/patient, other (specify); if implanted, give date; if explanted, give date; is this a single use device that was reprocessed and reused on a patient? (check one of the following): yes, no; if yes to previous item, enter name and address of reprocessor (textual description up to a total of 450 characters allowed)
- Product names and therapy dates (exclude treatment of event). Up to a total of 2000 characters allowed

- Reporter information: name; address; city; state; zip code; phone number; email; health professional (check one of the following): yes, no; also reported to (check as many as you want): manufacturer, user facility, distributor/importer; If you do NOT want your identity disclosed to the manufacturer, check here (checkbox)

Priority: Essential
Category: Optional
Inputs and pre-conditions: None
Outputs and post-conditions: The drug complaint is saved on the system
Main flow of events:

1. The citizen chooses *Drug* as the kind of complaint.
2. The system shows the *Drug* complaint screen.
3. The system registers the kind, date and time of the complaints.
4. The citizen provides the complaint specific data.
5. The system saves the complaint.
   (a) The information entered by the user is sent to the server.
   (b) The system parses the data entered by the user.
   (c) The system creates a new instance of the appropriate complaint type.
   (d) The system generates a unique identifier and assigns this to the new complaint.
   (e) The complainers address is parsed and saved.
   (f) The common complaint information is parsed and stored with the OPENED state.
   (g) The specific complaint data is then extracted and stored accordingly.
   (h) The system ensures the data is left in a consistent state.
6. The unique identifier is returned and presented to the user on their local display.

Special complaint specification [UC06]
Use Case Name: Special complaint specification
Description: Special Complaint - DVISA

- Cases related to several reasons, which are not mentioned above (restaurants with hygiene problems, leaking sewerage, suspicious water transporting trucks, etc.)

Priority: Essential
Category: Optional
Inputs and pre-conditions: none
Outputs and post-conditions: The drug complaint is saved on the system
Main flow of events:

1. The citizen chooses *Special* as the kind of complaint.

2. The system shows the *Special* complaint screen.
3. The system registers the kind, date and time of the complaints.
4. The citizen provides the complaint specific data.
5. The system saves the complaint.
   (a) The information entered by the user is sent to the server.
   (b) The system parses the data entered by the user.
   (c) The system creates a new instance of the appropriate complaint type.
   (d) The system generates a unique identifier and assigns this to the new complaint.
   (e) The complainers address is parsed and saved.
   (f) The common complaint information is parsed and stored with the OPENED state.
   (g) The specific complaint data is then extracted and stored accordingly.
   (h) The system ensures the data is left in a consistent state.
6. The unique identifier is returned and presented to the user on their local display.

Login [UC07]
Use Case Name: Login
Description: This use case allows an employee to have access to restricted operations on the Health-Watcher system.
Priority: Essential
Category: Optional
Inputs and pre-conditions: None
Outputs and post-conditions: Password validated by the system
Main flow of events:

1. The employee provides the login and password.
2. The login and password are sent to the server.
3. The system retrieves the employee details using the login as a unique identifier.
4. The system validates the entered password.
5. The result of the login attempt is presented to the employee on their local display.

Extension Points:
E1. Send data over the internet
The *Send data over the internet* extension point occurs before step 2

Logout [UC08]
Use Case Name: Logout
Description: The employee logouts the system
Priority: Essential

Category: Optional
Inputs and pre-conditions: The employee is logged in
Outputs and post-conditions: The employee is not logged in
Main flow of events:

1. The employee clicks on *Logout* button
2. The system logs out the employee

Register tables [UC09]
Use Case Name: Register tables
Description: This use case allows the registration of system tables. The following operations are possible: insert, update, delete, search and print. The available tables include:

- Health unit (unit code, unit description).
- Specialty (code and description).
- Health unit / Specialty (health unit and specialty).
- Employee (login, name and password).
- Type of disease (code, name, description, symptom and duration).
- Symptom (code and description).
- Type of disease / Symptom (type of disease and symptom).

Priority: Essential
Category:
Inputs and pre-conditions: Verified employees
Outputs and post-conditions: Updated data on tables
Main flow of events:

1. The employee chooses the option to register (insert/update) in one of the tables.
2. The employee enters the data.
3. The system saves the data.

Extension Points:
E1. Send data over the internet
The *Send data over the internet* extension point occurs before step 3

Update complaint [UC10]
Use Case Name: Update complaint
Description: This use case allows the state of a complaint to be updated.
Priority: Essential
Category: Mandatory
Inputs and pre-conditions:

- The complaint must be registered and have the OPENED state.
- Verified employee

Outputs and post-conditions: Complaint updated and with state CLOSED.
Main flow of events:

1. The employee selects the update complaint option.
2. The system retrieves the list of all registered complaints.
   (a) The complaint list is populated with general and complaint type specific data.
3. The list of complaints is returned to the employee.
4. The complaints are formatted and presented to the employee on their local display.
5. The employee selects the complaint they wish to update.
6. The complaint unique identifier is sent to the server.
7. The system ensures the complaint data is consistent.
8. The system retrieves the complaint entry.
9. The complaint is returned to the employee.
10. The complaint is formatted and presented to the employee on their local display.
11. The employee enters the conclusion.
12. The conclusion is sent to the server.
13. The complaint status is set to closed; the date the conclusion was entered is set in addition to the employee who dealt with the complaint.
14. The system ensures the complaint is left in a consistent state.
15. The complaint information is updated to store the new information.

Extension Points:
E1. Send data over the internet
The *Send data over the internet* extension point occurs before steps 5 and 12

Register new employee [UC11]
Use Case Name: Register new employee
Description: This use case allows new employees to be registered on the system.
Priority: Essential
Category: Mandatory
Inputs and pre-conditions: Verified employee
Outputs and post-conditions: New employee registered on the system
Main flow of events:

1. The employee selects the insert employee option.
2. The employee provides the following information about the new employee: Name, Login ID, Password (with second password field for confirmation).
3. The employee confirms the operation.
4. The entered data is transmitted to the server.
5. The system verifies the entered data.
6. The system ensures employee data is consistent.
7. The system saves the new employee's data.

Extension Points:
E1. Send data over the internet
The *Send data over the internet* extension point occurs before step 4

Update employee [UC12]
Use Case Name: Update employee
Description: This use case allows of the employee's data to be updated on the system.
Priority: Essential
Category: Mandatory
Inputs and pre-conditions: Verified employee
Outputs and post-conditions: Employee's data updated on the system
Main flow of events:

1. The employee chooses the update employee option.
2. The employee provides the data to be updated: *Name*, *New password* (with second password field for confirmation), *Current password*
3. The employee confirms the update.
4. The entered data is sent to the server.
5. The system verifies the entered data.
6. The system ensures the employee data is consistent.
7. The system stores the updated employee information.

Extension Points:
E1. Send data over the internet
The *Send data over the internet* extension point occurs before step 4

Update health unit [UC13]
Use Case Name: Update health unit
Description: This use case allows the health unit's data to be updated.
Priority: Essential

Category: Mandatory
Inputs and pre-conditions: Verified employee
Outputs and post-conditions: Health unit's data updated on the system.
Main flow of events:

1. The employee chooses the update health unit option.
2. The system retrieves the list of all health units.
3. The list of health units is returned to the employee.
4. The list of health units is formatted and displayed on the employee's local display.
5. The employee selects the health unit to be updated.
6. The unique identifier for the selected health unit is sent to the server.
7. The system ensures the health unit data is consistent.
8. The system retrieves the data for the selected health unit.
9. The data retrieved is returned to the employee.
10. The health unit data is formatted and presented on the employee's local display.
11. The employee alters the necessary data.
12. The updated information is sent to the server.
13. The system ensures the health unit data is left in a consistent state.
14. The system stores the updated health unit information.

Extension Points:
E1. Send data over the internet
The *Send data over the internet* extension point occurs before steps 6 and 12

Change logged employee [UC14]
Use Case Name: Change logged employee
Description: This use case allows the currently logged employee to be changed.
Priority: Essential
Category: Mandatory
Inputs and pre-conditions: Verified employee
Outputs and post-conditions: First employee signed out and new employee logged-in.
Main flow of events:

1. The employee chooses the change logged employee option.
2. The system shows the login screen, and from this point on, the flow will follow the one described in [Login].

Receive alerts via feeds [UC15]
Use Case Name: Receive alerts via feeds
Priority: Desirable
Category: Optional
Inputs and pre-conditions: The citizen is using a browser which is capable of subscribing to feed
Outputs and post-conditions: The citizen receives feeds from Public Health Complaint SPL
Main flow of events:

1. The citizen clicks on the link to subscribe to Public Health Complaint SPL feed
2. The system displays some options for the citizen to choose how to subscribe the feeds
3. The citizen selects one option
4. The system registers the citizen

Publish feeds [UC16]
Use Case Name: Receive alerts via feeds
Priority: Desirable
Category: Optional
Inputs and pre-conditions: Update complaint (UC10)
Outputs and post-conditions: Feeds will be updated

Main flow of events:
This use case is triggered automatically after a complaint is updated by an employee [UC10]. Data related to the updated complaint is used to create a feed entry that will be published.

3.2.2 Nonfunctional Use Cases

Some nonfunctional requirements can be refined as nonfunctional use cases (also known as infrastructure use cases), such as persistence and encryption, which are described in this section. There are also other kinds of nonfunctional requirements that represent system wide qualities and are described simply as declarative statements during requirements [27, Chapter 7.3]. Although exception handling is a nonfunctional requirement and it would be possible to specify it as a nonfunctional use case, we follow the approach proposed by Rubira et al. [43] and specify the exceptional behavior as use case extensions (Section 3.2.3).

Persist Data [NUC01]
Use Case Name: Persist data
Priority: Essential
Category: Kernel
Actor: system
Inputs and pre-conditions: none
Outputs and post-conditions: data is stored on database
Extension flow of events: none

1. The system stores the data entered by the user on the database

Encrypt Data [NUC02]
Use Case Name: Encrypt data
Description: The system should use a security protocol when sending data over the internet. To have access to the complaint registration features, access must be allowed by the access control sub-system [7].
Priority: Important
Category: Mandatory
Actor: system
Inputs and pre-conditions: none
Outputs and post-conditions: data is stored on database
Extension flow of events:
This extension flow occurs at the extension points *Send data over the internet* in the *Query information*, *Login*, *Receive alerts via feeds*, *Find information on the website*, and *Specify complaint* use cases.

1. The system uses a security protocol to send data over the internet

Provide remote access [NUC03]
Use Case Name: Provide remote access
Priority: Essential
Category: Kernel
Actor: system
Inputs and pre-conditions: none
Outputs and post-conditions: data can be send over the Internet
Extension flow of events: This extension flow occurs at the extension points *Send data over the internet* in the *Query information*, *Receive alerts via feeds*, *Find information on the website*, and *Specify complaint* use cases.

1. The system send data over the Internet.

3.2.3 Exception Handling Flow in Use Cases

Query Information Use Case Name: Query Information
Exceptions:

1. **Exception #1:**
   - **Description of Exception Behavior:** Problems on Internet Connection
   - **Number of the Event that Identify the Exception:** 1 and 2
   - **Activity of Exception Handling:** Warn the user about the problem with a message
   - **Group of valid post-conditions after the Exception:** The query won't be executed and the system won't be changed
2. **Exception #2:**

- **Description of Exception Behavior:** A problem occurs retrieving data
- **Number of the Event that Identify the Exception:** 1.x.i, 2.x.iv
- **Activity of Exception Handling:** Warn the user about the problem with a message and the system retrieve the available information
- **Group of valid post-conditions after the Exception:** The query won't be executed and the system won't be changed

3. **Exception #3:**

   - **Description of Exception Behavior:** An invalid complaint code is entered
   - **Number of the Event that Identify the Exception:** 2.c.iii
   - **Activity of Exception Handling:** Warn the user with a message that the number of this complaint is invalid
   - **Group of valid post-conditions after the Exception:** The screen is showing the principal screen of Query Information

4. **Exception #4:**

   - **Description of Exception Behavior:** Consistent data cannot be assured
   - **Number of the Event that Identify the Exception:** 2.x.ii
   - **Activity of Exception Handling:** Warn the user with a message, the system abandon the retrieval
   - **Group of valid post-conditions after the Exception:** The screen is showing the principal screen of Query Information

Complaint Specification, Food Complaint, Animal Complaint, Special Complaint
Use Case Name: Complaint Specification
Exceptions:

1. **Exception #1:**

   - **Description of Exception Behavior:** Problems on Internet Connection
   - **Number of the Event that Identify the Exception:** 5.a
   - **Activity of Exception Handling:** Warn the user about the problem with a message
   - **Group of valid post-conditions after the Exception:** The screen is back to the main screen of complaint specification

2. **Exception #2:**

   - **Description of Exception Behavior:** Invalid data is entered by the user
   - **Number of the Event that Identify the Exception:** 5.b
   - **Activity of Exception Handling:** Warn the user about the problem with a message
   - **Group of valid post-conditions after the Exception:** The screen is back to the main screen of complaint specification

3. **Exception #3:**
    - **Description of Exception Behavior:** A problem occurs storing the complaint
    - **Number of the Event that Identify the Exception:** 5.e-5.g
    - **Activity of Exception Handling:** Warn the user about the problem with a message, and roll-back the complaint entry
    - **Group of valid post-conditions after the Exception:** The screen is back to the main screen of complaint specification
4. **Exception #4:**
    - **Description of Exception Behavior:** Data Consistency cannot be ensured
    - **Number of the Event that Identify the Exception:** 5.h
    - **Activity of Exception Handling:** Warn the user about the problem with a message, and roll-back the complaint entry
    - **Group of valid post-conditions after the Exception:** The screen is back to the main screen of complaint specification

Login Use Case Name: Login
Exceptions:

1. **Exception #1:**
    - **Description of Exception Behavior:** Problems on Internet Connection
    - **Number of the Event that Identify the Exception:** 2
    - **Activity of Exception Handling:** Warn the user about the problem with a message
    - **Group of valid post-conditions after the Exception:** The screen is back to the main screen of Login
2. **Exception #2:**
    - **Description of Exception Behavior:** Problem when the system was retrieving the employee details
    - **Number of the Event that Identify the Exception:** 3
    - **Activity of Exception Handling:** Warn the user about the problem with a message
    - **Group of valid post-conditions after the Exception:** The screen is back to the main screen of Login
3. **Exception #3:**
    - **Description of Exception Behavior:** The system cannot validate the employee (adapted from Healthwatcher - Use cases specification [7])
    - **Number of the Event that Identify the Exception:** 3
    - **Activity of Exception Handling:** Warn the user about the problem with a message

- **Group of valid post-conditions after the Exception:** The screen is back to the main screen of Login

Register Tables Use Case Name: Register Tables
Exceptions:

1. **Exception #1:**
   - **Description of Exception Behavior:** Problems on Internet Connection
   - **Number of the Event that Identify the Exception:** 5.a
   - **Activity of Exception Handling:** Warn the user about the problem with a message
   - **Group of valid post-conditions after the Exception:** The screen is back to the main screen of complaint specification
2. **Exception #2:**
   - **Description of Exception Behavior:** Invalid data is entered by the user
   - **Number of the Event that Identify the Exception:** 5.b
   - **Activity of Exception Handling:** Warn the user about the problem with a message
   - **Group of valid post-conditions after the Exception:** The screen is back to the main screen of complaint specification

Update Complaint Use Case Name: Update Complaint
Exceptions:

1. **Exception #1:**
   - **Description of Exception Behavior:** Error during the retrieve of registered complaints
   - **Number of the Event that Identify the Exception:** 2, 8
   - **Activity of Exception Handling:** Warn the user about the problem with a message
   - **Group of valid post-conditions after the Exception:** The screen is back to the main screen of update complaint
2. **Exception #2:**
   - **Description of Exception Behavior:** Data consistency cannot be ensured
   - **Number of the Event that Identify the Exception:** 7, 14
   - **Activity of Exception Handling:** Warn the user about the problem with a message, and roll-back the complaint entry
   - **Group of valid post-conditions after the Exception:** The screen is back to the main screen of complaint specification
3. **Exception #3:**
   - **Description of Exception Behavior:** Problems on Internet Connection

- **Number of the Event that Identify the Exception:** 3, 6, 9, 12
- **Activity of Exception Handling:** Warn the user about the problem with a message
- **Group of valid post-conditions after the Exception:** The screen is back to the main screen of update complaint

4. **Exception #4:**

- **Description of Exception Behavior:** Error during the update of complaint
- **Number of the Event that Identify the Exception:** 15
- **Activity of Exception Handling:** Warn the user about the problem with a message, and roll-back the complaint entry
- **Group of valid post-conditions after the Exception:** The screen is back to the main screen of update complaint

Register New Employee Use Case Name: Register New Employee
Exceptions:

1. **Exception #1**

- **Description of Exception Behavior:** Incomplete Data Entry (adapted from Healthwatcher - Use cases specification [7])
- **Number of the Event that Identify the Exception:** 2
- **Activity of Exception Handling:** Warn the user that he/she has missing/incorrect data
- **Group of valid post-conditions after the Exception:** The screen is back to the main screen of Register New Employee

2. **Exception #2**

- **Description of Exception Behavior:** Problems on Internet Connection
- **Number of the Event that Identify the Exception:** 4
- **Activity of Exception Handling:** Warn the user about the problem with a message
- **Group of valid post-conditions after the Exception:** The screen is back to the main screen of Register New Employee

3. **Exception #3**

- **Description of Exception Behavior:** The Employee is already entry
- **Number of the Event that Identify the Exception:** 5
- **Activity of Exception Handling:** Warn the user that already has this employee subscribed on the system and abandon the entry
- **Group of valid post-conditions after the Exception:** The screen is back to the main screen of Register New Employee

4. **Exception #4**

- **Description of Exception Behavior:** Data consistency cannot be ensured
- **Number of the Event that Identify the Exception:** 6
- **Activity of Exception Handling:** Warn the user about the problem with a message, and roll-back the complaint entry
- **Group of valid post-conditions after the Exception:** The screen is back to the main screen of Register New Employee

5. **Exception #5**

- **Description of Exception Behavior:** Error when the system was storing the new employee's details
- **Number of the Event that Identify the Exception:** 6
- **Activity of Exception Handling:** Warn the user about the problem with a message, and roll-back the complaint entry
- **Group of valid post-conditions after the Exception:** The screen is back to the main screen of Register New Employee

Update Employee Use Case Name: Update Employee
Exceptions:

1. **Exception #1**

- **Description of Exception Behavior:** Password is missing or invalid (adapted from Healthwatcher - Use cases specification [7])
- **Number of the Event that Identify the Exception:** 2
- **Activity of Exception Handling:** Warn the user that he/she has missing/invalid password
- **Group of valid post-conditions after the Exception:** The screen is back to the main screen of Update Employee

Update Health Unit Use Case Name: Update Health Unit
Exceptions:

1. **Exception #1**

- **Description of Exception Behavior:** Problem when the system was retrieving the information of the health unit
- **Number of the Event that Identify the Exception:** 2, 8
- **Activity of Exception Handling:** Warn the user that he/she has missing/incorrect data
- **Group of valid post-conditions after the Exception:** The screen is back to the main screen of Update Health Unit

2. **Exception #2**

- **Description of Exception Behavior:** Problems on Internet Connection

- **Number of the Event that Identify the Exception:** 3, 6, 9, 12
- **Activity of Exception Handling:** Warn the user about the problem with a message
- **Group of valid post-conditions after the Exception:** The screen is back to the main screen of Update Health Unit

3. **Exception #3**

- **Description of Exception Behavior:** Data consistency cannot be ensured
- **Number of the Event that Identify the Exception:** 7, 13
- **Activity of Exception Handling:** Warn the user about the problem with a message, and roll-back the complaint entry
- **Group of valid post-conditions after the Exception:** The screen is back to the main screen of Update Health Unit

4. **Exception #4**

- **Description of Exception Behavior:** Error when the system was storing the update of the health unit
- **Number of the Event that Identify the Exception:** 14
- **Activity of Exception Handling:** Warn the user about the problem with a message, and roll-back the complaint entry
- **Group of valid post-conditions after the Exception:** The screen is back to the main screen of Update Health Unit

Change Logged Employee Use Case Name: Change Logged Employee

1. **Exception #1**

- **Description of Exception Behavior:** On step 2, in case the password or the employee is not valid, an error message should be shown (adapted from Healthwatcher - Requirements [7]).
- **Number of the Event that Identify the Exception:** 2
- **Activity of Exception Handling:** Warn the user about the problem with a message
- **Group of valid post-conditions after the Exception:** The screen is back to the main screen of Login

Drug Complaint Specification Use Case Name: Drug Complaint Specification
Exceptions:

1. **Exception #1:**

- **Description of Exception Behavior:** Problems on Internet Connection
- **Number of the Event that Identify the Exception:** 2, 5
- **Activity of Exception Handling:** Warn the user about the problem

- **Group of valid post-conditions after the Exception:** The data could not be saved

2. **Exception #2:**

- **Description of Exception Behavior:** Database is not Accessable
- **Number of the Event that Identify the Exception:** 2, 5
- **Activity of Exception Handling:** Warn the user about the problem and ask him/her to access the system in a few minutes
- **Group of valid post-conditions after the Exception:** The data could not be saved

Find Information on Website Use Case Name: Find Information on Website
Exceptions:

1. **Exception #1:**

- **Description of Exception Behavior:** Problems on Internet Connection
- **Number of the Event that Identify the Exception:** 2
- **Activity of Exception Handling:** Warn the user about the problem
- **Group of valid post-conditions after the Exception:** The data can't be showed

2. **Exception #2:**

- **Description of Exception Behavior:** Database is not Accessable
- **Number of the Event that Identify the Exception:** 2
- **Activity of Exception Handling:** Warn the user about the problem and ask him/her to access the system in a few minutes
- **Group of valid post-conditions after the Exception:** The data can't be showed

Receive Alerts via Feeds Use Case Name: Receive Alerts via Feeds
Exceptions:

1. **Exception #1:**

- **Description of Exception Behavior:** Problems on Internet Connection
- **Number of the Event that Identify the Exception:** 2, 4
- **Activity of Exception Handling:** Warn the user about the problem
- **Group of valid post-conditions after the Exception:** The subscribe can't be commited

2. **Exception #2:**

- **Description of Exception Behavior:** Database is not Accessable
- **Number of the Event that Identify the Exception:** 4
- **Activity of Exception Handling:** Warn the user about the problem and ask him/her to access the system in a few minutes
- **Group of valid post-conditions after the Exception:** The subscribe can't be commited

3.2.4 Crosscutting Use Cases

Moreira et al. [37] proposed an approach to identify crosscutting concerns among the quality attributes that affect functional use cases. Araújo and Moreira [11] augmented the identification of crosscutting concerns by using use cases specification to represent functional and nonfunctional requirements. Based on the work of Araújo and Moreira [11], Eler [20, Chapter 4.4.10] proposed guidelines to identify crosscutting use cases based on the use case diagram, which does not necessarily mean that these crosscutting use cases will be implemented by aspects. The criteria to identify crosscutting use cases are the number of relationships with other use cases. The criteria are listed below.

- Use cases included by two or more use cases;
- Use cases that extend two or more use cases;
- Use cases that constrain two or more use cases. The *constrain* relationship between use cases was proposed proposed by Araújo and Moreira [11].

Thus, based on the use case diagram (Figure 5), two crosscutting use cases have been identified. Table 3 presents crosscutting use cases and the criteria applied to identify them.

Use cases	**Criteria**
Encrypt data	It extends two or more use cases
Persist data	It is included by two or more use cases
Handle exception	It extends two or more use cases
Provide remote access	It extends two or more use cases

Table 3: Crosscutting use cases

3.2.5 Relationship between nonfunctional requirements and use cases

Moreira et al. [37] proposed to establish a relationship between use cases and nonfunctional requirements in order to support the identification of crosscutting concerns.

Table 4 supports the understanding of Table 5.

3.2.6 Mapping Features to Use Cases

There is a symbiotic relationship between features and use cases. Feature models focus on specifying the features variability by means of a graphical user-friendly and hierarchical structure. On the other hand, use cases specify the interaction between user and system, and also the system behavior. Thus, feature models support defining the variability of each use case and feature dependencies can be depicted in terms of the dependencies between the use cases [25, Chapter 5.3.2].

Identifier	Use case name	Identifier	Use case name
UC01	Query information	UC02	Specify complaint
UC03	Food complaint specification	UC04	Animal complaint specification
UC05	Drug complaint specification	UC06	Special complaint specification
UC07	Login	UC08	Logout
UC09	Register tables	UC10	Update complaint
UC11	Register new employee	UC12	Update employee
UC13	Update Health Unit	UC14	Change logged employee
UC15	Receive alerts via feeds		
Identifier	**Nonfunctional requirement**	**Identifier**	**Nonfunctional requirement**
NFR01	Usability	NFR02	Availability/EH
NFR03	Response time/Performance	NFR04	Security/Encryption
NFR05	Standard/Compatibility	NFR06	Hardware and Software/Operational Environment
NFR07	Distribution	NFR08	Storage medium/Persistence
NFR09	Concurrency		

Table 4: Use case and nonfunctional requirements identifiers

Use Cases	NFR01	NFR02	NFR03	NFR04	NFR05	NFR06	NFR07	NFR08	NFR09
UC01	✓	✓	✓		✓		✓	✓	✓
UC02	✓	✓	✓	✓	✓		✓	✓	✓
UC03	✓	✓	✓	✓	✓		✓	✓	✓
UC04	✓	✓	✓	✓	✓		✓	✓	✓
UC05	✓	✓	✓	✓	✓		✓	✓	✓
UC06		✓		✓	✓		✓	✓	✓
UC07					✓		✓	✓	✓
UC08	✓	✓					✓		✓
UC09	✓	✓	✓	✓	✓			✓	
UC10		✓			✓			✓	
UC11		✓			✓			✓	
UC12		✓			✓			✓	
UC13		✓			✓			✓	
UC14	✓	✓							
UC15		✓					✓		✓

Table 5: Use cases influenced by nonfunctional requirements

Feature Name	Feature Category	Use case Name	Use case category/ variation point	Variation point name
Public Health Complaint SPL	kernel	Specify complaint	kernel	
Persistence	kernel	Persist data	kernel	
Database	kernel	Persist data	kernel	
MySQL	kernel	Persist data	kernel	
Oracle	kernel	Persist data	kernel	
Computer infrastructure	kernel	Specify complaint	kernel	
Hardware	kernel	Specify complaint	kernel	
2.0GHz, 1GB RAM, netCard 3Com	kernel	Specify complaint	kernel	
Software	kernel	Specify complaint	kernel	
Ubuntu	kernel	Specify complaint	kernel	
Java Servlets	kernel	Provide remote access	kernel	
Java RMI	kernel	Provide remote access	kernel	
Distribution	kernel	Provide remote access	kernel	
Usability	kernel	Specify complaint	kernel	
User interface	kernel	Specify complaint	kernel	
Compatibility	kernel	Specify complaint	kernel	
Standards	kernel	Specify complaint	kernel	

3.3 Identifying and composing crosscutting concerns

Identifying and managing early as apects helps to improve modularity in the requirements and architecture design and to detect conflicting concerns early, when trade-offs can be resolved more economically [12].

Some scattered concerns might be too trivial or heterogeneous to capture separately. For instance, if a different type of auditing is required for every transaction, it's unhelpful to decouple auditing's description from its transaction description. However, if some core concepts related to auditing crosscut the other concerns, those should be collected and their points of impact recorded [12].

We identified crosscutting concerns based on crosscutting use cases (Section 3.2.4), since use cases can represent crosscutting concerns, and on legacy requirements documents [4–7, 44], which have already identified some crosscutting concerns. These legacy documents were revisited in order to keep the consistency with current requirements. Thus, the identified crosscutting concerns are concurrency, distribution, exceptional handling, persistence, encryption, standards, performance, and usability. Note that we use the same name of the features (see Figure 3) in order to list the crosscutting concerns, because a concern is a kind of feature [16, 42].

According to Moreira et al. [38] it is important to identify coarse-grained relationships among crosscutting concerns. It facilitates negotiation and decision-making among stakeholders. Table 6 shows how each croscutting concern affects other crosscutting concerns. The creation of Table 6 was based on use cases specification (Section 3.2) and legacy requirements documents [4–7, 44].

Concerns	Concurrency	Distribution	Exc.Handling	Performance	Persistence	Encryption	Standards	Usability
Concurrency	x		✓	✓	✓			✓
Distribution		x	✓					✓
Exception handling			x					✓
Performance				x	✓			✓
Persistence			✓	✓	x			
Encryption		✓		✓		x		
Standards		✓					x	✓
Usability								x

Table 6: This table shows the influence of one concern on the others

Note that *Security* feature (see Figure 3) is composed by *Encryption* and *Captcha* features. Although some legacy requirements documents define *Security* as a crosscutting concern (e.g. MDSOC [5] and Viewpoints [44]), we believe that only *Encryption* cuts across the system while *Captcha* does not.

3.4 Aspect-oriented Feature Analysis

Since we are using an aspect-oriented approach to perform a feature-oriented reengineering process [29], we use Aspect-oriented Feature View (AOFV) to specify crosscutting features affects other crosscutting and regular (i.e. non-crosscutting) features [48]. Crosscutting features are features that

cut across other features. The AOFV promotes the traceability of both crosscutting concerns and variability from feature model to aspect-oriented PLA model, thus supporting its design and enhancing system's evolvability. The AOFV enables modeling crosscutting concerns as crosscutting features thereby representing their variability. This view is derived from the feature model and its role is not to replace the feature model, but to complement it. Note that the notation is a little bit different from the feature model, since it is not hierarchical. Thus, feature variability must be represented in isolation. For instance, *Encryption* is an alternative feature (see Figure 3), but as the others features of the same group are not represented in the AOFV because they are neither crosscutting features nor crosscut by a feature, the variability of *Encryption* feature is represented by a crossed-circle.

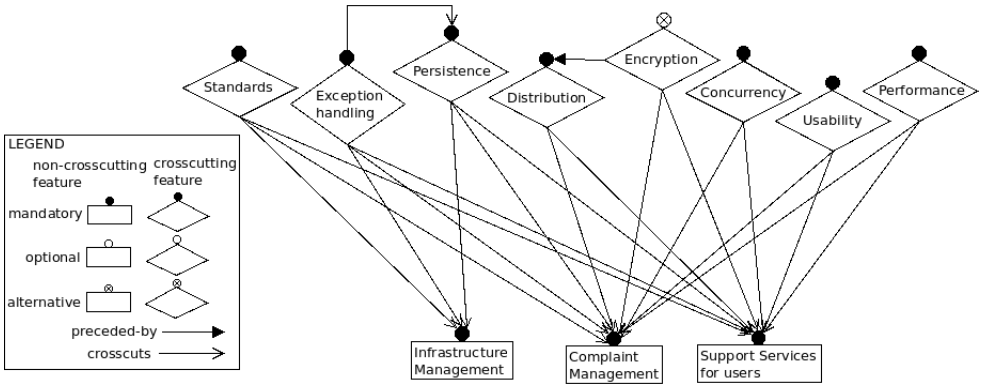

Figure 6: Aspect-oriented feature view

4 Design

Figure 7 shows activities necessary to build the PLA based on analysis documents. In order to create the PLA, the feature diagram endures four transformations, which are based on the Feature-Architecture Mapping (FArM) method [45]. FArM considers only the feature diagram to create the PLA. We extended FArM in order to consider the crosscutting concerns which are represented in the AOFV. Thus, transformations applied to the feature diagram are also applied to the AOFV.

4.1 From feature model to architecture model

Deriving the PLA from use cases induces feature scattering and tangling [45]. The Feature-Architecture Mapping (FArM) method [45] supports the design of PLA based on feature diagram. FArM defines 4 transformations to build the architecture based on feature diagram:

1. **Tranformation 1.** Removing non-architecture related features and Resolving quality features (Section 4.1.1)
2. **Tranformation 2.** Transforming based on architectural requirements (Section 4.1.2)
3. **Tranformation 3.** Transforming based on interacts relations (Section 4.1.3)
4. **Tranformation 4.** Transform based on hierarchy relations (Section 4.1.4)

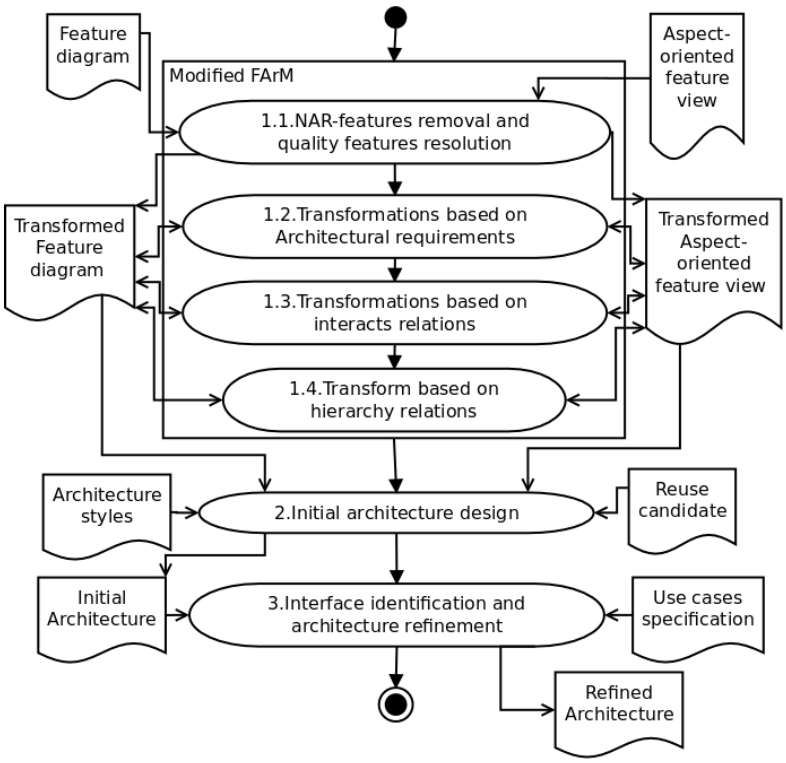

Figure 7: Design activities

4.1.1 Removing non-architecture related features and Resolving quality features

According to van der Linden et al. [36, Chapter 3.1.2], one important issue related to architectures is architectural significant requirements, which encompass two types of requirements: functional requirements and quality requirements. Functional requirements determine *what* is realised and quality requirements determine *how* it is realised [36, Chapter 3.1.2]. Non-architecture related (NAR) features are the opposite of architectural significant requirements.

In the Public Health Complaint SPL, all subfeatures of *Computer infrastructure* (except *Java RMI*, *Java Servlets*, and *Database*) and subfeatures of *Compatibility* are NAR features.

FArM method resolves quality features (i.e. a feature that represents a quality attribute) through the integration with existing functional features. In the Public Health Complaint SPL, the following quality features are integrated with existing functional features: *Persistence*, *Usability*, *Distribution*, *Concurrency*, *Availability*, *Encryption*, and *Performance*. *Persistence* is composed by *Database*, which can be implemented by either *MySQL* or *Oracle*. *Usability* is composed by *User interface*, which is implemented by *Java Servlets*. For instance, *Distribution* can be implemented by both *Java RMI* and *Java Servlets* features. *Java RMI* can also implement *Encryption* feature, a subfeature of *Security*. *Concurrency* is implemented by *Java Synchronization* mechanisms [8]. *Availability* is supported by the implementation of *Exception Handling*.

Performance feature and its subfeature depends on architecture styles [13, Chapter 4.1]. However, we believe that, for Public Health Complaint SPL, the performance requirement (see NFR03) should not be a driving requirement for defining the product line architecture (PLA). Healthwatcher

legacy architecture (Section 2.1) is a layered architecture, which is not performance-friendly architecture style. Furthermore, Healthwatcher implementation [2] does not support any mechanism to improve performance, like the implementation of cache. Figure 13 in Appendix A shows the result of transformation 1, which includes removing NAR features (Section 4.1.1) and resolving quality attributes.

4.1.2 Transforming based on architectural requirements

There may exist architectural requirements that must be satisfied through direct resolution, that is, integrating with existing functional features. As architectural requirements have already been specified as features, no feature has been added.

4.1.3 Transforming based on interacts relations

After transformations described in steps 1 to 3, current feature model contains exclusively functional features. The communication among these features is represented by introducing new features relation, namely the interacts relation [45]. FArM defines the following interacts relation:

- **Type 1.** It connects two features where one features uses the other feature's functionality.
- **Type 2.** It connects two features where the correct operation of one feature alters the behavior of the other feature.
- **Type 3.** It connects two features where the correct operation of one feature contradicts with the correct operation of the other feature.

Before adding interacts relations among features, it is important to check if the relation already exists in the AOFV. If any type of interacts relation has a correspondent in AOFV, it must not be added to the feature model. For instance, *Complaint Management* and its subfeatures uses *Database* feature in order to persist data. According to FArM, an *uses* relation would be added between *Complaint Management* and *Database*. However, in AOFV it is already represented that *Database* (which is the implementation of *Persistence* crosscutting feature) crosscuts *Complaint Management*. Although the semantic of uses and crosscuts relations are different Thus, it is not necessary to add the *uses* relation because it is already represented in the AOFV.

Note that *crosscuts* relations (type 4) modifies features, which is similar to *alters* relation (type 2) defined in FArM method [45]. Whereas the *crosscuts* is a relation between one crosscutting feature and other features (1-n relation), *alters* is a relation between two features (1-1 relation).

After the identification of relations between features, they are transformed based on two criteria:

- Criterion 1. The type of interacts relations.
- Criterion 2. The number of interacts relations.

4.1.4 Transform based on hierarchy relations

In this step, features are grouped to become components. The module hierarchy largely corresponds to the feature hierarchy [30]

1. If a feature does not have any sibling and it is not the root feature, merge the feature with its parent recursively;

2. All parent-features should become components (except the root feature) and their subfeatures should become either subcomponents or classes depending on how complex it is to implement them. Note that this step can be recursively applied.

3. All the relationships between subfeatures must be represented as relationships between components. If a subfeature has become a subcomponent their respective superfeatures of the same level[3].

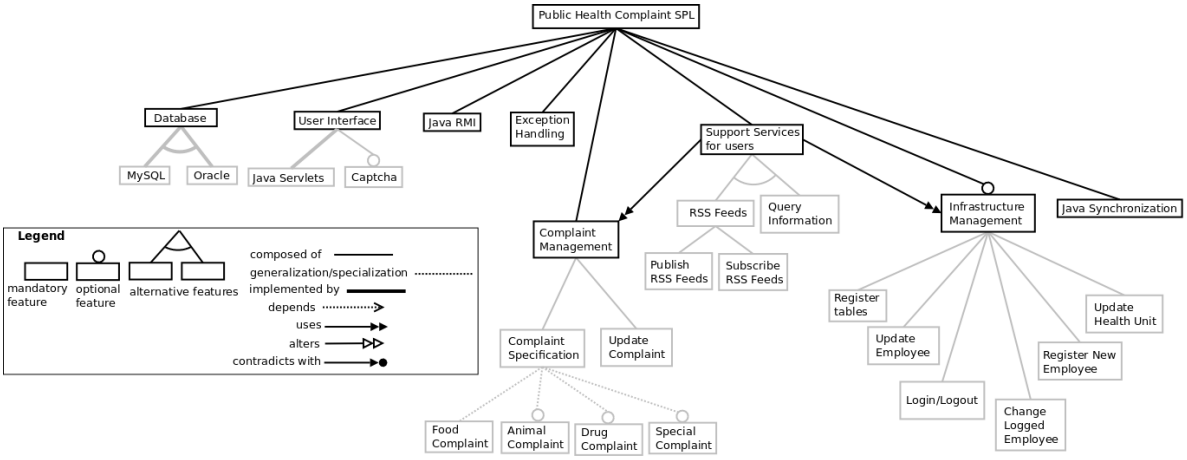

Figure 8: Transformation 4: Grouped features and their relationships

4.1.5 Aspect-oriented feature transformations

The AOFV extends the feature model in order to support reasoning about crosscutting features. Thus, the transformations describe in Section 4.1 must also be applied to the AOFV. For instance, the first transformation of FArM (Section 4.1.1), NAR features are removed from the feature model causing the same effect on AOFV.

4.2 Initial architecture design

After transforming the feature diagram according to FArM (Section 4.1) and propagating these transformations to the AOFV (Section 4.1.5), we merged both diagrams in order to define all relations among features.

4.3 Interface identification and architecture refinement

Since we intend to Public Health Complaint SPL based on components, it is known that components communicate through their interfaces. In this context, interfaces are usually classified in

[3]The level of a feature is given by the shortest distance between the feature and the root feature.

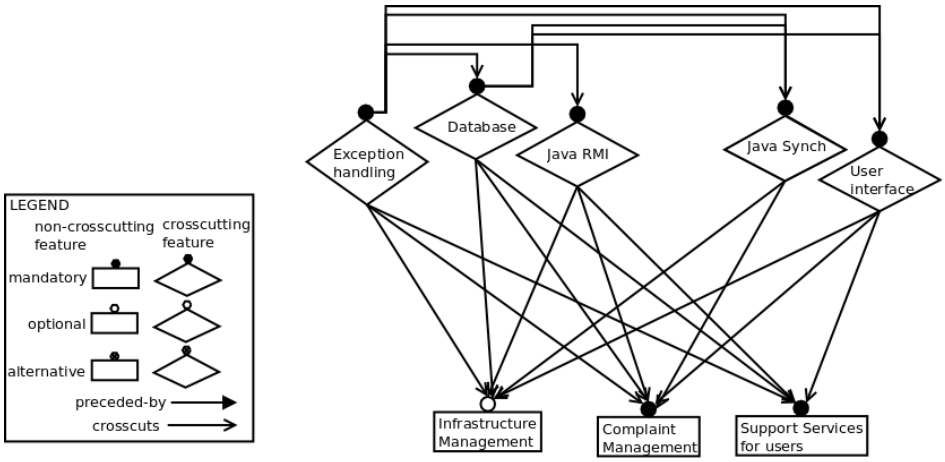

Figure 9: Aspect-oriented feature view after FArM transformations

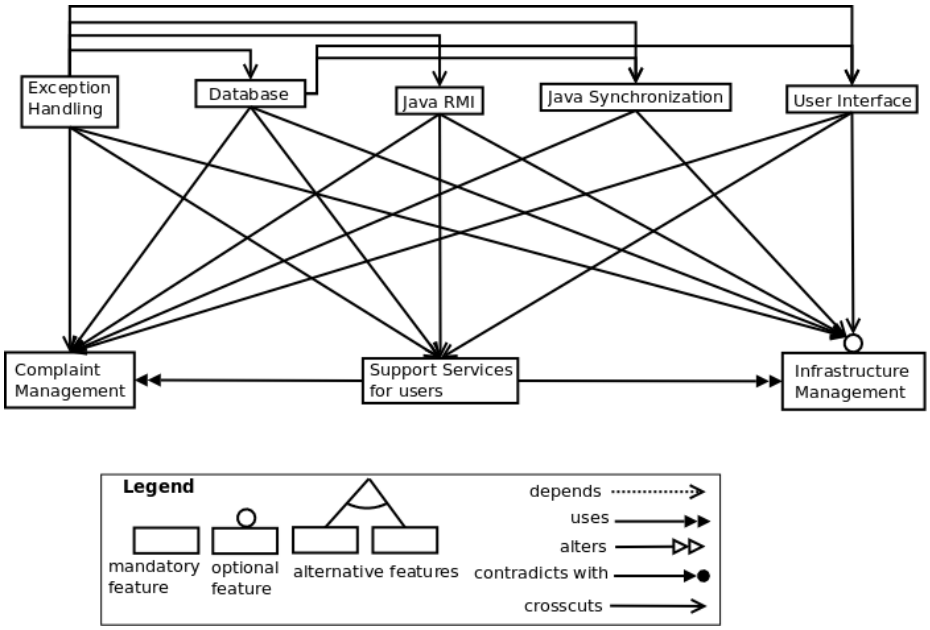

Figure 10: Concept model

provided and required interfaces [46]. However, aspects provide another mechanism to establish the communication among components, because an aspect can capture a method call to an interface, for instance. Some works in the literature propose a new aspect-oriented interface, which Griswold et al. called crosscutting programing interface (XPI) [26], which allows the communication among components. Thus, there are two types of interfaces: regular interfaces which are designed and implemented by object-oriented techniques and XPIs which are designed and implemented by aspect-oriented techniques.

The *crosscuts* relations have at one end an XPI which exposes the joinpoints of the component and at other end an Abstract aspect which has the advices. Abstract aspects are connected to

XPIs through aspect-connectors.

- *crosscuts* relations between components must be mediated by using aspect-orientation, that is, the crosscut component must have an XPI and the component which cuts across other components must have an Abstract aspect.

- *uses*, *alters*, and *contradicts with* relations must be mediated by using object-orientation, that is, regular provided and required interfaces.

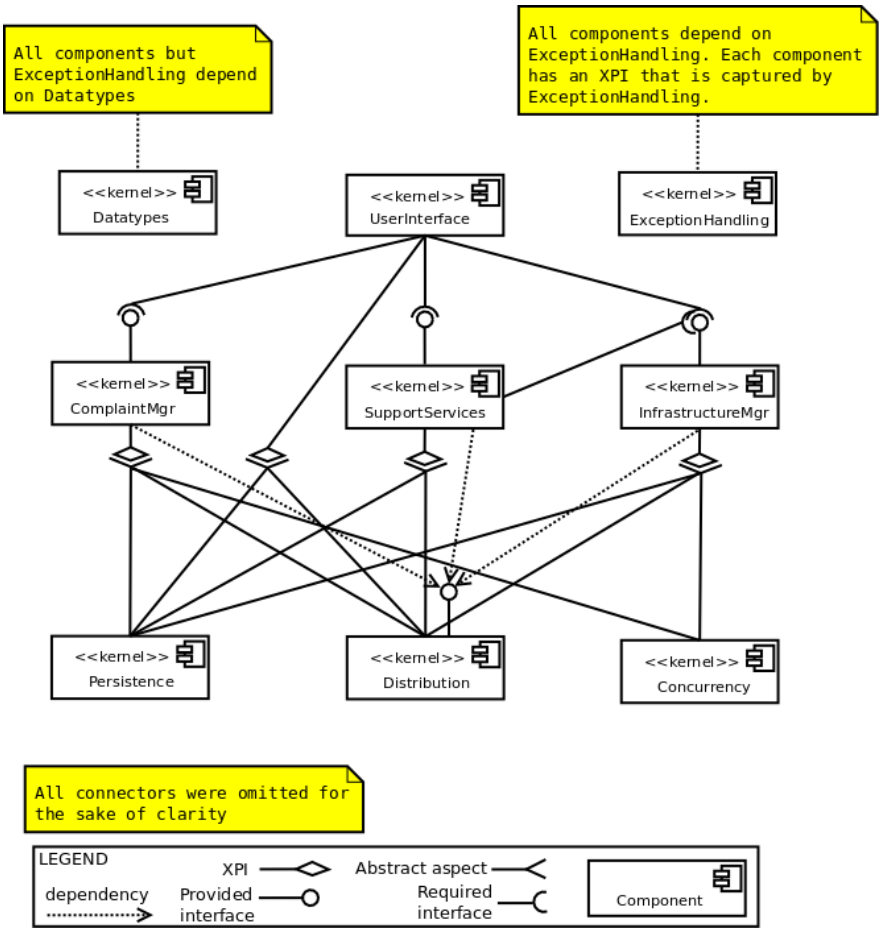

Figure 11: Initial product line architecture specification

Architecture refinement is performed by refining the identified interfaces using UML communication diagrams. Eler and Masiero proposed an approach to refine component interfaces using communication diagrams [19]. We extended their approach to use XPIs to realize use case extensions (see Section 3). The use of XPIs and abstract aspects minimizes the coupling between components [18]. One input for this activity is the *Use case specification*, which allows to reason about the behavior of the architectural components and it specifies what information each interface needs.

5 Provisioning

As shown in Figure 12, in this approach *Provisioning* consists of the following activities: *Evaluation of legacy components*, *Component implementation and refactoring*, and *Connector specification and implementation*. The objective of evaluating legacy components is to increase reuse and reduce costs and time-to-market. The quality of legacy components must be evaluated. Metrics such as coupling, cohesion, complexity, and dependency can be collected by tools [33] and are useful to evaluate the quality of a component.

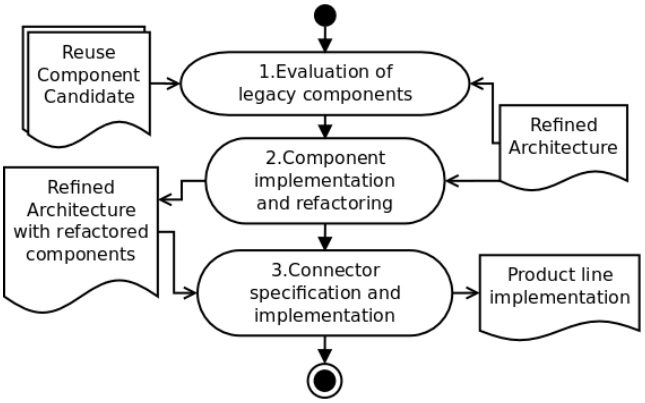

Figure 12: Provisioning activities

5.1 Evaluation of legacy components

The objective of evaluating legacy components is to increase reuse and reduce costs and time-to-market. The quality of legacy components must be evaluated. Metrics such as coupling, cohesion, complexity, and dependency can be collected by tools [33] and are useful to evaluate the quality of a component.

5.2 Component implementation and refactoring

The *Component Implementation and Refactoring* activity is responsible for implementing new features and refactoring components that implement reused features. Some components might be reused without any modifications. Others might be wrapped using a different component model in order to make them compliant to the current PLA.

We managed to reuse components that implement the Healthwatcher features, but we also had to implement features that do not belong to Healthwatcher, such as *RSS Feeds* and *Captcha*. Reused components were refactored to adapt to current PLA. Refactoring is a technique for changing the internal structure of existing programs to make them easier to understand and cheaper to modify, wihtout changing their observable behavior [23]. During refactoring, we avoided changing legacy code. Instead, we wrapped legacy code using COSMOS*-VP component model [18].

5.3 Connectors specification and implementation

Finally, *Connectors specification and implementation* activity has a main role for variability specification and implementation, because connectors are used to materialize variability. Moving the implementation of variability decisions from components to connectors supports PLA evolution [49]. The COSMOS*-VP component implementation model [18] provides guidelines to implement components that handle both regular and aspectual communication and it also supports the design and implementation of such connectors to encapsulate variability decisions.

6 Data collection and analysis

Metrics have been collected based on legacy Healthwatcher (Legacy-HW) application and Healthwatcher product which was derived from Public Health Complaint SPL (PHC-HW).

We have collected 9 metrics, which can be divided in two groups: (i) size metrics, namely, number of modules, number of components, number of connectors, number of features, total number of Lines of Code (LOC), average LOC/module; and (ii) evolution metrics, namely, feature tangling, feature scattering, and average efferent coupling between modules (coupling for short). A module can be either a component or a connector.

Metrics of number of modules, number of components, and number of connectors have been manually collected based on architectures. Legacy-HW is presented in Section 2.1 while PHC-HW is an instantiation of Public Health Complaint PLA presented in Section 4.3. The number of features was collected by counting them in Table 1. We collected the total number of LOC using shell scripts (see Appendix B). Blank lines and comments were also counted.

Separation of concerns is one measurable attribute of evolution [14]. In order to measure separation of concerns, we have used metrics presented by Riebisch and Brcina [41]. Equation 1 shows how to measure the number of modules ($a \in A$) that implements a particular feature $f \in F$. Note that the equation considers the ideal case where no scattering of features exists by the subtraction of 1.

$$sca(f) := |a : f \rightsquigarrow a| - 1 \tag{1}$$

Equation shows how it is measured the scattering of one feature, while Equation 2 shows how it is measured the scattering of all features. The scattering of variation points and tangling of features are measured similarly. The higher is the scattering, the worse is the support for PLA evolution.

$$fsca(F) := \frac{\sum_{f \in F} sca\{f\}}{|F| \cdot |A|}, fsca \in [0, 1) \tag{2}$$

The complete matrix that shows the mapping between features and modules is available on Public Health Complaint website [9].

Efferent coupling refers to the degree of interdependence between parts of a design [15], which means that a high interdependence can harm maintainability. In this study, we have measured the efferent coupling between modules, that is, the number of modules each module knows. The collection of coupling metrics was supported by shell scripts (see Appendix B).

7 Related work

7.1 Features and aspects

Kastner

Lee et al. propose a way to combine feature-oriented analysis (FOA) and aspect-oriented programming (AOP). AOP can be used to implement crosscutting features (i.e. optional and alternative features) . AOP addresses the dependency between features by encapsulating dependencies related to a variable feature into the aspect implementing the variable feature

7.2 Extractives approaches

Lee ICSR'10

Niu RE'08

7.3 Empirical studies

Acknowlegedments

Leonardo P. Tizzei is supported by CAPES - Brazil. Cecília M.F. Rubira is partially supported by Fapesp - Brazil and CNPQ - Brazil. We also thank the Healthwatcher developers: Phil Greenwood, Thiago Bartolomei, Eduardo Magno, Uirá Kulesza, Sérgio Soares, Nelio Cacho, and Marcos Dósea.

A From feature diagram and aspect-oriented feature view to product line architecture

B Scripts

To count the number of LOC, the script described in Figure 17 was executed. It counts the number of LOC, including blank lines and comments, of all files whose extensions are either `.java` (Java files) or `.aj` (AspectJ files). However, the result has a minor error, because the execution of this script adds 3 lines to each file it reads. Thus, the second script, described in Figure 18, counts the number of files read in order to collect the correct value.

Figure 19 shows the script we created to support the collection of coupling metric. The script below was executed to collect data from PHC-HW, but the script to collect data from Legacy-HW was slightly different. The script reads all `.java` and `.aj` files and selects only references to other packages. As in COSMOS*-VP each module has a corresponding package, then it is possible to know the relations between modules analyzing the `import` of all classes and aspects of each module. The result is stored in a file, one for each module. This file is analyzed to find out the coupling between modules.

References

[1] Department of Public Health of Los Angeles county. Available from: URL: http://publichealth.lacounty.gov/phcommon/complaints/phcomp.cfm.

[2] HealthWatcher - A testbed for Aspect Oriented Software Development. Available from: URL: http://www.comp.lancs.ac.uk/~greenwop/tao/.

[3] HealthWatcher - architecture - aosd-europe notation - version 2. Available from: URL: http://www.comp.lancs.ac.uk/~greenwop/tao/architecture.htm.

[4] HealthWatcher - Aspect-oriented Requirements Engineering Model - version 2. Available from: URL: http://www.comp.lancs.ac.uk/~greenwop/tao/requirements.htm.

[5] HealthWatcher - MDSoC - Multi-Dimensional Separation of Concerns - version 2. Available from: URL: http://www.comp.lancs.ac.uk/~greenwop/tao/requirements.htm.

[6] HealthWatcher - requirements alignment document. Available from: URL: http://www.comp.lancs.ac.uk/~greenwop/tao/requirements.htm.

[7] HealthWatcher - Requirements Document - version 2. Available from: URL: http://www.comp.lancs.ac.uk/~greenwop/tao/requirements.htm.

[8] Java SE Development Kit. Available from: URL: http://download.oracle.com/javase/6/docs/.

[9] Public Health Complaint website. Available from: URL: http://www.ic.unicamp.br/~tizzei/phc/sac2012/.

[10] United States Food and Drug Administration (FDA) - Medwatch. Available from: URL: https://www.accessdata.fda.gov/scripts/medwatch/medwatch-online.htm.

[11] Jo ao Araújo and Ana Moreira. An aspectual use case driven approach. In *VIII Jornadas de Ingeniería de Software y Bases de Datos*, Alicante, Spain, November 2003. Thompson (Spain).

[12] Elisa Baniassad, Paul C. Clements, João Araújo, Ana Moreira, Awais Rashid, and Bedir Tekinerdogan. Discovering early aspects. *IEEE Softw.*, 23:61–70, January 2006.

[13] Len Bass, Paul Clements, and Rick Kazman. *Software architecture in practice*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[14] Robert Brcina, Stephan Bode, and Matthias Riebisch. Optimisation process for maintaining evolvability during software evolution. In *ECBS '09: Proceedings of the 2009 16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, pages 196–205, Washington, DC, USA, 2009. IEEE Computer Society.

[15] S.R. Chidamber and C.F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20:476–493, 1994.

[16] Thomas Cottenier, Aswin van den Berg, and Tzilla Elrad. Motorola weavr: Aspect orientation and model-driven engineering. *Journal of Object Technology*, 6(7):51–88, 2007.

[17] M.V. Couto, M.T. Valente, and E. Figueiredo. Extracting software product lines: A case study using conditional compilation. In *15th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 191 –200, 2011.

[18] Marcelo Dias, Leonardo Tizzei, Cecília M. F. Rubira, Alessandro Garcia, and Jaejoon Lee. Leveraging aspect-connectors to improve stability of product line variabilities. In *4th Intl. Workshop on Variability Modelling of Software-intensive Systems*, pages 21–28, 2010.

[19] Marcelo Eler and Paulo Masiero. Aspects as components. In Maurizio Morisio, editor, *Reuse of Off-the-Shelf Components*, volume 4039 of *Lecture Notes in Computer Science*, pages 411–414. Springer Berlin / Heidelberg, 2006.

[20] Marcelo Medeiros Eler. Um método para o desenvolvimento de software baseado em componentes e aspectos. Master's thesis, Instituto de Ciências Matemáticas e de Computação - ICMC/USP, 2006. In Portuguese.

[21] Alessandro Fantechi, Stefania Gnesi, Isabel John, Giuseppe Lami, and Jörg Dörr. Elicitation of use cases for product lines. In Frank van der Linden, editor, *Software Product-Family Engineering*, volume 3014 of *Lecture Notes in Computer Science*, pages 152–167. Springer Berlin / Heidelberg, 2004.

[22] Eduardo Figueiredo, Nelio Cacho, Claudio Sant'Anna, Mario Monteiro, Uira Kulesza, Alessandro Garcia, Sérgio Soares, Fabiano Ferrari, Safoora Khan, Fernando Castor Filho, and Francisco Dantas. Evolving software product lines with aspects: an empirical study on design stability. In *Proceedings of the 30th international conference on Software engineering*, pages 261–270, New York, NY, USA, 2008. ACM.

[23] Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Publishing Company, 1st edition, July 1999.

[24] William Frakes and Carol Terry. Software reuse: metrics and models. *ACM Comput. Surv.*, 28:415–435, June 1996.

[25] Hassan Gomaa. *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004.

[26] William G. Griswold, Kevin Sullivan, Yuanyuan Song, Macneil Shonle, Nishit Tewari, Yuanfang Cai, and Hridesh Rajan. Modular software design with crosscutting interfaces. *IEEE Softw.*, 23(1):51–60, 2006.

[27] Ivar Jacobson and Pan-Wei Ng. *Aspect-Oriented Software Development with Use Cases (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2004.

[28] Isabel John and Jörg Dörr. Elicitation of requirements from user documentation. In *Ninth International Workshop on Requirements Engineering: Foundation for Software Quality. Refsq '03. Klagenfurt/Velden*, 2003.

[29] Kyo Kang, Moonzoo Kim, Jaejoon Lee, and Byungkil Kim. Feature-oriented re-engineering of legacy systems into product line assets : a case study. In *Software Product Line Conference*, 2006.

[30] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, and Moonhang Huh. Form: A feature-oriented reuse method with domain-specific reference architectures. *Ann. Softw. Eng.*, 5:143–168, January 1998.

[31] Charles W. Krueger. Easing the transition to software mass customization. In *PFE '01: Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, pages 282–293, London, UK, 2002. Springer-Verlag.

[32] Axel Anders Kvale, Jingyue Li, and Reidar Conradi. A case study on building cots-based system using aspect-oriented programming. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 1491–1498, New York, NY, USA, 2005. ACM.

[33] Hyesun Lee, Hyunsik Choi, Kyo C. Kang, Dohyung Kim, and Zino Lee. Experience report on using a domain model-based extractive approach to software product line asset development. In *ICSR '09: Proceedings of the 11th International Conference on Software Reuse*, pages 137–149, Berlin, Heidelberg, 2009. Springer-Verlag.

[34] Kwanwoo Lee and Kyo C. Kang. Feature dependency analysis for product line component design. 3107:69–85, 2004.

[35] Kwanwoo Lee, Kyo Chul Kang, and Jaejoon Lee. Concepts and guidelines of feature modeling for product line software engineering. In *Proceedings of the 7th International Conference on Software Reuse: Methods, Techniques, and Tools*, ICSR-7, pages 62–77, London, UK, 2002. Springer-Verlag.

[36] Frank J. van der Linden, Klaus Schmid, and Eelco Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[37] Ana Moreira, João Araújo, and Isabel Brito. Crosscutting quality attributes for requirements engineering. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, SEKE '02, pages 167–174, New York, NY, USA, 2002. ACM.

[38] Ana Moreira, João Araújo, and Awais Rashid. A concern-oriented requirements engineering model. In Oscar Pastor and João Falcão e Cunha, editors, *Advanced Information Systems Engineering*, volume 3520 of *Lecture Notes in Computer Science*, pages 55–100. Springer Berlin / Heidelberg, 2005.

[39] Klaus Pohl, Günter Böckle, and Frank van der Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques.* Springer, Berlin Heidelberg New York, 2005.

[40] Awais Rashid, Ana Moreira, and João Araújo. Modularisation and composition of aspectual requirements. In *AOSD '03: Proceedings of the 2nd international conference on Aspect-oriented software development*, pages 11–20, New York, NY, USA, 2003. ACM.

[41] Matthias Riebisch and Robert Brcina. Optimizing design for variability using traceability links. In *Proceedings of the 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, pages 235–244. IEEE Computer Society, Washington, DC, USA, 2008.

[42] Martin P. Robillard and Gail C. Murphy. Representing concerns in source code. *ACM Trans. Softw. Eng. Methodol.*, 16, February 2007.

[43] Cecília M. F. Rubira, Rogerio de Lemos, Giselle R. M. Ferreira, and Fernando Castor Filho. Exception handling in the development of dependable component-based systems. *Software: Practice and Experience*, 35(3):195–236, 2005.

[44] Americo Sampaio. Analysis of the HealthWatcher system using Viewpoint-based AORE and the EA-Miner tool - version 2. Available from: URL: http://www.comp.lancs.ac.uk/~greenwop/tao/requirements.htm.

[45] Periklis Sochos, Matthias Riebisch, and Ilka Philippow. The feature-architecture mapping (farm) method for feature-oriented development of software product lines. In *ECBS '06: Proceedings of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems*, pages 308–318, Washington, DC, USA, 2006. IEEE Computer Society.

[46] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002.

[47] Leonardo P. Tizzei, Marcelo Dias, Cecília M.F. Rubira, Alessandro Garcia, and Jaejoon Lee. Components meet aspects: Assessing design stability of a software product line. *Information and Software Technology*, 53(2):121 – 136, 2011.

[48] Leonardo P. Tizzei, Jaejoon Lee, and Cecília M.F. Rubira. An aspect-oriented feature view to support feature-oriented reengineering process. In *13th International Workshop on Aspect-Oriented Modeling, co-located with MODELS*, Oslo, Norway, October 2010.

[49] Leonardo P. Tizzei and Cecília M.F. Rubira. Aspect-connectors to support the evolution of component-based product line architectures: a comparative study. In *ECSA'11: European Conference on Software Engineering*. September 2011.

[50] Eddy Truyen, Bo Nørregaard Jørgensen, Wouter Joosen, and Pierre Verbaeten. Aspects for run-time component integration. In *In Proceedings of the ECOOP 2000 Workshop on Aspects and Dimensions of Concerns*, Cannes, France, 2000.

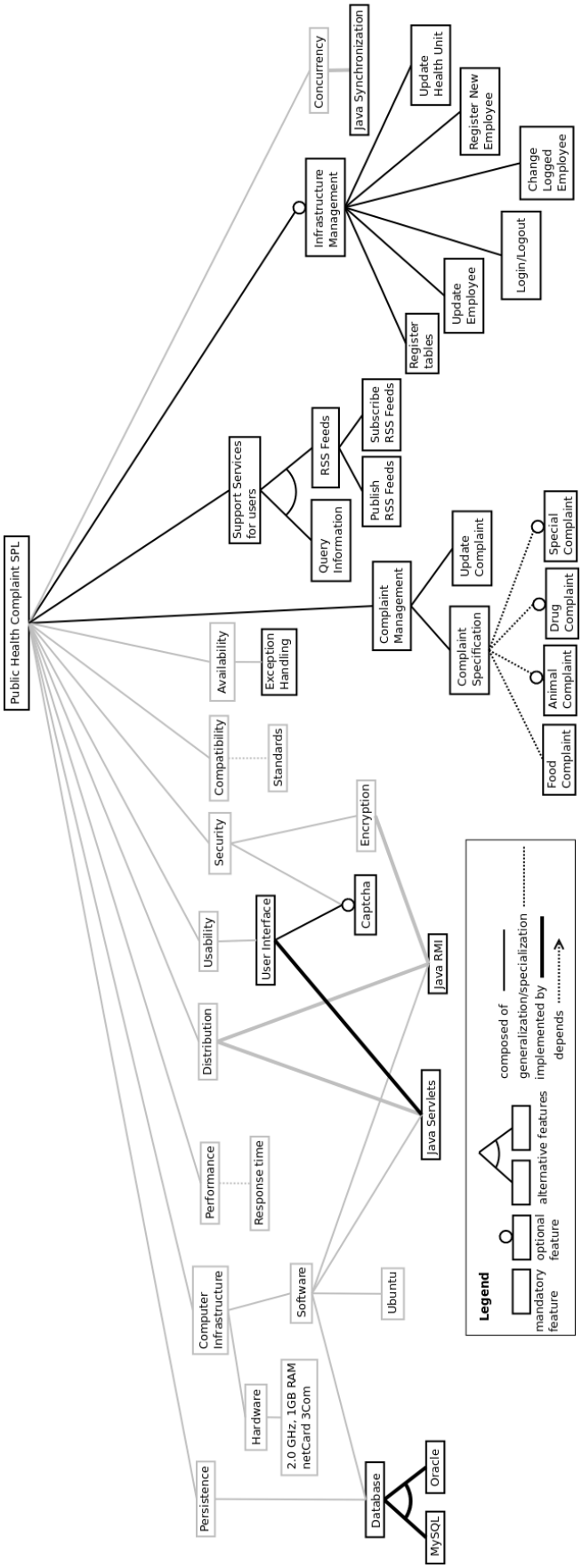

Figure 13: FArM transformation 1

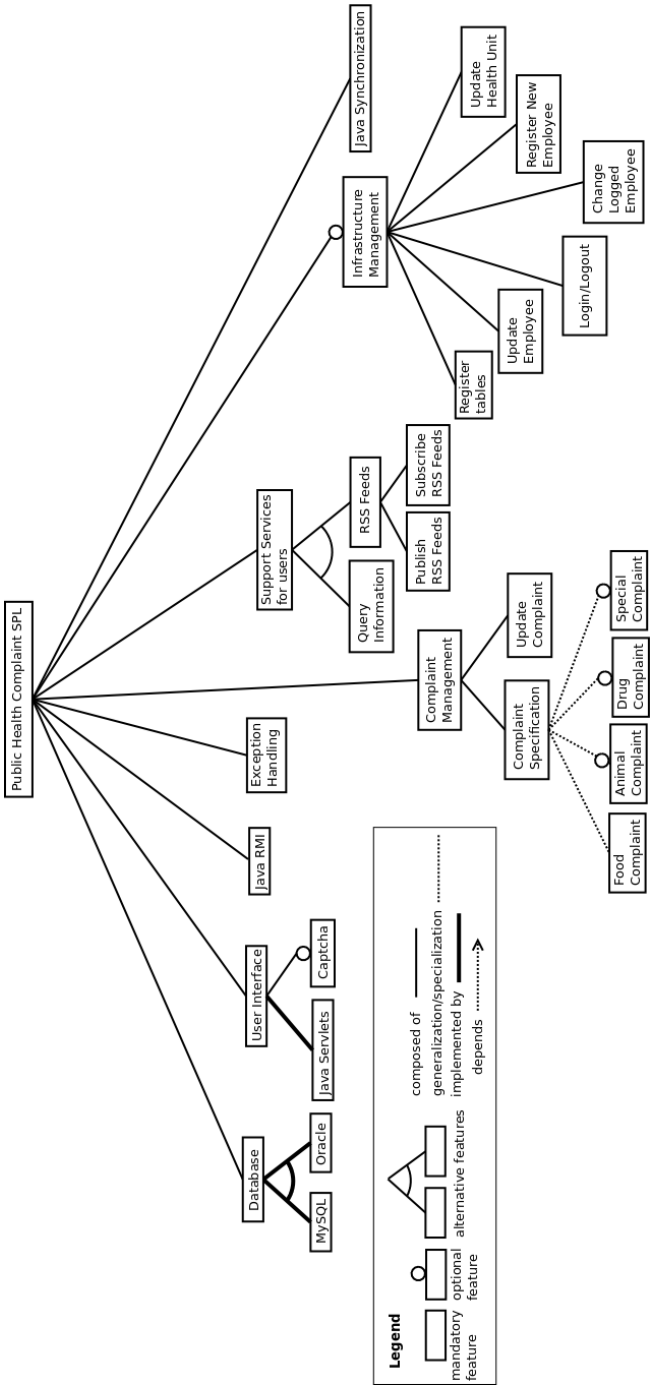

Figure 14: FArM transformation 2

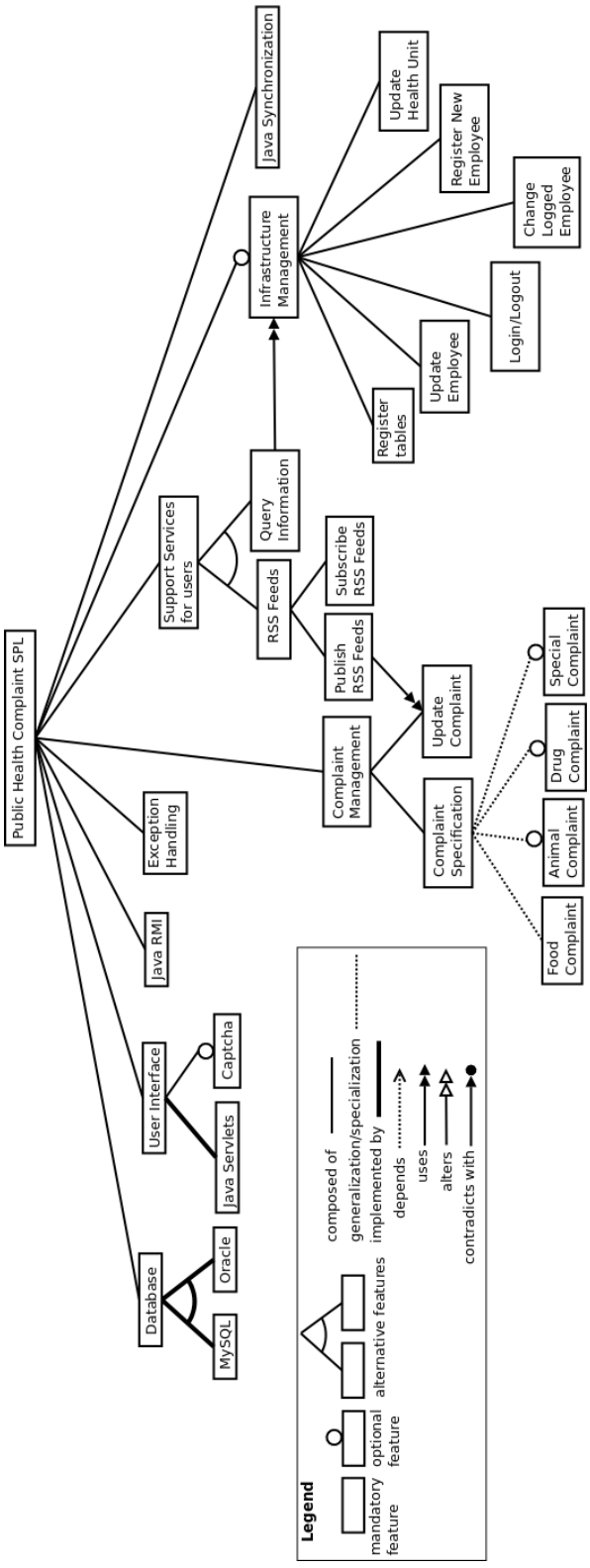

Figure 15: FArM transformation 3

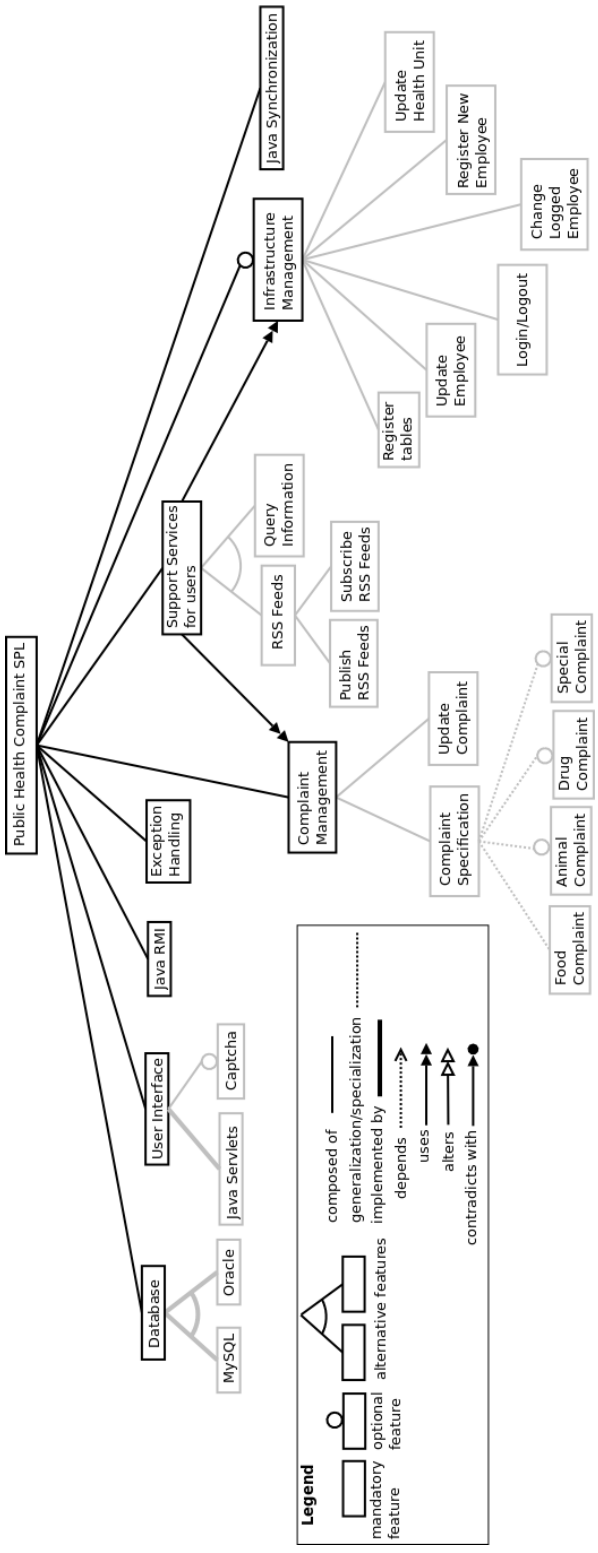

Figure 16: FArM transformation 4

```
find <application source folder> -name "*.java" -or -name "*.aj" | xargs more | wc
```

Figure 17: The above script counts the LOC of all files whose extensions are either .java or .aj

```
find <application source folder> -name "*.java" -or -name "*.aj" | wc
```

Figure 18: The above script counts the number of files whose extensions are either .java or .aj

```
#!/bin/bash/
    for i in $( ls ); do
        find $i -name "*.java" -or -name "*.aj" | xargs more | grep -E publichealthcomplaint.
        | grep -v publichealthcomplaint.$i | sort -nk 2 > $i.txt
    done
```

Figure 19: The above script supports the collection of coupling metric