Balzano Informatik AG
Zwicky-Platz 4, 8304 Wallisellen

www.balzano.net
info@balzano.ch

IPA-DOCUMENTATION 2021

IMPLEMENT A DICOM CLIENT CONTEXT CLASS FOR UNIT TESTING

Author	Gabriel Schafflützel
Responsible specialist	Enes Mujak
Main expert	Melanie Meneghini
Secondary expert	Johannes Biederstädt
Vocational trainer	Rolf Ryser

CONTENTS

Part 1 7
1 Preface 7
2 Work Environment and Preparations 8
2.1 Initial situation 8
2.2 Detailed Work Instructions 8
2.3 Project Organization 10
2.4 Means and Methods 11
2.5 Prior Knowledge 11
2.6 Preliminary Work 11
2.7 New Learning Content 11
2.8 Work In The Last 6 Months 11
2.9 Organization of The Work Results 11
3 Timetable 12
4 Daily Work Journal 13
4.1 7th of April 2021 13
4.1.1 Daily Meeting with Responsible specialist 13
4.1.2 Completed Work 13
4.1.3 Achievements 13
4.1.4 Problems 13
4.1.5 Assistance 13
4.1.6 Conclusion 13
4.2 8th of April 2021 14
4.2.1 Daily Meeting with Responsible specialist 14
4.2.2 Completed Work 14
4.2.3 Achievements 14
4.2.4 Problems 14
4.2.5 Assistance 14
4.2.6 Conclusion 14
4.3 9th of April 2021 15
4.3.1 Daily Meeting with Responsible specialist 15
4.3.2 Completed Work 15
4.3.3 Achievements 15
4.3.4 Problems 15
4.3.5 Assistance 15
4.3.6 Conclusion 15
4.4 12th of April 2021 16
4.4.1 Daily Meeting with Responsible specialist 16

4.4.2 Completed Work 16
4.4.3 Achievements 16
4.4.4 Problems 16
4.4.5 Assistance 16
4.4.6 Conclusion 16
4.5 13th of April 2021 17
4.5.1 Daily Meeting with Responsible specialist 17
4.5.2 Completed Work 17
4.5.3 Achievements 17
4.5.4 Problems 17
4.5.5 Assistance 17
4.5.6 Conclusion 17
4.6 14th of April 2021 18
4.6.1 Daily Meeting with Responsible specialist 18
4.6.2 Completed Work 18
4.6.3 Achievements 18
4.6.4 Problems 18
4.6.5 Assistance 18
4.6.6 Conclusion 18
4.7 15th of April 2021 19
4.7.1 Daily Meeting with Responsible specialist 19
4.7.2 Completed Work 19
4.7.3 Achievements 19
4.7.4 Problems 19
4.7.5 Assistance 19
4.7.6 Conclusion 19
4.8 16th of April 2021 20
4.8.1 Daily Meeting with Responsible specialist 20
4.8.2 Completed Work 20
4.8.3 Achievements 20
4.8.4 Problems 20
4.8.5 Assistance 20
4.8.6 Conclusion 20
4.9 19th of April 2021 21
4.9.1 Daily Meeting with Responsible specialist 21
4.9.2 Completed Work 21
4.9.3 Achievements 21
4.9.4 Problems 21

4.9.5 Assistance 21
4.9.6 Conclusion 21
4.10 20th of April 2021 22
4.10.1 Daily Meeting with Responsible specialist 22
4.10.2 Completed Work 22
4.10.3 Achievements 22
4.10.4 Problems 22
4.10.5 Assistance 22
4.10.6 Conclusion 22
4.11 21th of April 2021 23
4.11.1 Daily Meeting with Responsible specialist 23
4.11.2 Completed Work 23
4.11.3 Achievements 23
4.11.4 Problems 23
4.11.5 Assistance 23
4.11.6 Conclusion 23
Part 2 24
5 Summary 24
5.1 Remark 24
5.2 Initial situation 24
5.3 Implementation 24
5.4 Outcome 24
6 Informieren – Inform 25
6.1 Dicom ..... **Error! Bookmark not defined.**
6.1.1 General 25
6.1.2 Networking 25
6.2 WRAPPING 26
6.2.1 General 26
7 Planen – Plan 28
7.1 Functional Requirements 28
7.1.1 Requests 28
7.1.2 Wrapping 28
7.1.3 Modified methods 28
7.2 Non-Functional Requirements 28
7.2.1 Exception handling 28
7.2.2 Clean Code 28
7.3 Test Concept 28
7.3.1 Test Objectives 28

7.3.2 Test Strategy and Test Level 30
7.3.3 Test Environment 30
7.3.4 What won’t be tested 30
7.3.5 Test Objects 30
7.3.6 Test Types 31
7.3.7 Overview of test cases 31
7.3.8 Test Case Descriptions 32
8 Entscheiden – Decide 39
8.1 How many energy drinks should I drink per day? 39
8.2 How should the existing services be modified? 39
8.2.1 If there were no Dicom Client Context 39
8.2.2 Adjustments using the Dicom client context 43
8.2.3 The thing you might have noticed 44
8.2.4 The other problem, which the context could solve 45
8.2.5 The question of the state 47
8.2.6 Which Parameters does the context need 49
9 Realisieren – Implement 50
9.1 Preface 50
9.2 DicomClientContext.DicomContext 50
9.2.1 CreateClient method 51
9.3 DicomGenerator.PACSTarget 52
9.3.1 Constructor 52
9.4 DicomGenerator.PACSTargetTest 53
9.4.1 SaveTest 53
9.4.2 SaveDatasetsIsNullTest 54
9.4.3 SaveDatasetsCountEqualsRequestCountTest 54
9.5 DicomToPacsLoader.DicomToPacsLoader 55
9.6 DicomTopacsloader.Topacsloadertest 55
9.7 ReportSender.pacstarget 56
9.8 ReportsSender.Test 56
9.8.1 Constructor and dispose 56
9.9 IDicomClient 57
9.10 PlayGround.Dicomfacade 58
9.11 Playground.Playgroundclient 60
9.12 Playground.Program 60
10 Kontrollieren – Control 62
10.1 Planned tests 62
11 Auswerten – Evaluate 63

11.1 Dicomgenerator 63
11.2 Jobreceiver 63
11.3 DicomTopacsloader 64
11.4 Reportsender 65
11.5 Unit tests 65
Appendix 66
12 Glossary 66
13 Table of Figures 67
14 Bibliography 68
15 Code 69
15.1 Dicomclientcontext.IDicomContext 69
15.2 DicomClientcontext.dicomcontext 73
15.3 Playground.program 80
15.4 playground.playgroundclient 82
15.5 playground.playgroundserver 83
15.6 Playground.constants 86
15.7 Playground.idicomfacade 87
15.8 playground.dicomfacade 88
15.9 Dicomgenerator.pacstarget 91
15.10 Dicomgenerator.pacstargettest 93
15.11 dicomtopacsloader.dicomtopacsloader 95
15.12 dicomtopacsloader.topacsloadertest 100
15.13 jobreceiver.pacstarget 103
15.14 Jobreceiver.pacstargettest 106
15.15 reportsender.pacstarget 108
15.16 Reportsender.reportsendertest 111
16 Git versioning 113

PART 1

1 PREFACE

This IPA documentation was written by Gabriel Schafflützel. The IPA was initiated on the 6th of April and completed on the 20th of April. Some preliminary work such as this preface and the document structure preparation was done as early as the 31st of March. The whole IPA was done in home office, due to covid measures.

The first part of this documentation covers the tools I will use to complete all the tasks of the IPA and the prior knowledge I possess relevant to the IPA. I will also keep a daily journal in which I document my progress, the problems that arise and the solutions I will hopefully find to those problems. The focus of the first part is the workflow.

In the second part of this documentation, I will show in depth how the Dicom client context operates, how the implementation of said context works and how the unit tests to test the methods with the newly implemented context function. The focus of the second part is to explain the project to the reader.

The implementation of the IPA will be organized using the IPERKA method. Each letter in the word IPERKA stand for a specific step in the process:

Letter	German Word	English translation
I	Informieren	Inform
P	Planen	Plan
E	Entscheiden	Decide
R	Realisieren	Implement
K	Kontrollieren	Check
A	Auswerten	Evaluate

Table 1: IPERKA

2 WORK ENVIRONMENT AND PREPARATIONS

2.1 INITIAL SITUATION

The ScanDiags software service provides AI (Artificial Intelligence) for augmented diagnosis from musculoskeletal MRI (Magnetic Resonance Imaging). The service acts as a Dicom node (Digital Imaging and Communications in Medicine, a widely used protocol in medicine) that can be accessed by other Dicom nodes, like a PACS (Picture Archiving and Communication System, a system in which radiologists manage images). Up until now we cannot unit test our Dicom node fully since we cannot mock the [Dicom.Network.Client.DicomClient] class inside a unit test. Using a real Dicom client does not make sense in a unit test since such a test environment depends on many other components working together.

Framework used for development:

1. The services that shall be unit tested are implemented as worker processes using the [Microsoft.NET.Sdk.Worker] Sdk of the .Net Core 3.1 infrastructure in the programming lanuage C#.
2. The Dicom libraries are available on the NUGET package manager for .NET under https://www.nuget.org/packages?q=fo-Dicom.
3. The test environment is using the packages [Microsoft.NET.Test.Sdk], [xunit], [xunit.runner.visualstudio] and [Moq], all available on the NUGET package manager. The aim of the IPA is to implement a Dicom client context class to make unit testing possible. In addition, unit tests are to be created that use the Dicom client context class.

2.2 DETAILED WORK INSTRUCTIONS

1. Implement a new .Net Core 3.1 library named [Balzano.Common.DicomContext] using the SDK [Microsoft.NET.Sdk]. This library shall contain the following class and interface:

1.1. [Common.IDicomContext] The interface of the [DicomContext] class exposes

1.1.1. a wrapper method to each public method of the [Dicom.Network.Client.DicomClient] class.

1.1.2. a wrapper event to each public event of the [Dicom.Network.Client.DicomClient] class.

1.1.3. a wrapper property to each public property of the [Dicom.Network.Client.DicomClient] class.

1.2. [Common.DicomContext] The implementation of the [IDicomContext] interface:

1.2.1. It must contain a reference to a [Dicom.Network.Client.DicomClient] class. This reference must be initialized in the constructor of the [DicomContext].

1.2.2. All public methods of the [Dicom.Network.Client.DicomClient] class shall be implemented as wrapper to the [DicomClient] reference.

1.2.3. If necessary, private helper methods shall be implemented to expose the wrapped [DicomClient] methods.

2. Implement a new .Net Core 3.1 library named [Balzano.Common.DicomContext.Test] using the [Microsoft.NET.Sdk] SDK, the [Microsoft.NET.Test.Sdk], [xunit], [xunit.runner.visualstudio] and [Moq] packages, as well as the [Balzano.Common.DicomContext] library. This library serves as the unit test solution for the [Balzano.Common.DicomContext] library and shall test each method of the [Common.IDicomContext] interface.

2.1. Test each wrapped method of the [Dicom.Network.Client.DicomClient] class with:

2.1.1. a positive test with straight method parameters.

2.1.2. a negative test with [null] value method parameters.

2.1.3. parameter edge cases if possible.

3. Modify the service classes in a separate git branch, to use the implementation:
   - DicomGenerator.PacsTarget
   - DicomToPACSLoader.DicomToPACSLoader
   - JobReceiver.PacsTarget
   - ReportSender.PacsTarget

3.1. Change the implementation of relevant methods to use the interface as a parameter.
3.2. Describe the reason why this is good practice.
3.3. Write the unit tests for the modified methods.
4. Documentation
4.1. Document the [Balzano.Common.DicomContext] library in the code.
4.2. Document the [Balzano.Common.DicomContext.Test] test library in the code.
4.3. Document the solution as part of the IPA.
5. Code Style
5.1. Code must be formatted according to the file [210301_DotNet_Code_Style_Rules.md].
6. Definition of done
6.1. All methods are fully implemented.
6.2. The code fulfills the code quality guidelines described in 5.
6.3. The code is well documented. The rules are part of the guidelines described in 5.
7. Testing
7.1. Tests are already part of the tasks, see 2. and 3.3.
7.2. Tests must be reliable and reproducible.
7.3. Tests must be described, including the expected results.

2.3 PROJECT ORGANIZATION

Company:

Balzano Informatik AG
Zwicky-Platz 4, 8304 Wallisellen
info@balzano.net

Place of execution:

Home office

Candidate:

Gabriel Schafflützel
Walderstrasse 42, 8630 Rüti
076 813 29 98

Vocational trainer:

Rolf Ryser
rolf.ryser@wiss.ch
058 404 42 69

Responsible specialist:

Enes Mujak
enesm@balzano.net
0795458747

Main expert:

Melanie Meneghini
melipfister72@gmail.com
0764355635

Secondary expert:

Johannes Biederstädt
jonnybie@gmail.com
0041 76 419 40 42

2.4 MEANS AND METHODS

The following instruments are used for development work: Visual Studio Code and GIT.

2.5 PRIOR KNOWLEDGE

The candidate has used all relevant technologies and instruments in other projects.

2.6 PRELIMINARY WORK

No preparation work is necessary.

2.7 NEW LEARNING CONTENT

There is no entirely new learning content.

2.8 WORK IN THE LAST 6 MONTHS

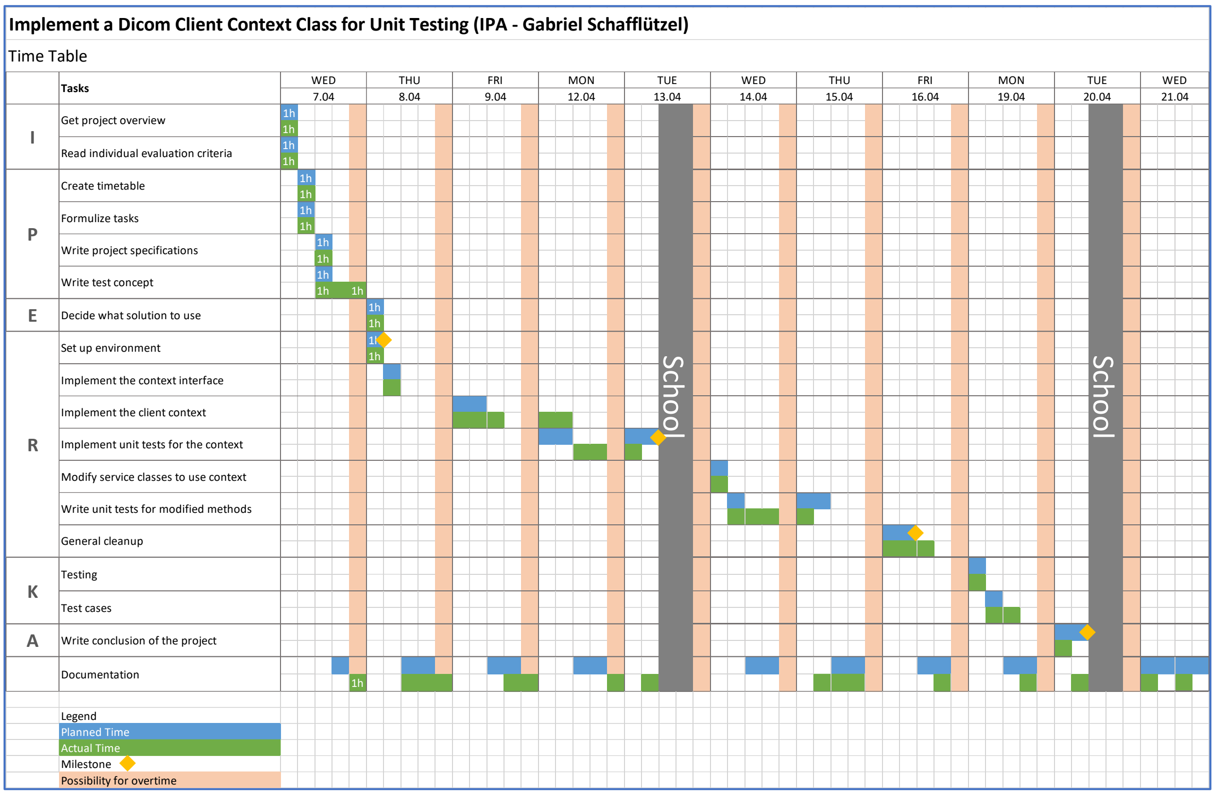
The candidate worked on User Stories with .Net Core 3.1.

2.9 ORGANIZATION OF THE WORK RESULTS

The IPA documentation as well as the actual implementation is stored on a Git repository provided by the company's Microsoft Azure account. This enables examining the progress by looking at the different versions of the IPA. A push will be done at least once every day during the IPA.

The company has multiple such Git repositories. The one the IPA gets implemented in is the *services* repository, which is used to store the code of every service relevant to the company. The IPA and its documentation are on the 11989 Git-branch.

3 TIMETABLE


Implement a Dicom Client Context Class for Unit Testing (IPA - Gabriel Schafflützel)
Time Table
Tasks
WED 7.04
THU 8.04
FRI 9.04
MON 12.04
TUE 13.04
WED 14.04
THU 15.04
FRI 16.04
MON 19.04
TUE 20.04
WED 21.04
I
Get project overview
Read individual evaluation criteria
P
Create timetable
Formulize tasks
Write project specifications
Write test concept
E
Decide what solution to use
R
Set up environment
Implement the context interface
Implement the client context
Implement unit tests for the context
Modify service classes to use context
Write unit tests for modified methods
General cleanup
K
Testing
Test cases
A
Write conclusion of the project
Documentation
1h
School
School
Legend
Planned Time
Actual Time
Milestone
Possibility for overtime

4 DAILY WORK JOURNAL

4.1 7TH OF APRIL 2021

4.1.1 DAILY MEETING WITH RESPONSIBLE SPECIALIST

In the daily meeting I asked what steps usually must be done on the first day of the IPA. Enes, the responsible specialist, responded to this saying that it makes sense to do the planning phase on the first day and added that it should not take longer than the time it takes until the first meeting with the experts takes place.

4.1.2 COMPLETED WORK

I was able to get an overview of the whole project, read the individual evaluation criteria, created a timetable, formulized all tasks on dev.azure.com and wrote down the project specifications as well as the test concept. I also informed myself more about the wrapping pattern. I did this mainly by watching the following YouTube video about the topic: https://www.youtube.com/watch?v=9ZFN8DrvcYA

4.1.3 ACHIEVEMENTS

I now have all the above-mentioned work done and a better understanding of how the wrapping/adapter pattern functions.

4.1.4 PROBLEMS

I badly underestimated the time it took to create the test concept, so I worked overtime. The reason it took so long is because of the amount of test possibilities I had to come up with, not only to satisfy the expectations documented in the task description but also because this whole IPA has a big focus on unit testing. I came up with over a dozen tests I'll implement later and thinking of a single one and writing it down took at least ten minutes. Besides that, I was unsure of what I was supposed to write in the *Test steps* fields in the test concept. I ended up leaving them empty.

4.1.5 ASSISTANCE

I asked my co-worker Andreas two questions:

The first was about what was meant with the word *reference* in the point 1.2.1 in the detailed work instructions. I knew what the word itself meant but not in a technical sense in that context. Andreas explained to me, that this meant a reference to an instance of a class, which makes sense to me.

The second question was about the wrapper pattern. Andreas wrote his own wrapper class a few months prior but for an SQL client. As I analyzed his code, I wondered why he gave all information necessary to the client context over parameters and had so few public methods in the context itself, instead of using setters or more methods. His response was that wrapper classes should be stateless if possible and therefore all information should be provided through parameters. I informed myself further about stateless classes on the internet after he mentioned that and will describe my findings later in the IPA documentation.

4.1.6 CONCLUSION

Overall, it was a productive day, and I am satisfied with the fact that I was able to do all planning steps on the first day, even though I assume that I will still have to change a few things in the test concept.

4.2 8TH OF APRIL 2021

4.2.1 DAILY MEETING WITH RESPONSIBLE SPECIALIST

During the meeting Enes told it would make sense to adjust the timetable in a way, that I plan more time in the evenings to document. Besides that, we talked about how to document unit tests in the test concept. I told them I was unsure on what I should write in the *test steps* field for unit tests and Enes and Andreas agreed that it would be best for me to write a short explanation in that field on why there aren't any test steps involved.

4.2.2 COMPLETED WORK

I adjusted the timetable according to the suggestions of Enes yet reserved even more time in the evenings for documenting than he recommended. I informed myself more about the wrapping pattern. I also created the interface IDicomContext and wrote the constructor for the DicomContext. As I implemented these things, I found myself asking various questions and noting them down. For example, what is the benefit of multiple constructors vs. having optional parameters in one constructor? I informed myself on the internet and will write about my findings later in the IPA. Something else I wondered was, whether a function called SendRequestsAsyncWithCustomClientConfig would make any sense. I dismissed the idea though. If I find time, I will write about why this doesn't make sense to me later on in the IPA as well. I also wrote about Dicom, mocking and dependency injection in the IPA-documentation.

4.2.3 ACHIEVEMENTS

I now have a more accurate and better timetable, more knowledge about the wrapping pattern, the IDicomContext interface, an experimental constructor for the DicomContext, more questions I will try to answer myself and more information about Dicom, mocking and dependency injection in the IPA-documentation.

4.2.4 PROBLEMS

I was unsure what the Dicom association release was. I eventually found out by reading the code comments in the DicomClient code on FO-Dicoms GitHub page.

4.2.5 ASSISTANCE

I asked Enes how much information about Dicom I should write into the IPA documentation. He responded that I should write a short introduction with all relevant information and that if I use descriptions from the internal Wiki, I should reference it.

4.2.6 CONCLUSION

I'm satisfied with the code I wrote today and think it's a good start. I was a bit slow with documenting though because I genuinely didn't know *how* I'm supposed to write things. Maybe things should be super easy to understand and I should make small steps in explaining. Perhaps I should write about things in depth and assume everyone knows what I'm talking about. Hopefully I will find a good balance in my writing style tomorrow or else I will consult Enes on this issue if it persists.

4.3 9TH OF APRIL 2021

4.3.1 DAILY MEETING WITH RESPONSIBLE SPECIALIST

In the meeting Enes suggested to me, that I could write a reflection of why the test concept took so long. I adjusted the journal entry of the first day so it is clear what exactly the reason was the test concept took so long. I also told Enes about my realization, that the original reason for the implementation of the Dicom client was faulty. We discussed the problem and if it makes sense to change to a different, more simple solution. I gave a few arguments on why it still makes sense to go for the more complex solution, that being implementing the Dicom client context and we decided to proceed with the original plan due to my newly found reasons, even though the original argument, the notion that it wouldn't be unit testable otherwise, was simply wrong. I will describe this in further detail later in the IPA. We also talked about my writing style. I mentioned that I was unsure how simple it should be to understand but we agreed it's better to keep it easily understandable.

4.3.2 COMPLETED WORK

I implemented a first version of the DicomContext and created a testing environment that includes a simple Dicom server and a class implementing the DicomContext, enabling me to try out the DicomContext without having to start a real service.

I also did a lot more research about wrapper patterns. This took way longer than expected, because the whole purpose of the DicomContext had changed and therefore I had to find information aiding me in deciding any future steps. All of this obviously also hindered the actual implementation of the DicomContext, prompting me to require a lot more knowledge before proceeding.

4.3.3 ACHIEVEMENTS

With the testing environment I now have a great way of testing new versions of the DicomContext. I also have a way better understanding of wrapping patterns.

4.3.4 PROBLEMS

One obvious problem is the fact, that the next steps in the IPA are a lot fuzzier, now that the original purpose of the DicomContext has been defeated.

4.3.5 ASSISTANCE

I shared my thoughts about why wrapping all public fields and events would make the context stateful and not stateless with Andreas. He told me it might be an idea to pretend they don't exist, which would make sense, since they are optional for the client to function.

4.3.6 CONCLUSION

Even though the task description has been deemed faulty it's still very much possible to do the IPA, especially since mistakes can be more interesting than things that go as intended. This doesn't only apply to software engineering.

4.4 12TH OF APRIL 2021

4.4.1 DAILY MEETING WITH RESPONSIBLE SPECIALIST

We talked about whether the implementation should be stateless or stateful and if I should implement my pattern ideas. We agreed upon sticking with the original plan but documenting my other ideas for potential future implementation.

We also talked about the namespace. My problem was that I couldn't make the namespace of the client context be the same as the actual class name. Well, I could but it would make the implementation unnecessarily complicated, because I wouldn't be able to import the context library anywhere and I'd have to specify the namespace as well as the class anywhere I use it i.e., DicomContext.DicomContext or else it wouldn't know which library and class I'm talking about. We concluded that I should continue implementing it the way I already did, namely calling the namespace DicomClientContext.

4.4.2 COMPLETED WORK

I wrote the ClientContext. This took up unplanned time because I had to rewrite the context. I made a stateless version last Friday but since the decision was made, that I should implement the stateful version instead and because the stateful version differs a lot from the stateless one I rewrote it. This might still have saved a lot of time in the future though, because implementing the stateless version in the services would have been a lot riskier and might have ended up in me having to debug a service for hours to make it work with the stateless version. The stateful versions interface looks like the original DicomClient, so it's a lot safer to implement, since one doesn't have to adjust the code to match any new interface.

I also wrote all unit tests for the new version of the ClientContext.

4.4.3 ACHIEVEMENTS

I now have a functioning ClientContext inheriting from IDicomClient.

4.4.4 PROBLEMS

I tried writing a unit test that serves as an automated end-to-end test. Sadly, that unit test fails for unknown reasons, rendering it useless for now.

4.4.5 ASSISTANCE

None.

4.4.6 CONCLUSION

All in all, it was an okay day. I couldn't document as much as I would have liked to, because rewriting the DicomContext took a up most time and I generally was tired and inefficient.

4.5 13TH OF APRIL 2021

4.5.1 DAILY MEETING WITH RESPONSIBLE SPECIALIST

Because Enes is sick I only talked with Andreas, a coworker, instead. We looked at the timetable and I explained why I spent so much time on implementing the context on Monday. I also told him, that I was practically done with the unit testing of the client context. There was still a small problem with one unit test, but I was going to try solving it on my own.

4.5.2 COMPLETED WORK

I switched the inheritance of the DicomContext to IDicomContext and implemented the IDicomContext, because it turns out the interface IDicomClient doesn't have all the properties we need. I also finished writing the unit tests for the DicomContext and continued with the IPA documentation.

4.5.3 ACHIEVEMENTS

I now have all the unit tests for the DicomContext, the IDicomContext interface and more content in the IPA documentation.

4.5.4 PROBLEMS

None.

4.5.5 ASSISTANCE

None.

4.5.6 CONCLUSION

All in all, it was a good day and I was able to reach the planned milestone.

4.6 14TH OF APRIL 2021

4.6.1 DAILY MEETING WITH RESPONSIBLE SPECIALIST

There was nothing to discuss during the meeting. Andreas, a coworker, simply informed me that Enes will soon be back from being sick. I hope he'll be in better health and not overworking himself.

4.6.2 COMPLETED WORK

I wrote unit tests for the Dicom generator service and started writing unit tests for DicomToPACSLoader.

4.6.3 ACHIEVEMENTS

I now have all unit tests required for the Dicom generator service.

4.6.4 PROBLEMS

I first was unsure how to disable the validation of the DicomFiles during runtime because I was feeding the DicomGenerators `Save` method with empty DicomDatasets and it would throw an exception on the line, where I added the datasets to the requests list. The workaround was to leave away the conversion from DicomDataset to DicomFile. Hopefully, this didn't ruin the service. I will have to test that later.

There were tons of problems as I started unit testing the `Load` method, that aren't worth mentioning but took up a lot of time.

4.6.5 ASSISTANCE

None.

4.6.6 CONCLUSION

I am unsatisfied with today's work, because I couldn't do any documenting and because unit testing the `Load` method is taking forever. I love trying to figure out ways to unit test it, but ultimately, I'm just too greedy and should maybe just leave the method alone due to its complexity and because someone should instead refactor it at some point it's useless to unit test it now. Even though I slept enough I also was very tired in general today making me a lot less efficient.

4.7 15TH OF APRIL 2021

4.7.1 DAILY MEETING WITH RESPONSIBLE SPECIALIST

During the meeting I explained why writing the unit tests took so long. The reason being the complex `Load` method. We agreed upon me skipping the unit tests for this method and refactoring the method after the IPA. I also asked how I should obtain example Dicom datasets in my unit tests. Enes recommended me generating some and storing them in the project as files. He also thought it'd make sense for me to concentrate on documenting today, since I didn't get to that yesterday and I agree with him on that.

4.7.2 COMPLETED WORK

I was able to complete six more pages for the IPA documentation. I mostly documented the code I wrote up until then.

4.7.3 ACHIEVEMENTS

I now have more content in the IPA documentation.

4.7.4 PROBLEMS

None.

4.7.5 ASSISTANCE

None.

4.7.6 CONCLUSION

Six pages are underwhelming in my opinion. The reason I only was able to do six pages is because documenting code snippets is a somewhat mundane task and therefore progress was slow and I was unsure with which tool and diagram type I wanted to do the flowchart for the LoadTest method. Ironically, this is the unit test that took me forever to try finish in the first place, once again taking up my time and energy.

4.8 16TH OF APRIL 2021

4.8.1 DAILY MEETING WITH RESPONSIBLE SPECIALIST

On one hand there was the usual meeting with the responsible specialist, of which there is nothing of interest to report and on the other hand there was a meeting with the main expert. I gave them a rough overview of my progress so far and the main problems I had on the way. They also gave me some advice for the IPA documentation and the presentation. Namely that it might make sense to show the test result in form of a table, so I don't have to explain every test again and that I could use a role book during the presentation so I don't lose my trail.

4.8.2 COMPLETED WORK

I didn't do any documentation, because I had some slight health issues due to drinking too many energy drinks in the last days. I felt more comfortable completing the last unit tests, finishing the refactoring of the services and doing a general cleanup and decided to do the documenting during the weekend instead, when I can concentrate better.

I also went on FO-Dicoms Gitter page and asked why certain properties were missing in the IDicomClient interface. Reinhard Gruber, one of the main contributors to the library, explained the reasons to me and invited me to do a pull requests with the added properties and I did just that.

4.8.3 ACHIEVEMENTS

All the code for the IPA is written now, so I can concentrate on the documentation. I also created the pull request I mentioned above. Technically this isn't relevant to the IPA, because the IPA shouldn't have any dependencies with other people, but I might as well mention it.

4.8.4 PROBLEMS

I had a namespace problem. Basically the PacsTarget class wasn't recognized in the JobReceiver.Test namespace.

4.8.5 ASSISTANCE

I asked my coworker Andreas for a solution to my problem mentioned above and he quickly realized that I forgot to add the following line to JobReceiver.Test.csproj:

```
<ProjectReference Include="..\JobReceiver\JobReceiver.csproj" />
```

All this really does is include the JobReceiver namespace, so I have access to classes such as the PacsTarget.

4.8.6 CONCLUSION

All in all, it was a successful day but I drew the conclusion that drinking too many energy drinks for over a week is indeed unhealthy.

4.9 19TH OF APRIL 2021

4.9.1 DAILY MEETING WITH RESPONSIBLE SPECIALIST

We discussed whether or not I have free in the afternoon, because there would be public holidays due to “Sechseläuten”. The responsible specialist concluded that it wasn’t thought of during the planning and we should stick to the plan and I agree with that. There was nothing further to discuss.

4.9.2 COMPLETED WORK

I documented the code of the `Playground` and checked if all unit tests and services still ran and wrote down the results.

4.9.3 ACHIEVEMENTS

I now have more content in the IPA documentation and have all the test results.

4.9.4 PROBLEMS

One major problem is the fact, that the ReportSender service showed some very odd behavior and I don’t know what causes the issue.

4.9.5 ASSISTANCE

None.

4.9.6 CONCLUSION

All in all, it was a productive day and I am mostly content with the results of the tests. Documenting the test cases took a bit longer than anticipated in the timetable, especially because the ReportSender service didn’t exactly work though.

4.10 20TH OF APRIL 2021

4.10.1 DAILY MEETING WITH RESPONSIBLE SPECIALIST

In the daily meeting Andreas volunteered to proofread my IPA documentation. Besides that nothing important was discussed.

4.10.2 COMPLETED WORK

I did all the inline documentation of the code, wrote the summary of the project, finished the cleanup process, did some grammar corrections according to Andreas's suggestions and copied the code into the IPA documentation.

Writing the conclusion of the project didn't take as long as anticipated, because it only had to be one page long.

4.10.3 ACHIEVEMENTS

I now have cleaner code, inline documentation and better grammar in my documentation.

4.10.4 PROBLEMS

None.

4.10.5 ASSISTANCE

Andreas proofread my IPA documentation.

4.10.6 CONCLUSION

I was generally content with my IPA and looked forward to finalizing it on the next day.

4.11 21TH OF APRIL 2021

4.11.1 DAILY MEETING WITH RESPONSIBLE SPECIALIST

There wasn’t anything interesting to discuss. I simply mentioned that I will have a job interview starting at 10.30 am.

4.11.2 COMPLETED WORK

I finalized the IPA-documentation and checked all the evaluation criteria again.

4.11.3 ACHIEVEMENTS

I now have finished the IPA-documentation.

4.11.4 PROBLEMS

None.

4.11.5 ASSISTANCE

None.

4.11.6 CONCLUSION

I really didn’t have to do much anymore on the last day so I decided to upload the IPA-documentation a few hours earlier than necessary.

PART 2

5 SUMMARY

5.1 REMARK

I recommend skipping the summary if you haven't read the rest yet, because I am using a lot of technical terms that get explained later in the IPA. If I'd explain them here it wouldn't be a summary anymore.

5.2 INITIAL SITUATION

At the beginning of the IPA there were four different services that used a Dicom client provided by a library called FO-Dicom. The methods in which the Dicom client got used, simply weren't unit testable, because one would have to set up a real connection to a real Dicom server. You want to test the methods themselves and not the connection to servers when you write unit tests.

Another problem is the fact, that the implementations of the Dicom client were done differently in every service, although the end goals were almost identical. This however isn't the main focus of the IPA, but an interesting side quest.

5.3 IMPLEMENTATION

By writing a wrapper class called DicomContext containing the Dicom client and an interface for that wrapper class and by doing dependency injection to every class that uses the new Dicom client wrapper you can enable unit testing by mocking the DicomContext's interface so the connection can be faked.

And by simplifying that same DicomContext using the facade pattern and making the context stateless you can unify the usages of the DicomContext to enhance maintainability and testability.

5.4 OUTCOME

Because FO-Dicom already provided an interface for the Dicom client, the client technically already was mockable. Some properties of the Dicom client are still missing in their interface, so it turns out it was still necessary to create a wrapper class to enable mocking. After discussing it with FO-Dicoms main developer I created a pull request enhancing their interface with the missing properties. After that pull request has been done, the original purpose of the Dicom context will indeed be defeated but having a wrapper class still brings some advantages and is a good steppingstone for adding a version with the facade pattern.

At the end of the IPA the DicomContext was written resembling the proxy pattern, every class using the new DicomContext had dependency injection and their methods had unit tests in which the DicomContext was mocked using the IDicomContext interface. I also created a prototype in which the facade pattern was used for the DicomContext instead. This might get used in the future.

I am satisfied with the outcome and was able to learn a lot during the process, but it is important to keep in mind, that the future of the DicomContext is uncertain.

6 INFORMIEREN – INFORM

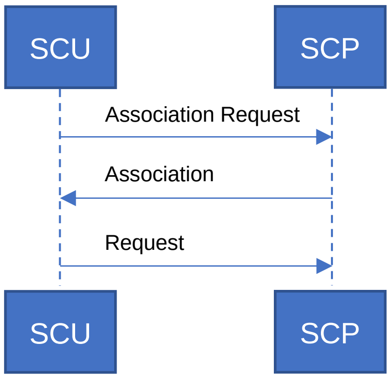
6.1 DICOM

6.1.1 GENERAL

Dicom is a medical imaging software integration standard and can either be encountered in the form of a file format with the ending ".dcm" or used as a network protocol. Both options are based on so called Dicom datasets, which are collections of information about the patient, the equipment used and much more. Medical imaging equipment in hospitals generate Dicom files and for sharing the Dicom datasets contained in those files it uses the Dicom network protocol. (Schafflützel, 2021)

6.1.2 NETWORKING

Only the networking part is relevant to the IPA. Therefore, this is the only topic I give a small overview of.



SCU (Service Class User): Client
SCP (Service Class Provider): Server

(Schafflützel, 2021)

As you can see in the diagram sending a request to a SCP ("server") requires two prior steps.

In the first step, the association request, the SCU ("client") asks what kind of requests the SCP allows, meaning what kind of commands and in which format they can be. In Dicom language the command type is called SOP which stands for Service Object Pair and the format options is called the presentation context.

Now like in dating the SCP can either respond by rejecting or accepting any further advances, the difference to dating being an SCP will never ghost you completely. This is the second step.

In C# you can use the FO-Dicom library to initialize a Dicom client. The required parameters for doing this are the following.

Paramter name	Type	Description
host	string	This is the Dicom host you want to send the data to. This can be an IP or a hostname like “localhost”.
port	int	This is the port of the host you want to send your request to.
useTls	bool	This is whether you want TLS security to be enabled. What this entails isn’t relevant to the IPA.
callingAe	string	This is the so-called calling application title. It’s basically how your SCU is named in the Dicom world.
calledAe	string	This is the name of the SCP.

6.2 WRAPPING

6.2.1 GENERAL

What I learned from researching this topic is that one does not simply wrap something. There are a lot of different ways to go about it and various similar patterns with completely different intentions you can implement. Therefore, it’s important to know exactly what you want to achieve before you start wrapping.

The following are patterns that have similar purposes and might be confused with each other.

Pattern name	Short description
Adapter pattern	An intermediate interface to make two incompatible interfaces work with each other.
Facade pattern	A simplified interface hiding a complex collection of objects working together.
Proxy pattern	Controls access to an interface so you have a placeholder for an object and can initialize the object itself later.
Decorator pattern	An object is placed inside of a wrapper object, that calls the original objects methods but adds some new logic around it.

All the above patterns are part of a bigger group called structural patterns. There are two more groups of patterns called the creational and behavioral patterns, but I won't go into further detail about those. I'm unable to say which of the mentioned structural patterns are true wrapper patterns because different sources on the internet claim different things. I assume wrapping just universally means placing an object inside of a wrapper object. The adapter pattern seems to be the most straight forward example of this concept.

6.2.1.1 ADAPTER PATTERN

The basic premise of the adapter pattern is to create an interface of an adapter that allows an interface to call an otherwise incompatible interface by acting as an intermediary. The goal is not however to change any behavior during this process. The most obvious example is a literal power adapter. It allows you to travel to another country where the power sockets have a completely different interface than what you need for your phone charger but still be able to charge it. On one hand you have the interface of which the power originates, i.e., the power socket and on the other hand you have the interface of the adapter it connects to, that allows you to connect your phone charger.

Here's how this concept would translate to a simple UML.

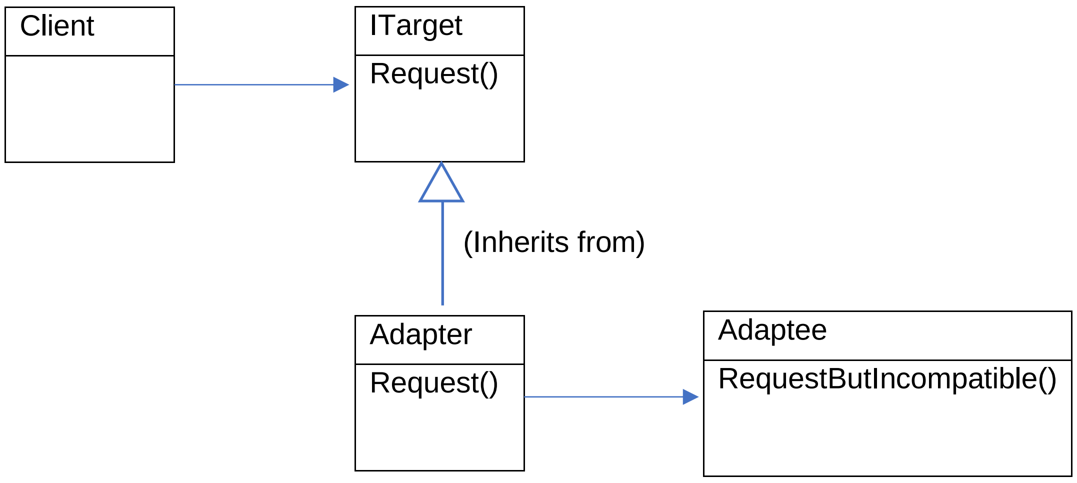


Figure 1: Adapter example

(Wikipedia, 2021)

The client in the diagram wants to call the adaptees method `RequestButIncompatible()` but for some compatibility reasons can't. The solution here is to create an interface on top of the adapter. The Client now can call the adapters `Request()` method without having to bother about compatibility, because the adapter already does that.

The reasons for using the adapter pattern go as follows.

- It allows reusing a class that doesn't have the interface a client requires.
- It allows two classes with incompatible interfaces to work together.
- It can provide an alternative interface for a class.

(Wikipedia, 2021)

7 PLANEN – PLAN

7.1 FUNCTIONAL REQUIREMENTS

7.1.1 REQUESTS

Sending requests with the Dicom client context to a Dicom server should give the correct response.

7.1.2 WRAPPING

All 5 public methods, 5 events and 18 properties of the Dicom client are wrapped in the Dicom client context.

7.1.3 MODIFIED METHODS

The newly modified methods in the services should still run.

7.2 NON-FUNCTIONAL REQUIREMENTS

7.2.1 EXCEPTION HANDLING

Every function with a potential of failing should be surrounded by a `try catch` clause. This enables developers and users to identify problems easier and make them less severe, because like this the rest of the program will still be able to run without crashing.

7.2.2 CLEAN CODE

The code should be clean and easy to understand for other developers. The guidelines to be followed are documented in the file `210301_DotNet_Code_Style_Rules.md` which is in PkOrg.

7.3 TEST CONCEPT

7.3.1 TEST OBJECTIVES

No.	Description	Comparable Result	Priority 1 is highest, 3 is lowest
1	Test if successful association can be done with the wrapped `SendAsync` method.	`DicomClient.AssociationAccepted`	1
2	Test if an association can be done with the wrapped `AddRequestsAsync` method using	`DicomClient.AssociationRejected` AND	1

No.	Description	Comparable Result	Priority 1 is highest, 3 is lowest
	a falsely or partially falsely initialized Dicom client context.	`DicomClient.RequestTimedOut`	
3	Test if all public properties in the Dicom client context can be set correctly.	All public properties in the Dicom client context.	1
4	Test if wrapped events function.	All public events in the Dicom client context.	1
5	Test the provided requests validity of the modified `Save` method in the `PACSTarget`.	The callback of the client context mock containing the requests, which were provided to the mock object over its parameters.	2
6	Test the Load method in `DicomToPacsLoader` by mocking both the SQL data context and Dicom client context and controlling the data `sqlDataContext.ExecuteReaderAsync` returns to test the parameters provided to `SendCStoreRequest`.	The callback of the client context mock containing the request, which was provided to the mock object over its parameters.	2
7	Test the Load method of test nr. 5 using the same technique as described in test nr. 5 but with the goal to test the value of the SQL parameter called `SPParameterExportID`.	The callback of the sql data context mock containing the `SPParameterExportID` SQL parameter which was provided to the mock object over its parameters.	3
8	Test the Dicom requests validity in `JobReceiver.PacsTarget.sendReports`.	The callback of the client context mock containing the requests, which were provided to the mock object over its parameters.	2
9	Test the Dicom requests validity in `ReportSender.PacsTarget.sendReports`.	The callback of the client context mock containing the requests, which were provided to the mock object over its parameters.	2

Table 2: Test descriptions

7.3.2 TEST STRATEGY AND TEST LEVEL

I must test the Dicom client context itself. Because it is a client it does not have much logic and return values to test. Therefore, I will do automated integration tests to check if the context uses the wrapped methods correctly and at least one unit test to make sure the class variables are set correctly when initializing a client context.

I also must test the modified methods in the services. I intend to write a unit test for each occurrence of the client context. Because none of those methods have testable return values, I will instead test the parameters provided to the context by mocking the context and comparing a callback of the parameters with the expected results.

7.3.3 TEST ENVIRONMENT

The test environment for testing the Dicom client context itself is a .Net Core 3.1 library named `Balzano.Common.DicomContext.Test` using the `Microsoft.NET.Sdk` SDK, the `Microsoft.NET.Test.Sdk`, `xunit`, `xunit.runner.visualstudio` and `Moq` packages, as well as the `Balzano.Common.DicomContext` library. To enable automated integration tests there is a rudimentary Dicom server in the project.

The tests for the modified methods in the services will be implemented in the services own test folders. The namespace of those test folders is the combination of the capitalized service name with the suffix `.Test`. The `Microsoft.NET.Sdk` SDK, the `Microsoft.NET.Test.Sdk`, `xunit`, `xunit.runner.visualstudio` and `Moq` packages are used in those as well.

No internet connection is needed and you need at least 1 GB of RAM to run VS Code and the tests. (Microsoft, 2021)

7.3.4 WHAT WON'T BE TESTED

- I won't be writing unit tests that test the actual connection to a real Dicom server for the services, because I want to test the logic of the methods using the DicomContext.
- I won't test any other methods of the services besides the ones using the DicomContext, because during the IPA my only responsibility is that the DicomContext works where it is being used.

7.3.5 TEST OBJECTS

No.	Item	Description
1	Common.DicomContext	Enables unit testing of classes using it due to its interface being mockable.
2	DicomGenerator.PacsTarget	Sends the generated DicomDatasets to a DicomNode.
3	DicomToPACSLoader.DicomToPACSLoader	Uses a given list of StudyInstanceUIDs to load SOPInstances from the database

No.	Item	Description
		and write them as Dicom files to the filesystem.
4	JobReceiver.PacsTarget	Sends encapsulated PDF reports to the PACS.
5	ReportSender.PacsTarget	Sends encapsulated PDF reports to the PACS.

Table 3: Test objects

7.3.6 TEST TYPES

No.	Test type	Description
1	Positive Test	The provided parameters are in an expected and correct manner.
2	Negative Test	The provided parameters expected are `null` values.
3	Parameter Edge Cases	The provided parameters are either the highest or lowest possible values.
4	Unit Test	A unit i.e., a class is being tested.
5	Integration Test	An end-to-end test where not a single unit but an outcome is being tested.

7.3.7 OVERVIEW OF TEST CASES

No.	Test object	Test case
1	Common.DicomContext.SendRequestsAsync	A single request
2	Common.DicomContext.SendRequestsAsync	Invalid parameters
3	Common.DicomContext.SendRequestsAsync	Max. number of requests
4	Common.DicomContext.SendRequestsAsync	Too many requests
5	Common.DicomContext.SendRequestsAsync	Parameters null or “”
6	DicomGenerator.PacsTarget	Datasets = null
7	DicomGenerator.PacsTarget	requests amount == datasets amount
8	DicomToPACSLoader.DicomToPACSLoader	SQL reader gives correct dataset

No.	Test object	Test case
9	DicomToPACSLoader.DicomToPACSLoader	SQL reader gives null values
10	Jobreceiver.PacsTarget	reports = null
11	Jobreceiver.PacsTarget	requests amount == reports amount
12	ReportSender.PacsTarget	reports = null
13	ReportSender.PacsTarget	requests amount == reports amount
14	DicomGenerator.PacsTarget	Run DicomGenerator
15	DicomToPACSLoader.DicomToPACSLoader	Run DicomToPACSLoader
16	Jobreceiver.PacsTarget	Run JobReceiver
17	ReportSender.PacsTarget	Run ReportSender

Table 4: Test cases overview

7.3.8 TEST CASE DESCRIPTIONS

Test Case	No. 1 A single request
Description	Provide the SendAsync method with a single request as a parameter.
Test prerequisites	• Have an instance of the Dicom client context. • Have any kind of request e.g., a CEchoRequest. • Have all other required parameters for SendRequestsAsync.
Test steps	No manual test steps needed due to it being a unit test. See implementation.
Expected outcome	The Dicom server should answer with a successful response.

Test Case	**No. 2 Invalid parameters**
Description	Provide the SendAsync method with invalid parameters.
Test prerequisites	• Have an instance of the Dicom client context. • Have any kind of request e.g., a CEchoRequest. • Have all other required parameters for SendRequestsAsync.
Test steps	No manual test steps needed due to it being a unit test. See implementation.
Expected outcome	SendRequestsAsync should throw an error describing the problem for as many mistakes as possible.

Test Case	**No. 3 Max. number of requests**
Description	Provide the SendAsync method with the maximum allowed number of requests that SendAsync should still be able to process as a parameter.
Test prerequisites	• Have an instance of the Dicom client context. • Have the maximum possible number of requests that SendRequestsAsync should still be able to process of any kind of request e.g., a CEchoRequest in a list. • Have all other required parameters for SendRequestsAsync.
Test steps	No manual test steps needed due to it being a unit test. See implementation.
Expected outcome	The Dicom server should answer with a successful response.

Test Case	**No. 4 Too many requests**
Description	Provide the SendAsync method with more requests than it should be able to handle as a parameter.
Test prerequisites	• Have an instance of the Dicom client context. • Have a totally exaggerated amount of any kind of request e.g., a CEchoRequest in a list. • Have all other required parameters for SendRequestsAsync.
Test steps	No manual test steps needed due to it being a unit test. See implementation.
Expected outcome	SendRequestsAsync should throw an error describing the problem.

Test Case	**No. 5 Parameters null or “”**
Description	Provide the SendRequestsAsync method with all correct parameters yet make one of them the value null or “”. Do this with every possible parameter once.
Test prerequisites	• Have an instance of the Dicom client context. • Have all other required parameters for SendRequestsAsync.
Test steps	No manual test steps needed due to it being a unit test. See implementation.
Expected outcome	SendRequestsAsync should throw an error describing the problem or the Dicom server should answer with a successful response, depending on the importance of the parameter.

Test Case	**No. 6 Datasets = null**
Description	Provide the Save method with the value null in the datasets parameter.
Test prerequisites	• Have a mock of the Dicom client context.
Test steps	No manual test steps needed due to it being a unit test. See implementation.
Expected outcome	SendRequestsAsync should throw an error describing the problem.

Test Case	**No. 7 requests amount == datasets amount**
Description	Provide the Save method with a certain number of datasets. Check if the same amount of DicomCStoreRequests are given to the SendRequestsAsync method.
Test prerequisites	• Have a mock object of the Dicom client context. • Have multiple generated datasets.
Test steps	No manual test steps needed due to it being a unit test. See implementation.
Expected outcome	The amount of requests should be equal to the number of datasets.

Test Case	No. 8 SQL reader gives correct dataset
Description	Mock the SQL data context and make the SQL reader return an id and a studyInstanceUID. Check if the SendRequestsAsync call in SendCStoreRequests gets a dataset with the same id and studyInstanceUID as defined earlier.
Test prerequisites	• Have a mock object of the Dicom client context. • Have a mock object of the SQL data context.
Test steps	No manual test steps needed due to it being a unit test. See implementation.
Expected outcome	The SendRequestsAsync call in SendCStoreRequests should get a dataset with the same id and studyInstanceUID as defined in the return values of the mocked SQL reader.

Test Case	No. 9 SQL reader gives null values
Description	Mock the SQL data context and make the SQL reader return null as an id and null as a studyInstanceUID.
Test prerequisites	• Have a mock object of the Dicom client context. • Have a mock object of the SQL data context.
Test steps	No manual test steps needed due to it being a unit test. See implementation.
Expected outcome	The Load method should throw an error describing the problem.

Test Case	No. 10 reports = null
Description	Provide the SendReports method with null as the reports parameter but give the other parameters valid values.
Test prerequisites	• Have a mock object of the Dicom client context.
Test steps	No manual test steps needed due to it being a unit test. See implementation.
Expected outcome	The SendReports method should throw an error describing the problem.

Test Case	No. 11	requests amount = reports amount
Description	Provide the SendReports method with a certain amount of valid DicomDatasets as the reports parameter and give the other parameters valid values as well. Check if the same amount of DicomCStoreRequests are given to the SendRequestsAsync method.	
Test prerequisites	• Have a mock object of the Dicom client context. • Have generated DicomDatasets.	
Test steps	No manual test steps needed due to it being a unit test. See implementation.	
Expected outcome	The amount of requests should be equal to the number of datasets.	

Test Case	No. 12	reports = null
Description	Provide the SendReports method with null as the reports parameter but give the other parameters valid values.	
Test prerequisites	• Have a mock object of the Dicom client context.	
Test steps	No manual test steps needed due to it being a unit test. See implementation.	
Expected outcome	The SendReports method should throw an error describing the problem.	

Test Case	No. 13	requests amount = reports amount
Description	Provide the SendReports method with a certain amount of valid DicomDatasets as the reports parameter and give the other parameters valid values as well. Check if the same amount of DicomCStoreRequests are given to the SendRequestsAsync method.	
Test prerequisites	• Have a mock object of the Dicom client context. • Have generated DicomDatasets.	
Test steps	No manual test steps needed due to it being a unit test. See implementation.	
Expected outcome	The amount of requests should be equal to the number of datasets.	

Test Case	**No. 14** **Run DicomGenerator**
Description	Follow the steps to run the DicomGenerator to check if the modification doesn't disable the service.
Test prerequisites	• Have a .NET Core runtime installed.
Test steps	• Set the variable `Target` to `Dicom` in the `Config.json` file located in the `Config` folder of the DicomGenerator project. • Make sure the `Config.json` references your own Dicom server you want to send the data to. • Open the project root folder in CMD. • Type dotnet run and hit enter.
Expected outcome	The generated Dicom datasets should arrive at the server.

Test Case	**No. 15** **Run DicomToPACSLoader**
Description	Follow the steps to run the DicomToPACSLoader to check if the modification doesn't disable the service.
Test prerequisites	• Have a .NET Core runtime installed. • Have the services configuration point to your local Orthanc.
Test steps	• Open the project root folder in CMD. • Type dotnet run and hit enter.
Expected outcome	The service should run without crashing and the log should show that a successful connection with the local Orthanc was established.

Test Case	**No. 16** **Run JobReceiver**
Description	Follow the steps to run the JobReceiver to check if the modification doesn't disable the service.
Test prerequisites	• Have a .NET Core runtime installed. • Have the services configuration point to your local Orthanc.
Test steps	• Open the project root folder in CMD. • Type dotnet run and hit enter.
Expected outcome	The service should run without crashing and the log should show that a successful connection with the local Orthanc was established.

Test Case	No. 17	Run ReportSender
Description	Follow the steps to run the ReportSender to check if the modification doesn’t disable the service.	
Test prerequisites	• Have a .NET Core runtime installed. • Have the services configuration point to your local Orthanc.	
Test steps	• Open the project root folder in CMD. • Type dotnet run and hit enter.	
Expected outcome	The service should run without crashing and the log should show that a successful connection with the local Orthanc was established.	

8 ENTSCHEIDEN – DECIDE

8.1 HOW MANY ENERGY DRINKS SHOULD I DRINK PER DAY?

Even though I wrote this question as a joke at the start of the IPA I really should have tried to answer it. Whatever the answer is, it's less than I drank.

8.2 HOW SHOULD THE EXISTING SERVICES BE MODIFIED?

8.2.1 IF THERE WERE NO DICOM CLIENT CONTEXT

8.2.1.1 PROBLEMS WITH CURRENT IMPLEMENTATION

The main problem the services initializing the DicomClient currently have is, that the DicomClient gets initialized in Constructors like the PACSTarget class in the DicomGenerator:

```
        public PACSTarget () {
            this.callingTitle = Program.config.CallingAET;
            this.calledTitle = Program.config.CalledAET;
            this.dicomHost = Program.config.DicomHost;
            this.dicomPort = Program.config.DicomPort;
            this.client = new DicomClient(
                dicomHost,
                dicomPort,
                false,
                callingTitle,
                calledTitle
            );
        }
```

Or at the start of methods like in the DicomToPACSLoader:

```
        public async Task Load()
        {
            client = new DicomClient(
                Program.config.PACS_IP,
                Convert.ToInt32(Program.config.PACS_PORT),
                false,
                Program.config.SERVICE_AE,
                Program.config.PACS_AE
            );
            client.Logger = LogManager.GetLogger("DicomClient");
            await PingPACS();
```

In both variants the methods initializing the DicomClient become untestable if there is no real Dicom server running somewhere which you could reference in the test. In the case of unit tests this is a major problem since you want to test single methods of a unit (class) and not the Dicom connection. And even if you would reference a Dicom server eligible for sending test data to, you'd have to find a way of hijacking the configuration variables for the DicomClient to direct the connection to said server, which is a difficult task to say the least. Passing in null values when initializing the DicomClient isn't an option either, because the program would simply crash or the method would throw an exception preventing you from testing the rest of the logic in the method you want to test.

8.2.1.2 WHAT WE NEED INSTEAD FOR ENABLING UNIT TESTING

The only way of unit testing such methods without connecting to real Dicom servers is by doing something called mocking. When you mock an object, you create a fake object that does nothing if you tell it to do something. By injecting such a dummy object instead of using a real client object you don't have to bother preparing a server the client can connect to, because if you tell the mocked client to send data to a server it'll just ignore your request and the program moves on to the next line of code.

It is important to keep in mind that mocking can only be done to interfaces of a class. The reason for this is because most mocking frameworks use the proxy pattern. (matt, 2020) I mentioned the proxy pattern in chapter 5.2.1. In the proxy pattern the proxy object inherits from the other objects interface so you could technically initialize an object later, redirect the calls to the object or add some own functionality. Basically, a mock is an object inheriting from the interface you feed it, but the methods in this object don't do anything, unless you want the mock to fake some return values for example. Meaning the mock only does the inheritance and adding own functionality part from my understanding though. The functionality you can add being fake responses you control. But you obviously need the interface to create a proxy.

Here’s a visual representation of how things work without mocking the client.

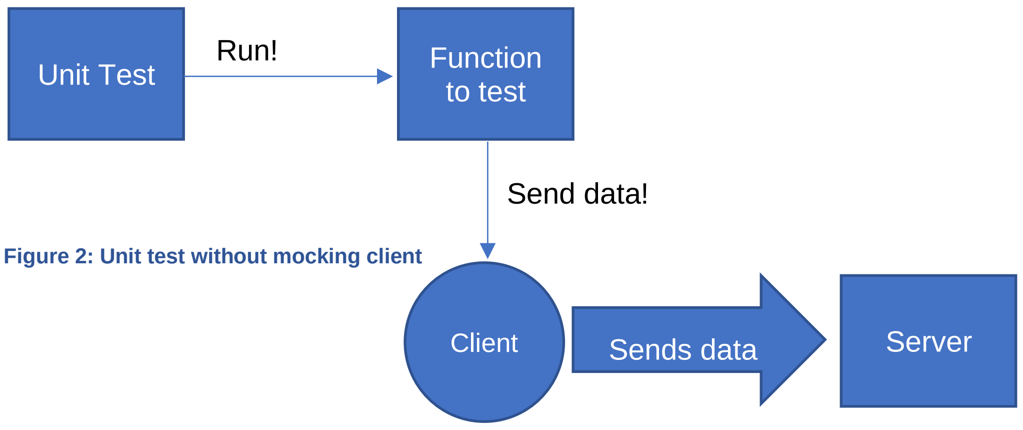


Figure 2: Unit test without mocking client

In this case you’d litter a server with unnecessary data. You either delete the data you sent to the server for testing reasons or you don’t test this method, unless you mock the client:

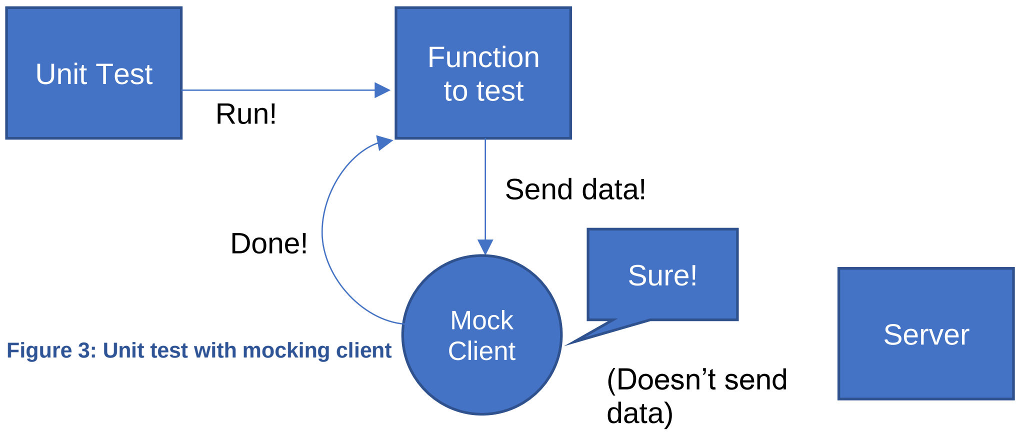


Figure 3: Unit test with mocking client

8.2.1.3 POSSIBILITES TO MAKE THE DICOM CLIENTS IN THE SERVICES MOCKABLE

You must be able to get that mocked client into the method somehow first though. Here’s two examples how you should not approach this problem followed by an actual solution.

8.2.1.3.1 BAD IDEA 1

What if we initialized the DicomClient in the constructor but used the interface IDicomClient as the type of the client class field, since we could use the interface to mock the DicomClient in a unit test?

```
private IDicomClient client;

public PacsTarget() {
    this.client = new DicomClient(
        "someHost",
        0000,
        false,
        "thisService",
        "someServer"
    );
}
```

Code Snippet 1: Bad idea 1

This isn't an option since the moment we initialize a PacsTarget in a unit test the client field is a real DicomClient and there's no possibility to swap the client out with a mock of IDicomClient, unless the client field was public or had a setter method, which should be avoided at all costs for security reasons.

8.2.1.3.2 BAD IDEA 2

Another thought experiment would be if there was a separate method in the PacsTarget class that initializes the DicomClient but it gets called after the PacsTarget gets initialized.

```
private IDicomClient client;

public void initializeDicomClient() {
    this.client = new DicomClient(
        "someHost",
        0000,
        false,
        "thisService",
        "someServer"
    );
}
```

Code Snippet 2: Bad idea 2

This would mean that if you'd initialize the pacsTarget inside of a unit test there wouldn't be a real DicomClient if you don't call the initializeDicomClient method as well. However there still would be no way of mocking the client because the class field for the client still is private. In case of unit testing, where you don't want to call the `initializeDicomClient` method, because you don't want a real server connection, the client would have the value null and an exception would be thrown in the method in which you use the client.

8.2.1.3.3 SOLUTION - DEPENDENCY INJECTION

The only reasonable alternative would be to do dependency injection for either the constructors or the methods of the classes requiring a DicomClient. For example, by adjusting the PacsTarget class in the ReportBuilder service to something like this:

```
private IDicomClient client;

public PacsTarget(IDicomClient client) {
    this.client = client;
}
```

Code Snippet 3: Dependency injection

It makes the class unit testable by enabling the mocking of the IDicomClient interface and passing it to the PacsTarget constructor. By doing that you move the problem upward in the hierarchy of classes. You'll have to initialize the DicomClient eventually after all meaning you'd end up having a new untestable method or constructor higher up in the class hierarchy where it gets initialized.

Here is how you would mock this in a unit test using Moq:

```
var dicomClientMock = new Mock<IDicomClient>();
var testablePacsTarget = new PacsTarget(dicomClientMock.Object);
```

Code Snippet 4: Creating and injecting a mock

The Object property of the dicomClientMock is the actual object that will fool the program into thinking it's a Dicom client.

Methods you run in the unit tests containing lines such as `await client.SendAsync();` are suddenly a lot less scary now.

The IPA task description says I should inject the Dicom client by adding it as a parameter for the method where the client gets used, instead of through the constructor like one usually does with dependency injection. An advantage of doing it through the constructor is, that it would be available to all methods in the class. I'll stick to the task description though.

8.2.2 ADJUSTMENTS USING THE DICOM CLIENT CONTEXT

The Dicom client context faces the exact same problems described above, only that instead of passing in a IDicomClient into the constructors we will pass in the IDicomContext:

```
private IDicomContext dicomContext;

public PacsTarget(IDicomContext dicomContext) {
    this.dicomContext = dicomContext;
}
```

Code Snippet 5: Injecting the DicomContext

And instead of mocking the IDicomClient we can now mock the IDicomContext:

```
var dicomContext = new Mock<IDicomContext>();
var testablePacsTarget = new PacsTarget(dicomContextMock.Object);
```

Code Snippet 6: Mocking the DicomContext

8.2.3 THE THING YOU MIGHT HAVE NOTICED

Let me remind you of the title of this IPA.

Implement a Dicom Client Context Class for Unit Testing

Notice how this might imply we require the Dicom client context class for unit testing?

But I was able to show how it’s completely possible to mock a DicomClient and therefore enable safe unit testing without the Dicom client context, simply by using the IDicomClient interface in chapter 7.2.1.3.3, wasn’t I? Doesn’t it render the original purpose of the Dicom Client Context completely useless?

Correct. And here’s how that came about.

When I came up with the idea for the IPA topic a few months ago I was facing the problem I described in chapter 7.2.1. I wanted to write unit tests for the methods implementing the DicomClient and I thought I couldn’t, simply because I was unaware an interface IDicomClient existed. If this interface didn’t exist, I would have been completely right about the fact that I was unable to write unit tests. And it would have been true that I would have needed to implement a wrapper class, to create an interface of the new wrapper class, which I can then mock.

The reason you need to create a wrapper class to make an interface is because you can’t just make an interface of a class you get from an external library that doesn’t have one yet, since the class of which you’d make an interface of wouldn’t inherit it. We can’t add inheritance to a class we don’t have write access to.

Here’s a visual representation of the wrapper solution if the DicomClient class wouldn’t have built in inheritance yet.

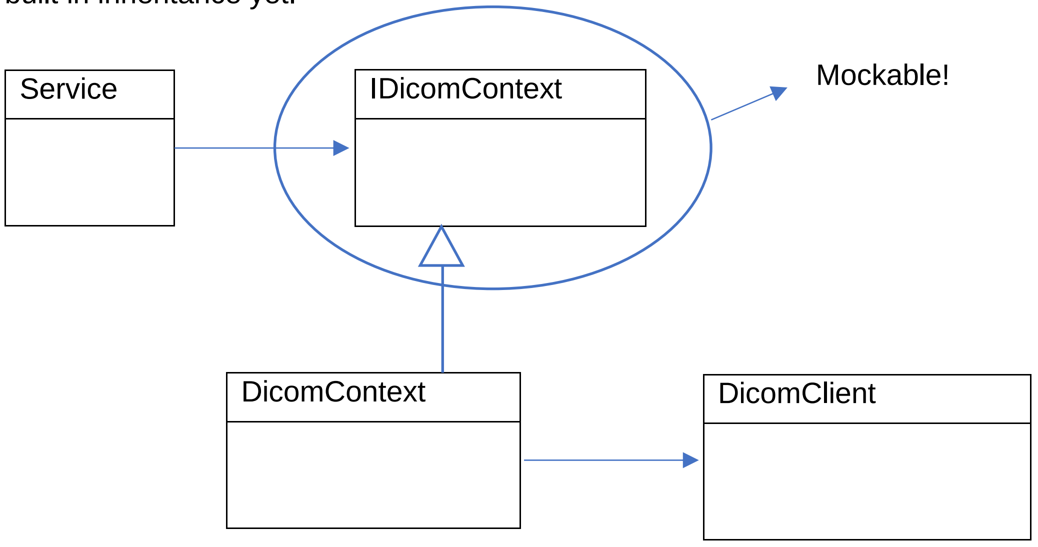


Figure 4: DicomContext concept

If this diagram feels familiar it’s because you’ve seen something similar in chapter 5.2.2.1. Just like in the adapter pattern we are wrapping an object inside of an object, that being the DicomClient inside of the DicomContext. In the DicomContext we then call methods of the actual DicomClient, but more importantly the purpose for all of this was to finally have an interface which we can mock and still, albeit indirectly, use the DicomClient in the services.

BUT

Ever since the 10th of April 2019, the day FO-Dicom pushed the first commit of their new DicomClient including a shiny interface, things would look more like this after I’d implement the DicomContext:

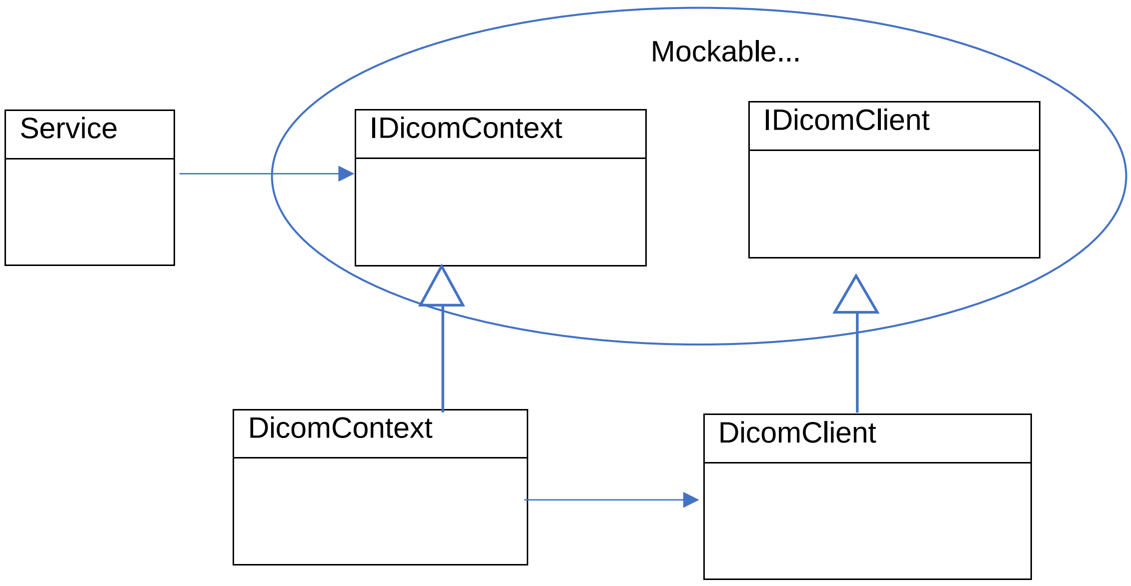


Figure 5: Client and Context, two interfaces, one problem

Having two mockable interfaces and no discernible difference is just slothful and goes against the YAGNI (You aren't gonna need it) principle. The original purpose of the IDicomContext interface is therefore defeated. Or so I thought at first. During the implementation I noticed, that IDicomClient doesn't contain definitions for a few properties, that the actual DicomClient has and that we require in the services. To make a mock that can fake having those properties, we need to make the IDicomContext containing all the properties or FO-Dicoms interface gets enhanced.

8.2.4 THE OTHER PROBLEM, WHICH THE CONTEXT COULD SOLVE

See this code snippet?

```
        public async Task Save(List<DicomDataset> datasets) {

            List<DicomRequest> requests = new List<DicomRequest>();

            foreach(DicomDataset dataset in datasets) {

                requests.AddRange(new List<DicomRequest>
                    {
                        new DicomCEchoRequest(),
                        new DicomCStoreRequest(new DicomFile(dataset))
                    });

            }

            await client.AddRequestsAsync(requests);
            await client.SendAsync();

        }
```

Code Snippet 7: DicomGenerator.PACSTarget.Save()

And this one?

```
        public async Task SendReports(List<DicomDataset> reports, string remot
eAE, string remoteHost)
        {
            client = new Dicom.Network.Client.DicomClient(remoteHost, Program.
config.PACS_PORT, false, Program.config.SERVICE_AE, remoteAE);
            client.Logger = LogManager.GetLogger("DicomClient");
            await PingPACS();
            foreach (DicomDataset ds in reports)
            {
                await sendCStoreRequest(ds);
            }
        }
```

Code Snippet 8: JobReceiver.PacsTarget.SendReports()

Those methods do the exact same, only completely differently, thinking that being unique makes them quirky. Maybe if PACSTarget and PacsTarget were a bit more alike, managing the code would be a lot simpler. This goes for all four of the classes I must modify for the IPA. It's the same concept with all of them though:

1. We initialize a new DicomClient.
2. We ping a Dicom server using a DicomCEchoRequest.
3. We send more requests.

There's currently no consistency in how the requests get sent and how often we ping the Dicom server, even though there's barely any difference in the end goals.

I'm pretty sure that pinging the server after every single request like in the PACSTarget or sending every request one by one like in the PacsTarget isn't optimal. Balzano AG isn't a hospital though, so not every developer should have to be required to know the most optimal solution for sending their Dicom requests and most requests will always stay rather simple.

I think this calls for a simplified interface. This is where the Dicom context could help.

In chapter 5.2.1 I mentioned the facade and decorator patterns. Now I might not be able to understand those patterns completely in the timespan of this IPA, but as some random internet stranger once said in a comment section; Patterns are supposed to be descriptive, not prescriptive. (superpig, 2020) Yes, I was surprised as well when I realized patterns aren't those pure magic forces you have to adhere to, that I imagined them to be. Maybe those two patterns I mentioned earlier might somewhat describe where I could go with the Dicom context. I want to simplify the interface and maybe add a specific pinging behavior, so most if not all requests in the services can be greatly shortened and standardized.

I will not actually implement this idea during the IPA however, but will make a prototype and store it for later use, so the chances of me being able to implement everything in time before the IPA is over don't drop. This means I will only do a simple wrapper class containing all public methods, events and properties instead. This simpler implementation is probably best described as the proxy pattern. In the proxy pattern you usually inherit from the interface of the class you want to create a proxy from, meaning the proxy interface looks the same as the class it wraps. I also will only be able to implement the idea of making the class stateless in form of a prototype for now. I will go into further detail what this means in the next chapter.

8.2.5 THE QUESTION OF THE STATE

One of the main questions is whether this new client context should be stateless or stateful.

A stateful class stores information about things that happened in the past and when you call a method of the stateful class this stored information impacts the result of your method call. Here's an example:

```
public class IAmStateful {

    private int number;

    public IAmStateful(int number) {
        this.number = number;
    }

    public int square() {
        return number *= number;
    }

}
```

Code Snippet 9: Stateful class

Every time you call the `square()` method, the number field in the class gets changed and stored. Depending on how many times you have called the method, your result will be different. This makes sense in scenarios, where an object should store important information relevant to itself after it has been initialized.

A stateless class never stores information that happened in the past and when you call a method of a stateless class using the same parameters you will always get the same result. (kgiannakakis, 2011) Here's an example:

```
public class IAmStateless {

    public int square(int number) {
        return number * number;
    }

}
```

Code Snippet 10: Stateless class

Every time you call the `square()` method, the only number that gets used for calculating, is the number you supply as a parameter. This makes sense in scenarios, where the information required for executing the method isn't dependent on past calls of methods or changes in the class.

The way to find out whether the ClientContext needs to be stateful or stateless is by analyzing the places the DicomClient gets implemented. Like that we can figure out how many changes are being done to the client that influence the outcome of its method calls and if it's reasonable to supply that information as parameters instead.

Here's a list of all public methods, public fields and public events of the Dicom client that get used after the initialization of the DicomClient by the different ScanDiags services:

```
Task AddRequestAsync(DicomRequest dicomRequest);

Task AddRequestsAsync(IEnumerable<DicomRequest> dicomRequests);

Task SendAsync(CancellationToken cancellationToken = default, DicomClientCance
llationMode cancellationMode = DicomClientCancellationMode.ImmediatelyReleaseA
ssociation);

Logger Logger { get; set; }
```

Code Snippet 11: Used methods, fields and events from IDicomClient

Usually, the methods adding requests to the client get called right before calling the method to send the data to the server and the logger can easily be provided as a parameter. Now if a lot more fields and events of the client would have been used and changed by the services after the initialization of the client, making the Dicom client context stateless would have been problematic, since you'd have to add all the information, that slightly changes the behavior of the called method, over parameters for the methods. But because most necessary information gets provided over the constructor when the client gets initialized for the first time and all that information doesn't change, making it stateless makes sense.

However, if it were a reoccurring theme, that a lot of important information of the client gets changed between two method calls that influences the outcome, it'd wouldn't be a good idea to make it stateless. Here's an example of such a situation:

```
var client = new DicomClient(
    "someHost",
    0000,
    false,
    "thisService",
    "someServer"
);
await client.AddRequestAsync(dicomRequest1)
await client.SendAsync();
client.Options.RequestTimeout = new System.TimeSpan(1000);
client.Options.MaxCommandBuffer = 100;
await client.AddRequestAsync(dicomRequest2)
await client.SendAsync();
client.Options.RequestTimeout = new System.TimeSpan(800);
client.Options.MaxCommandBuffer = 150;
client.Options.TcpNoDelay = true;
await client.AddRequestAsync(dicomRequest3)
await client.SendAsync();
```

Code Snippet 12: Situation where stateful is good

If we'd want this client to be stateless, we'd need a method that has all possible parameters that change the behavior of the method, because this information wouldn't get stored in the client itself anymore. But that would mean we'd have to make a huge method with tons of parameters. Even if those parameters are declared as optional, this isn't a clean solution. Here's how such a method might look in action:

```
await client.AddAndSendAsync(
    DicomRequest,
    new System.TimeSpan(800),
    150,
    true,
    true,
    false,
    200);
```

Code Snippet 13: Bad usage of stateless

The reasons I combined add and send here is, that by adding the requests in a prior step, you already change the state of the class, therefore you'd have to combine those steps to make it stateless. The point however is, that because a lot of parameters are required to satisfy the needs of a hypothetical program, the usage of a stateless version of the class gets very messy.

If all that information stayed constant but you still wanted your custom values to be used, you could just pass it to the constructor when you first initialize, meaning this only is a problem when your client object constantly must change.

Luckily, the services I must modify for the IPA don't have changing clients and barely any advanced options of the client that gets set, so making it stateless is easy. The advantage of having it stateless is code readability, it allows you to combine a lot of steps and you don't have to worry about settings you changed in the client that might affect later usage of it. For example, you might change some setting for a request you make but then later completely forget about the changed setting, while you're still using the same client and this old setting interferes with your new request.

I won't actually make it stateless during the IPA yet, because for now the focus is making it possible to make a mock of a DicomClient and not to completely refactor the code in the services.

8.2.6 WHICH PARAMETERS DOES THE CONTEXT NEED

In code snippet 11 you can see that we only need the methods to add and send requests. For simplification we can simply combine those steps. Besides that, the only things that get used are the Logger and the CancellationToken. The DicomCancellationMode doesn't get used and is optional. The Logger constantly stays the same during the usage of the client. This means we can simply add that to the constructor. This leaves us with those two parameters we must provide for sending requests to a server.

```
IEnumerable<DicomRequest> requests,
CancellationToken cancellationToken
```

Code Snippet 14: Parameters needed for sending requests

By adding some pinging behavior, I described in chapter 7.2.4 and combining the option to send a single request or multiple requests at once we'd end up with a method called something like this:

```
public async Task EchoAndSendRequestsAsync(
            IEnumerable<DicomRequest> requests,
            CancellationToken cancellationToken);
```

Code Snippet 15: EchoAndSendRequestsAsync parameters

The reason I want to combine those two options is, because it simplifies the interface even more.

9 REALISIEREN – IMPLEMENT

9.1 PREFACE

In this chapter I will describe as much code as possible. Every secondary heading stands for the class I'm about to explain.

9.2 DICOMCLIENTCONTEXT.DICOMCONTEXT

```
public class DicomContext : IDicomClient
```

The DicomContext inherits from the IDicomClient. This makes sense, because the DicomContext must wrap every public property, event and method of the Dicomclient described in its interface. This enables me to use the DicomContext in places where an object of the type IDicomClient is expected.

```
private readonly Dicom.Network.Client.DicomClient dicomClient;
```

This is the private class field in which the actual wrapped DicomClient gets stored. The reason the whole namespace is written before `.DicomClient` is because FO-Dicom has an obsolete version of the DicomClient in `Dicom.Network` and if I don't give the specific namespace the compiler won't know which one I want.

I could alternatively set an alias for the namespace to shorten the code a bit, but I'd rather keep it like this, because there aren't too many occurrences of this problem in the class and like this it's more obvious for other developers where this type is coming from, without having to check what the alias stands for.

```
public DicomContext(string host,
    int port, bool useTls, string callingAe, string calledAe,
    int associationRequestTimeoutInMs = DicomClientDefaults.DefaultAssociation
RequestTimeoutInMs,
    int associationReleaseTimeoutInMs = DicomClientDefaults.DefaultAssociation
ReleaseTimeoutInMs,
    int associationLingerTimeoutInMs = DicomClientDefaults.DefaultAssociationL
ingerInMs,
    int? maximumNumberOfRequestsPerAssociation = null)
    {
    this.dicomClient = CreateClient(
        host,
        port,
        useTls,
        callingAe,
        calledAe,
        associationRequestTimeoutInMs,
        associationReleaseTimeoutInMs,
        associationLingerTimeoutInMs,
        maximumNumberOfRequestsPerAssociation);
    }
```

Code Snippet 16: DicomContext constructor

This is the constructor. It takes all required parameters for initializing a DicomClient and passes it on to the CreateClient method.

9.2.1 CREATECLIENT METHOD

```
private static Dicom.Network.Client.DicomClient CreateClient(params object[] a
rgs) {
```

This is the start of the CreateClient method. It takes multiple parameters but interprets them as an array of objects. The object type is something every type inherits from, allowing you to bundle all kinds of different types, e.g., strings and integers, in one single list or array. Usually this isn't recommendable, but it's alright because it's a private method and human error is therefore prevented since nobody will be able to give the parameters in a wrong manner because the constructor sorts them correctly. The name `args` stands for arguments.

```
foreach (object arg in args) {
```

Now we loop through each parameter and call the current argument we are looping through `arg`.

```
if (arg is string paramAsString && string.IsNullOrWhiteSpace(paramAsString)) {
```

If one of those parameters is both a string and this string has either the value null, is empty or consists only of white spaces it will do the following:

```
throw new ArgumentNullException("One of the string parameters is null, empty,
or consists only of white-space.");
```

It throws an exception explaining the problem.

After checking whether the strings are valid and throwing an exception if one of them isn't, the method simply initializes a new DicomClient using the array of parameters and casting the right types to the objects inside of the array:

```
var client = new Dicom.Network.Client.DicomClient(
     (string)args[0],
     (int)args[1],
     (bool)args[2],
     (string)args[3],
     (string)args[4],
     (int)args[5],
     (int)args[6],
     (int)args[7],
     (int?)args[8]);
return client;
```

Because the wrapping of the methods, properties and events are repetitive I will only give one example for each.

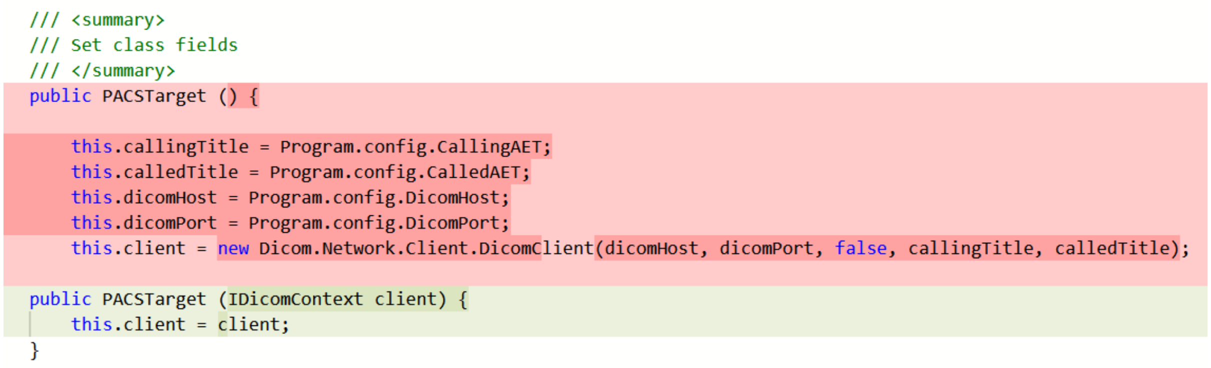
```
public Logger Logger {
    get {
        return dicomClient.Logger;
    }
    set {
        dicomClient.Logger = Logger;
    }
}
```

Here I created the Logger that matches the DicomClients interfaces Logger but it gets the Logger of the privately wrapped DicomClient when you want to retrieve the DicomContexts Logger property and sets the DicomClient Logger property if you give the DicomContexts property a value. This is done by the get and set methods, which are also called setters and getters. There are some properties that have no setters in the actual DicomClient, therefore there isn't one in the DicomContext either for those.

9.3 DICOMGENERATOR.PACSTARGET

9.3.1 CONSTRUCTOR

Originally the DicomClient was initialized in the constructor. I already explained in chapter 7.2.1 how this is bad practice and disables us from injecting a mock into the class when we try to unit test it. The solution is as I explained dependency injection. This is how the change looks when you compare the old version on git with the new modified one.



```
/// <summary>
/// Set class fields
/// </summary>
public PACSTarget () {

    this.callingTitle = Program.config.CallingAET;
    this.calledTitle = Program.config.CalledAET;
    this.dicomHost = Program.config.DicomHost;
    this.dicomPort = Program.config.DicomPort;
    this.client = new Dicom.Network.Client.DicomClient(dicomHost, dicomPort, false, callingTitle, calledTitle);

public PACSTarget (IDicomContext client) {
    this.client = client;
}
```

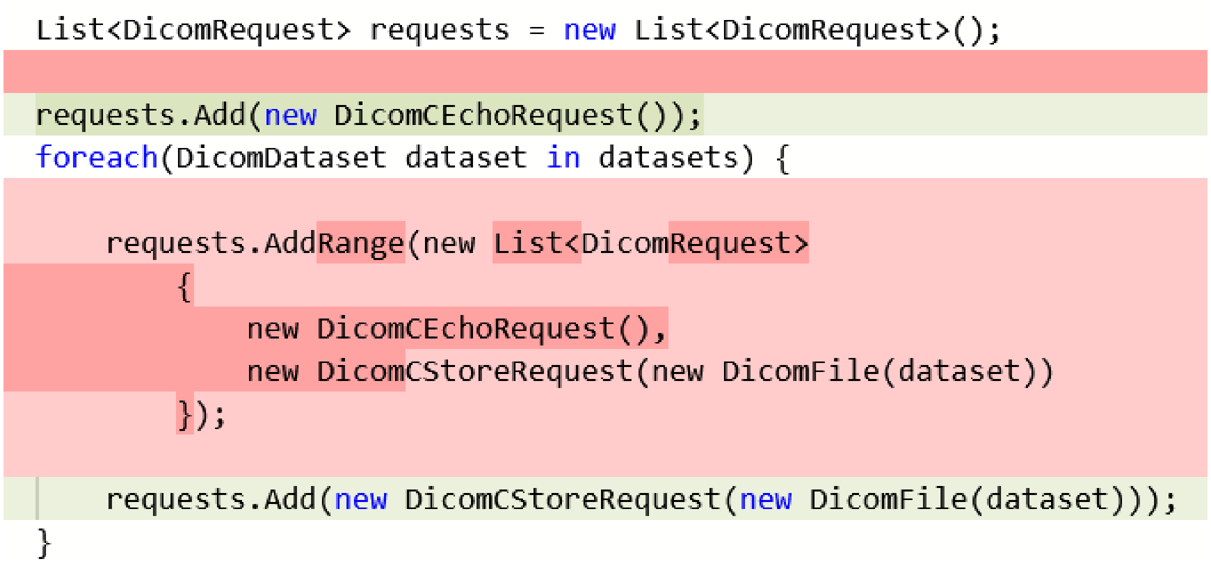
Figure 6: DicomGenerator.PACSTarget constructor git changes

The red part is what I removed and the green one is what I added. As you can see the modification is very straight forward and allows us to inject the Dicom context when we initialize the PACSTarget:

```
return new PACSTarget(
    new DicomContext(
        Program.config.DicomHost,
        Program.config.DicomPort,
        false,
        Program.config.CallingAET,
        Program.config.CalledAET
    )
);
```

In the `Save` method of the PACSTarget I added a simple check that makes sure the parameter `datasets` isn't null before it proceeds. This is in general a good practice, because it allows you to catch the error early on and gives you the possibility to give a more detailed error message. (Bayers, 2012) If you wouldn't do the null check for parameters that shouldn't be null and the parameter provided is null, an error would occur later in the code where it'd be used. Waiting for that error to occur is a waste of time and the description of that error might not be as straight forward.

I also changed the pinging behavior by only adding one DicomCEchoRequest to each batch of DicomDatasets. This is how the changes to the `Save` method look like after the null check:



```
List<DicomRequest> requests = new List<DicomRequest>();

requests.Add(new DicomCEchoRequest());
foreach(DicomDataset dataset in datasets) {

    requests.AddRange(new List<DicomRequest>
        {
            new DicomCEchoRequest(),
            new DicomCStoreRequest(new DicomFile(dataset))
        });

    requests.Add(new DicomCStoreRequest(new DicomFile(dataset)));
}
```

Figure 7: DicomGenerator.PACSTarget Save method git changes

9.4 DICOMGENERATOR.PACSTARGETTEST

This is a unit tests class. Let me go through the code.

9.4.1 SAVETEST

The basic premise of this test method is, that it injects a mocked DicomContext and runs the `Save` method just to check if it doesn't throw any errors.

```
[Fact]
public async Task SaveTest() {
```

This is the start of the first test method and as you can see it has the attribute [Fact]. This is an attribute provided by the xUnit and it signifies, that this is in fact a test method and will run once without any parameters.

```
var dicomContextMock = new Mock<IDicomContext>();
```

Here I create the DicomContext mock by using the IDicomContext interface. I describe how mocks work in chapter 7.2.1.2.

```
var testDatasetList = new List<DicomDataset>(){
                new DicomDataset(),
                new DicomDataset(),
                new DicomDataset(),
                new DicomDataset(),
                new DicomDataset()
            };
```

This is the list filled with DicomDatasets, that I will pass over to the `Save` function as a parameter. The reason that function needs DicomDatasets is because those datasets are the data we want to send to a SCP. All we must do first is pack those datasets into requests. Usually DicomDatasets contain a lot of information and some of it is mandatory. Luckily when initializing new DicomDatasets the FO-Dicom library doesn't check if that mandatory information is in the datasets yet, so for unit testing we can just insert empty DicomDatasets into the list. If we required DicomDatasets containing the mandatory information I'd have to save those DicomDatasets inside of DicomFiles and load those when starting the unit test. There are alternatives to loading test data from a file, but it is the most straight forward solution in this case.

```
var pacsTarget = new PACSTarget(dicomContextMock.Object);
```

In this line of code I simply initialize a new `PACSTarget` and inject the mocked Dicom context.

The last line of the `TestSave` method simply runs the `Save` method providing the list of empty DicomDatasets as fodder:

```
await pacsTarget.Save(testDatasetList);
```

9.4.2 SAVEDATASETSISNULLTEST

Just like the test method before it, this one also has the [Fact] attribute:

```
[Fact]
public async Task SaveDatasetsIsNullTest() {
```

And I inject a Dicom context mock here as well:

```
var dicomContextMock = new Mock<IDicomContext>();
var pacsTarget = new PACSTarget(dicomContextMock.Object);
```

This time however I want to test if it throws an error if I give null as a parameter instead of DicomDatasets. This error should be an ArgumentNullException. There are a lot of different kinds of exceptions, but this one matches this specific situation.

```
await Assert.ThrowsAsync<ArgumentNullException>(async () => {
                await pacsTarget.Save(null);
            });
```

`Assert.ThrowsAsync` simply checks if the given exception in the <> brackets matches the one being thrown by the method. The part that comes afterwards is called a lambda. A lambda is basically a block of code you can pass in as a parameter of a method. (Fowler, 2004) You might be wondering what the Async in ThrowsAsync means and the what the `await` is all about. Those are important keywords in asynchronous programming. I will not explain in detail what they do in this IPA, but it basically allows a part of code to not block the rest of the Program. So during the time a method is *awaiting* an *async*hronous task to finish other parallel running tasks can still run.

This was already the last line of code for this test so let's move on to the next.

9.4.3 SAVEDATASETSCOUNTEQUALSREQUESTCOUNTTEST

The basic idea of this test is, that we give the `Save` method a certain amount of DicomDatasets to pack inside of requests and then we check if the amount of requests – 1 (ping request) matches the amount of DicomDatasets we provided the `Save` method with.

```
[Fact]
public async Task SaveDatasetsCountEqualsRequestCountTest() {
```

You might be getting tired of seeing the [Fact] attribute, but facts don't care about your feelings.

Anyways, we start off by defining the following variable:

```
int actualRequestsNum = 0;
```

In this variable we will store the amount of store requests that will get passed to the `AddRequestsAsync` method.

```
var dicomContextMock = new Mock<IDicomContext>();
dicomContextMock
    .Setup(
        context => context.AddRequestsAsync(It.IsAny<List<DicomRequest>>()))
            .Callback<IEnumerable<DicomRequest>>((requests) => {
                foreach(var request in requests) {
                    ++actualRequestsNum;
                }
                --actualRequestsNum;
            });
```

As you can see above we continue to create a Dicom context mock and setting it up in a way, that if the AddRequestsAsync method gets called, the list of requests gets looped through and for each request it increments the `actualRequestsNum` by one, but removes one after the looping is finished, so the request responsible for pinging doesn't get count, because we will only want to compare amount of store requests with the amount of DicomDatasets we provided the method with.

Like with all previous method we inject the Dicom context mock and like in the `SaveTest` method we create a list of DicomDatasets, which we count and then add the length of this list as an integer to the `expectedRequestsNum.` I won't add the respective code-snippets here because those steps are straight forward.

We run the `Save` method and then compare the expected number of requests with the actual number of requests like so:

```
Assert.Equal(expectedRequestsNum, actualRequestsNum);
```

9.5 DICOMTOPACSLOADER.DICOMTOPACSLOADER

The changes to this service are very similar to the ones I mentioned in DicomGenerator.PACSTarget, therefore I won't show the modifications here again. I simply added dependency injection for the Dicom context.

9.6 DICOMTOPACSLOADER.TOPACSLOADERTEST

Because the `Load` method, the one I have to unit test is extremely complex for being a single method, I couldn't write a working unit test for it. This is one of the reasons for keeping methods simpler. Of course it rarely makes sense to make a one-liner method and that's not what I'm implying. It's important to find a balance of how complex a unit of code should be, but it should be easily digestible for other developers and it should be possible to write unit tests for the code you write. (mortalapeman, 2014)

After the IPA is over, me or a coworker will have to refactor the `Load` method to enable unit testing. I was able to create something that would work if it wouldn't require hours of debugging to get it running. I will add the code of the unfinished unit test in the appendix, but won't go into further detail, because explaining it would require giving a lot of context irrelevant to the IPA.

9.7 REPORTSENDER.PACSTARGET

The modification of this service is very similar to the changes in DicomGenerator.PacsTarget. The only difference is that I'm not able to do dependency injection over the constructor, but instead must pass in the Dicom client context as a parameter in the method where it gets used:

```
public async Task SendReports(List<DicomDataset> reports, IDicomContext client
) {
```

The other modifications are too similar to the ones in the DicomGenerator to explain here again, but the above-mentioned change does have an effect to the unit tests.

9.8 REPORTSSENDER.TEST

9.8.1 CONSTRUCTOR AND DISPOSE

```
public class ReportSenderTest : IDisposable {
```

As you can see the class for the unit tests inherits from the IDisposable interface. The only method that this interface defines is the following.

```
void Dispose();
```

This method gets called after the unit tests you executed are done and you can add whatever code you want in it. It's very useful if you want to reset things to the way they were if a unit test changed something important. I use it for doing the following.

```
public void Dispose() {
    testDatasetList = null;
}
```

If you're wondering what that variable comes from, it's a class property:

```
private List<DicomDataset> testDatasetList;
```

The reason I set it to null in the `Dispose` method is, because it generally makes sense to keep test classes clean and clear things you don't need any more after the unit tests are done. It may be useless for this specific property though.

I initialize the `testDatasetList` property in the constructor:

```
public ReportSenderTest() {
    testDatasetList = new List<DicomDataset>();
    DirectoryInfo testFilesDir = new DirectoryInfo(
    "..\\..\\..\\testFiles\\");
    foreach (var file in testFilesDir.GetFiles("*.dcm"))
    {
        testDatasetList.Add(DicomFile.Open(file.FullName).Dataset);
    }
}
```

Code Snippet 17: ReportSenderTest constructor

In the constructor I first initialize the `testDatasetList` property, then I initialize a new `DirectoryInfo` object pointing to the folder containing test Dicom files and then I loop through that folder and for every file that ends with `.dcm` I extract its dataset and add it to the class property `testDatasetList`. The test files were generated using the `DicomGenerator` and contain mostly random values instead of sensitive information.

The unit tests are almost identical to those in the `DicomGenerator.PacsTargetTest` class. One difference though is that I use the datasets with fake information as shown above, because it would otherwise throw an error when the DicomGenerator tries to convert the DicomDatasets to DicomFiles if I would just use empty DicomDatasets instead. Another major difference is, as I described earlier, that the Dicom client context has to be passed as a parameter to the method instead of the constructor. This is also how I inject the mock:

```
await pacsTarget.SendReports(testDatasetList, dicomContextMock.Object);
```

9.9 IDICOMCLIENT

I mentioned earlier that I had no write access to the FO-Dicom library, so you might be wondering how I did changes to it then.

Well, it started off when I went on FO-Dicoms Gitter page, a public website where people discuss GitHub projects and asked the following question:

Me, Apr 16 14:35

Hi, just a question out of curiosity. The IDicomClient interface doesn't define all properties of the DicomClient. Namely it misses the Host, Port, UseTls, CallingAe, CalledAe and IsSendRequired properties. Is there some specific reason for this? I just realized when I wanted to do dependency injection. It's not a problem because there's workarounds for that, but I'm just wondering

I received an answer to this question by one of the main developers of the FO-Dicom library:

Reinhard Gruber, Apr 16 19:56

IDicomClient is an older interface. When it was created, these properties you mention have not been properties of DicomClient, but they were passed as parameter of the method SendAsync. This made it possible to use the same DicomClient instance for several different connections, which caused errors. So we changed the behavior in that way that the connection parameters are now passed via constructor, so having one DicomClient per connection. With that step the properties you mentioned where added to DicomClient class, but we did not forward it do IDicomClient interface for no specific reason. If you would need it in interface, then feel free to create a pull request and add them. (Gruber, 2021)

This basically means, that the properties aren't intentionally missing in the interface and just weren't added, when there were changes to the DicomClient. It wasn't a hassle for me to add some properties to an interface and it would be beneficial for the IPA because it would allow me to change the inheritance of the DicomContext to the libraries IDicomClient interface. That would make our code a lot less redundant, because it doesn't make sense to have multiple interfaces that look pretty much the same.

Me, Apr 16 22:01

Thank you for explaining @gofal ! I've created a PR with the added properties

9.10 PLAYGROUND.DICOMFACADE

This class is just a prototype for potential future use. Since the IDicomClient interface has been fixed by me, there is little purpose for the DicomContext since there is no difference in its IDicomContext and the IDicomClients interface. But having implemented the DicomContext in form of the proxy pattern creates some independence, better exception handling and is a good steppingstone for future improvement, for example in case we decide to implement a Dicom client facade like my prototype.

The prototype starts with the following line:

```
public class DicomFacade : IDicomFacade {
```

It inherits from the IDicomFacade. I created this interface to enable dependency injection if the DicomFacade ever gets used. In general it makes sense to make interfaces for all important classes that get used by other classes so you can mock in your unit tests. This isn't the primary purpose of interfaces though and maybe at some point in the future one will be able to mock things without needing an interface.

```
private readonly DicomClient dicomClient;
```

This is the wrapped DicomClient. All calls will eventually be redirected to this client, even though the interface of the facade looks different.

The constructor of the DicomFacade looks almost the same as the on from the DicomContext. The only difference is that a Logger is also provided as a parameter of the constructor. This partially enables the class to be stateless, because I don't have to add a property in which you can set the Logger from outside. In the `CreateClient()` method I added the following line:

```
if (args[5] != null) client.Logger = (Logger)args[5];
```

`args[5]` is the Logger handed down as a parameter of the constructor. So, if this logger isn't null, I set the privately wrapped client's logger as the one from the constructor.

The real magic happens in the following method:

```
public async Task EchoAndSendRequestsAsync(IEnumerable<DicomRequest> requests,
            CancellationToken cancellationToken)
        {
```

As you can see from the method name it is responsible to ping the SCP first and then send a list of requests. This is beneficial in the sense, that it would greatly unify and simply the implementations of the interface, as I explained earlier. The list of requests is of the type IEnumerable because this enables any kind of list or array to be provided as a parameter, so nobody has to convert it to a specific type of collection first.

```
var echoRequest = new DicomCEchoRequest();
```

This is the first line of the method. It initializes a new DicomCEchoRequest. A C-echo request serves as a kind pinging functionality.

```
echoRequest.OnResponseReceived = (request, response) => {
    dicomClient.Logger.Info("C-
Echo request to PACS with IP: {ip}, Port: {port}, AE-
Title: {ae} with status {status}", dicomClient.Host, dicomClient.Port, dicomCl
ient.CalledAe, response.Status.ToString());
};
echoRequest.OnTimeout = (request, response) => {
    dicomClient.Logger.Error("C-
Echo request to PACS with IP: {ip}, Port: {port}, AE-
Title: {ae} timed out", dicomClient.Host, dicomClient.Port, dicomClient.Called
Ae);
    throw new DicomNetworkException("C-Echo request timed out");
};
```

Code Snippet 18: echo request OnResponseReceived

As you can see I add a lambda method to the `OnResponseReceived` event of the C-echo request, that simply logs the response details and another lambda method to the `OnTimeout` event that logs some information and throws an error. I partially copied the code from a previous implementation of the Dicom client written by Michael Grossmann, a former co-worker. The parameters both lambda method receive are the request itself and the response it received after having been sent.

The rest of the method simply looks like the following for now:

```
await this.dicomClient.AddRequestsAsync(requests);
await this.dicomClient.SendAsync(cancellationToken);
```

This should be self-explanatory by now. It is important to note though, that FO-Dicoms DicomClient treats requests added with `AddRequestsAsync()` and `AddRequestAsync()` differently on first look. Besides only being able to add a singular request per call of the `AddRequestAsync()` method I want to point out something else:

`AddRequestAsync()`:

```
State.AddRequestAsync(dicomRequest);
```

`AddRequestsAsync()`:

```
foreach (var request in requests)
    await State.AddRequestAsync(request).ConfigureAwait(false);
```

As you can see the `AddRequestsAsync()` calls `ConfigureAwait(false)` on every request that gets added. The `ConfigureAwait()` method is available to every asynchronous task and it determines on which Thread the program should continue on. When developers create libraries it's important for them to set it to `false` for asynchronous task to prevent deadlocks. This isn't important to the IPA so I won't explain in detail what this means, but it's relevant to me to know, that the requests don't really get handled differently in the two methods besides one of them being safer against deadlocks, so it's okay to just use the `AddRequestsAsync()` even for single requests.

9.11 PLAYGROUND.PLAYGROUNDCLIENT

This is a class with which I tested my DicomFacade. It has a private field for the DicomFacade object it uses:

```
private IDicomFacade dicomFacade;
```

And it has a constructor in which this DicomFacade gets initialized:

```
public PlayGroundClient() {
            // set up connection with server
            var host = "127.0.0.1";
            var port = 55555;
            var callingAE = "PlayGroundClient";
            var calledAe = "PlayGroundServer";
            dicomFacade = new DicomFacade(host, port, false, callingAE, called
Ae);
        }
```

Code Snippet 19: DicomFacade constructor

The reason I don't do any fancy dependency injection is because this is a throw-away class that will never join in on the develop branches party, so it's unimportant to make it unit testable. As you can see, I use real values for the Dicom SCP information like for example the host IP. The reason is because I have an actual Dicom SCP with which I can test things with.

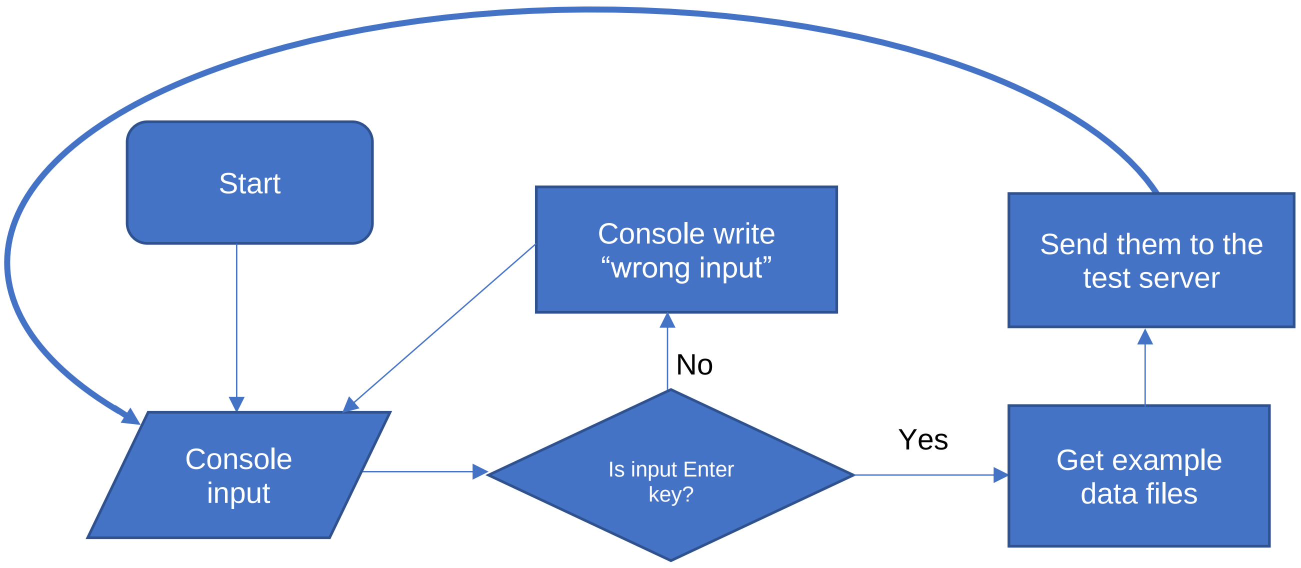
```
public async Task sendDicoms(DicomFile[] dicomFiles, CancellationToken cancell
ationToken) {
            var storeRequests = new List<DicomCStoreRequest>();
            foreach(var dicomFile in dicomFiles) {
                storeRequests.Add(new DicomCStoreRequest(dicomFile, DicomPrior
ity.Medium));
            }
            await dicomContext.EchoAndSendRequestsAsync(storeRequests, cancell
ationToken);
        }
```

Code Snippet 20: sendDicoms method

This is the method in which the test data gets sent to the test server. It creates a list of C-store requests and adds every DicomFile provided through the parameters by first converting them to C-store requests. It then calls the `EchoAndSendRequestsAsync()` method of the Dicom facade with the requests as a parameter.

9.12 PLAYGROUND.PROGRAM

Because the code of the test Dicom server requires a lot of knowledge of Dicom that is irrelevant to the IPA I won't explain it in detail. I will however show how I test the facade by making a flowchart of how the program operates.



Now all I have to do is debug the program and check what kind of responses I get to figure out if the facade works. Turns out it does.

10 KONTROLLIEREN – CONTROL

10.1 PLANNED TESTS

Test ID	Result and test method or tested class	Person who tested	Date
1	TestAddRequestAsyncNullAsParam()	Gabriel S.	19.04.2021
2	Not implemented	Gabriel S.	19.04.2021
3	Not implemented	Gabriel S.	19.04.2021
4	Not implemented	Gabriel S.	19.04.2021
5	AutomatedEndToEndTest()	Gabriel S.	19.04.2021
6	SaveDatasetsIsNullTest()	Gabriel S.	19.04.2021
7	SaveDatasetsCountEqualsRequestCountTest()	Gabriel S.	19.04.2021
8	LoadTest()	Gabriel S.	19.04.2021
9	Not implemented	Gabriel S.	19.04.2021
10	SaveDatasetsIsNullTest()	Gabriel S.	19.04.2021
11	SaveDatasetsCountEqualsRequestCountTest()	Gabriel S.	19.04.2021
12	SaveDatasetsIsNullTest()	Gabriel S.	19.04.2021
13	SaveDatasetsCountEqualsRequestCountTest()	Gabriel S.	19.04.2021
14	DicomGenerator	Gabriel S.	19.04.2021
15	DicomToPacsLoader (only tested pinging)	Gabriel S.	19.04.2021
16	JobReceiver	Gabriel S.	19.04.2021
17	ReportSender	Gabriel S.	19.04.2021

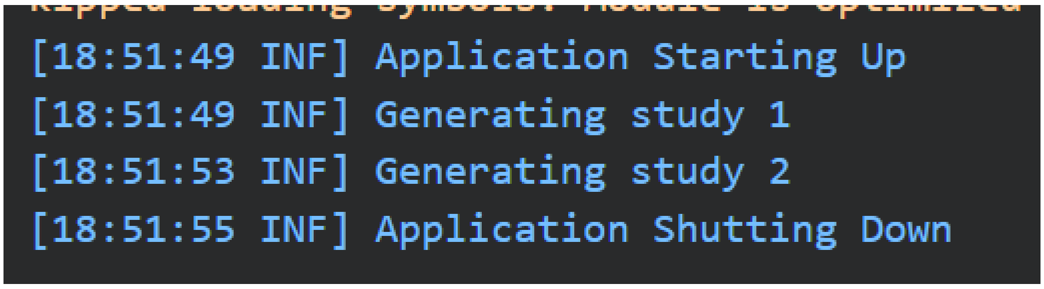
Table 5: Test results

When the field in the above table is green it was a success, if it's Orange it hasn't been implemented and if its red the result didn't meet the expectations. All unplanned unit tests succeeded, but, due to being less important, aren't in the table.

11 AUSWERTEN – EVALUATE

11.1 DICOMGENERATOR

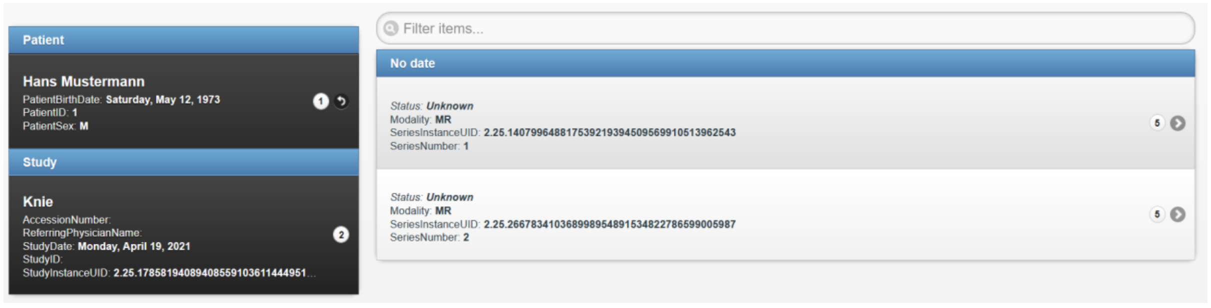
I tested the DicomGenerator by setting the target of the DicomGenerator to my local Orthanc server. The Orthanc is a PACS (Picture Archiving And Communication System) which serves as a Dicom SCP and SCU. I then ran the DicomGenerator service and got the following logs documenting my success:



```
[18:51:49 INF] Application Starting Up
[18:51:49 INF] Generating study 1
[18:51:53 INF] Generating study 2
[18:51:55 INF] Application Shutting Down
```

Figure 8: DicomGenerator proof of test

Then I checked if the data arrived at my PACS and I wasn't disappointed:



Hans Mustermann
PatientBirthDate: Saturday, May 12, 1973
PatientID: 1
PatientSex: M

Knie
AccessionNumber:
ReferringPhysicianName:
StudyDate: Monday, April 19, 2021
StudyID:
StudyInstanceUID: 2.25.17858194089408559103611444951...

Status: Unknown
Modality: MR
SeriesInstanceUID: 2.25.140799648817539219394509569910513962543
SeriesNumber: 1

Status: Unknown
Modality: MR
SeriesInstanceUID: 2.25.266783410368998954891534822786599005987
SeriesNumber: 2

Figure 9: DicomGenerator result

Two studies arrived, just as I anticipated it.

11.2 JOBRECEIVER

I tested the JobReceiver by starting it up and then going into my Orthanc. From there I sent a generated test study to the JobReceiver by selecting it as the target:

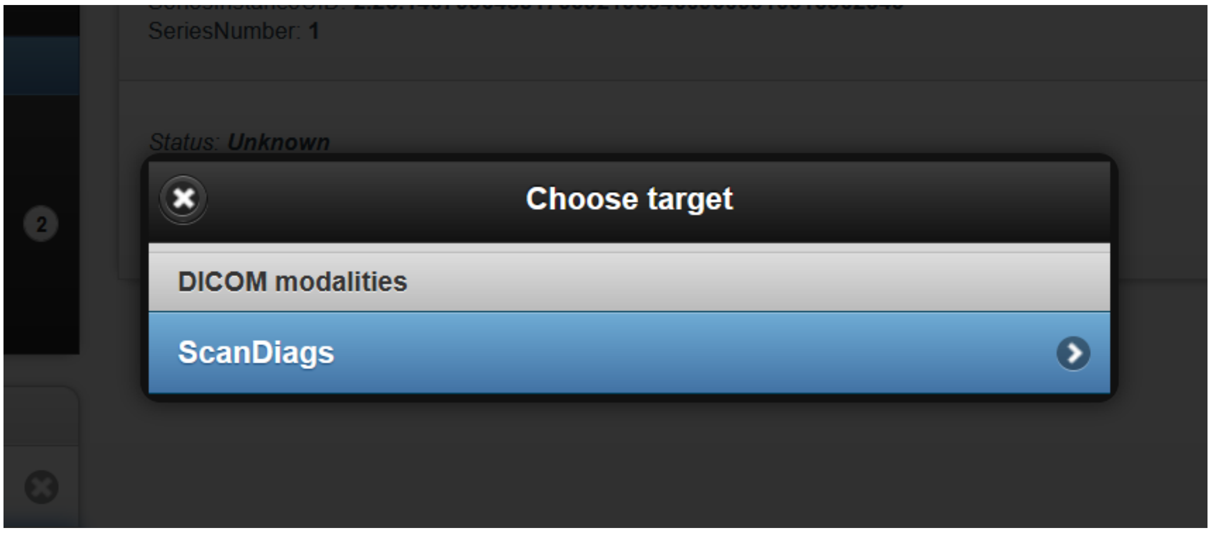
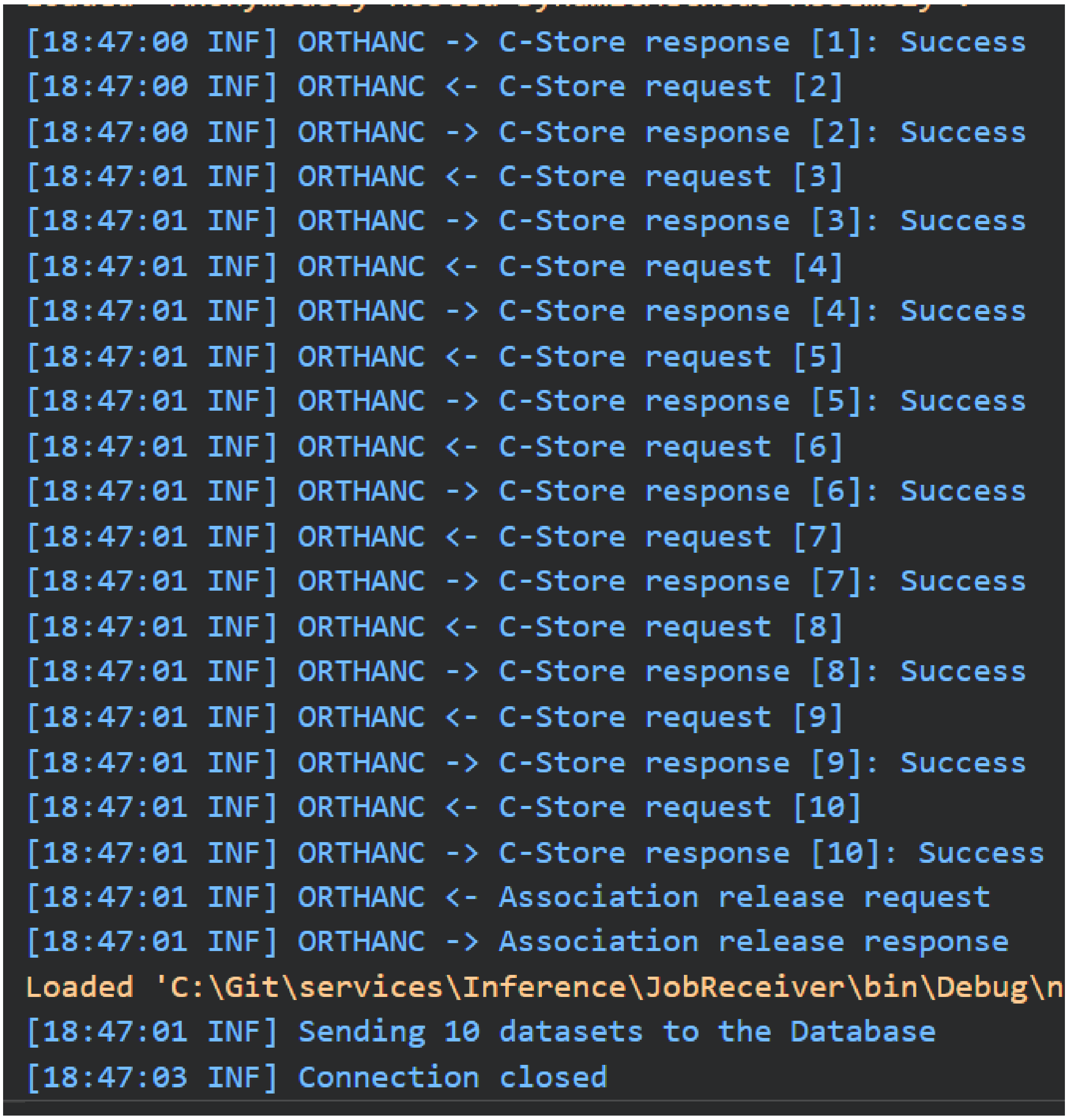


Figure 10: JobReceiver way of testing

After I clicked on the target ScanDiags it sent it to the JobReceiver and the logs showed a full-blown success:

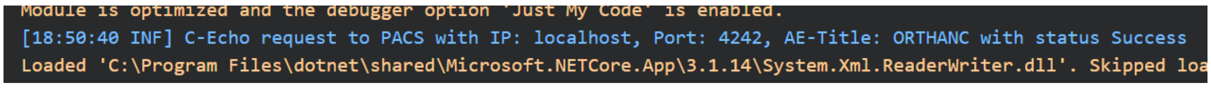


```
[18:47:00 INF] ORTHANC -> C-Store response [1]: Success
[18:47:00 INF] ORTHANC <- C-Store request [2]
[18:47:00 INF] ORTHANC -> C-Store response [2]: Success
[18:47:01 INF] ORTHANC <- C-Store request [3]
[18:47:01 INF] ORTHANC -> C-Store response [3]: Success
[18:47:01 INF] ORTHANC <- C-Store request [4]
[18:47:01 INF] ORTHANC -> C-Store response [4]: Success
[18:47:01 INF] ORTHANC <- C-Store request [5]
[18:47:01 INF] ORTHANC -> C-Store response [5]: Success
[18:47:01 INF] ORTHANC <- C-Store request [6]
[18:47:01 INF] ORTHANC -> C-Store response [6]: Success
[18:47:01 INF] ORTHANC <- C-Store request [7]
[18:47:01 INF] ORTHANC -> C-Store response [7]: Success
[18:47:01 INF] ORTHANC <- C-Store request [8]
[18:47:01 INF] ORTHANC -> C-Store response [8]: Success
[18:47:01 INF] ORTHANC <- C-Store request [9]
[18:47:01 INF] ORTHANC -> C-Store response [9]: Success
[18:47:01 INF] ORTHANC <- C-Store request [10]
[18:47:01 INF] ORTHANC -> C-Store response [10]: Success
[18:47:01 INF] ORTHANC <- Association release request
[18:47:01 INF] ORTHANC -> Association release response
Loaded 'C:\Git\services\Inference\JobReceiver\bin\Debug\n
[18:47:01 INF] Sending 10 datasets to the Database
[18:47:03 INF] Connection closed
```

Figure 11: JobReceiver proof of test

11.3 DICOMTOPACSLOADER

I only tested the pinging mechanism, because it does that automatically and I thought it would be a clear enough indication, that the DicomContext actually works. Here are the log results for when I set the target for the pinging to the local Orthanc server and started the service:

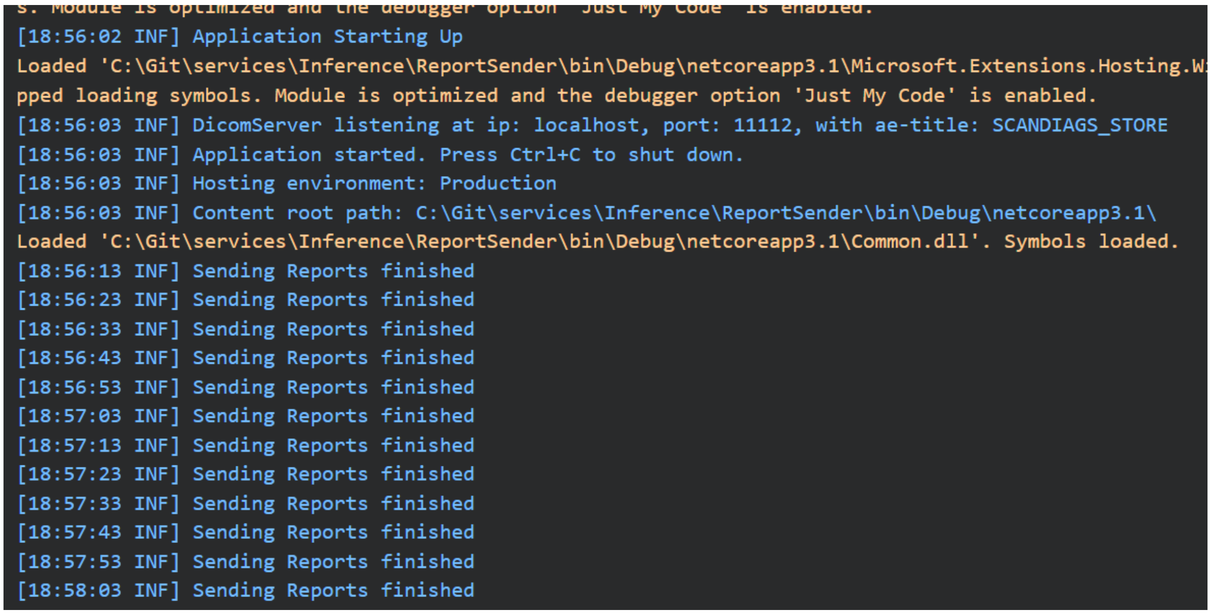


```
Module is optimized and the debugger option 'Just My Code' is enabled.
[18:50:40 INF] C-Echo request to PACS with IP: localhost, Port: 4242, AE-Title: ORTHANC with status Success
Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.1.14\System.Xml.ReaderWriter.dll'. Skipped loa
```

Figure 12: DicomToPacsLoader proof of test

11.4 REPORTSENDER

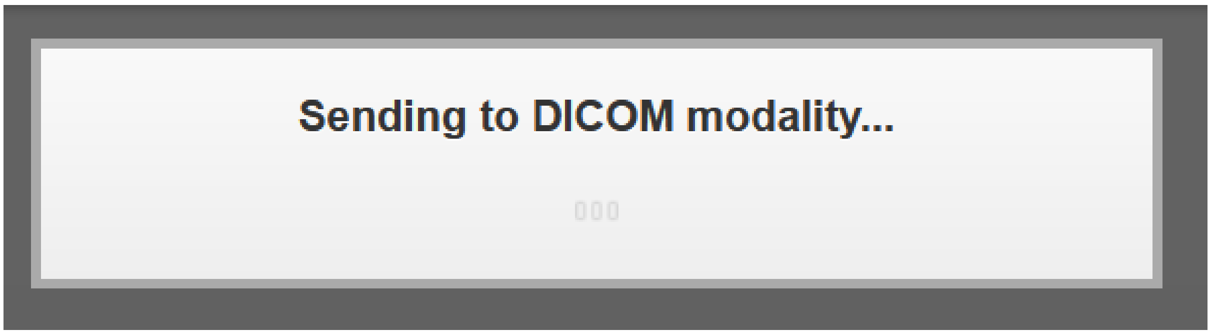
The report sender showed some very odd behavior. Even though the configuration was pretty much the same as of the JobReceiver, all it really did is log that sending the reports was finished, which I don't believe at all:



```
s. Module is optimized and the debugger option 'Just My Code' is enabled.
[18:56:02 INF] Application Starting Up
Loaded 'C:\Git\services\Inference\ReportSender\bin\Debug\netcoreapp3.1\Microsoft.Extensions.Hosting.W
pped loading symbols. Module is optimized and the debugger option 'Just My Code' is enabled.
[18:56:03 INF] DicomServer listening at ip: localhost, port: 11112, with ae-title: SCANDIAGS_STORE
[18:56:03 INF] Application started. Press Ctrl+C to shut down.
[18:56:03 INF] Hosting environment: Production
[18:56:03 INF] Content root path: C:\Git\services\Inference\ReportSender\bin\Debug\netcoreapp3.1\
Loaded 'C:\Git\services\Inference\ReportSender\bin\Debug\netcoreapp3.1\Common.dll'. Symbols loaded.
[18:56:13 INF] Sending Reports finished
[18:56:23 INF] Sending Reports finished
[18:56:33 INF] Sending Reports finished
[18:56:43 INF] Sending Reports finished
[18:56:53 INF] Sending Reports finished
[18:57:03 INF] Sending Reports finished
[18:57:13 INF] Sending Reports finished
[18:57:23 INF] Sending Reports finished
[18:57:33 INF] Sending Reports finished
[18:57:43 INF] Sending Reports finished
[18:57:53 INF] Sending Reports finished
[18:58:03 INF] Sending Reports finished
```

Figure 13: Report sender odd behavior

I tried testing it by sending a text report to the service using Orthanc, but Orthanc simply got stuck trying to send it to the service:



Sending to DICOM modality...

Figure 14: Orthanc stuck sending

I'm quite sure the configuration was correct and the data I tried sending to the service wasn't corrupt, but there's still a possibility I made a mistake somewhere. I switched to the develop branch instead, where my DicomContext isn't implemented yet and tried the same steps as I described above and got the same results. This means the problem doesn't lie in the DicomContext and the service needs a general debugging. My IPA isn't about debugging services as a whole though, so I will leave it at that for now.

11.5 UNIT TESTS

I can't give a conclusion and write about the repercussions of every single unit test because there's not much to say about them and I already documented how they work and why one of them failed. Some of them weren't implemented, because they didn't say much about whether the method they tested would work in a realistic setting.

APPENDIX

12 GLOSSARY

Dependency Injection	When you give an object another object it depends on through the constructor. (Wikipedia, 2021)
FO-Dicom	Fellow Oak Dicom is a library under the Microsoft Public License and used in all C# ScanDiags services.
Interface	Either a point where two subjects interact or a type a class can inherit from e.g., a VW can inherit of the type *car*.
Mock	A fake object resembling a real one, that doesn't do anything when you call the same methods.
Orthanc	A PACS application.
PACS	Picture Archiving and Communication System. Often used in hospitals to manage radiological data.
Pattern	A reusable software design solution to a commonly occurring problem. (Wikipedia, 2021)
SCP	Service Class Provider (Kind of a server)
SCU	Service Class User (Kind of a client)

13 TABLE OF FIGURES

Table 1: IPERKA 7
Table 2: Test descriptions 29
Table 3: Test objects 31
Table 4: Test cases overview 32
Table 5: Test results 62

Figure 1: Adapter example 27
Figure 2: Unit test without mocking client 41
Figure 3: Unit test with mocking client 41
Figure 4: DicomContext concept 44
Figure 5: Client and Context, two interfaces, one problem 45
Figure 6: DicomGenerator.PACSTarget constructor git changes 52
Figure 7: DicomGenerator.PACSTarget Save method git changes 53
Figure 8: DicomGenerator proof of test 63
Figure 9: DicomGenerator result 63
Figure 10: JobReceiver way of testing 63
Figure 11: JobReceiver proof of test 64
Figure 12: DicomToPacsLoader proof of test 64
Figure 13: Report sender odd behavior 65
Figure 14: Orthanc stuck sending 65

Code Snippet 1: Bad idea 1 42
Code Snippet 2: Bad idea 2 42
Code Snippet 3: Dependency injection 43
Code Snippet 4: Creating and injecting a mock 43
Code Snippet 5: Injecting the DicomContext 43
Code Snippet 6: Mocking the DicomContext 43
Code Snippet 7: DicomGenerator.PACSTarget.Save() 45
Code Snippet 8: JobReceiver.PacsTarget.SendReports() 46
Code Snippet 9: Stateful class 47
Code Snippet 10: Stateless class 47
Code Snippet 11: Used methods, fields and events from IDicomClient 48
Code Snippet 12: Situation where stateful is good 48
Code Snippet 13: Bad usage of stateless 49
Code Snippet 14: Parameters needed for sending requests 49
Code Snippet 15: EchoAndSendRequestsAsync parameters 49
Code Snippet 16: DicomContext constructor 50

Code Snippet 17: ReportSenderTest constructor 56
Code Snippet 18: echo request OnResponseReceived 59
Code Snippet 19: DicomFacade constructor 60
Code Snippet 20: sendDicoms method 60

14 BIBLIOGRAPHY

Bayers, M. (2012, 12 01). *Is it a good or bad practice to check for NULL? [closed]*. From stackoverflow.com: https://stackoverflow.com/questions/8347163/is-it-a-good-or-bad-practice-to-check-for-null

Fowler, M. (2004, 09 08). *Lambda*. From martinfowler.com: https://martinfowler.com/bliki/Lambda.html

Gruber, R. (2021, 04 16). *fo-dicom/fo-dicom*. From https://gitter.im/: https://gitter.im/fo-dicom/fo-dicom?at=5dab9e7b714b8b0538233564

kgiannakakis. (2011, 11 22). *stackoverflow.com*. From Stateless vs Stateful: https://stackoverflow.com/questions/5329618/stateless-vs-stateful

matt. (2020, December 5). *How do I mock a class without an interface?* From stackoverflow.com: https://stackoverflow.com/questions/20400734/how-do-i-mock-a-class-without-an-interface

Microsoft. (2021, 04 21). *requirements*. From code.visualstudio.com: https://code.visualstudio.com/docs/supporting/requirements

mortalapeman. (2014, 09 13). *Why are we supposed to use short functions to sectionalize our code?* From softwareengineering.stackexchange.com: https://softwareengineering.stackexchange.com/questions/210372/why-are-we-supposed-to-use-short-functions-to-sectionalize-our-code

Okhravi, C. (2017, 08 06). *Adapter Pattern – Design Patterns (ep 8)*. From YouTube: https://www.youtube.com/watch?v=2PKQtcJjYvc

Schafflützel, G. (2021, January 15). *Dicom Know How.* From Azure: https://dev.azure.com/balzano/ScanDiags/_wiki/wikis/ScanDiags.wiki/388/Introduction

superpig. (2020, May 09). *forum.unity.com*. From Design Patterns Mix-up?: https://forum.unity.com/threads/design-patterns-mix-up.886345/

Wikipedia. (2021, 04 08). *Adapter pattern*. From Wikipedia: https://en.wikipedia.org/wiki/Adapter_pattern

Wikipedia. (2021, 04 21). *Software design pattern*. From Wikipedia: https://en.wikipedia.org/wiki/Software_design_pattern

15 CODE

15.1 DICOMCLIENTCONTEXT.IDICOMCONTEXT

```
using System;
using System.Threading.Tasks;
using Dicom.Network;
using Dicom.Network.Client;
using System.Threading;
using System.Collections.Generic;
using Dicom.Log;
using System.Text;


namespace DicomClientContext
{

    /// <summary>
    /// The interface for the Dicom client context
    /// </summary>
    public interface IDicomContext
    {

        /// <summary>
        /// The host you want to connect to
        /// </summary>
        string Host { get; }

        /// <summary>
        /// The port to use for connecting to the host
        /// </summary>
        int Port { get; }

        /// <summary>
        /// True if Transport Layer Security should be on, false if not
        /// </summary>
        bool UseTls { get; }

        /// <summary>
        /// Application entity name of your client
        /// </summary>
        string CallingAe { get; }

        /// <summary>
        /// Application entity name of the server you want to connect to
        /// </summary>
        string CalledAe { get; }

        /// <summary>
        /// True if sending another request is required, false if not
        /// </summary>
        bool IsSendRequired { get; }

        /// <summary>
        /// Additional negotiation items to present during an association
        /// </summary>
        List<DicomExtendedNegotiation> AdditionalExtendedNegotiations { get; s
et; }

        /// <summary>
        /// Additional presentation contexts to present during an association
        /// </summary>
        List<DicomPresentationContext> AdditionalPresentationContexts { get; s
et; }

        /// <summary>
        /// How long the association should stay alive after all requests have
 been processed
        /// </summary>
        int AssociationLingerTimeoutInMs { get; }

        /// <summary>
        /// The amount of time to wait for an association release response aft
er sending an association release request
        /// </summary>
        int AssociationReleaseTimeoutInMs { get; }

        /// <summary>
        /// The amount of time to wait for an association response after sendi
ng an association request
        /// </summary>
        int AssociationRequestTimeoutInMs { get; }

        /// <summary>
        /// The encoding to fallback to when all else fails
        /// </summary>
        Encoding FallbackEncoding { get; set; }

        /// <summary>
        /// The logger with which to log events
        /// </summary>
        Logger Logger { get; set; }

        /// <summary>
        /// The maximum amount of requests that can be done in one association
.
        /// Starts a new association if it's over the max value.
        /// </summary>
        int? MaximumNumberOfRequestsPerAssociation { get; }

        /// <summary>
        /// The network manager
        /// </summary>
        NetworkManager NetworkManager { get; set; }

        /// <summary>
        /// The handler for when a C-store request is made to the client
        /// </summary>
        DicomClientCStoreRequestHandler OnCStoreRequest { get; set; }

        /// <summary>
        /// The handler for when a N-
event report request is made to the client
        /// </summary>
        DicomClientNEventReportRequestHandler OnNEventReportRequest { get; set
; }

        /// <summary>
        /// Options for the Dicom service
        /// </summary>
        DicomServiceOptions Options { get; set; }

        /// <summary>
        /// The event handler for when an association is accepted
        /// </summary>
        event EventHandler<Dicom.Network.Client.EventArguments.AssociationAcce
ptedEventArgs> AssociationAccepted;

        /// <summary>
        /// The event handler for when an association is rejected
        /// </summary>
        event EventHandler<Dicom.Network.Client.EventArguments.AssociationReje
ctedEventArgs> AssociationRejected;

        /// <summary>
        /// The event handler for when an association is released
        /// </summary>
        event EventHandler AssociationReleased;

        /// <summary>
        /// The event handler for when a request times out
        /// </summary>
        event EventHandler<Dicom.Network.Client.EventArguments.RequestTimedOut
EventArgs> RequestTimedOut;

        /// <summary>
        /// The event handler for when the state of the client changes
        /// </summary>
        event EventHandler<Dicom.Network.Client.EventArguments.StateChangedEve
ntArgs> StateChanged;

        /// <summary>
        /// Add a single request to the clients request que
        /// </summary>
        /// <param name="dicomRequest">The request to add</param>
        Task AddRequestAsync(DicomRequest dicomRequest);

        /// <summary>
        /// Add multiple requests to the clients requests que
        /// </summary>
        /// <param name="dicomRequests">The requests to add</param>
        Task AddRequestsAsync(IEnumerable<DicomRequest> dicomRequests);

        /// <summary>
        /// Add multiple requests to the client requests que
        /// </summary>
        /// <param name="dicomRequests">As many requests as you want, each a s
eparate parameter</param>
        Task AddRequestsAsync(params DicomRequest[] dicomRequests);

        /// <summary>
        /// The amount of requests you can send and process at the same time.
If it has to do more than defined here, it'll do them synchronously
        /// </summary>
        /// <param name="invoked">How many requests you can send at the same t
ime</param>
        /// <param name="performed">How many requests you can process at the s
ame time</param>
        void NegotiateAsyncOps(int invoked = 0, int performed = 0);

        /// <summary>
        /// Send the queued requests to the Dicom server
        /// </summary>
        /// <param name="cancellationToken">The cancellation token to stop the
 task</param>
        /// <param name="cancellationMode">If the connection should be aborted
 or cleanly released</param>
        Task SendAsync(CancellationToken cancellationToken = default, DicomCli
entCancellationMode cancellationMode = default);
    }
}
```

15.2 DICOMCLIENTCONTEXT.DICOMCONTEXT

```
using System;
using System.Threading.Tasks;
using Dicom.Network;
using Dicom.Network.Client;
using System.Threading;
using System.Collections.Generic;
using Dicom.Log;
using System.Text;


namespace DicomClientContext
{

    /// <summary>
    /// The Dicom context according to proxy pattern
    /// </summary>
    public class DicomContext : IDicomContext
    {

        /// <summary>
        /// The
        /// </summary>
        private readonly Dicom.Network.Client.DicomClient dicomClient;

        public string Host {
            get {
                return dicomClient.Host;
            }
        }

        public int Port {
            get {
                return dicomClient.Port;
            }
        }

        public bool UseTls {
            get {
                return dicomClient.UseTls;
            }
        }

        public string CallingAe {
            get {
                return dicomClient.CallingAe;
            }
        }
        public string CalledAe {
            get {
                return dicomClient.CalledAe;
            }
        }

        public bool IsSendRequired {
            get {
                return dicomClient.IsSendRequired;
            }
        }

        public Logger Logger {
            get {
                return dicomClient.Logger;
            }
            set {
                dicomClient.Logger = Logger;
            }
        }

        public DicomServiceOptions Options {
            get {
                return dicomClient.Options;
            }
            set {
                dicomClient.Options = Options;
            }
        }

        public List<DicomPresentationContext> AdditionalPresentationContexts {
            get {
                return dicomClient.AdditionalPresentationContexts;
            }
            set {
                dicomClient.AdditionalPresentationContexts = AdditionalPresent
ationContexts;
            }
        }

        public List<DicomExtendedNegotiation> AdditionalExtendedNegotiations {
            get {
                return dicomClient.AdditionalExtendedNegotiations;
            }
            set {
                dicomClient.AdditionalExtendedNegotiations = AdditionalExtende
dNegotiations;
```

```
            }
        }

        public Encoding FallbackEncoding {
            get {
                return dicomClient.FallbackEncoding;
            }
            set {
                dicomClient.FallbackEncoding = FallbackEncoding;
            }
        }

        public int AssociationRequestTimeoutInMs {
            get {
                return dicomClient.AssociationLingerTimeoutInMs;
            }
        }

        public int AssociationReleaseTimeoutInMs {
            get {
                return dicomClient.AssociationReleaseTimeoutInMs;
            }
        }

        public int AssociationLingerTimeoutInMs {
            get {
                return dicomClient.AssociationLingerTimeoutInMs;
            }
        }

        public int? MaximumNumberOfRequestsPerAssociation {
            get {
                return dicomClient.MaximumNumberOfRequestsPerAssociation;
            }
        }

        public DicomClientCStoreRequestHandler OnCStoreRequest {
            get {
                return dicomClient.OnCStoreRequest;
            }
            set {
                dicomClient.OnCStoreRequest = OnCStoreRequest;
            }
        }

        public DicomClientNEventReportRequestHandler OnNEventReportRequest {
            get {
                return dicomClient.OnNEventReportRequest;
            }
            set {
                dicomClient.OnNEventReportRequest = OnNEventReportRequest;
            }
        }

        public NetworkManager NetworkManager {
            get {
                return dicomClient.NetworkManager;
            }
            set {
                dicomClient.NetworkManager = NetworkManager;
            }
        }

        public event EventHandler<Dicom.Network.Client.EventArguments.Associat
ionAcceptedEventArgs> AssociationAccepted {
            add {
                dicomClient.AssociationAccepted += value;
            }
            remove {
                dicomClient.AssociationAccepted -= value;
            }
        }

        public event EventHandler<Dicom.Network.Client.EventArguments.Associat
ionRejectedEventArgs> AssociationRejected {
            add {
                dicomClient.AssociationRejected += value;
            }
            remove {
                dicomClient.AssociationRejected -= value;
            }
        }

        public event EventHandler AssociationReleased {
            add {
                dicomClient.AssociationReleased += value;
            }
            remove {
                dicomClient.AssociationReleased -= value;
            }
        }

        public event EventHandler<Dicom.Network.Client.EventArguments.StateCha
ngedEventArgs> StateChanged {
            add {
                dicomClient.StateChanged += value;
            }
            remove {
                dicomClient.StateChanged -= value;
            }
        }

        public event EventHandler<Dicom.Network.Client.EventArguments.RequestT
imedOutEventArgs> RequestTimedOut {
            add {
                dicomClient.RequestTimedOut += value;
            }
            remove {
                dicomClient.RequestTimedOut -= value;
            }
        }

        public void NegotiateAsyncOps(int invoked = 0, int performed = 0)
        {
            dicomClient.NegotiateAsyncOps(invoked, performed);
        }

        public async Task AddRequestAsync(DicomRequest dicomRequest)
        {
            if (dicomRequest == null) {
                var paramName = nameof(dicomRequest);
                throw new ArgumentNullException(paramName, $"Parameter {paramN
ame} is null.");
            }
            await dicomClient.AddRequestAsync(dicomRequest);
        }

        public async Task AddRequestsAsync(IEnumerable<DicomRequest> dicomRequ
ests)
        {
            if (dicomRequests == null) {
                var paramName = nameof(dicomRequests);
                throw new ArgumentNullException(paramName, $"Parameter {paramN
ame} is null.");
            }
            await dicomClient.AddRequestsAsync(dicomRequests);
        }

        public async Task AddRequestsAsync(params DicomRequest[] dicomRequests
)
        {
            if (dicomRequests == null) {
                var paramName = nameof(dicomRequests);
                throw new ArgumentNullException(paramName, $"Parameter {paramN
ame} is null.");
            }
            foreach (DicomRequest request in dicomRequests) {
                if (request == null) {
                    var paramName = nameof(request);
                    throw new ArgumentNullException(paramName, $"Parameter {pa
ramName} is null.");
                }
            }
            await dicomClient.AddRequestsAsync(dicomRequests);
        }

        public async Task SendAsync(CancellationToken cancellationToken = defa
ult(CancellationToken),
            DicomClientCancellationMode cancellationMode = DicomClientCancella
tionMode.ImmediatelyReleaseAssociation)
        {
            if (cancellationToken == null) {
                var paramName = nameof(cancellationToken);
                throw new ArgumentNullException(paramName, $"Parameter {paramN
ame} is null.");
            }
            if (cancellationToken == null) {
                var paramName = nameof(cancellationMode);
                throw new ArgumentNullException(paramName, $"Parameter {paramN
ame} is null.");
            }
            await dicomClient.SendAsync(cancellationToken, cancellationMode);
        }

        /// <summary>
        /// Takes all required parameters for initializing a DicomClient and p
asses it on to the CreateClient method
        /// </summary>
        /// <param name="host">Host</param>
        /// <param name="port">Port</param>
        /// <param name="useTls">Tls security on or off</param>
        /// <param name="callingAe">Calling application entity title</param>
        /// <param name="calledAe">Called application entity title</param>
        /// <param name="associationRequestTimeoutInMs">Association request ti
meout</param>
        /// <param name="associationReleaseTimeoutInMs">Association release ti
meout</param>
        /// <param name="associationLingerTimeoutInMs">Association linger time
out</param>
        /// <param name="maximumNumberOfRequestsPerAssociation">Max number of
requests per association</param>
```

```
        public DicomContext(string host, int port, bool useTls, string calling
Ae, string calledAe,
            int associationRequestTimeoutInMs = DicomClientDefaults.DefaultAss
ociationRequestTimeoutInMs,
            int associationReleaseTimeoutInMs = DicomClientDefaults.DefaultAss
ociationReleaseTimeoutInMs,
            int associationLingerTimeoutInMs = DicomClientDefaults.DefaultAsso
ciationLingerInMs,
            int? maximumNumberOfRequestsPerAssociation = null)
        {
            this.dicomClient = CreateClient(
                host,
                port,
                useTls,
                callingAe,
                calledAe,
                associationRequestTimeoutInMs,
                associationReleaseTimeoutInMs,
                associationLingerTimeoutInMs,
                maximumNumberOfRequestsPerAssociation);
        }

        private static Dicom.Network.Client.DicomClient CreateClient(params ob
ject[] args)
        {
            foreach (object arg in args) {
                if (arg is string paramAsString && string.IsNullOrWhiteSpace(p
aramAsString)) {
                    throw new ArgumentNullException("One of the string paramet
ers is null, empty, or consists only of white-space.");
                }
            }
            var client = new Dicom.Network.Client.DicomClient(
                (string)args[0],
                (int)args[1],
                (bool)args[2],
                (string)args[3],
                (string)args[4],
                (int)args[5],
                (int)args[6],
                (int)args[7],
                (int?)args[8]);
            return client;
        }
    }
}
```

15.3 PLAYGROUND.PROGRAM

```
using System;
using Dicom;
using System.Threading.Tasks;
using System.Collections.Generic;
using System.IO;
using System.Threading;

namespace PlayGround
{

    class Program
    {

        static async Task Main(string[] args)
        {
            var PlayGroundClient = new PlayGroundClient();
            var PlayGroundServer = new PlayGroundServer();

            var cancellationToken = new CancellationToken();

            Console.Clear();
            Console.ForegroundColor = ConsoleColor.Blue;
            Console.WriteLine("----- Instructions -----");
            Console.ForegroundColor = ConsoleColor.Gray;
            Console.WriteLine("Press Enter to send all Dicom files to the play
ground server.");
            Console.ForegroundColor = ConsoleColor.Blue;
            Console.WriteLine("------------------------");
            while (true) {
                Console.ForegroundColor = ConsoleColor.Gray;
                Console.WriteLine("\nPress Enter. ");
                var enteredKey = Console.ReadKey(true);
                if (enteredKey.Key.Equals(ConsoleKey.Enter)) {
                    var dicomFiles = new List<DicomFile>();
                    string[] filesInDicomDirectory = Directory.GetDirectories(
Constants.dicomFilesPath);
                    foreach (string subdir in filesInDicomDirectory) {
                        foreach (string dcm in Directory.GetFiles(subdir)) {
                            dicomFiles.Add(DicomFile.Open(dcm));
                        }
                    }
                    Console.WriteLine($"Sending {dicomFiles.Count} Dicom files
 to the server.");
                    Console.ForegroundColor = ConsoleColor.Red;
                    await PlayGroundClient.sendDicoms(dicomFiles.ToArray(), ca
ncellationToken);
                    Console.WriteLine($"Dicom datasets sent.");
                }
                else {
                    Console.WriteLine("Wrong input. ");
                }
            }
        }
    }
}
```

15.4 PLAYGROUND.PLAYGROUNDCLIENT

```
using Dicom;
using Dicom.Network;
using System.Threading;
using System.Threading.Tasks;
using System.Collections.Generic;

namespace PlayGround
{
    class PlayGroundClient
    {

        private IDicomFacade dicomFacade;

        public PlayGroundClient()
        {
            // set up connection with server
            var host = "127.0.0.1";
            var port = 55555;
            var callingAE = "PlayGroundClient";
            var calledAe = "PlayGroundServer";
            dicomFacade = new DicomFacade(host, port, false, callingAE, called
Ae);
        }


        public async Task sendDicoms(DicomFile[] dicomFiles, CancellationToken
 cancellationToken)
        {
            var storeRequests = new List<DicomCStoreRequest>();
            foreach (var dicomFile in dicomFiles) {
                storeRequests.Add(new DicomCStoreRequest(dicomFile, DicomPrior
ity.Medium));
            }
            await dicomFacade.EchoAndSendRequestsAsync(storeRequests, cancella
tionToken);
        }

    }

}
```

15.5 PLAYGROUND.PLAYGROUNDSERVER

```
using Dicom;
using Dicom.Network;
using System;
using System.Threading.Tasks;
using System.Text;
using Dicom.Log;
using System.Collections.Generic;

namespace PlayGround
{
    class PlayGroundServer
    {

        public PlayGroundServer()
        {
            var port = 55555;
            var server = DicomServer.Create<CStoreSCP>(port);
        }

    }

    class CStoreSCP :
    DicomService,
    IDicomServiceProvider,
    IDicomCStoreProvider,
    IDicomCEchoProvider
    {

        private static readonly DicomTransferSyntax[] AcceptedImageTransferSyn
taxes = new DicomTransferSyntax[]
        {
               // Lossless
               DicomTransferSyntax.JPEGLSLossless,
               DicomTransferSyntax.JPEG2000Lossless,
               DicomTransferSyntax.JPEGProcess14SV1,
               DicomTransferSyntax.JPEGProcess14,
               DicomTransferSyntax.RLELossless,
               // Lossy
               DicomTransferSyntax.JPEGLSNearLossless,
               DicomTransferSyntax.JPEG2000Lossy,
               DicomTransferSyntax.JPEGProcess1,
               DicomTransferSyntax.JPEGProcess2_4,
               // Uncompressed
               DicomTransferSyntax.ExplicitVRLittleEndian,
               DicomTransferSyntax.ExplicitVRBigEndian,
               DicomTransferSyntax.ImplicitVRLittleEndian
        };
        public CStoreSCP(INetworkStream stream, Encoding fallbackEncoding, Log
ger log)
            : base(stream, fallbackEncoding, log)
        {
        }

        /* ---------- */
        private static readonly List<DicomUID> RejectedUIDs = new List<DicomUI
D>
        {
                DicomUID.EncapsulatedPDFStorage,
                DicomUID.BasicTextSRStorage
            };

        public Task OnReceiveAssociationRequestAsync(DicomAssociation associat
ion)
        {
            if (association.CalledAE != "PlayGroundServer") {
                return SendAssociationRejectAsync(
                    DicomRejectResult.Permanent,
                    DicomRejectSource.ServiceUser,
                    DicomRejectReason.CalledAENotRecognized);
            }

            foreach (var pc in association.PresentationContexts) {
                if (RejectedUIDs.Contains(pc.AbstractSyntax)) {
                    pc.SetResult(DicomPresentationContextResult.RejectAbstract
SyntaxNotSupported);
                }
                else {
                    pc.AcceptTransferSyntaxes(AcceptedImageTransferSyntaxes);
                    pc.SetResult(DicomPresentationContextResult.Accept);
                }
            }

            return SendAssociationAcceptAsync(association);
        }
        /* ---------- */

        public Task OnReceiveAssociationReleaseRequestAsync()
        {
            return SendAssociationReleaseResponseAsync();
        }

        public void OnReceiveAbort(DicomAbortSource source, DicomAbortReason r
eason)
        {
        }

        public void OnConnectionClosed(Exception exception)
        {
        }

        public DicomCStoreResponse OnCStoreRequest(DicomCStoreRequest request)
        {

            // storage of file

            return new DicomCStoreResponse(request, DicomStatus.Success);
        }

        public void OnCStoreRequestException(string tempFileName, Exception e)
        {
            // let library handle logging and error response
        }

        public DicomCEchoResponse OnCEchoRequest(DicomCEchoRequest request)
        {
            return new DicomCEchoResponse(request, DicomStatus.Success);
        }


    }

}
```

15.6 PLAYGROUND.CONSTANTS

```
namespace PlayGround
{
    public static class Constants
    {
        public static string dicomFilesPath = "files/";

    }
}
```

15.7 PLAYGROUND.IDICOMFACADE

```
using System.Threading.Tasks;
using Dicom.Network;
using System.Threading;
using System.Collections.Generic;

namespace PlayGround
{

    /// <summary>
    /// Interface for the Dicom client facade
    /// </summary>
    public interface IDicomFacade
    {

        /// <summary>
        /// Sends an echo request first before sending the requests provided t
hrough the parameters
        /// </summary>
        /// <param name="requests">The requests to send</param>
        /// <param name="cancellationToken">The cancellation token to stop the
 task</param>
        Task EchoAndSendRequestsAsync(
            IEnumerable<DicomRequest> requests,
            CancellationToken cancellationToken);

    }

}
```

15.8 PLAYGROUND.DICOMFACADE

```
using System;
using System.Threading.Tasks;
using Dicom.Network;
using Dicom.Network.Client;
using System.Threading;
using System.Collections.Generic;
using Dicom.Log;

namespace PlayGround {

    /// <summary>
    /// The wrapper class for the Dicom client according to facade pattern
    /// </summary>
    public class DicomFacade : IDicomFacade {
        private readonly Dicom.Network.Client.DicomClient dicomClient;

        /// <summary>
        /// Takes all required parameters for initializing a DicomClient and p
asses it on to the CreateClient method
        /// </summary>
        /// <param name="host">Host</param>
        /// <param name="port">Port</param>
        /// <param name="useTls">Tls security on or off</param>
        /// <param name="callingAe">Calling application entity title</param>
        /// <param name="calledAe">Called application entity title</param>
        /// <param name="logger">Logger to log events</param>
        /// <param name="associationRequestTimeoutInMs">Association request ti
meout</param>
        /// <param name="associationReleaseTimeoutInMs">Association release ti
meout</param>
        /// <param name="associationLingerTimeoutInMs">Association linger time
out</param>
        /// <param name="maximumNumberOfRequestsPerAssociation">Max number of
requests per association</param>
        public DicomFacade(string host, int port, bool useTls, string callingA
e, string calledAe,
            Logger logger = null,
            int associationRequestTimeoutInMs = DicomClientDefaults.DefaultAss
ociationRequestTimeoutInMs,
            int associationReleaseTimeoutInMs = DicomClientDefaults.DefaultAss
ociationReleaseTimeoutInMs,
            int associationLingerTimeoutInMs = DicomClientDefaults.DefaultAsso
ciationLingerInMs,
            int? maximumNumberOfRequestsPerAssociation = null)
        {
            this.dicomClient = CreateClient(
                host,
                port,
                useTls,
                callingAe,
                calledAe,
                logger,
                associationRequestTimeoutInMs,
                associationReleaseTimeoutInMs,
                associationLingerTimeoutInMs,
                maximumNumberOfRequestsPerAssociation);
        }

        private static Dicom.Network.Client.DicomClient CreateClient(params ob
ject[] args) {
            foreach (object arg in args) {
                var paramName = nameof(arg);
                if (arg is string paramAsString && string.IsNullOrWhiteSpace(p
aramAsString)) {
                    throw new ArgumentNullException(paramName, $"Parameter {pa
ramName} is null, empty, or consists only of white-space.");
                }
            }
            var client = new Dicom.Network.Client.DicomClient(
                (string)args[0],
                (int)args[1],
                (bool)args[2],
                (string)args[3],
                (string)args[4],
                (int)args[6],
                (int)args[7],
                (int)args[8],
                (int?)args[9]);
            if (args[5] != null) client.Logger = (Logger)args[5];
            return client;
        }

        public async Task EchoAndSendRequestsAsync(IEnumerable<DicomRequest> r
equests,
            CancellationToken cancellationToken)
        {
            var echoRequest = new DicomCEchoRequest();
            echoRequest.OnResponseReceived = (request, response) => {
                dicomClient.Logger.Info("C-
Echo request to PACS with IP: {ip}, Port: {port}, AE-
Title: {ae} with status {status}", dicomClient.Host, dicomClient.Port, dicomCl
ient.CalledAe, response.Status.ToString());
            };
            echoRequest.OnTimeout = (request, response) => {
                dicomClient.Logger.Error("C-
Echo request to PACS with IP: {ip}, Port: {port}, AE-
Title: {ae} timed out", dicomClient.Host, dicomClient.Port, dicomClient.Called
Ae);
                throw new DicomNetworkException("C-Echo request timed out");
            };
            await this.dicomClient.AddRequestsAsync(requests);
            await this.dicomClient.SendAsync(cancellationToken);
        }


    }

}
```

15.9 DICOMGENERATOR.PACSTARGET

```
using System.Collections.Generic;
using System.Threading.Tasks;
using Dicom;
using Dicom.Network;
using DicomClientContext;
using System;

namespace DicomGenerator
{

    /// <summary>
    /// Sends the generated DicomDatasets to a DicomNode, usually a PACS syste
m, but it can be any capable Dicom entity
    /// </summary>
    public class PACSTarget : ITarget
    {

        /// <summary>
        /// Calling application entity title, "client" name
        /// </summary>
        private string callingTitle;
        /// <summary>
        /// Called application entity title, "server" name
        /// </summary>
        private string calledTitle;
        /// <summary>
        /// The host ip address of the called DicomNode
        /// </summary>
        private string dicomHost;
        /// <summary>
        /// The port address of the called DicomNode
        /// </summary>
        private int dicomPort;
        /// <summary>
        /// The DicomClient to send data from
        /// </summary>
        private IDicomContext client;

        /// <summary>
        /// Set class fields
        /// </summary>
        /// <param name="client">DicomContext acting as Dicom client</param>
        public PACSTarget(IDicomContext client)
        {
            this.client = client;
        }
        /// <summary>
        /// Sends the DicomDatasets to a DicomNode
        /// </summary>
        /// <param name="datasets">The list of datasets to send</param>
        public async Task Save(List<DicomDataset> datasets)
        {
            if (datasets == null) {
                var paramName = nameof(datasets);
                throw new ArgumentNullException(paramName, $"Parameter {paramN
ame} is null.");
            }
            List<DicomRequest> requests = new List<DicomRequest>();
            requests.Add(new DicomCEchoRequest());
            foreach (DicomDataset dataset in datasets) {
                requests.Add(new DicomCStoreRequest(new DicomFile(dataset)));
            }
            await client.AddRequestsAsync(requests);
            await client.SendAsync();
        }
    }
}
```

15.10 DICOMGENERATOR.PACSTARGETTEST

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Threading.Tasks;
using Dicom.Network;
using Xunit;
using Dicom;
using Moq;
using DicomClientContext;

namespace DicomGenerator
{

    public class PACSTargetTest
    {

        private List<DicomDataset> testDatasetList;

        public PACSTargetTest()
        {
            testDatasetList = new List<DicomDataset>();
            DirectoryInfo testFilesDir = new DirectoryInfo("..\\..\\..\\testFi
les\\");
            foreach (var file in testFilesDir.GetFiles("*.dcm")) {
                testDatasetList.Add(DicomFile.Open(file.FullName).Dataset);
            }
        }

        public void Dispose()
        {
            testDatasetList = null;
        }

        [Fact]
        public async Task SaveTest()
        {
            // Arrange
            var dicomContextMock = new Mock<IDicomContext>();
            var pacsTarget = new PACSTarget(dicomContextMock.Object);
            // Act
            await pacsTarget.Save(testDatasetList);
        }

        [Fact]
        public async Task SaveDatasetsIsNullTest()
        {
            // Arrange
            var dicomContextMock = new Mock<IDicomContext>();
            var pacsTarget = new PACSTarget(dicomContextMock.Object);
            // Assert
            await Assert.ThrowsAsync<ArgumentNullException>(async () => {
                // Act
                await pacsTarget.Save(null);
            });
        }

        [Fact]
        public async Task SaveDatasetsCountEqualsRequestCountTest()
        {
            // Arrange
            int actualRequestsNum = 0;
            var dicomContextMock = new Mock<IDicomContext>();
            dicomContextMock.Setup(context => context.AddRequestsAsync(It.IsAn
y<List<DicomRequest>>()))
            .Callback<IEnumerable<DicomRequest>>((requests) => {
                foreach (var request in requests) {
                    ++actualRequestsNum;
                }
                --actualRequestsNum;
            });
            var pacsTarget = new PACSTarget(dicomContextMock.Object);
            var expectedRequestsNum = testDatasetList.Count;
            // Act
            await pacsTarget.Save(testDatasetList);
            // Assert
            Assert.Equal(expectedRequestsNum, actualRequestsNum);
        }
    }
}
```

15.11 DICOMTOPACSLOADER.DICOMTOPACSLOADER

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.Threading;
using System.Threading.Tasks;
using Common;
using DicomClientContext;
using Dicom;
using Dicom.Log;
using Dicom.Network;
using Dicom.Serialization;
using Newtonsoft.Json;
using Serilog;
using Dicom.Network.Client;

namespace DicomToPACSLoader
{
    /// <summary>
    /// Uses a given list of StudyInstanceUIDs to load SOPInstances from the d
atabase and write them as Dicom files to the filesystem
    /// </summary>
    public class DicomToPACSLoader
    {
        #region Properties

        /// <summary>
        /// Context to the database
        /// </summary>
        private ISqlDataContext sqlDataContext;

        /// <summary>
        /// Asyncronous token to stop the mask importer.
        /// </summary>
        private CancellationToken stoppingToken;

        /// <summary>
        /// The DicomClient with which the connection is set up
        /// </summary>
        /// <value>DicomClient</value>
        private IDicomContext client;

        /// <summary>
        /// The logger for context of the class, derived from the global logge
r
        /// </summary>
        /// <returns>Serilog.ILogger logging interface</returns>
        private readonly ILogger _logger = Log.ForContext<DicomToPACSLoader>()
;

        #endregion

        #region ctor

        /// <summary>
        /// Constructor, saves the sqlDataContext and the stoppingToken
        /// </summary>
        public DicomToPACSLoader(ISqlDataContext sqlDataContext, IDicomContext
 client, CancellationToken stoppingToken = default)
        {
            this.sqlDataContext = sqlDataContext;
            this.client = client;
            this.stoppingToken = stoppingToken;
        }

        #endregion

        #region Methods

        /// <summary>
        /// Loads SOPInstances from the DB and saves them to the FileSystem
        /// </summary>
        /// <returns>nothing</returns>
        public async Task Load()
        {
            client.Logger = LogManager.GetLogger("DicomClient");
            await PingPACS();

            var studyInstanceUIDs = new Dictionary<Int64, String>();

            using (var reader = await sqlDataContext.ExecuteReaderAsync(
                Program.config.SPNameGETPacsExports,
                CommandType.StoredProcedure,
                new[] {
                    new SqlParameter(Program.config.SPParameterStateID, SqlDbT
ype.BigInt) { Value = 100 }
                },
                stoppingToken
            )) {
                while (reader.Read()) {
                    var id = reader.GetInt64(0);
                    var studyInstanceUID = reader.GetString(1);
                    // var stateId = reader.GetInt64(2);
                    // var stateTitle = reader.GetString(3);
                    studyInstanceUIDs.Add(id, studyInstanceUID);
                }
            }

            if (studyInstanceUIDs.Count == 0) {
                return;
            }

            foreach (var studyInstanceUID in studyInstanceUIDs) {
                try {
                    using (SqlDataReader reader = await sqlDataContext.Execute
ReaderAsync(
                        Program.config.SPNameGETDicom,
                        CommandType.StoredProcedure,
                        new[] {
                        new SqlParameter(Program.config.SPParameterStudyInstan
ceUID, SqlDbType.VarChar, 64) { Value = studyInstanceUID.Value },
                        },
                        stoppingToken
                    ) as SqlDataReader) {
                        while (reader != null && reader.Read()) {
                            var dsJson = reader.GetString(0);
                            var dsPixel = reader.GetStream(1);
                            await SendCStoreRequest(new SDDataset(dsJson, dsPi
xel));
                        }
                    }

                    await sqlDataContext.ExecuteNonQueryAsync(
                        Program.config.SPNamePUTPacsExportState,
                        CommandType.StoredProcedure,
                        new[] {
                        new SqlParameter(Program.config.SPParameterExportID, S
qlDbType.BigInt) { Value = studyInstanceUID.Key },
                        new SqlParameter(Program.config.SPParameterStateID, Sq
lDbType.BigInt) { Value = 300 }
                        },
                        stoppingToken
                    );
                }
                catch (Exception ex) {
                    _logger.Error(ex.ToString());

                    await sqlDataContext.ExecuteNonQueryAsync(
                        Program.config.SPNamePUTPacsExportState,
                        CommandType.StoredProcedure,
                        new[] {
                        new SqlParameter(Program.config.SPParameterExportID, S
qlDbType.BigInt) { Value = studyInstanceUID.Key },
                        new SqlParameter(Program.config.SPParameterStateID, Sq
lDbType.BigInt) { Value = 900 }
                        },
                        stoppingToken
                    );
                }
            }
        }

        /// <summary>
        /// Pings the PACS to see if it is reachable, if not it throws an erro
r
        /// </summary>
        /// <returns>nothing</returns>
        private async Task PingPACS()
        {
            var EchoRequest = new DicomCEchoRequest();
            EchoRequest.OnResponseReceived = (EchoRequest, response) => {
                _logger.Information("C-
Echo request to PACS with IP: {ip}, Port: {port}, AE-
Title: {ae} with status {status}", Program.config.PACS_IP, Convert.ToInt32(Pro
gram.config.PACS_PORT), Program.config.PACS_AE, response.Status.ToString());
                return;
            };
            EchoRequest.OnTimeout = (EchoRequest, response) => {
                _logger.Error("C-
Echo request to PACS with IP: {ip}, Port: {port}, AE-
Title: {ae} timed out", Program.config.PACS_IP, Convert.ToInt32(Program.config
.PACS_PORT), Program.config.PACS_AE);
                throw new DicomNetworkException("C-Echo request timed out");
            };
            await this.client.AddRequestAsync(EchoRequest);
            await this.client.SendAsync();
        }

        /// <summary>
        /// Sends Dicom Dataset to PACS System
        /// </summary>
        /// <returns>nothing</returns>
        private async Task SendCStoreRequest(ISDDataset sDDataset)
        {
            DicomFile dicomFile = CreateDicomFile(sDDataset);
            DicomCStoreRequest StoreRequest = new DicomCStoreRequest(dicomFile
);
            await client.AddRequestAsync(StoreRequest);

            StoreRequest.OnResponseReceived = (StoreRequest, response) => {
                return;
            };
            StoreRequest.OnTimeout = (StoreRequest, response) => {
                _logger.Error("C-
Store request to PACS with IP: {ip}, Port: {port}, AE-
Title: {ae} timed out", Program.config.PACS_IP, Convert.ToInt32(Program.config
.PACS_PORT), Program.config.PACS_AE);
                throw new DicomNetworkException("C-Move request timed out");
            };
            await client.SendAsync();
        }

        /// <summary>
        /// Creates a DicomFile from the sd Dataset
        /// </summary>
        /// <returns>dicom file</returns>
        public DicomFile CreateDicomFile(ISDDataset sDDataset)
        {
            DicomDataset dicomDataset = JsonConvert.DeserializeObject<DicomDat
aset>(sDDataset.DSJson, new JsonDicomConverter());
            dicomDataset.AddOrUpdate(DicomTag.PixelData, sDDataset.DSPixel);
            DicomFile dicomFile = new DicomFile(dicomDataset);
            return dicomFile;
        }
        #endregion
    }
}
```

15.12 DICOMTOPACSLOADER.TOPACSLOADERTEST

```
using Common;
using DicomClientContext;
using Xunit;
using Moq;
using Dicom;
using Dicom.Network;
using System.IO;
using System.Threading;
using System.Data;
using System.Data.SqlClient;
using System.Threading.Tasks;
using System.Collections.Generic;

namespace DicomToPACSLoader
{
    public class ToPACSLoaderTest
    {
        [Fact(DisplayName = "Load Test")]
        public void CreateDicomFileTest()
        {
            // Arrange
            var sqlDataContext = new Mock<ISqlDataContext>();
            var dicomContext = new Mock<IDicomContext>();
            var toPACSLoader = new DicomToPACSLoader(sqlDataContext.Object, di
comContext.Object, CancellationToken.None);

            var SDDatasetMoq = new Mock<ISDDataset>();

            var DSJson = File.ReadAllText("DSJson.json");

            SDDatasetMoq
                .Setup(m => m.DSPixel)
                .Returns(new byte[] { 0x20 });

            SDDatasetMoq
                .Setup(m => m.DSJson)
                .Returns(DSJson);

            // Act
            var result = toPACSLoader.CreateDicomFile(SDDatasetMoq.Object);

            var tag = result.Dataset.GetValues<string>(DicomTag.SOPClassUID);

            // Assert
            Assert.Equal("1.2.840.10008.5.1.4.1.1.7", tag[0]);
        }
        [Fact]
        public async Task LoadTest()
        {
            //Arrange
            var expectedPatientName = "Dimitri";
            DicomDataset actualDicomDataset = null;
            var sqlDataContextMock = new Mock<ISqlDataContext>();
            var GETPacsExportsDataReaderMock = new Mock<IDataReader>();
            var GETDicomReaderMock = new Mock<IDataReader>();
            var imageStreamMock = new Mock<Stream>();

            GETPacsExportsDataReaderMock.Setup(reader => reader.GetInt64(0))
                .Returns(1);
            GETPacsExportsDataReaderMock.Setup(reader => reader.GetString(1))
                .Returns(DicomUIDGenerator.GenerateDerivedFromUUID().UID);

            var readGETPacsExportsDataReturnQueue = new Queue<bool>();
            readGETPacsExportsDataReturnQueue.Enqueue(true);
            readGETPacsExportsDataReturnQueue.Enqueue(false);

            GETPacsExportsDataReaderMock.Setup(reader => reader.Read())
                .Returns(() => readGETPacsExportsDataReturnQueue.Dequeue());
            sqlDataContextMock.Setup(context => context.ExecuteReaderAsync(Pro
gram.config.SPNameGETPacsExports, It.IsAny<CommandType>(), It.IsAny<SqlParamet
er[]>(), It.IsAny<CancellationToken>()))
                .Returns(Task.FromResult(GETPacsExportsDataReaderMock.Object))
;
            GETDicomReaderMock.Setup(reader => reader.GetString(0))
                .Returns("{\"00100010\":{\"vr\":\"PN\",\"Value\":[{\"Alphabeti
c\":\"Dimitri\"}]}}");

            //GETDicomReaderMock.Setup(reader => reader.GetStream(1))
            //    .Returns(imageStreamMock.Object);

            var readGETDicomReturnQueue = new Queue<bool>();
            readGETDicomReturnQueue.Enqueue(true);
            readGETDicomReturnQueue.Enqueue(false);

            GETPacsExportsDataReaderMock.Setup(reader => reader.Read())
                .Returns(() => readGETDicomReturnQueue.Dequeue());
            sqlDataContextMock.Setup(context => context.ExecuteReaderAsync(Pro
gram.config.SPNameGETDicom, It.IsAny<CommandType>(), It.IsAny<SqlParameter[]>(
), It.IsAny<CancellationToken>()))
                .Returns(Task.FromResult(GETDicomReaderMock.Object));

            var dicomContext = new Mock<IDicomContext>();
            dicomContext.Setup(context => context.AddRequestAsync(It.IsAny<Dic
omCStoreRequest>()))
```

```
                .Callback<DicomRequest>((request) => actualDicomDataset = requ
est.Command);

            var toPACSLoader = new DicomToPACSLoader(sqlDataContextMock.Object
, dicomContext.Object, CancellationToken.None);
            // Act
            await toPACSLoader.Load();
            // Assert
            Assert.Equal(expectedPatientName, actualDicomDataset.GetSingleValu
e<string>(DicomTag.PatientName));
        }
    }
}
```

15.13 JOBRECEIVER.PACSTARGET

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Serilog;
using Dicom;
using Dicom.Log;
using Dicom.Network;
using DicomClientContext;

namespace JobReceiver
{
    /// <summary>
    /// Class that sends the encapsulated PDF report to the PACS
    /// </summary>
    public class PacsTarget
    {

        #region Properties

        /// <summary>
        /// The DicomClient with which the connection is set up
        /// </summary>
        /// <value>DicomClient</value>
        private IDicomContext client;

        /// <summary>
        /// The logger for context of the class, derived from the global logge
r
        /// </summary>
        /// <returns>Serilog.ILogger logging interface</returns>
        private readonly ILogger _logger = Log.ForContext<PacsTarget>();

        #endregion


        /// <summary>
        /// Constructor, initializes the Dicom LogManager
        /// </summary>
        public PacsTarget()
        {
            LogManager.SetImplementation(new SerilogManager(Log.ForContext<Pac
sTarget>()));
        }

        /// <summary>
        /// Sends a C-Echo request to the PACS to check the connection
        /// </summary>
        /// <returns>Task that completes when the request times out or is succ
essful</returns>
        private async Task PingPACS()
        {
            var request = new DicomCEchoRequest();
            request.OnResponseReceived = (request, response) => {
                _logger.Information("C-
Echo request to PACS with IP: {ip}, Port: {port}, AE-
Title: {ae} with status {status}", client.Host, Program.config.PACS_PORT, clie
nt.CalledAe, response.Status.ToString());
                return;
            };
            request.OnTimeout = (request, response) => {
                _logger.Error("C-
Echo request to PACS with IP: {ip}, Port: {port}, AE-
Title: {ae} timed out", client.Host, Program.config.PACS_PORT, client.CalledAe
);
                throw new DicomNetworkException("C-Echo request timed out");
            };
            await this.client.AddRequestAsync(request);
            await this.client.SendAsync();
        }

        /// <summary>
        /// Sends a C-
Store request to the PACS with the encapsulated PDF report
        /// </summary>
        /// <param name="resultReport">The encapsulated PDF report that will b
e sent to the PACS</param>
        /// <returns>Task that completes when the request times out or is succ
essful</returns>
        private async Task sendCStoreRequest(DicomDataset resultReport)
        {
            DicomCStoreRequest request = new DicomCStoreRequest(new DicomFile(
resultReport), DicomPriority.Medium);
            await client.AddRequestAsync(request);
            request.OnResponseReceived = (request, response) => {
                _logger.Information("C-
Move request to PACS with IP: {ip}, Port: {port}, AE-
Title: {ae} with status {status}", client.Host, Program.config.PACS_PORT, clie
nt.CalledAe, response.Status.ToString());
                return;
            };
            request.OnTimeout = (request, response) => {
                _logger.Error("C-
Move request to PACS with IP: {ip}, Port: {port}, AE-
Title: {ae} timed out", client.Host, Program.config.PACS_PORT, client.CalledAe
);
```

```
        throw new DicomNetworkException("C-Move request timed out");
      };
      await client.SendAsync();
    }

    /// <summary>
    /// Pings the PACS and sends the encapsulated PDF reports to the PACS
    /// </summary>
    /// <param name="reports">A list of encapsulated PDF reports that will
 be sent to the PACS</param>
    /// <param name="client">DicomContext acting as Dicom client</param>
    public async Task SendReports(List<DicomDataset> reports, IDicomContex
t client)
    {

      if (reports == null) {
        var paramName = nameof(reports);
        throw new ArgumentNullException(paramName, $"Parameter {paramN
ame} is null.");
      }
      else if (client == null) {
        var paramName = nameof(client);
        throw new ArgumentNullException(paramName, $"Parameter {paramN
ame} is null.");
      }

      this.client = client;
      client.Logger = LogManager.GetLogger("DicomClient");
      await PingPACS();
      foreach (DicomDataset ds in reports) {
        await sendCStoreRequest(ds);
      }
    }
  }
}
```

15.14 JOBRECEIVER.PACSTARGETTEST

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Threading.Tasks;
using Xunit;
using Moq;
using DicomClientContext;
using Dicom;
using Dicom.Network;

namespace JobReceiver
{
    public class PacsTargetTest : IDisposable
    {

        private List<DicomDataset> testDatasetList;

        public PacsTargetTest()
        {
            testDatasetList = new List<DicomDataset>();
            DirectoryInfo testFilesDir = new DirectoryInfo("..\\..\\..\\testFi
les\\");
            foreach (var file in testFilesDir.GetFiles("*.dcm")) {
                testDatasetList.Add(DicomFile.Open(file.FullName).Dataset);
            }
        }

        public void Dispose()
        {
            testDatasetList = null;
        }

        [Fact]
        public async Task TestSendReports()
        {
            // Arrange
            var dicomContextMock = new Mock<IDicomContext>();
            var pacsTarget = new PacsTarget();
            // Act
            await pacsTarget.SendReports(testDatasetList, dicomContextMock.Obj
ect);
        }

        [Fact]
        public async Task SaveDatasetsIsNullTest()
        {
            // Arrange
            var dicomContextMock = new Mock<IDicomContext>();
            var pacsTarget = new PacsTarget();
            // Assert
            await Assert.ThrowsAsync<ArgumentNullException>(async () => {
                // Act
                await pacsTarget.SendReports(null, dicomContextMock.Object);
            });
        }

        [Fact]
        public async Task SaveDatasetsCountEqualsRequestCountTest()
        {
            // Arrange
            int actualRequestsNum = 0;
            var dicomContextMock = new Mock<IDicomContext>();
            dicomContextMock.Setup(context => context.AddRequestsAsync(It.IsAn
y<IEnumerable<DicomRequest>>()))
            .Callback<IEnumerable<DicomRequest>>((requests) => {
                foreach (var request in requests) {
                    ++actualRequestsNum;
                }
                --actualRequestsNum;
            });
            var pacsTarget = new PacsTarget();
            var expectedRequestsNum = testDatasetList.Count;
            // Act
            await pacsTarget.SendReports(testDatasetList, dicomContextMock.Obj
ect);

            // Assert
            Assert.Equal(expectedRequestsNum, actualRequestsNum);
        }
    }
}
```

15.15 REPORTSENDER.PACSTARGET

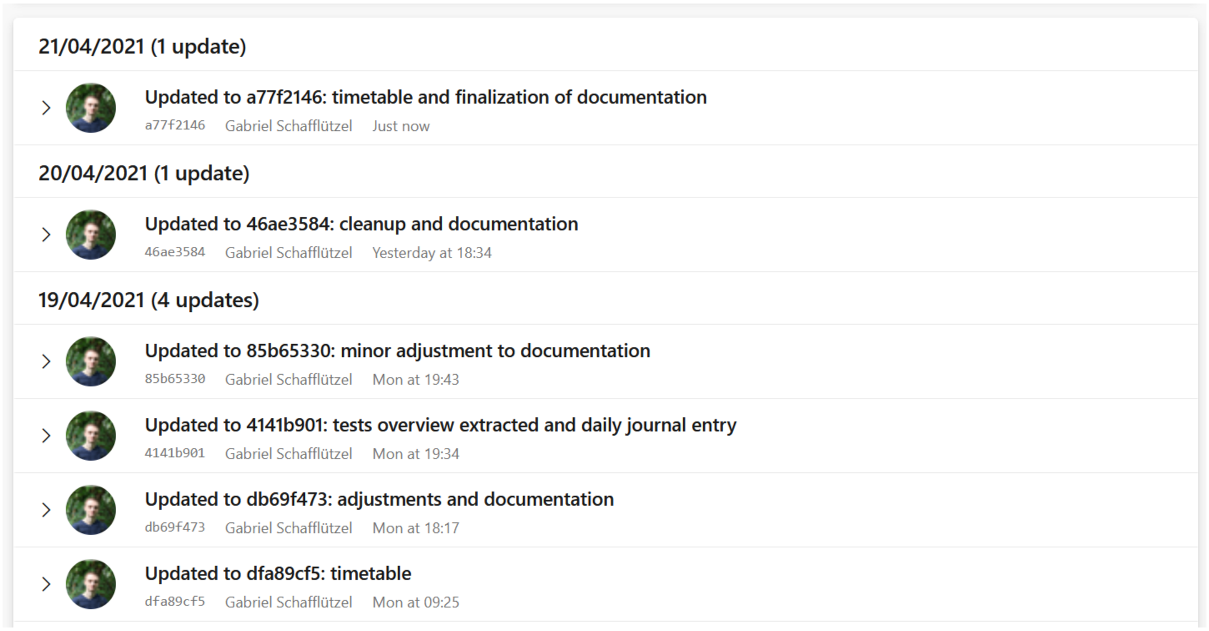
```
using Dicom.Network;
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Serilog;
using Dicom;
using Dicom.Log;
using DicomClientContext;

namespace ReportSender
{
    /// <summary>
    /// Class that sends the encapsulated PDF report to the PACS
    /// </summary>
    public class PacsTarget
    {

        #region Properties

        /// <summary>
        /// The DicomClient with which the connection is set up
        /// </summary>
        /// <value>DicomClient</value>
        private IDicomContext client;

        /// <summary>
        /// The logger for context of the class, derived from the global logge
r
        /// </summary>
        /// <returns>Serilog.ILogger logging interface</returns>
        private readonly ILogger _logger = Log.ForContext<PacsTarget>();

        #endregion


        /// <summary>
        /// Constructor, initializes the Dicom LogManager
        /// </summary>
        public PacsTarget()
        {
            LogManager.SetImplementation(new SerilogManager(Log.ForContext<Pac
sTarget>()));
        }

        /// <summary>
        /// Sends a C-Echo request to the PACS to check the connection
        /// </summary>
        /// <returns>Task that completes when the request times out or is succ
essful</returns>
        private async Task PingPACS()
        {
            var request = new DicomCEchoRequest();
            request.OnResponseReceived = (request, response) => {
                _logger.Information("C-
Echo request to PACS with IP: {ip}, Port: {port}, AE-
Title: {ae} with status {status}", client.Host, Program.config.PACS_PORT, clie
nt.CalledAe, response.Status.ToString());
                return;
            };
            request.OnTimeout = (request, response) => {
                _logger.Error("C-
Echo request to PACS with IP: {ip}, Port: {port}, AE-
Title: {ae} timed out", client.Host, Program.config.PACS_PORT, client.CalledAe
);
                throw new DicomNetworkException("C-Echo request timed out");
            };
            await this.client.AddRequestAsync(request);
            await this.client.SendAsync();
        }

        /// <summary>
        /// Sends a C-
Store request to the PACS with the encapsulated PDF report
        /// </summary>
        /// <param name="resultReport">The encapsulated PDF report that will b
e sent to the PACS</param>
        /// <returns>Task that completes when the request times out or is succ
essful</returns>
        private async Task sendCStoreRequest(DicomDataset resultReport)
        {
            DicomCStoreRequest request = new DicomCStoreRequest(new DicomFile(
resultReport), DicomPriority.Medium);
            await client.AddRequestAsync(request);
            request.OnResponseReceived = (request, response) => {
                _logger.Information("C-
Move request to PACS with IP: {ip}, Port: {port}, AE-
Title: {ae} with status {status}", client.Host, Program.config.PACS_PORT, clie
nt.CalledAe, response.Status.ToString());
                return;
            };
            request.OnTimeout = (request, response) => {
                _logger.Error("C-
Move request to PACS with IP: {ip}, Port: {port}, AE-
Title: {ae} timed out", client.Host, Program.config.PACS_PORT, client.CalledAe
);
```

```
                throw new DicomNetworkException("C-Move request timed out");
            };
            await client.SendAsync();
        }

        /// <summary>
        /// Pings the PACS and sends the encapsulated PDF reports to the PACS
        /// </summary>
        /// <param name="reports">A list of encapsulated PDF reports that will
 be sent to the PACS</param>
        /// <param name="client">DicomContext acting as Dicom client</param>
        public async Task SendReports(List<DicomDataset> reports, IDicomContex
t client)
        {

            if (reports == null) {
                var paramName = nameof(reports);
                throw new ArgumentNullException(paramName, $"Parameter {paramN
ame} is null.");
            }
            else if (client == null) {
                var paramName = nameof(client);
                throw new ArgumentNullException(paramName, $"Parameter {paramN
ame} is null.");
            }

            this.client = client;
            client.Logger = LogManager.GetLogger("DicomClient");
            await PingPACS();
            foreach (DicomDataset ds in reports) {
                await sendCStoreRequest(ds);
            }
        }
    }
}
```

15.16 REPORTSENDER.REPORTSENDERTEST

```
using System;
using System.IO;
using System.Threading.Tasks;
using System.Collections.Generic;
using Dicom;
using Dicom.Network;
using Moq;
using Xunit;
using DicomClientContext;

namespace ReportSender.Test
{
    public class ReportSenderTest : IDisposable
    {

        private List<DicomDataset> testDatasetList;

        public ReportSenderTest()
        {
            testDatasetList = new List<DicomDataset>();
            DirectoryInfo testFilesDir = new DirectoryInfo("..\\..\\..\\testFi
les\\");
            foreach (var file in testFilesDir.GetFiles("*.dcm")) {
                testDatasetList.Add(DicomFile.Open(file.FullName).Dataset);
            }
        }

        public void Dispose()
        {
            testDatasetList = null;
        }

        [Fact]
        public async Task TestSendReports()
        {
            // Arrange
            var dicomContextMock = new Mock<IDicomContext>();
            var pacsTarget = new PacsTarget();
            // Act
            await pacsTarget.SendReports(testDatasetList, dicomContextMock.Obj
ect);
        }

        [Fact]
        public async Task SaveDatasetsIsNullTest()
        {
            // Arrange
            var dicomContextMock = new Mock<IDicomContext>();
            var pacsTarget = new PacsTarget();
            // Assert
            await Assert.ThrowsAsync<ArgumentNullException>(async () => {
                // Act
                await pacsTarget.SendReports(null, dicomContextMock.Object);
            });
        }

        [Fact]
        public async Task SaveDatasetsCountEqualsRequestCountTest()
        {
            // Arrange
            int actualRequestsNum = 0;
            var dicomContextMock = new Mock<IDicomContext>();
            dicomContextMock.Setup(context => context.AddRequestsAsync(It.IsAn
y<IEnumerable<DicomRequest>>()))
            .Callback<IEnumerable<DicomRequest>>((requests) => {
                foreach (var request in requests) {
                    ++actualRequestsNum;
                }
                --actualRequestsNum;
            });
            var pacsTarget = new PacsTarget();
            var expectedRequestsNum = testDatasetList.Count;
            // Act
            await pacsTarget.SendReports(testDatasetList, dicomContextMock.Obj
ect);

            // Assert
            Assert.Equal(expectedRequestsNum, actualRequestsNum);
        }
    }
}
```

16 GIT VERSIONING



21/04/2021 (1 update)

Updated to a77f2146: timetable and finalization of documentation
a77f2146 Gabriel Schafflützel Just now

20/04/2021 (1 update)

Updated to 46ae3584: cleanup and documentation
46ae3584 Gabriel Schafflützel Yesterday at 18:34

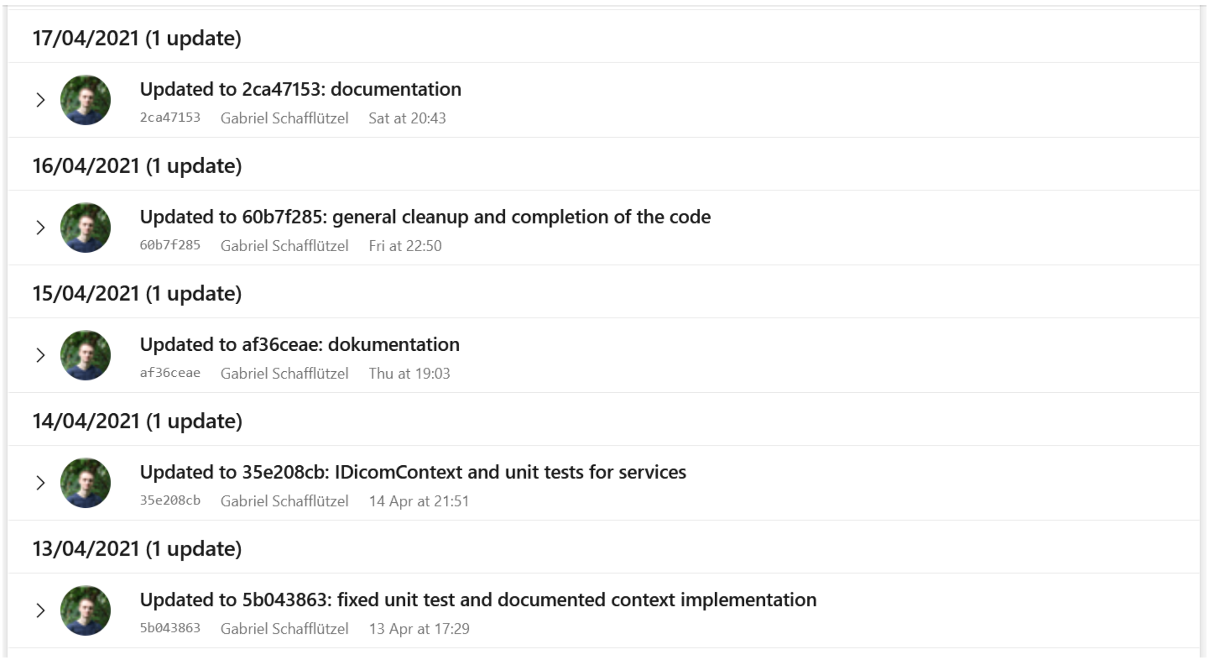
19/04/2021 (4 updates)

Updated to 85b65330: minor adjustment to documentation
85b65330 Gabriel Schafflützel Mon at 19:43

Updated to 4141b901: tests overview extracted and daily journal entry
4141b901 Gabriel Schafflützel Mon at 19:34

Updated to db69f473: adjustments and documentation
db69f473 Gabriel Schafflützel Mon at 18:17

Updated to dfa89cf5: timetable
dfa89cf5 Gabriel Schafflützel Mon at 09:25



17/04/2021 (1 update)

Updated to 2ca47153: documentation
2ca47153 Gabriel Schafflützel Sat at 20:43

16/04/2021 (1 update)

Updated to 60b7f285: general cleanup and completion of the code
60b7f285 Gabriel Schafflützel Fri at 22:50

15/04/2021 (1 update)

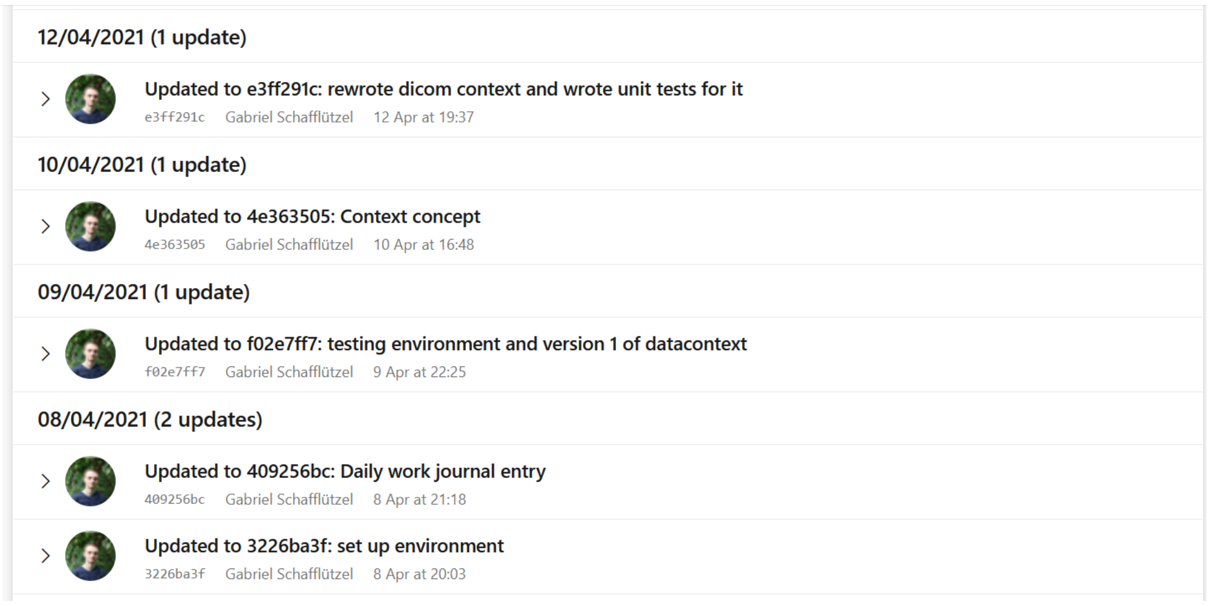
Updated to af36ceae: dokumentation
af36ceae Gabriel Schafflützel Thu at 19:03

14/04/2021 (1 update)

Updated to 35e208cb: IDicomContext and unit tests for services
35e208cb Gabriel Schafflützel 14 Apr at 21:51

13/04/2021 (1 update)

Updated to 5b043863: fixed unit test and documented context implementation
5b043863 Gabriel Schafflützel 13 Apr at 17:29



12/04/2021 (1 update)

Updated to e3ff291c: rewrote dicom context and wrote unit tests for it
e3ff291c Gabriel Schafflützel 12 Apr at 19:37

10/04/2021 (1 update)

Updated to 4e363505: Context concept
4e363505 Gabriel Schafflützel 10 Apr at 16:48

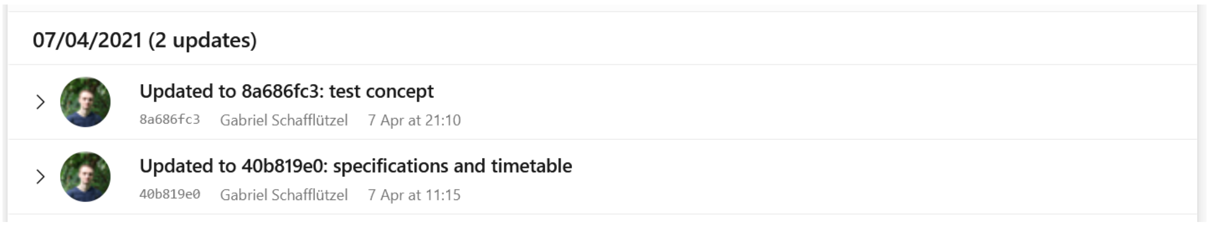
09/04/2021 (1 update)

Updated to f02e7ff7: testing environment and version 1 of datacontext
f02e7ff7 Gabriel Schafflützel 9 Apr at 22:25

08/04/2021 (2 updates)

Updated to 409256bc: Daily work journal entry
409256bc Gabriel Schafflützel 8 Apr at 21:18

Updated to 3226ba3f: set up environment
3226ba3f Gabriel Schafflützel 8 Apr at 20:03



07/04/2021 (2 updates)

Updated to 8a686fc3: test concept
8a686fc3 Gabriel Schafflützel 7 Apr at 21:10

Updated to 40b819e0: specifications and timetable
40b819e0 Gabriel Schafflützel 7 Apr at 11:15