



AARHUS UNIVERSITET

Software Architecture in Practice

Our Mandatory Case
TeleMed



- *“Software architecture is the ability to draw 7 boxes and connect them with lines...”* From ‘SA @Work’ interview with architect...
- ***My strong opinion: No, it is not just that!***
- *Why?*
 - *Because many central architectural quality attributes (like performance, security, modifiability) hinges on **specific decisions made at individual source code line level***



- Therefore: You will see code and *work* with code 😊



AARHUS UNIVERSITET

TeleMed

Intro to our Case Study system



Case - Challenges

- Demographic challenges
 - 2009: 70% of public health expenditure goes to chronic diseases
 - 2040: 100% more elderly
- Geographical challenges
 - Larger, fewer hospitals
 - Fewer general practitioners
- Leads to a need for tele medical solutions
 - ICT-supported healthcare services where some of the people participating in service delivery are not co-located with the receiver of the service
 - (Our research: [Net4Care project](#))

- Vision

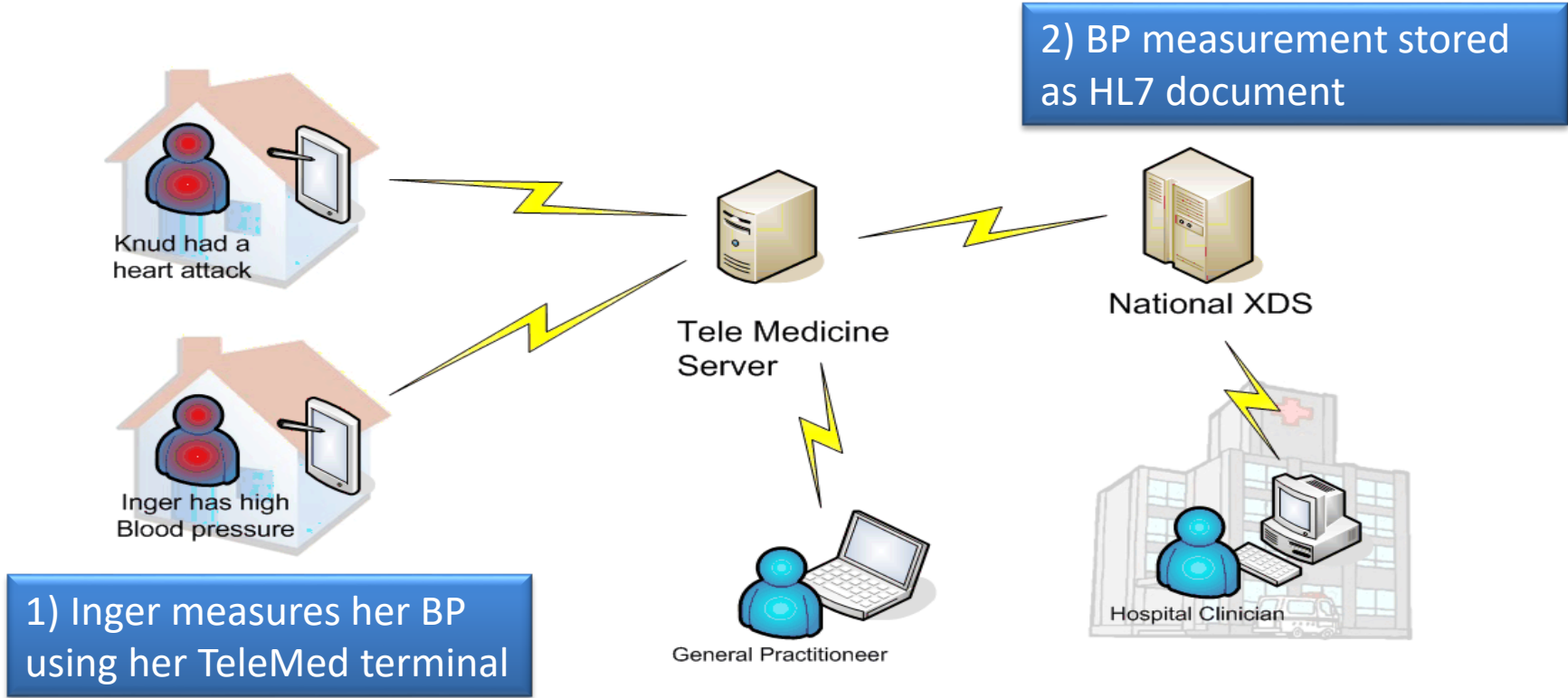
- Replace out-patient visits by measurements made by patients in their home
- *Move data from home to regional/national storage so all health care personal can view them...*

- Motivation

- Reduce out-patient visits
 - Better quality of life
 - Cost savings
 - Better traceability and visibility

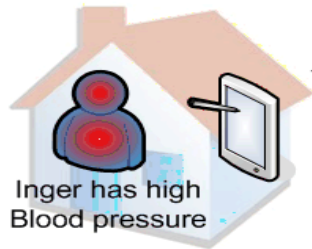
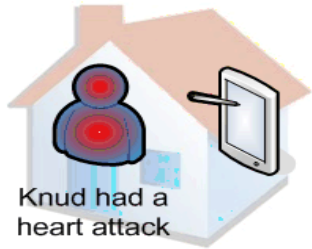


Use case 1



Use case 2

2) Query for all BP documents associated with Inger



1) GP queries last month's BP measurements for Inger using web browser



Tele Medicine Server



General Practitioner



National XDS



Hospital Clinician

(What is XDS)

- Cross-Enterprise Document Sharing:
 - One Registry + Multiple Repositories
 - Repository: Stores *clinical documents*
 - (id, document) pairs
 - Registry: Stores *metadata* with document *id*
 - Metadata (cpr, timeinterval, physician, measurement type,...)
 - Id of associate document and its repository
- Think
 - Registry = Google (index but no data)
 - Repository = Webserver (data but no index)



(What is HL7)

- HL7 is a standard (complex!) for clinical information storage and exchange.
 - Version 3 loves XML!
- Our version:

http://localhost:4567/bp/pid01

TeleMed

Observations for pid01

There are 1 observations.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<ClinicalDocument>
  <effectiveTime value="20160303121148"/>
  <patient>
    <id extension="pid01"/>
  </patient>
  <component>
    <observation>
      <code code="MSC88019" displayName="Systolisk BT"/>
      <value unit="mm(Hg)" value="115.0"/>
    </observation>
    <observation>
      <code code="MSC88020" displayName="Diastolisk BT"/>
      <value unit="mm(Hg)" value="65.0"/>
    </observation>
  </component>
</ClinicalDocument>
```

Real version:



Patient info

"Author", here interpreted as authorized/prescribed the treatment/monitoring

Custodian, the organization has HL7 "Stewardship" translate as they who responsible for storage

Documentation of the have only the Device guessed some and le

One each cont which is an me



Demo 1

AARHUS UNIVERSITET

- Start a server
 - gradle serverHttp
- Send an obs.
 - gradle homeHttp
 - -Psys=120 -Pdia=77 -Pid=pid17
- GP review in browser
 - http://localhost:4567/bp/pid01

```
csdev@m1: ~/proj/broker
Use ctrl-c to terminate!
2020-10-14T09:58:49.424+02:00 [INFO] org.eclipse.jetty.util.log :
og.Slf4jLog
2020-10-14T09:58:49.488+02:00 [INFO] sp
2020-10-14T09:58:49.488+02:00 [INFO] sp
2020-10-14T09:58:49.491+02:00 [INFO] of
8.989Z; git: e1bc35120a6617ee3df052294e
2020-10-14T09:58:49.514+02:00 [INFO] or
2020-10-14T09:58:49.514+02:00 [INFO] or
2020-10-14T09:58:49.516+02:00 [INFO] or
2020-10-14T09:58:49.531+02:00 [INFO] or
2020-10-14T09:58:49.531+02:00 [INFO] or
<=====--> 90% EXECUTING [21s]
> :telemed:serverHttp
```

Use case 1

```
csdev@m1: ~/proj/broker 89x18
csdev@m1: ~/proj/broker$ gradle homeHttp -Psys=127 -Pdia=77 -Pid=pid17
Starting a Gradle Daemon, 1 busy Daemon could not be reused, use --status for details
> Task :telemed:homeHttp
HomeClient: Asked to do operation store for patient pid17
HomeClient - completed.
Deprecated Gradle features were used in this build, making it incompatible with Gradle 7.0.
Use '--warning-mode all' to show the individual deprecation warnings.
See https://docs.gradle.org/6.5/userguide/command_line_interface.html#sec:command_line_warnings
BUILD SUCCESSFUL in 3s
6 actionable tasks:
csdev@m1: ~/proj/bro
```

Use case 2

localhost:4567/bp/pid17

← → ↻ 🏠 📄 localhost:4567/bp/pid17

TeleMed

Observations for pid17

There are 1 observations.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<ClinicalDocument>
  <effectiveTime value="2020-10-14T10:00:10+02:00"/>
  <patient>
    <id extension="pid17"/>
  </patient>
  <component>
    <observation>
      <code code="MSC88019" displayName="Systolic BP"/>
      <value unit="mm(Hg)" value="127.0"/>
    </observation>
    <observation>
      <code code="MSC88020" displayName="Diastolic BP"/>
      <value unit="mm(Hg)" value="77.0"/>
    </observation>
  </component>
</ClinicalDocument>
```



Demo 2

ifconfig ens33

- Can talk remotely
 - Fire up a new machine, note its IP

```

csdev@m104: ~
File Edit Tabs Help
TX packets:19036 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:2071861 (2.0 MB) TX bytes:2071861 (2.0 MB)

csdev@m104:~$ ifconfig ens33
ens33  Link encap:Ethernet  HWaddr 00:0c:29:87:5c:9d
        inet addr:10.11.96.127  Bcast:10.11.127.255  Mask:255.255.192.0
        inet6 addr: fd4a:544c:80d0:0:20f7:8744:68aa:ff97/64  Scope:Global
        inet6 addr: fd4a:544c:80d0:0:1862:dbf7:178a:94e3/64  Scope:Global
        inet6 addr: fd4a:544c:80d0:0:414d:37c7:4b4d:b54c/64  Scope:Global
        inet6 addr: fd4a:544c:80d0:0:6db1:297a:6194:7453/64  Scope:Global
  
```

- gradle homeHttp –Dhost=10.11.96.127

```

Command Prompt
24-06-2018 15:15 <DIR> demo
13-06-2018 09:33 <DIR> demo-rest
25-06-2018 15:05 <DIR> demo2
01-05-2018 09:37 5.468 gradlew
12-04-2018 11:05 2.260 gradlew.bat
01-05-2018 10:02 11.524 LICENSE
25-06-2018 15:05 7.619 README.md
07-06-2018 14:34 729 settings.gradle
25-06-2018 15:05 294 version.md
8 File(s) 28.683 bytes
7 Dir(s) 21.762.424.832 bytes free

d:\proj\broker>gradle home -Dhost=10.11.96.127
Starting a Gradle Daemon (subsequent builds will be faster)
BUILD FAILED: Build failed with an exception.

* What went wrong:
Task 'home' is ambiguous in root project 'broker'. Candidates are: 'homeHttp', 'homeSocket'.

* Try:
Run gradle tasks to get a list of available tasks. Run with --stacktrace option to get the stack trace. Run with --info or --debug option to get more log output.
Run with --scan to get full insights.

* Get more help at https://help.gradle.org

BUILD FAILED in 11s

d:\proj\broker>gradle home -Dhost=10.11.96.127
> Task :demo:homeHttp
HomeClient: Asked to do operation store for patient pid01
HomeClient - completed.

BUILD SUCCESSFUL in 7s
5 actionable tasks: 5 executed, 1 up-to-date
d:\proj\broker>
  
```

isen

- *Flexibility through Dependency Injection of delegates...*
- The default variant is an 'in-memory database'
 - Aka: 'Fake Object test double'
- Let us persist stuff, so restarting the server does not erase all patient data 😊
 - A real XDS (pain and agony)
 - *Let us try a document based NoSQL database engine:*
MongoDB

- Start a MongoDB
 - Option A: download lubuntu linux, create ISO image, create and install a new linux (virtual) machine, start it, download and install mongodb, note its hostname/ip address (1.5 hours)
 - Option B: Issue a single Docker command 😊 (20 secs)
 - `docker run -d --name db0 -p 27017:27017 mongo`
- Docker is a light-weight virtual machine monitor
 - Ala 'VMWare Workstation as linux command line'
 - *We will return to virtualization in next course...*

- Start server *with db*
 - gradle serverHttp `-Pdb=localhost`
- Upload a few blood pressures
- Verify contents in Mongo
 - docker exec `-ti db0 mongo`
 - And fire a few weird MongoDB console commands 😊
 - *MongoDb will be an example system in the next course...*

```
* dev@m1: ... x dev@m1: ~/... x
> use xds
switched to db xds
> show collections
tm16
> db.tm16.find().pretty()
{
  "_id" : ObjectId("5b8532163d65330b116cfb21"),
  "pid" : "pid01",
  "timestamp" : NumberLong("1535455764000"),
  "hl7" : "<?xml version='1.0' encoding='UTF-8' s
/>\n<patient>\n<id extension='pid01'>\n</patient>\n<comp
value unit='mm(Hg)\" value='156.0\">\n</observation>\n<o
t='mm(Hg)\" value='88.0\">\n</observation>\n</component>
}
}
{
  "_id" : ObjectId("5b8532223d65330b116cfb22"),
  "pid" : "pid01",
  "timestamp" : NumberLong("1535455777000"),
  "hl7" : "<?xml version='1.0' encoding='UTF-8' s
/>\n<patient>\n<id extension='pid01'>\n</patient>\n<comp
value unit='mm(Hg)\" value='154.0\">\n</observation>\n<o
t='mm(Hg)\" value='89.0\">\n</observation>\n</component>
}
}
>
```



- And ... Tada!
- Measurements survive a server shutdown and restart 😊



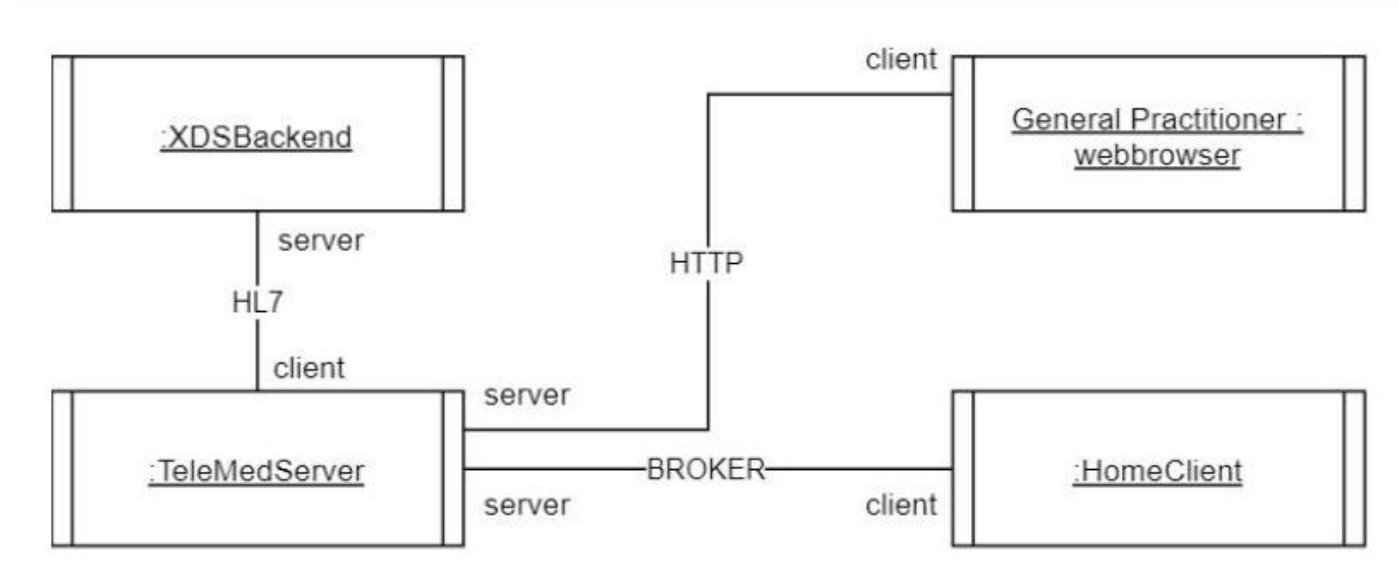
AARHUS UNIVERSITET

TeleMed Architecture

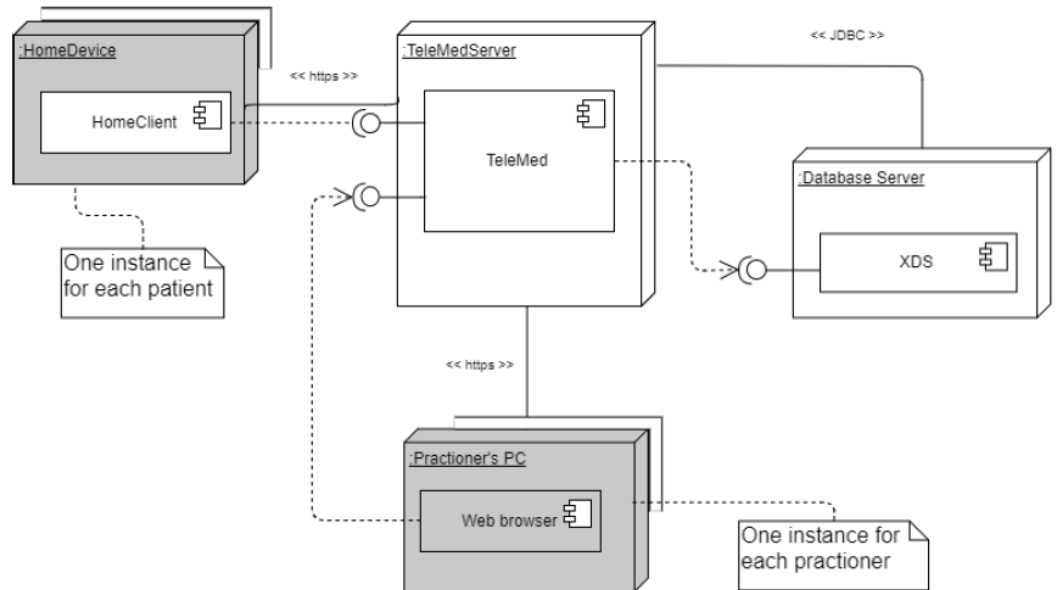
Views

Thanks to former SAiP students!

- Elements
 - The four standalone services by name
- Relations
 - Main protocol name and roles



- Elements
 - The machines and their deployed software units
- Relations
 - The network and interfaces



- Elements
 - Interfaces and implementing classes
- Relations
 - Associations etc.

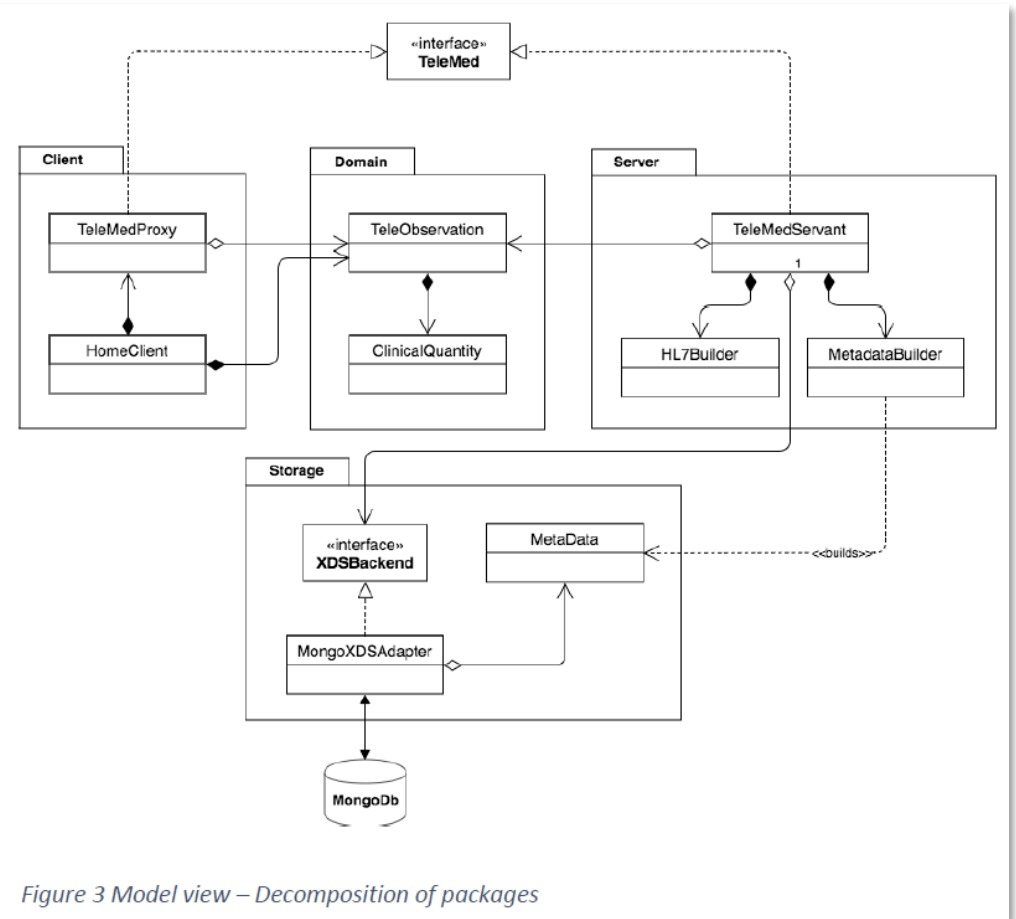


Figure 3 Model view – Decomposition of packages

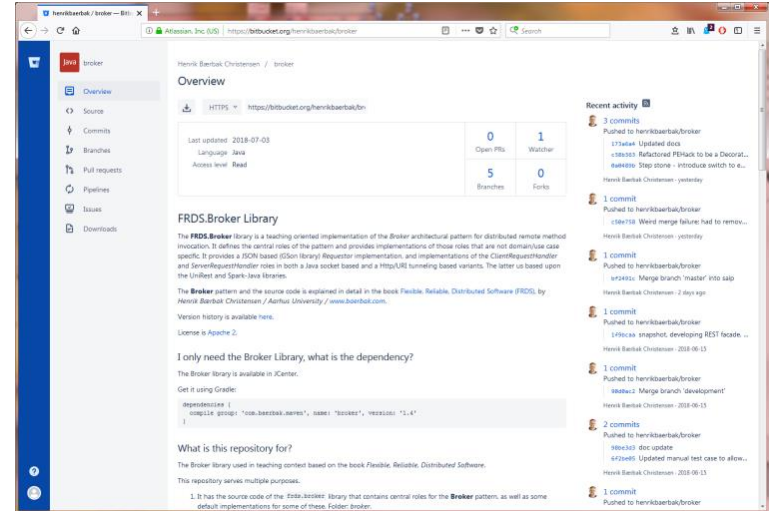


AARHUS UNIVERSITET

Getting TeleMed

- TM is on Bitbucket.org
 - bitbucket.org/henrikbaerbak/broker

- ... in
 - folder 'demo' ☹️
 - branch 'saip' !!!



- Fork it into a **private repository**
- Or clone it, and reset git origin
- Or just get the ZIP and use whatever SCM you like in the group...



More getting started...

- Read the 'README-SAIP.md' in root folder...
 - Tutorial on how to run
- Read 'Tools' web page from the blackboard course pages...



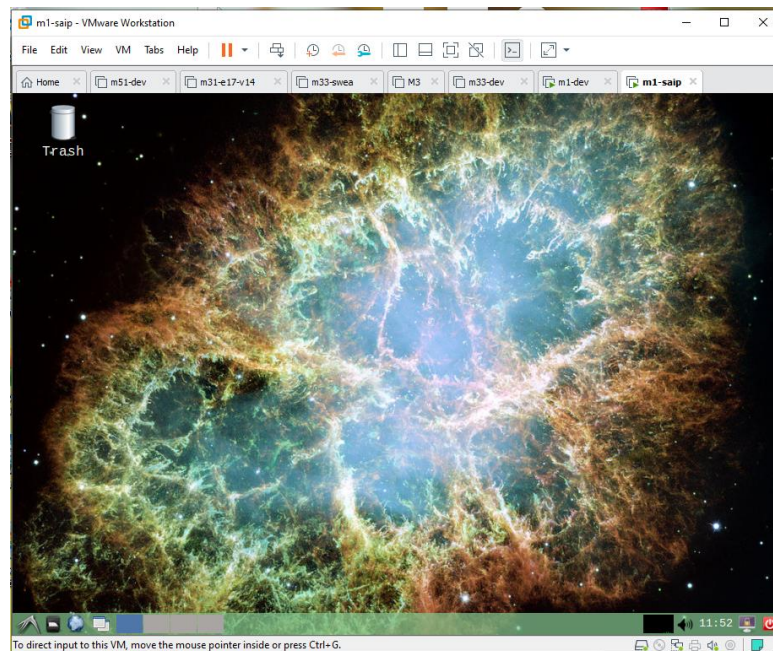
Toolchain

Edit, Compile, Debug,
throw something at the cat...



- Academia generally sticks to the Java world
 - It is generally open source and free of charge!
- TM tool chain
 - Java 8+, Gradle 7+, Junit, IntelliJ, Git
 - And Docker, MongoDB, JMeter, influx, Graphene, ...
- Installation options – **Follow the ‘tools’ web page**
 - A) Install it all on your machine OR
 - B) Get hold of VMWare Workstation and Mx
 - You can get a 1 year license for the full workstation as AU student

- “Mx for some x”
- Lubuntu **Minimal** desktop
- Ubuntu but
 - Much smaller
 - Pretty ‘raw’





Getting Started

- Getting TeleMed up and running
 - Start 'IntelliJ' and choose 'Open', browse to the 'broker' folder and click on the gradle icon.
 - The TeleMed code base is in the 'demo' folder
 - Standard maven/gradle folder structure for Java
 - Main/java = production code
 - Test/java = test code
- Learning test
 - Review 'TestStory1' in telemed.scenario in test folder!

*Details: See Tools
web page*



TestStory1

AARHUS

M104Dev on Rundo.cs.au.dk



- IntelliJ
 - Powertool, but
- It is like pilot'ing the Airbus 320 ☹️
 - One zillion handles to crank
- Ask at forum, review my guides, google
 - <https://baerbak.cs.au.dk/c/tutorial/intellij-gradle.html>
 - Tools page



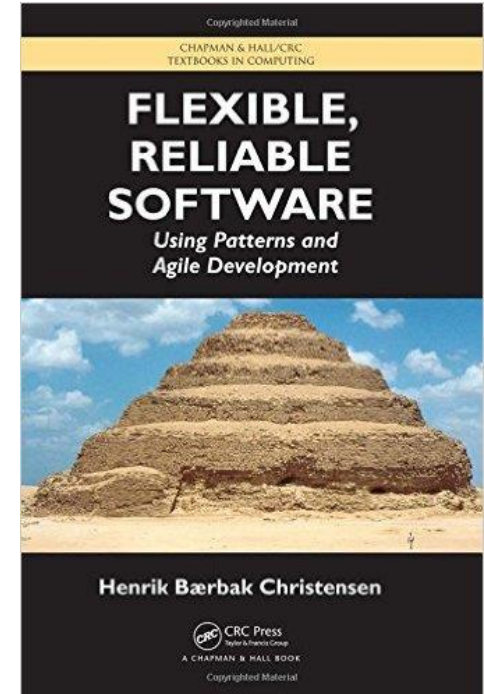
AARHUS UNIVERSITET

Architecture Prerequisites

Or – what I assume that you already know from the architect's toolbox...

Dependency Injection

- I have written a book on *Flexibility*
- *Program to interface*
 - **Role** is expressed by interface
- *Favor Object Composition*
 - Small services **collaborate** to form whole
- *Dependency injection*
 - The services you collaborate with are provided to you (injected)



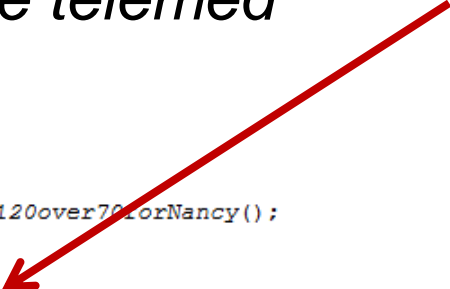
- TeleMed : Role of a full tele medicine system/server
- XDSBackend: Role of an XDS database system
- *Xds is injected into the telemed*

```
@Before
public void setup() {
    teleObs1 = HelperMethods.createObservation120over70forNancy();
    // Create server side implementations
    xds = new FakeObjectXDSDatabase();
    TeleMed tsServant = new TeleMedServant(xds);

    // Server side broker implementations
    Invoker invoker = new StandardJSONInvoker(tsServant);

    // Create client side broker implementations
    ClientRequestHandler clientRequestHandler = new LocalMethodCallClientRequestHandler(invoker);
    Requestor requestor = new StandardJSONRequestor(clientRequestHandler);

    // Finally, create the client proxy for the TeleMed
    telemed = new TeleMedProxy(requestor);
}
```



I can configure any suitable variant of the system by selecting the right implementations of roles to inject!



The Compositional Principles

- *Encapsulate what varies*
 - **Responsibilities that may vary** are encapsulated in a **Role**
- *Program to interface*
 - **Role** is expressed by interface
- *Favor Object Composition*
 - *Fine-grained roles* **collaborate** to form whole
- This design thinking naturally leads to Design Patterns
- *Basically the SOLID principles in operational format...*



- The 'Storage' role
 - Responsibilities:
 - To store patient's blood pressure measurement
 - To fetch sets of measurements
- *Program to interface:* Interface **XDSBackend**
- *Object Composition:*
 - *Instead of TeleMed object itself issuing, say, SQL statements, it delegates to its XDSBackend instance (injected) to perform its store and fetch operations*

```
public TeleMedServant(XDSBackend xds) { this.xds = xds; }

@Override
public String processAndStore(TeleObservation teleObs) {

    // Performance testing hack: Overwrite client side
    // time stamp with present time
    teleObs.setTime(LocalDateTime.now().truncatedTo(ChronoUnit.SECONDS));

    // Generate the XML document representing the
    // observation in HL7 (HealthLevel7) format.
    HL7Builder builder = new HL7Builder();
    Director.construct(teleObs, builder);
    Document hl7Document = builder.getResult();

    // Generate the metadata for the observation
    MetadataBuilder metaDataBuilder = new MetadataBuilder();
    Director.construct(teleObs, metaDataBuilder);
    Metadata metadata = metaDataBuilder.getResult();

    // Finally store the document in the XDS storage system
    String uniqueId = null;
    uniqueId = xds.provideAndRegisterDocument(metadata, hl7Document);

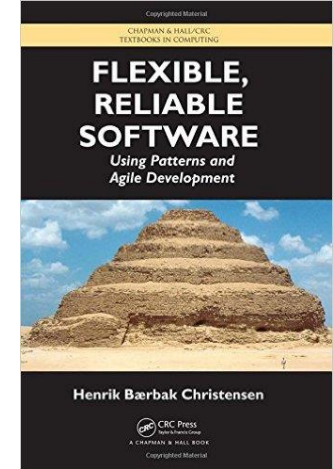
    return uniqueId;
}
```



- I can implement an XDSBackend by
 - Not using a real XDS storage system but
- MongoDB
 - Because it is NoSQL database and it is part of our curriculum...

Test Doubles

- I have written a book on *Reliability*
- *Test Doubles*
 - *Replacements for real 'depended-on units' that are under test control*
- *Test cases in JUnit*
 - *Inject test doubles instead of 'real' units to ease and control testing*



- XDSBackend: Role of an XDS database system
- FakeObjectXDSDatabase: A fake test double
 - No persistence, all in-memory!

```
@Before
public void setup() {
    teleObs1 = HelperMethods.createObservation120over70forNancy();
    // Create server side implementations
    xds = new FakeObjectXDSDatabase();
    TeleMed tsServant = new TeleMedServant(xds);

    // Server side broker implementations
    Invoker invoker = new StandardJSONInvoker(tsServant);

    // Create client side broker implementations
    ClientRequestHandler clientRequestHandler = new LocalMethodCallClientRequestHandler(invoker);
    Requestor requestor = new StandardJSONRequestor(clientRequestHandler);

    // Finally, create the client proxy for the TeleMed
    telemed = new TeleMedProxy(requestor);
}
```

I can test TeleMed code without starting a real XDS database server; it is much **faster** and initial state is **well-defined** = empty database

- Several kinds of test doubles exists (subtypes):
 - **Stub**: Get indirect input under control
 - **Spy**: Get indirect **output** under control
 - to validate that UUT use the proper protocol
 - count method calls, ensure proper call sequence
 - **Mock**: A spy with *fail fast* property
 - Frameworks exists that test code can ‘program’ mocks without every coding them in the native language
 - Fail fast: fail on first breaking of protocol
 - **Fake**: A lightweight but realistic double
 - when the UUT-DOU interaction is slow and tedious
 - when the Double interaction is not the purpose of test



AARHUS UNIVERSITET

Broker

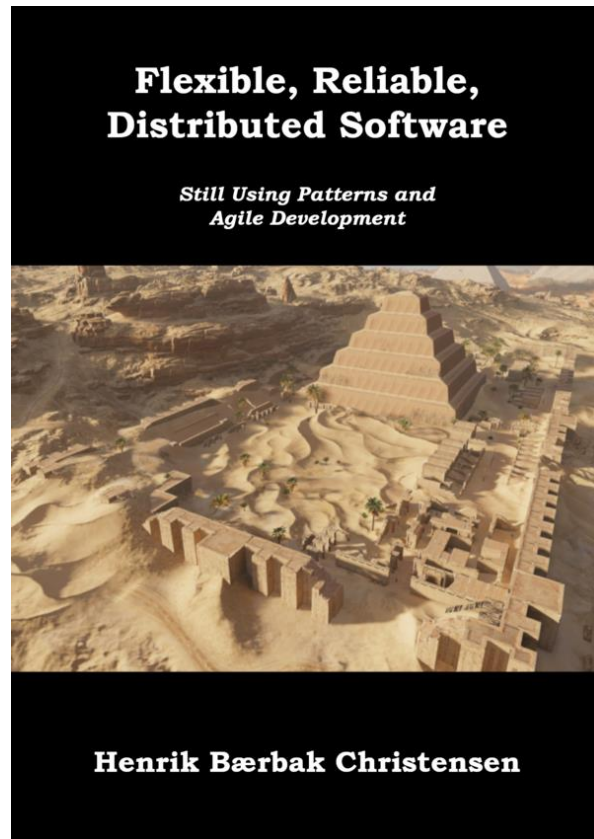
The Central Distribution Pattern



Distributed Computing

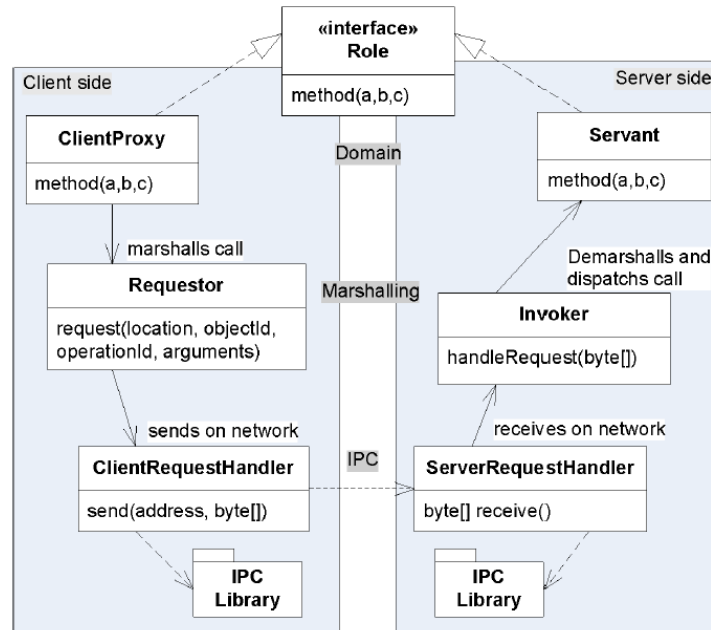
- I have written *Flexible, Reliable, Distributed Software...*
 - leanpub.com/frds
 - Costs ~12\$

- *Core contents:*
 - *The Broker pattern*
 - *REST based protocol*
 - *Will be curriculum later in the course*



- **Broker**

Intent Define an loosely coupled architecture that allows methods to be called on remote objects while having flexibility in choice of operating system, communication protocol, and marshaling format.

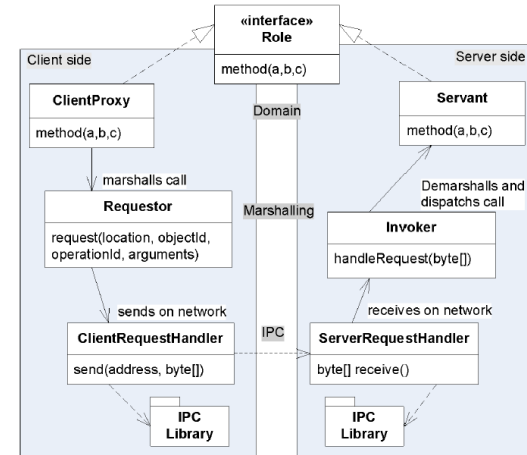




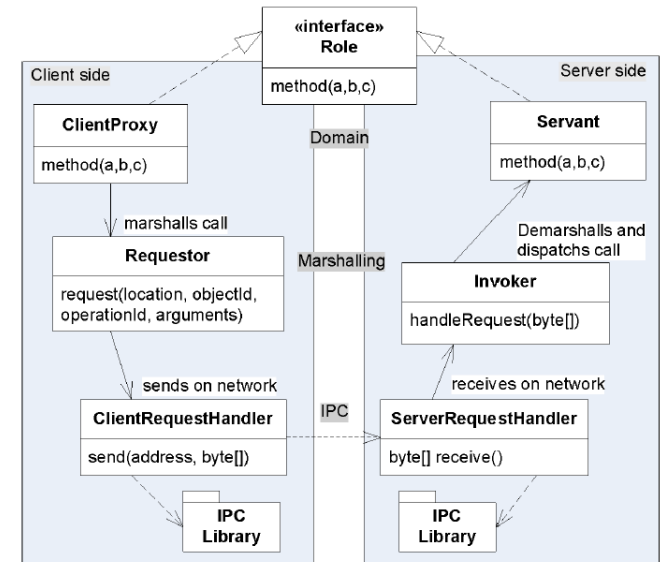
- Example
 - TeleMed object is on machine 'server'
 - On client we want to call
 - `teleMed.processAndStore(myBloodPressure);`
- But networks only have
 - `send(address, byte[]);`
 - `byte[] receive();`

Client Side Broker

- Broker dynamics on Client side
 - ClientProxy:
 - Implements TeleMed interface
 - Convert every call into *requests* to the Requestor
 - Requestor:
 - Does *marshalling* = convert data to byte array format
 - Typically JSON or XML
 - ClientRequestHandler
 - Binds to the OS and particular network protocol
 - Sockets, HTTP, Messaging
 - Does the 'send(payload) and blocks until answer returned



- Broker dynamics on Server side
 - ServerRequestHandler:
 - Binds to the OS and network protocol
 - Receives payload from ClientRequestHandler
 - Invoker:
 - Demarshalls byte[] into parameters
 - Dispatches to proper method and proper servant object
 - Servant:
 - Implements TeleMed interface
 - The *real* implementation!



Why all the trouble?

- Now we can *configure* our own Broker system
- A HTTP based client

```
// Configure the client side implementations of the Broker roles
ClientRequestHandler clientRequestHandler = new UriTunnelClientRequestHandler(hostname, 4567);
Requestor requestor = new StandardJSONRequestor(clientRequestHandler);
TeleMed teleMed = new TeleMedProxy(requestor);
```

- And HTTP webserver based server

```
// Create server side implementation of Broker roles
TeleMed tsServant = new TeleMedServant(xds);
Invoker invoker = new StandardJSONInvoker(tsServant);
UriTunnelServerRequestHandler srh =
    new UriTunnelServerRequestHandler(invoker, xds, port);
srh.registerRoutes(); // This will automatically spawn a tread for the web server
```

- (See the Broker code ('master' branch) for a socket based variant)

And Later

- A RabbitMQ based IPC layer
 - Just implement the two **roles**
 - ClientRequestHandler
 - ServerRequestHandler
- ... using RabbitMQ's RPC technique
- Messaging is curriculum later this year...





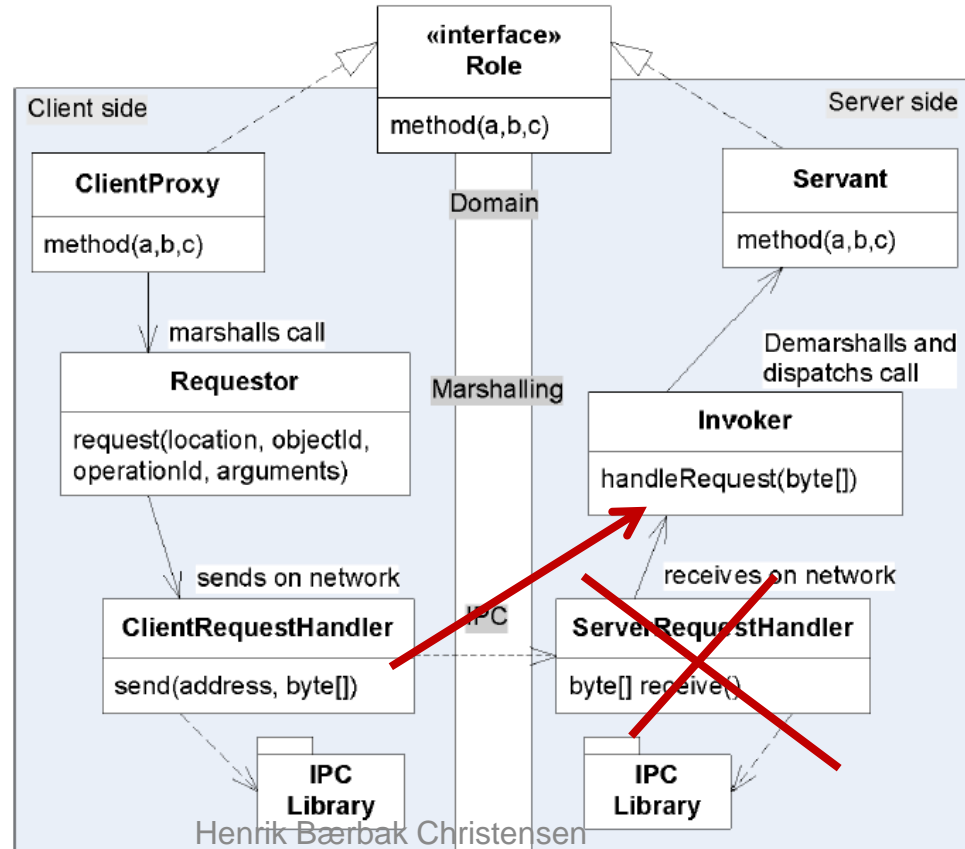
Unit Testing Distribution!

- Why code the broker myself?
- One big advantage:
 - *Test doubles*
 - *Now single VM system!*

```
public class LocalMethodCallClientRequestHandler implements ClientRequestHandler {  
  
    private Invoker invoker;  
    private ReplyObject lastReply;  
  
    public LocalMethodCallClientRequestHandler(Invoker invoker) {  
        this.invoker = invoker;  
    }  
  
    @Override  
    public ReplyObject sendToServer(String objectId, String operationName, String onTheWireFormat) {  
        // The send to the server can be mimicked by a direct method call...  
        lastReply = invoker.handleRequest(objectId, operationName, onTheWireFormat);  
        return lastReply;  
    }  
}
```

```
@Before  
public void setup() {  
    teleObs1 = HelperMethods.createObservation120over70forNancy();  
    // Create server side implementations  
    xds = new FakeObjectXDSDatabase();  
    TeleMed tsServant = new TeleMedServant(xds);  
  
    // Server side broker implementations  
    Invoker invoker = new StandardJSONInvoker(tsServant);  
  
    // Create client side broker implementations  
    ClientRequestHandler clientRequestHandler = new LocalMethodCallClientRequestHandler(invoker);  
    Requestor requestor = new StandardJSONRequestor(clientRequestHandler);  
  
    // Finally, create the client proxy for the TeleMed  
    telemed = new TeleMedProxy(requestor);  
}
```


- Or – using UML architecture:





AARHUS UNIVERSITET

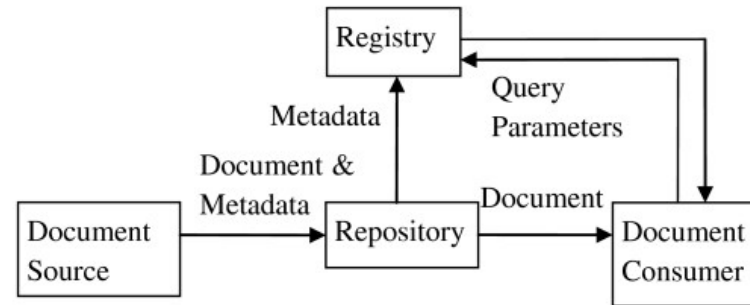
Mandatory 1

Intro to our Case Study system

Exercise 1

- Goal:
 - Get yourself started with the TeleMed system
 - Dig into the code base, understand the Broker
 - Do a *little bit of View based documentation*
 - *CC and Deployment views*
 - Of the XDS part
 - Get to know your group mates 😊

XDS is often documented using this 'rich picture':





Digging in...

- Where to start?
 - Lots of code, many variants
- Learning tests!
 - Demo/src/test TestScenario1.java
- Manual tests!
 - Review README-SAIP and build.gradle

- TeleMed report template
 - LaTeX 😊
 - But ***don't go there*** if you do not know LaTeX !!!
- *Find the link on the exercise page!*

- ***Remember timeboxing!***
 - *Struggle with an issue for one hour then **raise the white flag***
 - *That is, ask team members, ask me on forum*

Mandatory Project: Software Architecture of the TeleMed System

Software Architecture in Practice

Group: (Group name)

Members: (Names)

(Date)

Abstract

The TeleMed system implements an information system for supporting tele medicine, i.e. patients making measurements in their homes for review by general practitioners as well as hospital clinicians. This report gives a software architecture description of an architectural prototype of the TeleMed system. The techniques used for architectural description are taken from [Christensen et al., 2016].

1 Introduction

Figure 1 shows a schematic overview of TeleMed.

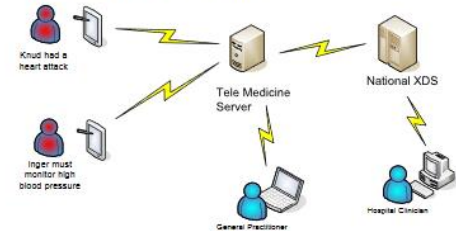


Figure 1: Rich picture of the TeleMed architecture

TeleMed is based on a device/tablet in the home that can make measurements of blood pressure and upload these to a TeleMed web server. Such a

No, you are not stupid!



AARHUS UNIVERSITET

Use IntelliJ for Digging Code

- Hit the 'Ctrl-Q' over an item, to see the JavaDocs

```
// And given the client side broker implementations, using the local
// method client request handler to avoid any real IPC layer.
ClientRequestHandler clientRequestHandler =
    new LocalRequestor req
    new Stand
// Then it is
// that voids
teleMed = new
}
```

frds.broker
public interface **ClientRequestHandler**
The Client Request Handler role in the Broker pattern. It is responsible for all inter-process-communication (IPC) on behalf of client objects. It is called by the Requestor role. It communicates over the network with an associated ServerRequestHandler on the server side.
broker.broker.main

Jump to Declaration

- Just hit 'ctrl-b' when cursor in any identifier to jump to declaration

```
// Then it is Given that we can create a client proxy
// that voids any real IPC communication
teleMed = new TeleMedProxy(requestor);
}
```

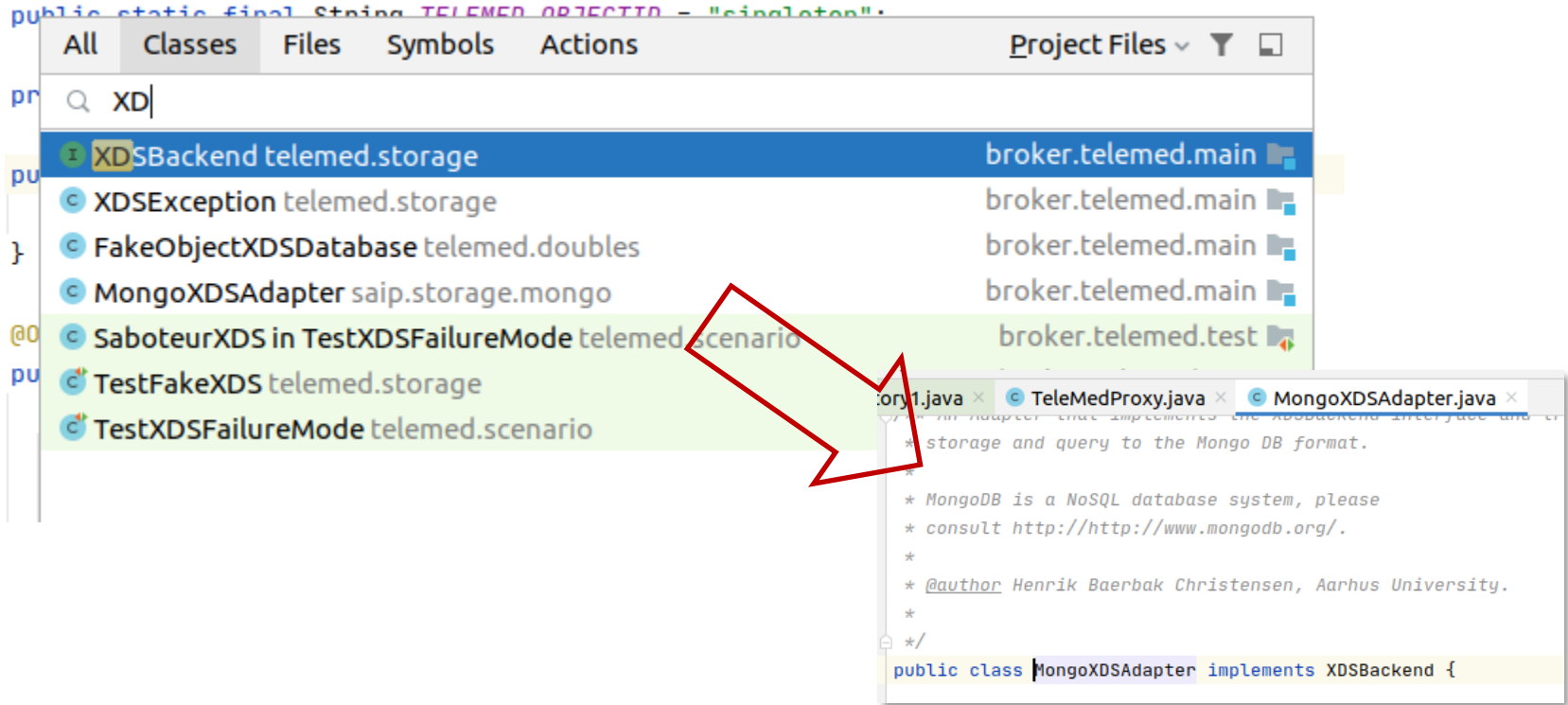


```
roxy > TeleMedProxy
testStory1.java x TeleMedProxy.java x
/* As there is only ONE telemed servant object, the
objectid is really not used in the TeleMed case, so
we just provide a 'dummy' string.
*/
public static final String TELEMED_OBJECTID = "single

private final Requestor requestor;

public TeleMedProxy(Requestor requestor) {
    this.requestor = requestor;
}
```


- Hit 'ctrl-n' and begin typing to find stuff quickly



The screenshot shows an IDE search window with the following search results:

Search Results	Location
XDSBackend	telemed.storage
XDSEException	telemed.storage
FakeObjectXDSDatabase	telemed.doubles
MongoXDSAdapter	saip.storage.mongo
SaboteurXDS in TestXDSFailureMode	telemed.scenario
TestFakeXDS	telemed.storage
TestXDSFailureMode	telemed.scenario

A red arrow points from the 'SaboteurXDS in TestXDSFailureMode telemed.scenario' result to a preview window showing the code for `MongoXDSAdapter.java`:

```
public class MongoXDSAdapter implements XDSBackend {
```