AARHUS UNIVERSITET

# Software Architecture in Practice

## Our Mandatory Case
## TeleMed

AARHUS UNIVERSITET

# Motivation

- *"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, energy efficiency, etc.) hinges on* ***specific decisions made at individual source code line level***

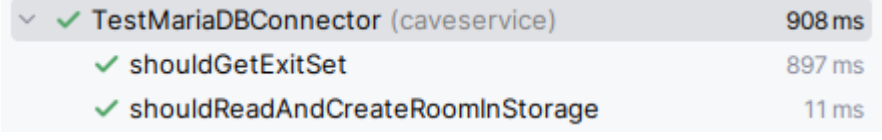
AARHUS UNIVERSITET

# Motivation

- Example: My CaveService implementation's connector to a SQL database
  - *I am no SQL expert*

```
statement.executeUpdate( s: "CREATE TABLE IF NOT EXISTS " + ROOMS_TABLE_NAME + " (" +
        "ID INT NOT NULL AUTO_INCREMENT, " +
        "POSITION VARCHAR(64) NOT NULL, " +
        "DESCRIPTION VARCHAR(256), " +
        "CREATOR_ID VARCHAR(256), " +
        "CREATE_TIME TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP, " +
        "PRIMARY KEY (ID))");
```

- Works perfectly *Functionally correct*



# However, …

- When subjected to high workloads
  - A) ‘Too many connections’ errors meant lost data
  - B) Corrected, introducing a C3PO connection pool
- *Functionally correct (again)*
  - *… but*
    - *Could not every force anything beyond 250 tps*
      - That is: performance is low
- The culprit?
  - All queries are based on criteria ‘POSITION’ which was not indexed…

AARHUS UNIVERSITET

# Performance

- So, a line of code is changed…

```
statement.executeUpdate( s: "CREATE TABLE IF NOT EXISTS " + ROOMS_TABLE_NAME + " (" +
        "ID INT NOT NULL AUTO_INCREMENT, " +
        "POSITION VARCHAR(64) NOT NULL, " +
        "DESCRIPTION VARCHAR(256), " +
        "CREATOR_ID VARCHAR(256), " +
        "CREATE_TIME TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP, " +
        "PRIMARY KEY (ID))");
```

```
statement.executeUpdate( s: "CREATE TABLE IF NOT EXISTS " + ROOMS_TABLE_NAME + " (" +
        "ID INT NOT NULL AUTO_INCREMENT PRIMARY KEY, " +
        "POSITION VARCHAR(64) NOT NULL, " +
        "DESCRIPTION VARCHAR(256), " +
        "CREATOR_ID VARCHAR(256), " +
        "CREATE_TIME TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP, " +
        "UNIQUE INDEX (POSITION))"); // Important as all queries are on position
```

- Functionally indifferent Architecturally significant

# Buttomline

Many architectural qualities are defined at code level; not at “box-and-line” level!

# Motivation

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

AARHUS UNIVERSITET

# TeleMed

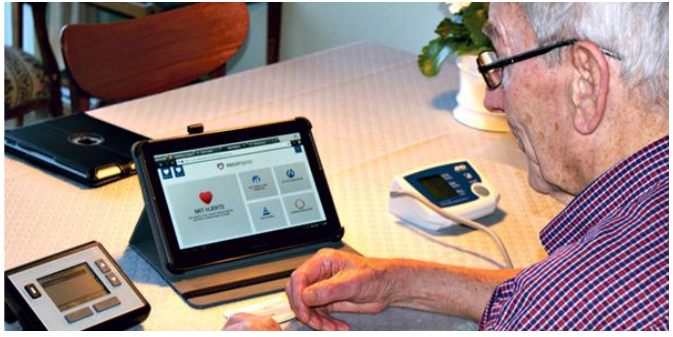
Intro to our Case Study system

# 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)

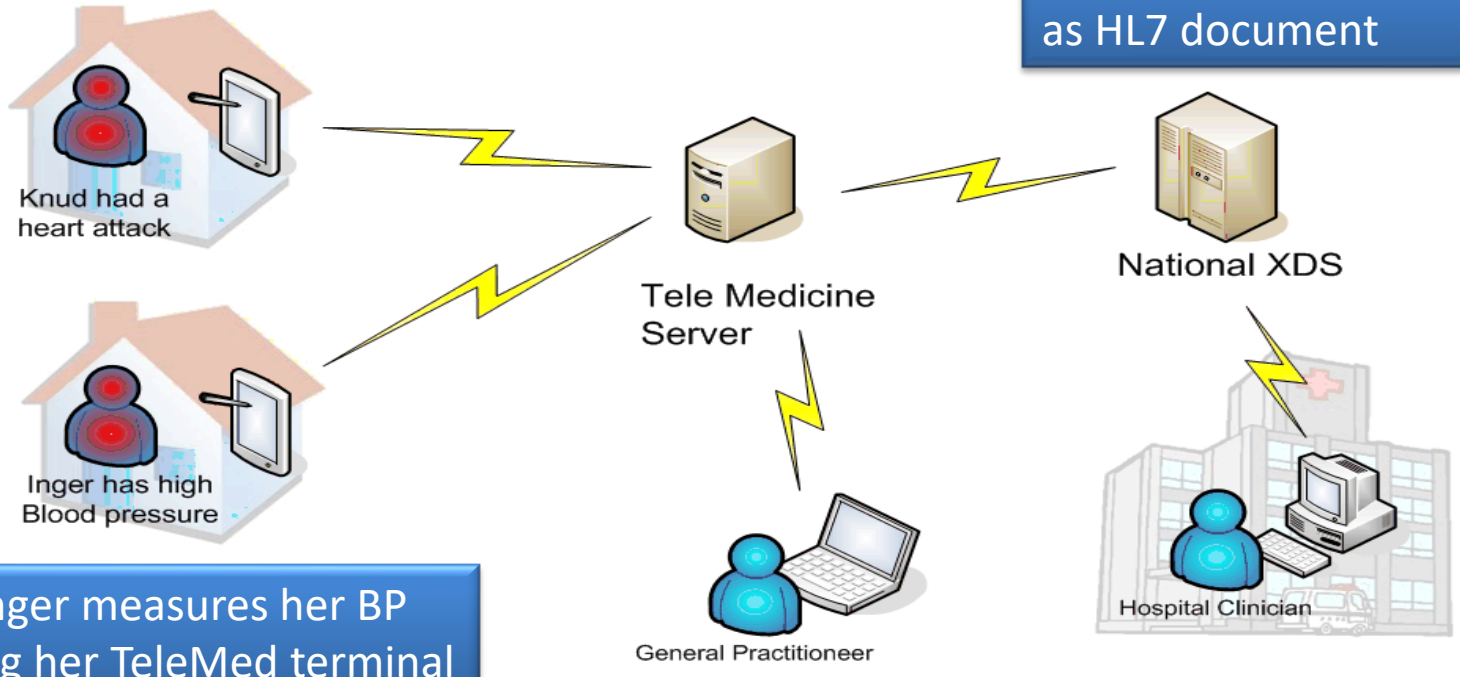
AARHUS UNIVERSITET

# Vision

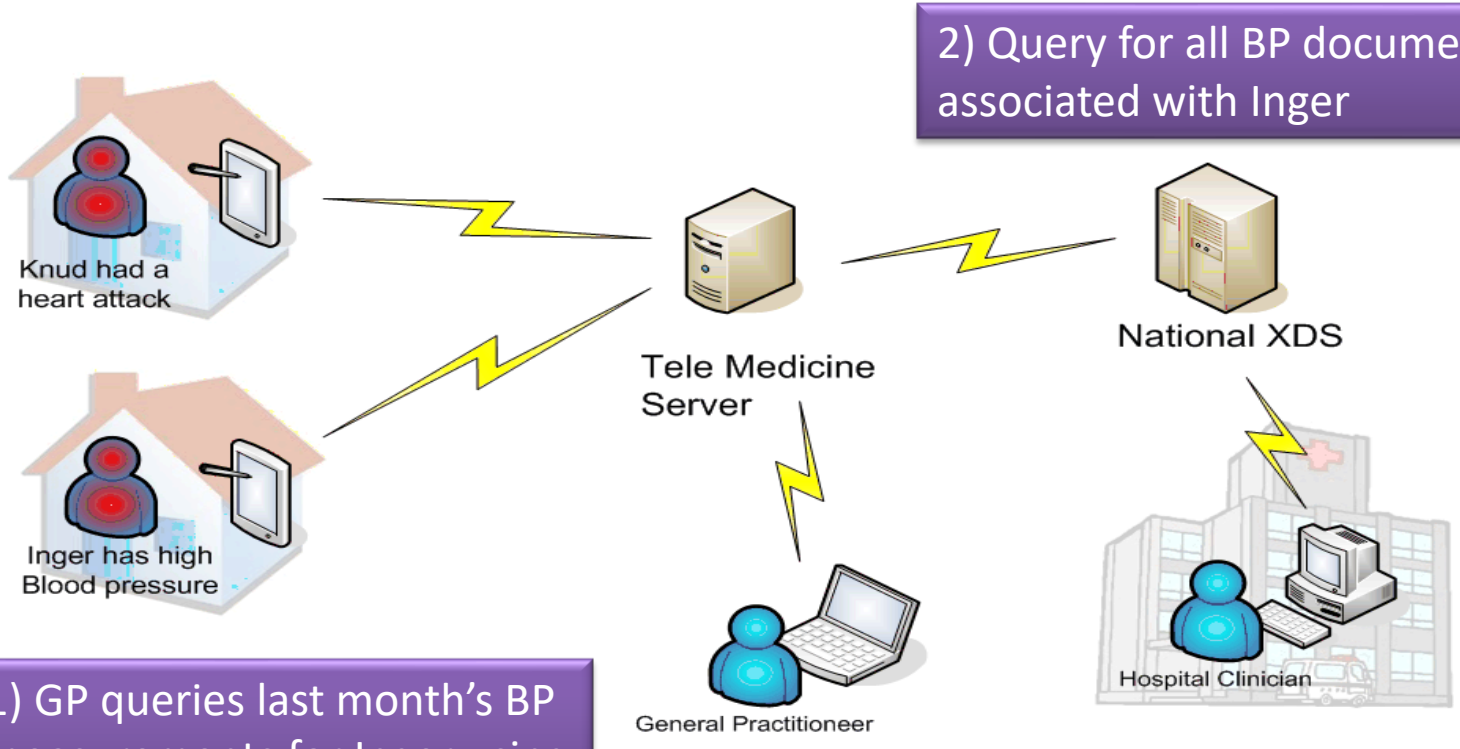


- Vision
  - Replace out-patient visits by measurements made by patients in their home
  - *Move data from home to regional/national storage so all health care personal can view them...*
- Motivation
  - Reduce out-patient visits
    - Better quality of life
    - Cost savings
    - Better traceability and visibility

# Use case 1


2) BP measurement stored as HL7 document
Knud had a heart attack
Inger has high Blood pressure
Tele Medicine Server
National XDS
General Practitioneer
Hospital Clinician
1) Inger measures her BP using her TeleMed terminal

# Use case 2


2) Query for all BP documents associated with Inger
Knud had a heart attack
Inger has high Blood pressure
Tele Medicine Server
National XDS
General Practitioneer
Hospital Clinician
1) GP queries last month's BP measurements for Inger using web browser

AARHUS UNIVERSITET

# (What is XDS)

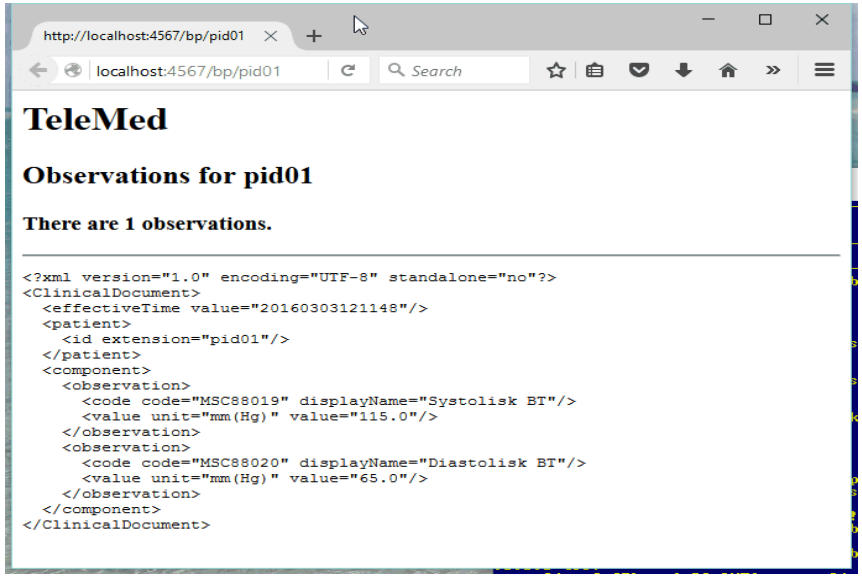
- Cross-Enterprise Document Sharing:
  - One Registry + Multiple Repositories
  - Repository: Stores *clinical documents*
    - (id, document) pairs
  - Registry: Stores *metadata* with document *id*
    - Metadata (cpr, timeinterval, physician, measurement type,...)
    - Id of associate document and its repository

- Think
  - Registry = Google (index but no data)
  - Repository = Webserver (data but no index)

AARHUS UNIVERSITET

# (What is HL7)

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




Real version:

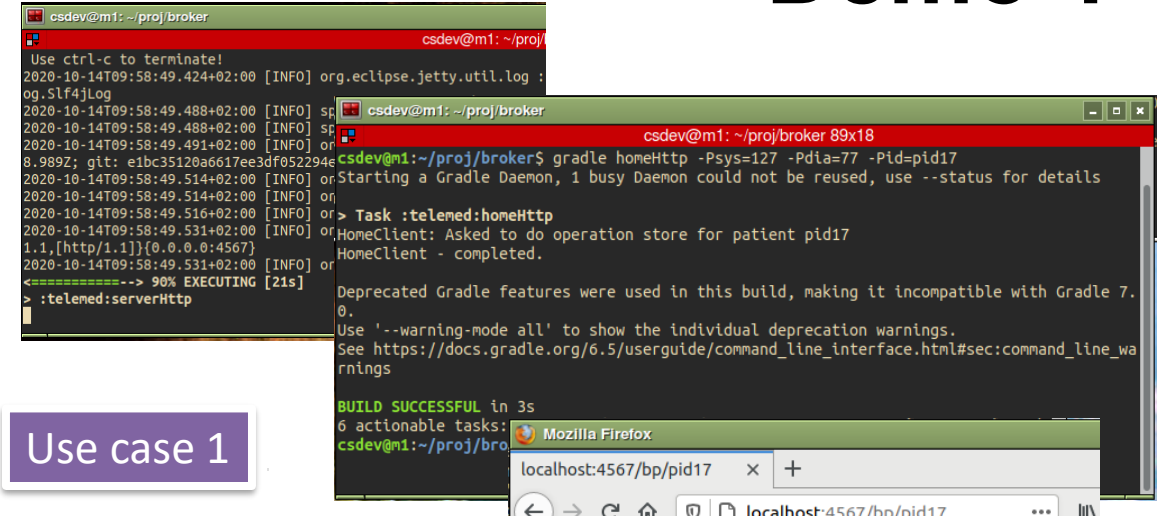
AARHUS UNIVERSITET

# Demo 1

- 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

Use case 1

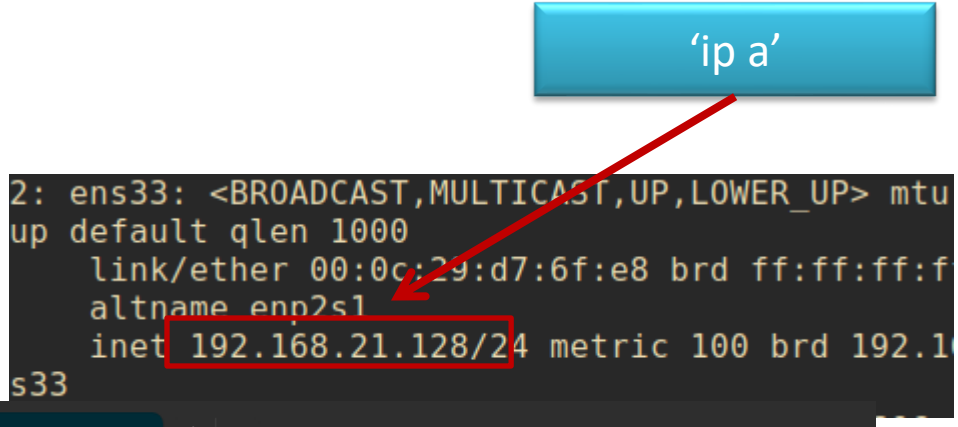
Use case 2

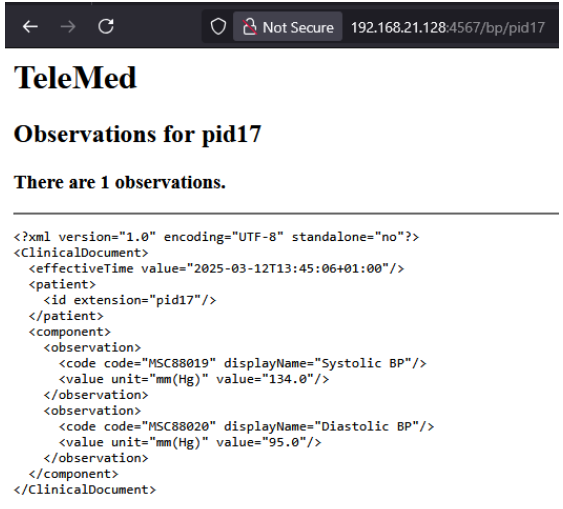


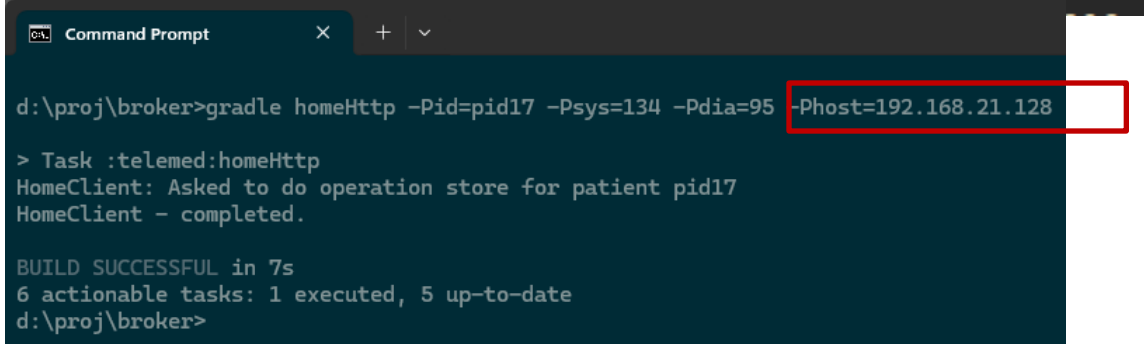
AARHUS UNIVERSITET

# Demo 2

- *Can* talk remotely
  - *Fire up a new machine, note its IP*
  - gradle homeHttp –Phost=(ip)



# Demo 3

- *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***

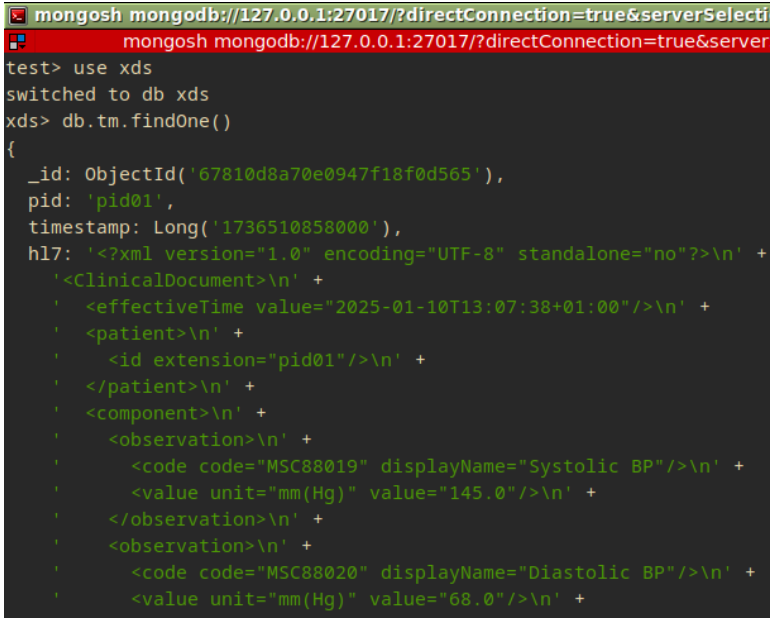
# Demo 3

- 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:8.0*
- Docker is a light-weight virtual machine monitor
  - Ala 'VMWare Workstation as linux command line'
  - *We will return to virtualization in next course…*

AARHUS UNIVERSITET

# Demo 3

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



# Demo3

- And ... Tada!

- Measurements survive a server shutdown and restart ☺

AARHUS UNIVERSITET

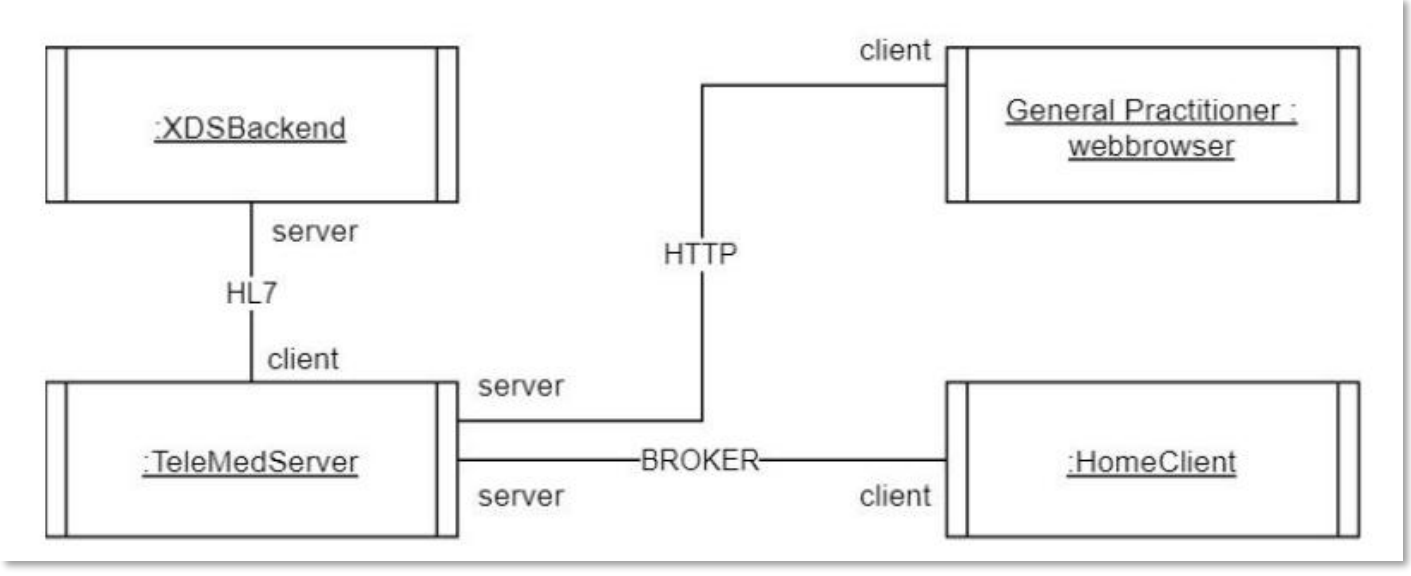
# TeleMed Architecture

Views

*Thanks to former SAiP students!*

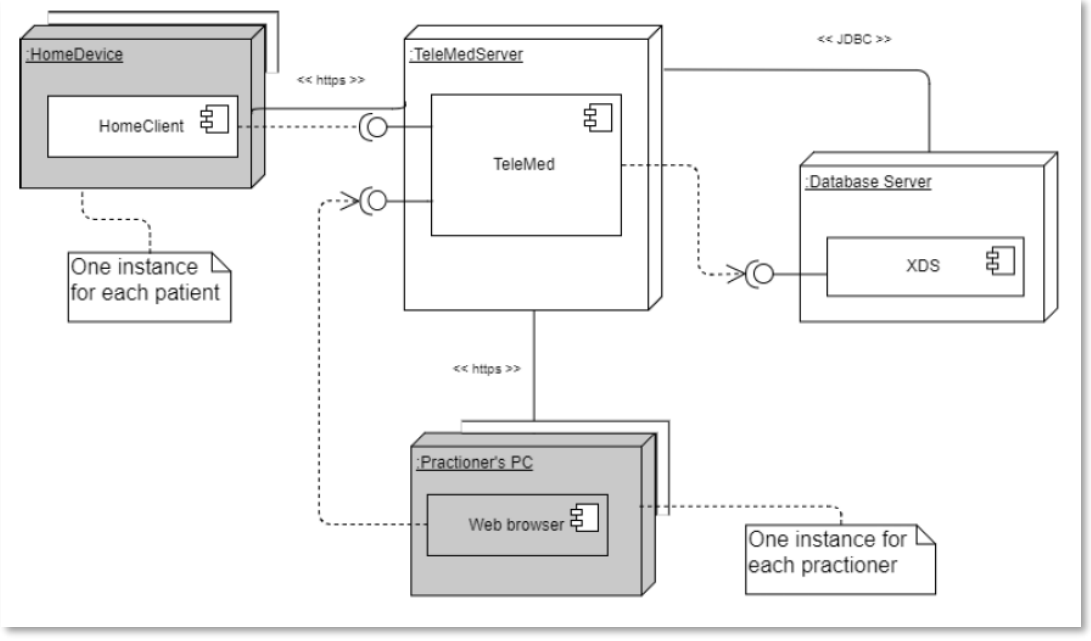
# CC View

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



# Allocation View

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



# Module View

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

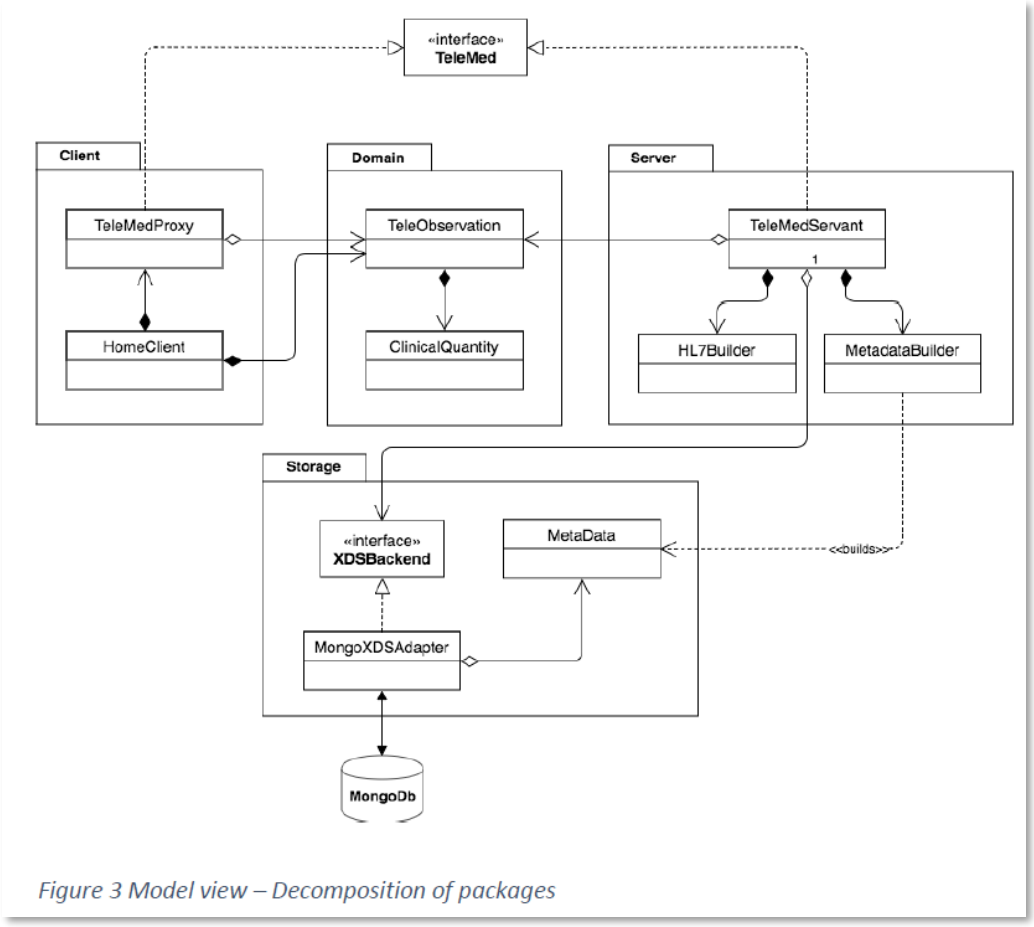



*Figure 3 Model view – Decomposition of packages*

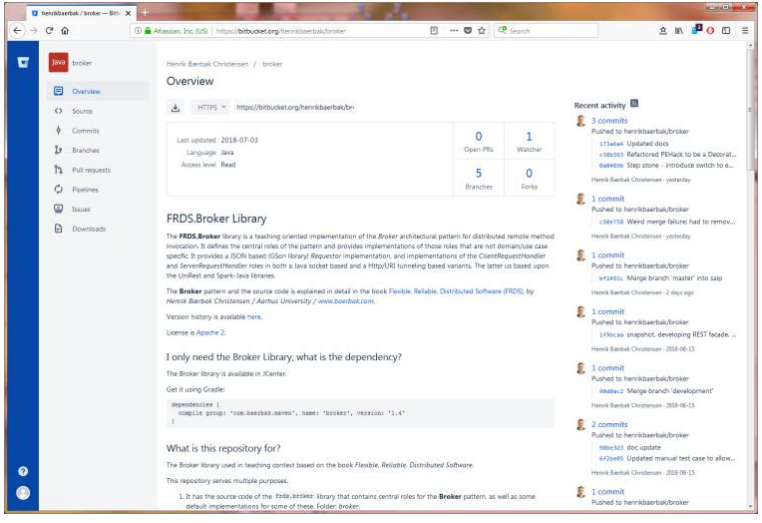
AARHUS UNIVERSITET

# Getting TeleMed

AARHUS UNIVERSITET

# TM Skeletal system

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




- … in
  - **folder ‘telemed’**
  - **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…

AARHUS UNIVERSITET

# 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…

AARHUS UNIVERSITET

# Tool Chain

- Academia generally sticks to the Java world
  - It is generally open source and free of charge!
- TM tool chain
  - Java 17+, Gradle 8+, Junit, IntelliJ, Git
    - And Docker, MongoDB, JMeter, …
- Installation options **– Follow the 'tools' web page**
  - A) Install it all on your machine OR
  - B) Get hold of VMWare Workstation and the 'csdev' machine
    - VMWare is now part of BroadCom. WorkStation is free for personal use.

AARHUS UNIVERSITET

# VM: csdev

- “Csdev-xxx.zip”
- Unzip, open in VMWare
- Hard work to make *minimal*
  - *Debian 12 – bookworm*
  - *LXDE window manager without applications*
  - *Waterfox as browser*
  - *Java, Gradle, Intellij*

AARHUS UNIVERSITET

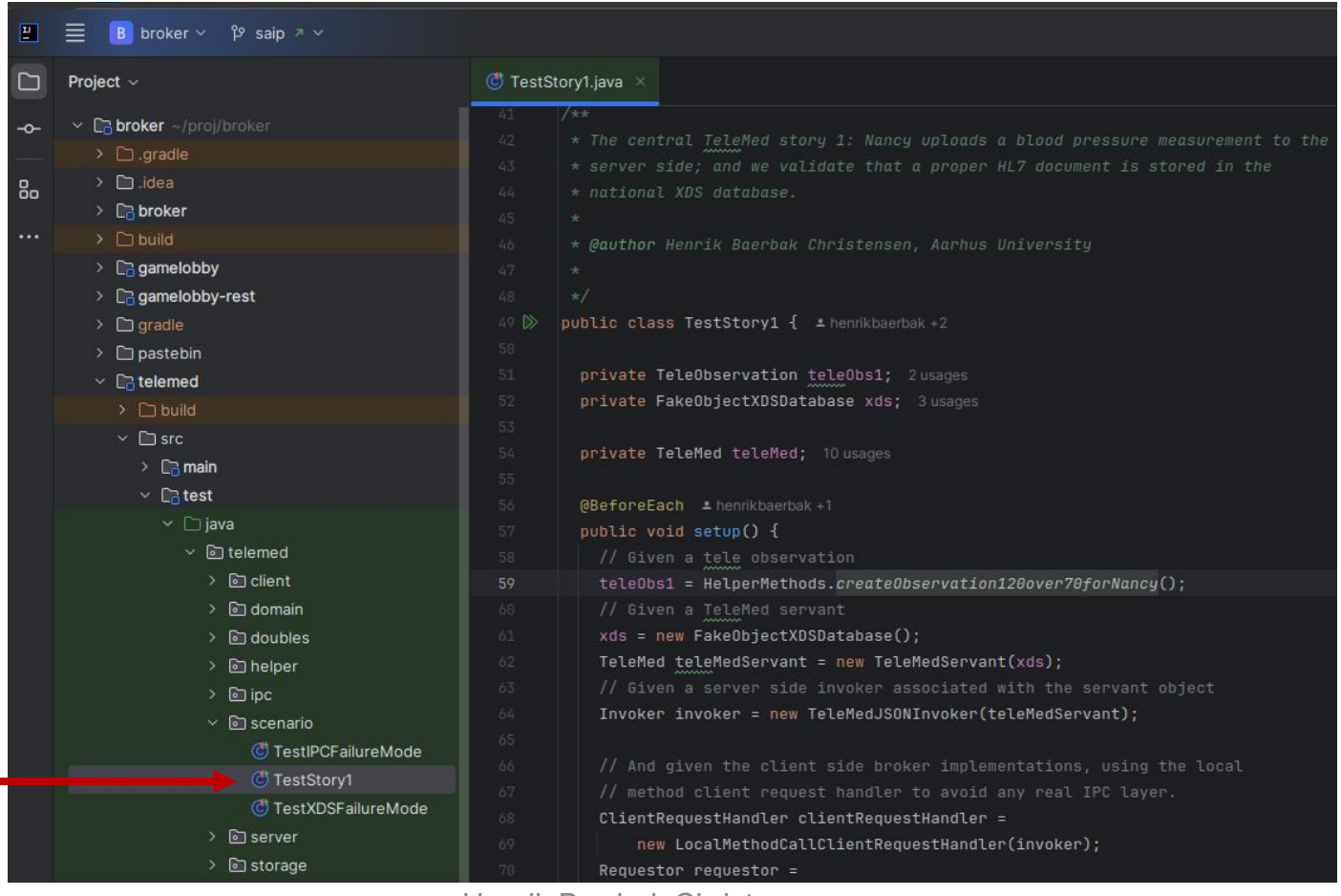
# Getting Started

- Getting TeleMed up and running
  - The TeleMed code base must be forked/cloned/unzipped from bitbucket.org
    - Standard maven/gradle folder structure for Java
      - Main/java = production code
      - Test/java = test code
  - Start 'IntelliJ' and choose 'Open', browse to the 'br and click on the gradle icon.

Details: See Tools web page

- Learning test
  - Review 'TestStory1' in telemed.scenario in test folder!

# TestStory1


broker
saip
Project
TestStory1.java
broker ~/proj/broker
.gradle
.idea
broker
build
gamelobby
gamelobby-rest
gradle
pastebin
telemed
build
src
main
test
java
telemed
client
domain
doubles
helper
ipc
scenario
TestIPCFailureMode
TestStory1
TestXDSFailureMode
server
storage
/**
* The central TeleMed story 1: Nancy uploads a blood pressure measurement to the
* server side; and we validate that a proper HL7 document is stored in the
* national XDS database.
*
* @author Henrik Baerbak Christensen, Aarhus University
*
*/
public class TestStory1 { henrikbaerbak +2
private TeleObservation teleObs1; 2 usages
private FakeObjectXDSDatabase xds; 3 usages
private TeleMed teleMed; 10 usages
@BeforeEach henrikbaerbak +1
public void setup() {
// Given a tele observation
teleObs1 = HelperMethods.createObservation120over70forNancy();
// Given a TeleMed servant
xds = new FakeObjectXDSDatabase();
TeleMed teleMedServant = new TeleMedServant(xds);
// Given a server side invoker associated with the servant object
Invoker invoker = new TeleMedJSONInvoker(teleMedServant);
// And given the client side broker implementations, using the local
// method client request handler to avoid any real IPC layer.
ClientRequestHandler clientRequestHandler =
new LocalMethodCallClientRequestHandler(invoker);
Requestor requestor =

# IDE

- 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

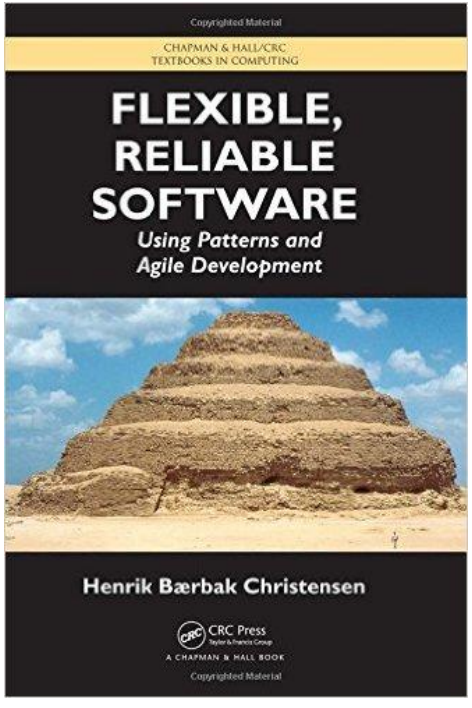
# Architecture Prerequisites

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

AARHUS UNIVERSITET

# 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)*
    - *You never define them yourself…*

# Example

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

```
@BeforeEach  henrikbaerbak +1
public void setup() {
  // Given a tele observation
  teleObs1 = HelperMethods.createObservation120over70forNancy();
  // Given a TeleMed servant
  xds = new FakeObjectXDSDatabase();
  TeleMed teleMedServant = new TeleMedServant(xds);
  // Given a server side invoker associated with the servant object
  Invoker invoker = new TeleMedJSONInvoker(teleMedServant);

  // And given the client side broker implementations, using the local
  // method client request handler to avoid any real IPC layer.
  ClientRequestHandler clientRequestHandler =
      new LocalMethodCallClientRequestHandler(invoker);
  Requestor requestor =
      new StandardJSONRequestor(clientRequestHandler);

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

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

AARHUS UNIVERSITET

# 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

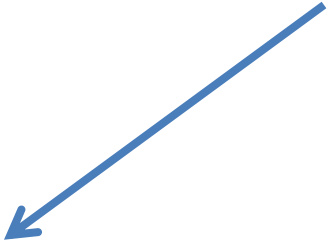
- *Basically, the SOLID principles in operational format…*

# Example

- 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*

# Example

```
@Override  baerbak
public String processAndStore(TeleObservation teleObs) {
  // 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;
}
```



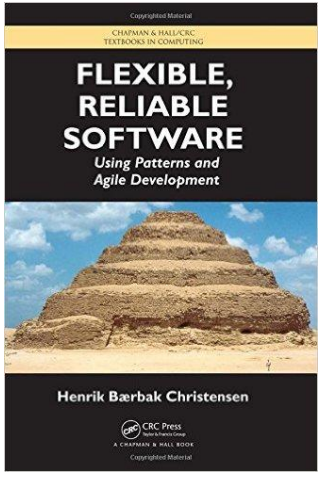
# And…

- 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*

# Example

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

```
@BeforeEach  henrikbaerbak +1
public void setup() {
  // Given a tele observation
  teleObs1 = HelperMethods.createObservation120overBeforNancy();
  // Given a TeleMed servant
  xds = new FakeObjectXDSDatabase();
  TeleMed teleMedServant = new TeleMedServant(xds);
  // Given a server side invoker associated with the servant object
  Invoker invoker = new TeleMedJSONInvoker(teleMedServant);

  // And given the client side broker implementations, using the local
  // method client request handler to avoid any real IPC layer.
  ClientRequestHandler clientRequestHandler =
      new LocalMethodCallClientRequestHandler(invoker);
  Requestor requestor =
      new StandardJSONRequestor(clientRequestHandler);

  // Then it is Given that we can create a client proxy
  // that voids any real IPC communication
  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

# Test Doubles

- 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

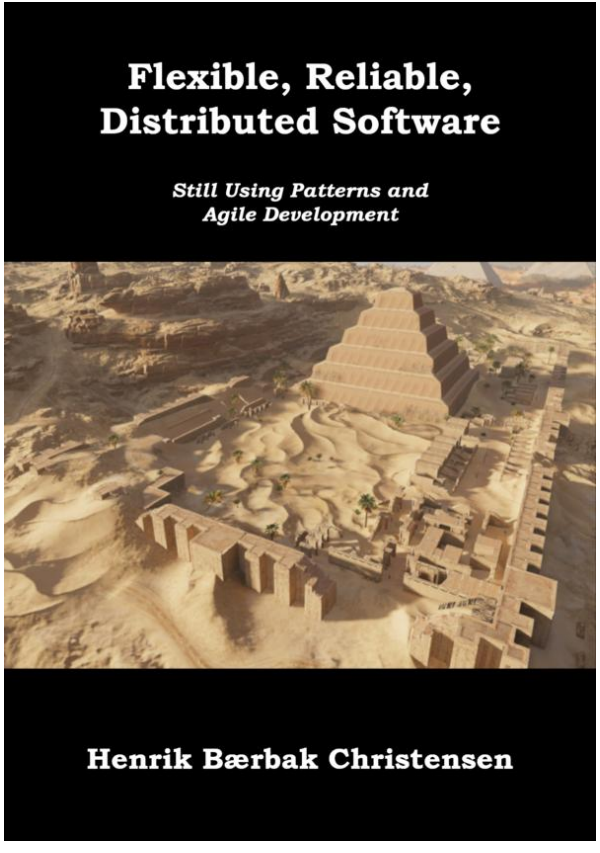
AARHUS UNIVERSITET

# Broker

The Central Distribution Pattern

# Distributed Computing

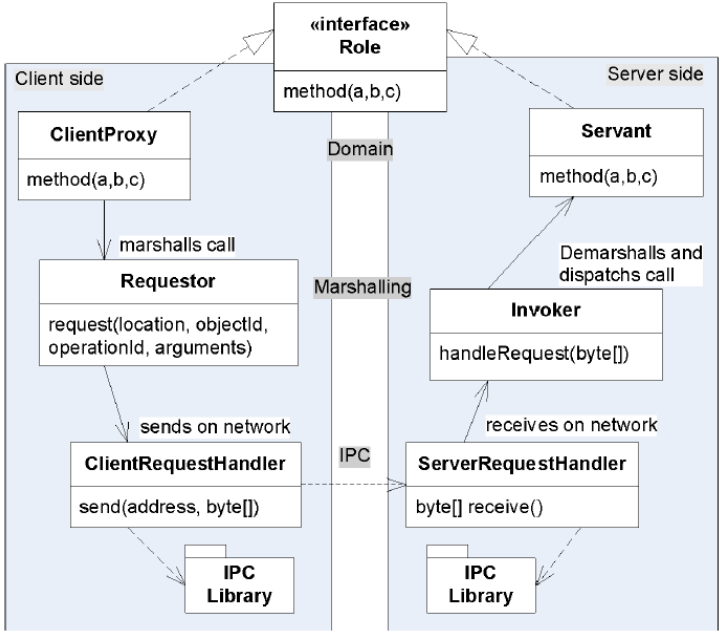
- I have written *Flexible, Reliable, **Distributed** Software*…
  - *leanpub.com/frds*
  - *Costs ~12$*




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

- **Broker**

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

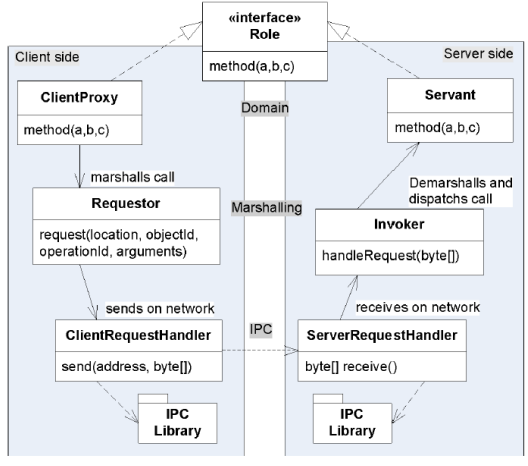


# Broker

- 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

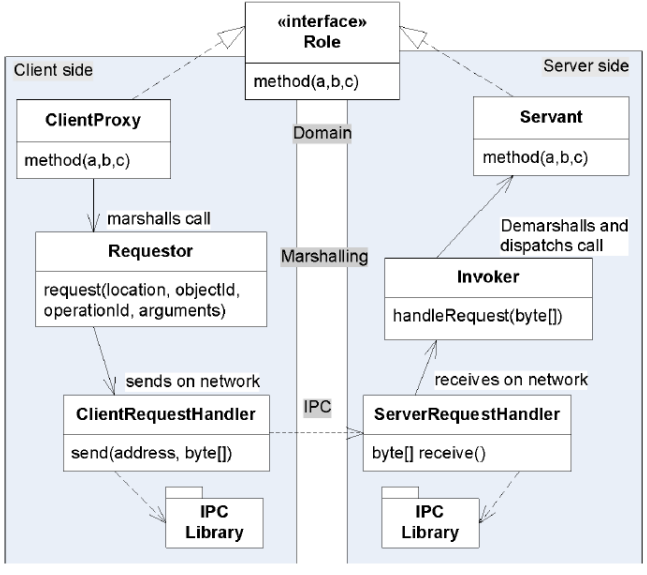
- Broker dynamics on Client side
  - ClientProxy:
    - Implements TeleMed interface
    - Convert every call into *requests* to the Requestor
  - Requestor:
    - Does *marshalling* = convert data to byte array format
      - Typically JSON or XML
  - ClientRequestHandler
    - Binds to the OS and particular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



AARHUS UNIVERSITET

# Server Side Broker

- 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!



AARHUS UNIVERSITET

# 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 …

AARHUS UNIVERSITET

# Unit Testing Distribution!

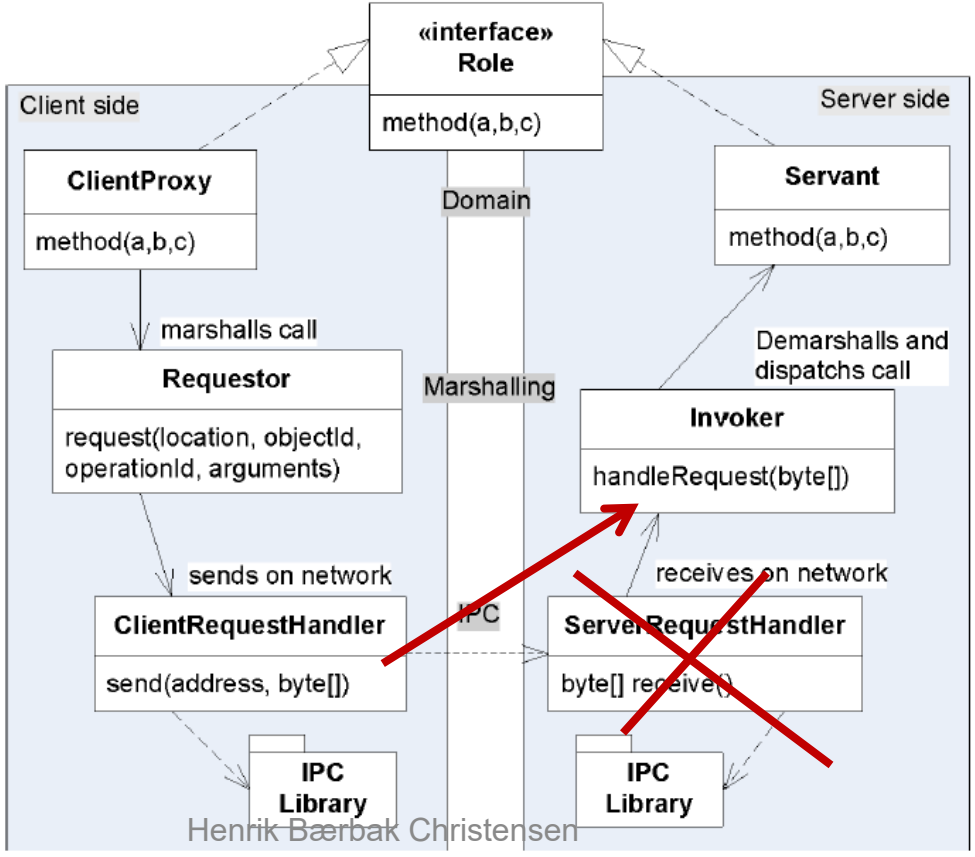
- Why code the broker myself?
- One big advantage:
  - *Test doubles*
  - *Now single VM system!*

```
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);
}
```

# Unit Testing

- Or – using UML architecture:



AARHUS UNIVERSITET

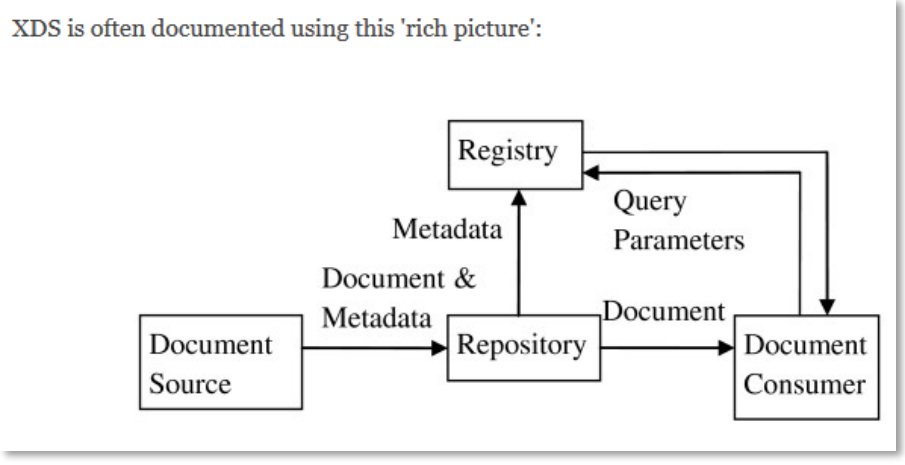
# Mandatory 1

Intro to our Case Study system

AARHUS UNIVERSITET

# 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 ☺



AARHUS UNIVERSITET

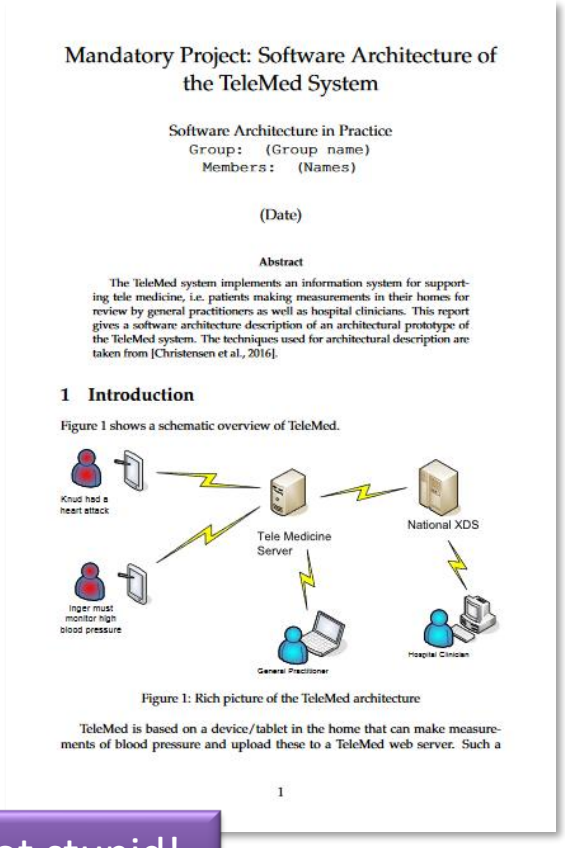
# Digging in...

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

AARHUS UNIVERSITET

# Provided:

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



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

1

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

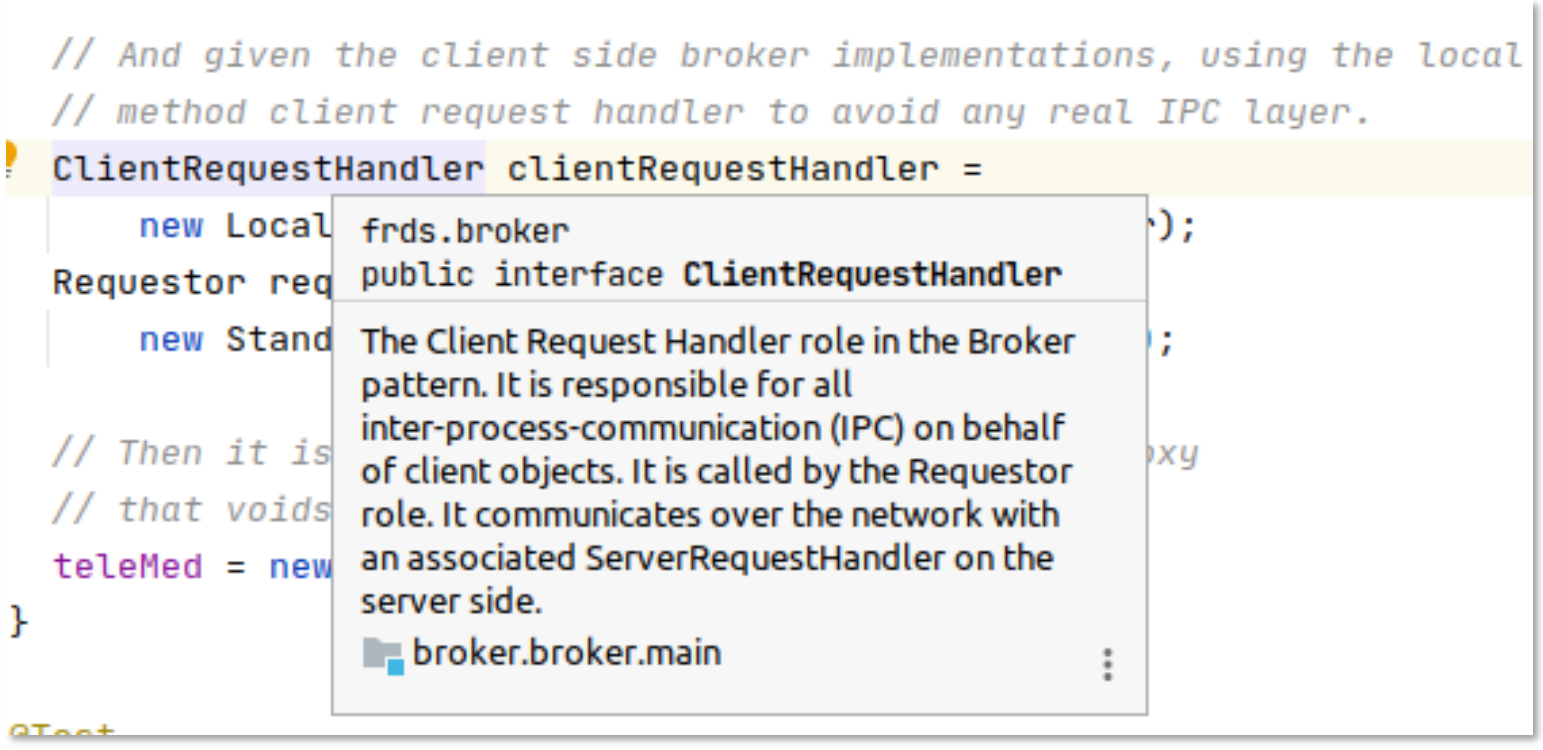
No, you are not stupid!

AARHUS UNIVERSITET

# Use IntelliJ for Digging Code

# Review the Docs

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

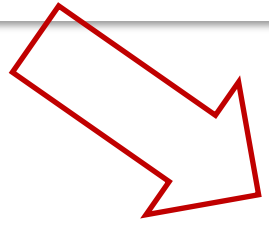


AARHUS UNIVERSITET

# 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 | TeleMedProxy.java

```
/* 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;
}
```

AARHUS UNIVERSITET

# Find Anything

- Hit ‘ctrl-n’ and begin typing to find stuff quickly