# Technical Presentation

The technical presentation is structured as followed:

* At 1st I´m going to show you an **overview** about the technologies we used
* Then we will look at the development process model **Unified process**
* After the introduction you will see our architecture on the basis of the **layer model**
* The next part is the **Hibernate** framework, which we used for the relational database – object – mapping
* Our **Dynamic Mapper** – which has the task to do a dynamic mapping between the persistence layer and the domain layer is show next
* Then we go on with the **State Pattern** and the User Interface technology **Swing**
* Last but not least you can see how we **integrated** the software modules of Team F
* And at the end of the technical presentation you get an overview about the technology we used for the web-reservation – **Java Server Faces**

# Overview (Technologies)

**Git** is a fast Revision (*version*) control and Source code management system

* We stored and distributed our data comfortably with github from where the whole team can access the code
* We created a different brunches – one for the Development and stable version after each timebox

**Java**

* “is an island of Indonesia. It is the most populous island of the world”
  + Indonesian capital city, Jakarta, is located on western Java
* Are you ready for the technical part?
* Is an Object orientated program language and Cross-platform computer environment from sun microsystems
* Java is for Free, very powerful and has a proper documentation with Java Doc

The development environment our team used is **Netbeans**

* This minimises problems around project consistence
  + And Netbeans has a good support for hibernate and Java Server Faces

We used the database **MySQL** – more about that later!

# Unified Process

The **Unified process** is a popular, iterative and incremental software development process. We used in this project and our team is really happy with this variant.

The process supports two dimensions – one for the management, the other for the programmer view.

The Management view includes the different phases Inception *(Kick off meeting, Definition of Use cases),* Elaboration *(Requirements engineering)*, Construction *(implementation of Use cases)* and Transition (*release*) phases *(divided into a series of timeboxed iterations).*

The programmer view includes Business Modeling, Requirements, Analysis and design, Implementation, Test, deployment.  
So the programmer can concentrate on his key sector!

The output after each iteration results in an increment, which is a release of the system that contains added or improved functionality compared with the previous release.

*The best-known and extensively documented refinement of the Unified Process is the Rational Unified Process (RUP)*

* *Markus shows you now the architecture of the application*

# State Pattern

The **State Pattern** is used in Use case controllers. To show you the benefit of this pattern I chose the “Create Invoice Use case”.

The User Interface has different screens. Each screen representing a state in the Use case controller. It is important that we are only able to pay an invoice, if we do the steps before – for example select a customer.

To do this in a clear way, the controller needs different states!

**Flip Chart**

Here you see a nice structure: The Invoice Controller has different states. A polymorphic problem-solution???

No, the controller must be always the same from outside (the GUI) to access it.  
Java does not support class change of an object during runtime (*alternative 🡪 Self*)!

So we have to extract the state from the controller…

Streiche Controller durch! Controller neu aggregiert State.

So the GUI can access always the same Controller, but with different states!

* + Solution: State Pattern (next Slide)

# Class Diagramm

Here we can see the Class-diagram of the Create Invoice Controller.

The controller looks now like an object which changed his class (from the outside – the GUI or something else)!

The different roles are:

* The context is our Use case controller. It is an interface for GUI
* The abstract state defines the functionality – depending on the state – of the controller. All methods throw an “IllegalStateException”.
* The concrete state implements the behaviour, which is connected to state of the context object. In example the payment only works in the Payment State. In all other states the Exception would be thrown!

To sum up, the State Pattern helps us to

* Structure the controllers. That prevents long switch-case statements in the operations and we get easy, readable and maintainable code!
* Also classification is possible, because super and sub classes in the states are supported

*Flip Chart*

* Another advantage of the state pattern is the easy way to create new states. You can just insert a new state-class and link it depending on the called functions together with the either states.
* Also the transitions become explicit, because the state is an external class; so – from the perspective of the context (Controller) – the state-transition is atomic, because there is only one specific state
* At least I can tell you the state pattern makes development more comfortably. During the testing phase (if you debug or test the program) you can see clearly, if you be in a false state. It is easier to locate faults, because…
  + A exception is thrown, if you are in an illegal state and
  + This structure reduces the fault scope

# Swing GUI

**Swing** is between SWT *THE Java GUI Framework!*

We used it in terms of the project, because it was a requirement, which makes integration in the group easier, if everybody has the same toolkit.

But there are *more reasons for Swing*:  
Swing is platform independent

* For example – AWT uses the Operating System native implementation for rendering the graphics. Swing does it independent and so it is possible to have the same design on different system (for instance MAC OSX, Windows or Linux)

Swing is extensible! We can create own components (for example the split and storno screen, which allows us to reuse it in a fine manner). Therefore we can use the whole power of JComponents (Swing) like fire events, bound properties or java beans binding. *this made problems in other teams (rotter)*

Swing is customizable. We get from the toolkit a standard set of elements, such as border, decorations and many other properties are available to customize tables, panels or buttons for a distinction to other products