



Katholieke
Universiteit
Leuven

Department of
Computer Science

PROJECT REPORT

DESIGN OF SOFTWARE SYSTEMS

Daniel Andrés Pérez Pérez
Jasper Marien
Robin Vanden Ecker
Romain Carlier

Contents

1	Design decisions	2
1.1	Iteration 1 & 2: Image Filtering	2
1.2	Iteration 3: Content Filtering	2
1.3	Iteration 4: Content Reporting	2
1.4	Iteration 5: Refactoring	2
1.4.1	org.parosproxy.paros.core.proxy.ProxyThread	2
2	Strengths of the design	3
2.1	Iteration 1 & 2: Image Filtering	3
2.2	Iteration 3: Content Filtering	3
2.3	Iteration 4: Content Reporting	3
2.4	Iteration 5: Refactoring	3
2.4.1	org.parosproxy.paros.core.proxy.ProxyThread	3
3	Weaknesses of the design	3
3.1	Iteration 1 & 2: Image Filtering	3
3.2	Iteration 3: Content Filtering	3
3.3	Iteration 4: Content Reporting	3
3.4	Iteration 5: Refactoring	3
3.4.1	org.parosproxy.paros.core.proxy.ProxyThread	3
4	Future improvements	4
4.1	Iteration 1 & 2: Image Filtering	4
4.2	Iteration 3: Content Filtering	4
4.3	Iteration 4: Content Reporting	4
4.4	Iteration 5: Refactoring	4
5	Design Diagrams	4
6	Overview pyramid and test coverage	4

1 Design decisions

1.1 Iteration 1 & 2: Image Filtering

1.2 Iteration 3: Content Filtering

We separated the distinct responsibilities among several (object) classes. *FilterHttpContent* is the link to the rest of ZAP. It extends *FilterAdaptor* in the parosproxy filter package. Upon receiving an HTTP response, it only verifies the *HttpMessage* has content. Then it instantiates a *FilterApplier* (concrete subclass) and calls one of its filtering methods with 2 parameters. Currently only one is available, but other methods can be added to offer various filtering algorithms. The 2 parameters are the *HttpMessage* and the url of the file containing the filter terms and additional info. The *FilterApplier* then instantiates 2 helpers. One, a *PageContent* represents a given (upon creation) *HttpMessage*'s content.

1.3 Iteration 4: Content Reporting

It was created a new package to be consistent with the naming that zapproxy has. The package *org.zaproxy.zap.extension.imgreport* encapsulates the classes used for the extension. The class *ExtensionImageReport* extends *ExtensionAdaptor* (creates, initializes and hooks a new extension), *XmlReporterExtension* (gets our XML format which will be added to the zapproxy report extension) and *HttpSenderListener* (converts our new extension in an observer object which is able to catch all the *HttpMessages*).

ExtensionImageReport validates whether *HttpMessage* content is an image content, stores *HttpImage* objects and delegate the creation of specific statistics format to the *ImageStatistics* classes.

ImageStatisticsFactory instantiates new concrete classes of *ImageStatistics*.

HttpImage processes *HttpMessage* and returns the corresponding object.

ImageDimensionStatistics is an template class implementation of *ImageStatistics* used by *ImageHeightStatistics*, *ImageSizeStatistics* and *ImageWidthStatistics*. *ImageTypeStatistics* is an implementation of *ImageStatistics* that creates a unique XML format.

This *ExtensionImageReport* was implemented as core functionality since the given XML format to the *ReportExtension* relies on XSL style sheets to add the new information in HTML and MarkDown reports thus those corresponding XSL files were properly updated.

1.4 Iteration 5: Refactoring

1.4.1 org.parosproxy.paros.core.proxy.ProxyThread

Response was created to handle the errors messages while *notification* package was created to delegate all the notification method used in *ProxyThread*. Due to the similarity in the algorithm, it was implemented using an abstract template class and the internal behavior was implemented in the concrete classes.

2 Strengths of the design

one
page

2.1 Iteration 1 & 2: Image Filtering

2.2 Iteration 3: Content Filtering

2.3 Iteration 4: Content Reporting

The extension is encapsulated in its package and relies in the interfaces provides by zaproxy. Due to the Strategy pattern applied for the image statistics, developers can create new concrete classes either using the template class or implementing a new one. Developers can select specific image statistics types via the *ImageStatisticsFactory*.

2.4 Iteration 5: Refactoring

2.4.1 org.parosproxy.paros.core.proxy.ProxyThread

Due to the template class *ProxyListenerNotifier*, developers can create new notification method using the concrete class without affecting the behavior of the others. The *notification* package can also be reused in other parts of the code since it does not depends on *ProxyThread*.

3 Weaknesses of the design

one
page

3.1 Iteration 1 & 2: Image Filtering

3.2 Iteration 3: Content Filtering

3.3 Iteration 4: Content Reporting

The extension can add the new images statistics to the XML report in a straightforward way but it is not the case for HTML and Markdown reports which are highly coupled to the XSL files. Whenever new XML image statistics format is created in the Image-Extension, the XSL files must be modified; the main issue is that those classes/files are not even directly related, making difficult to convert this *ImageExtension* into an add-on plugin.

3.4 Iteration 5: Refactoring

3.4.1 org.parosproxy.paros.core.proxy.ProxyThread

ProxyThread still has to create the concrete notification classes and stores them in a data structure. Hence the responsibilities were turned from calling internal methods to managing *ProxyListenerNotifier* classes.

4 Future improvements

one
page

4.1 Iteration 1 & 2: Image Filtering

4.2 Iteration 3: Content Filtering

4.3 Iteration 4: Content Reporting

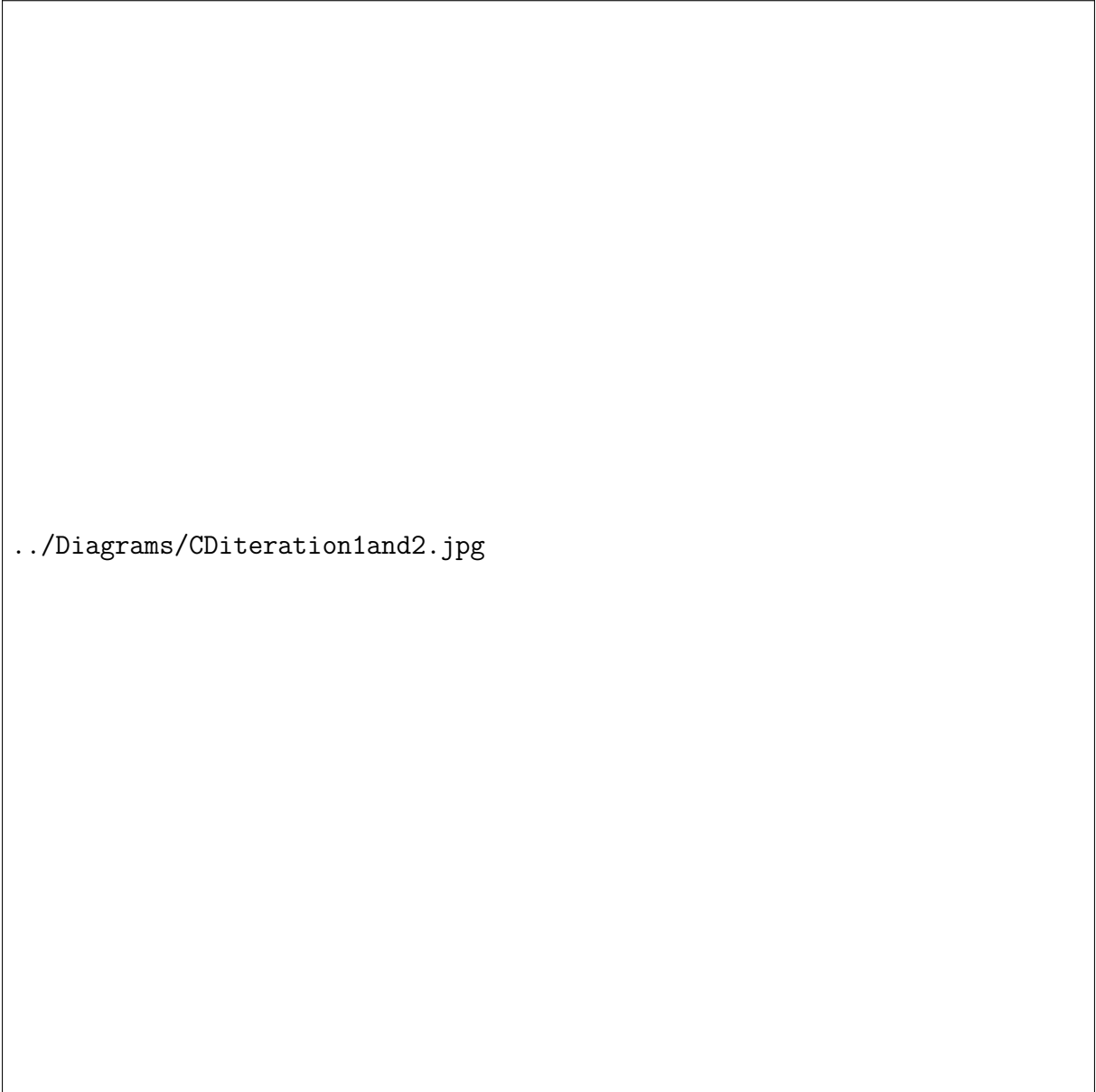
The current implementation does not allow the final user to select the specific image statistic type in the report. Next improvement considers a GUI implementation using the *hook* abstract method provide by *ExtensionAdaptor*. *ImageStatisticsFactory* can be adapted to add those responsibilities: keep tracking the final user image statistic type selections and instantiating/removing the corresponding *ImageStatistics* classes in runtime.

4.4 Iteration 5: Refactoring

5 Design Diagrams

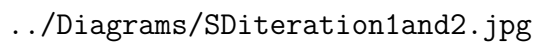
6 Overview pyramid and test coverage

one
page



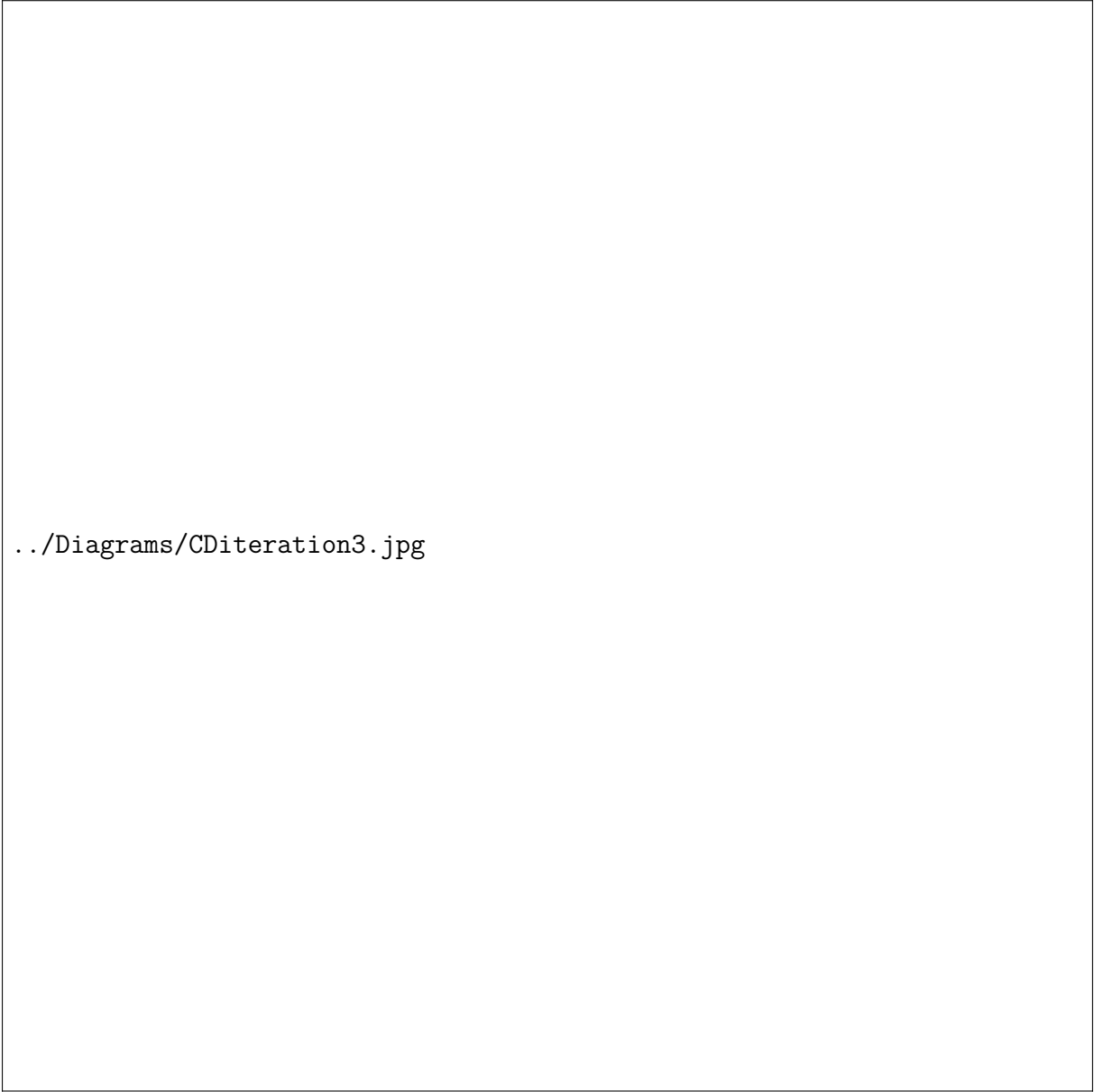
../Diagrams/CDiteration1and2.jpg

Figure 1: Class diagram iteration 1 & 2

A large rectangular box containing a broken image link.

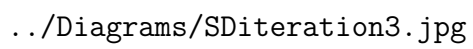
../Diagrams/SDiteration1and2.jpg

Figure 2: Sequence diagram of iteration 1 & 2



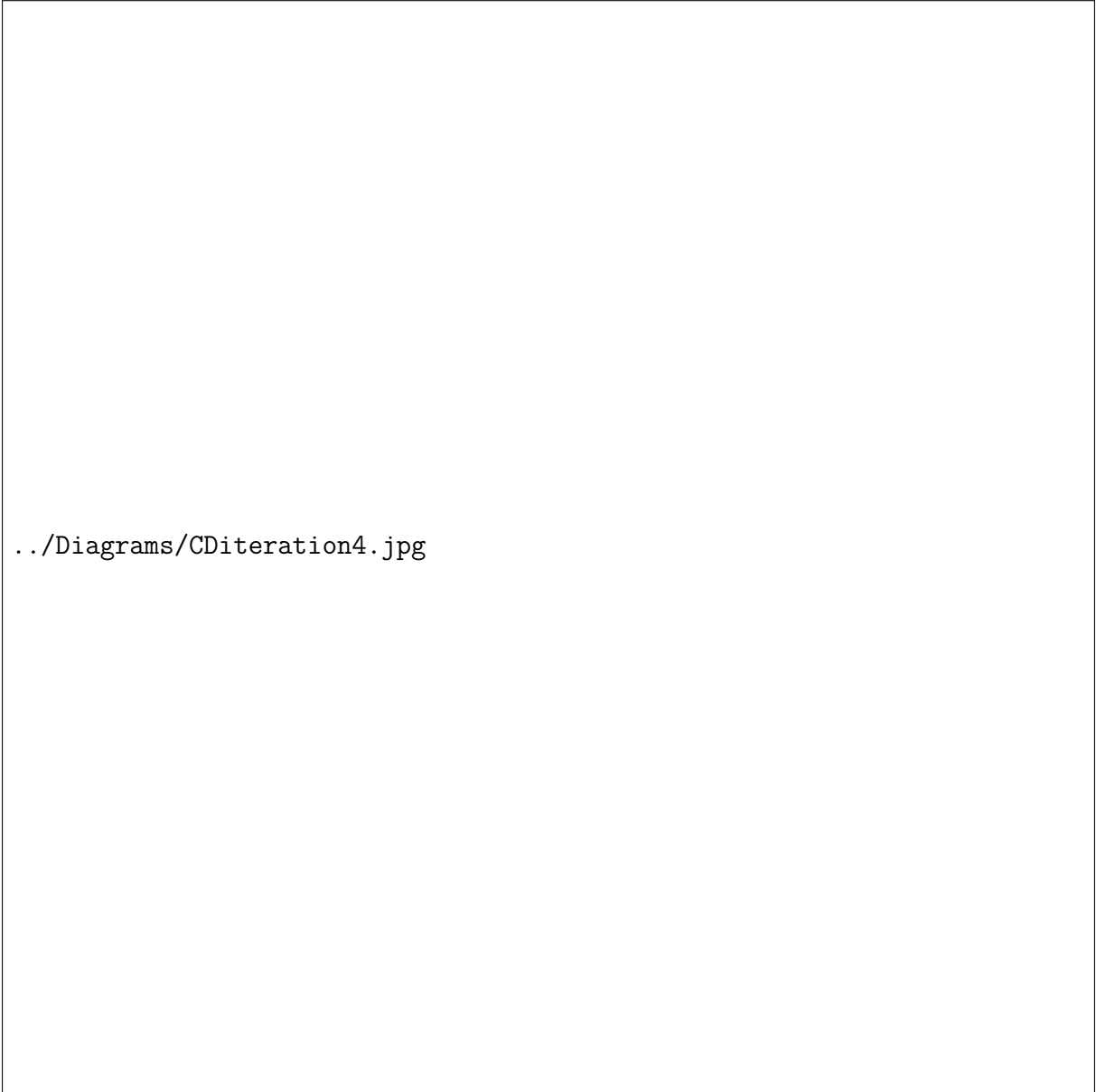
../Diagrams/CDiteration3.jpg

Figure 3: Class diagram of iteration 3

A large rectangular box containing a reference to a sequence diagram file.

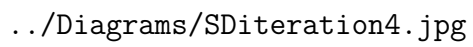
../Diagrams/SDiteration3.jpg

Figure 4: Sequence diagram of iteration 3



../Diagrams/CDiteration4.jpg

Figure 5: Class diagram of iteration 4

A large rectangular box containing a sequence diagram for iteration 4. The diagram is not visible, only the placeholder text is shown.

../Diagrams/SDiteration4.jpg

Figure 6: Sequence diagram of iteration 4