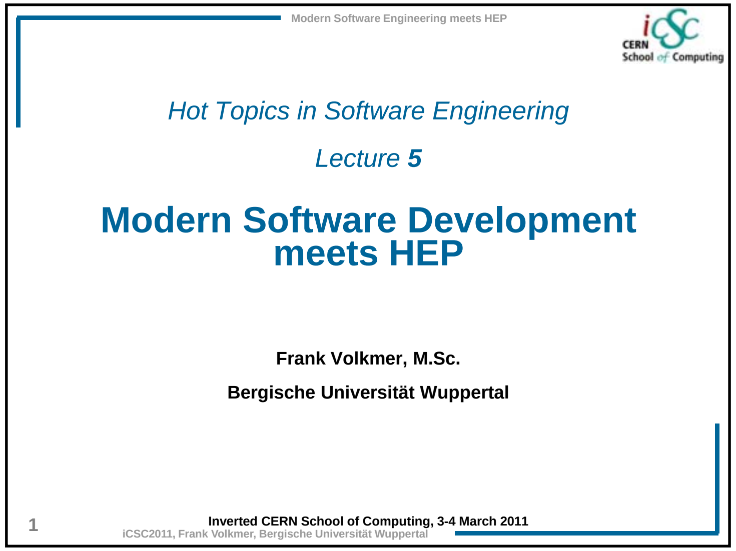


Hot Topics in Software Engineering

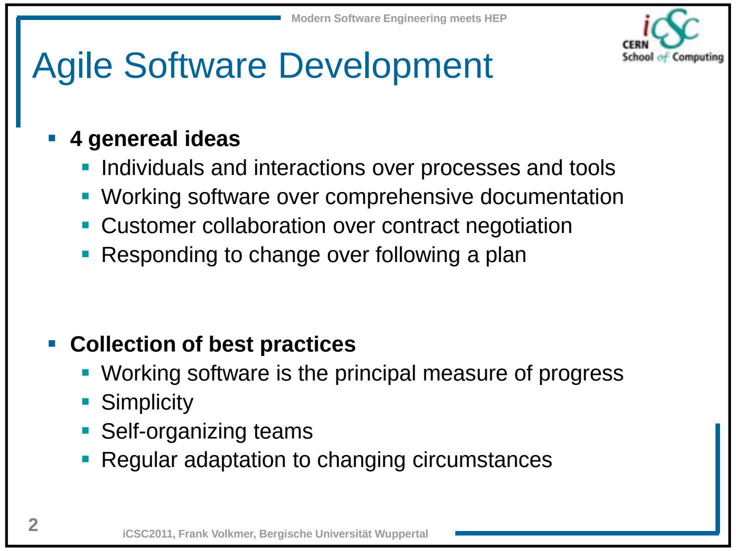
*Lecture **5***

Modern Software Development meets HEP

Frank Volkmer, M.Sc.

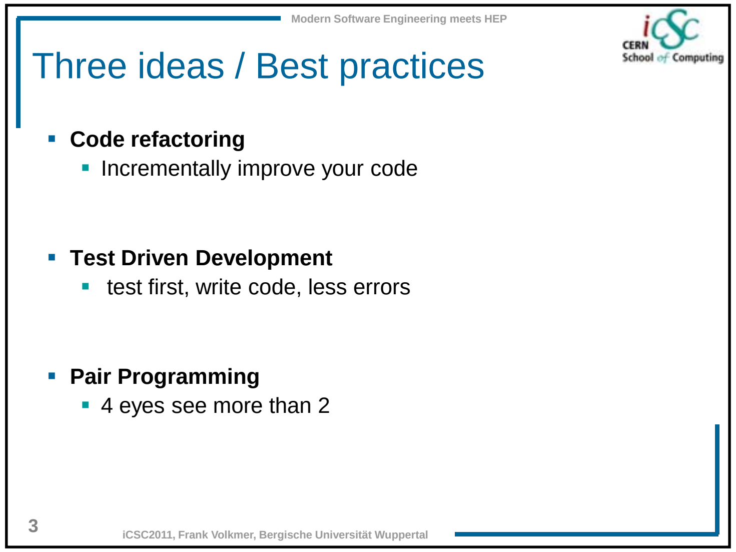
Bergische Universität Wuppertal

Inverted CERN School of Computing, 3-4 March 2011



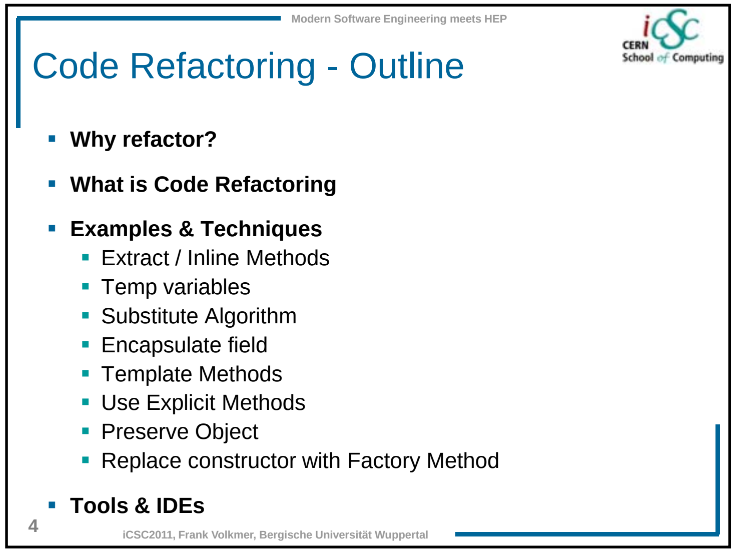
Agile Software Development

- **4 genereal ideas**
  - Individuals and interactions over processes and tools
  - Working software over comprehensive documentation
  - Customer collaboration over contract negotiation
  - Responding to change over following a plan
- **Collection of best practices**
  - Working software is the principal measure of progress
  - Simplicity
  - Self-organizing teams
  - Regular adaptation to changing circumstances



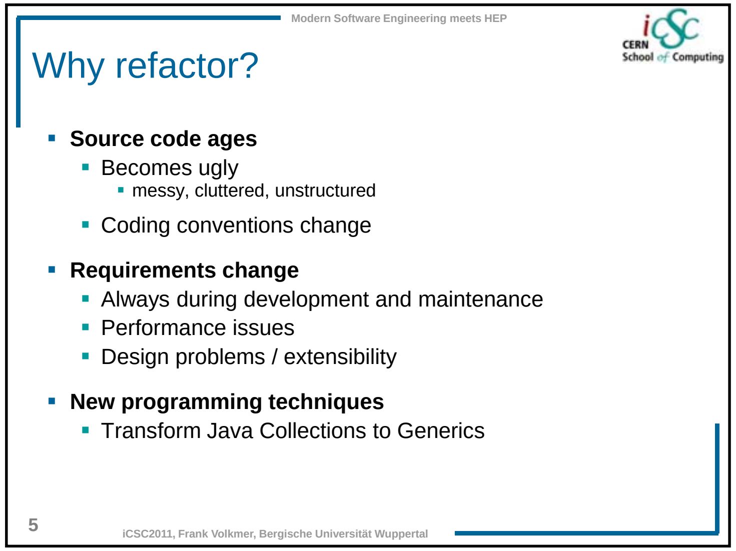
Three ideas / Best practices

- **Code refactoring**
  - Incrementally improve your code
- **Test Driven Development**
  - test first, write code, less errors
- **Pair Programming**
  - 4 eyes see more than 2



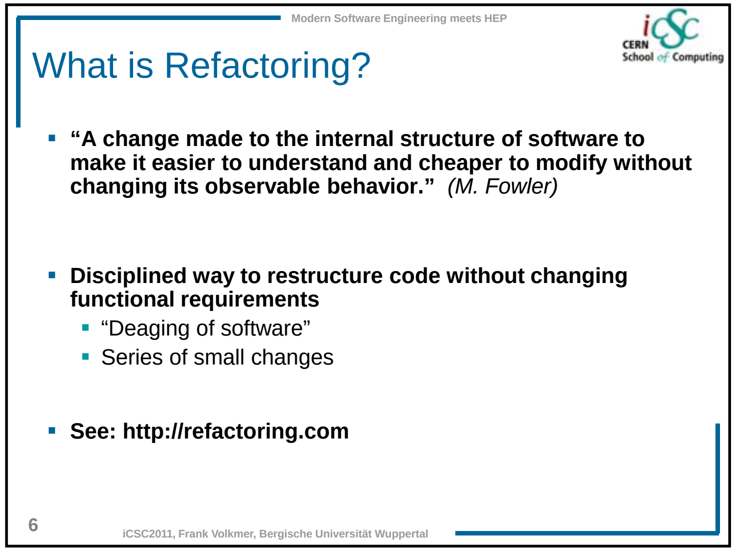
Code Refactoring - Outline

- **Why refactor?**
- **What is Code Refactoring**
- **Examples & Techniques**
  - Extract / Inline Methods
  - Temp variables
  - Substitute Algorithm
  - Encapsulate field
  - Template Methods
  - Use Explicit Methods
  - Preserve Object
  - Replace constructor with Factory Method
- **Tools & IDEs**



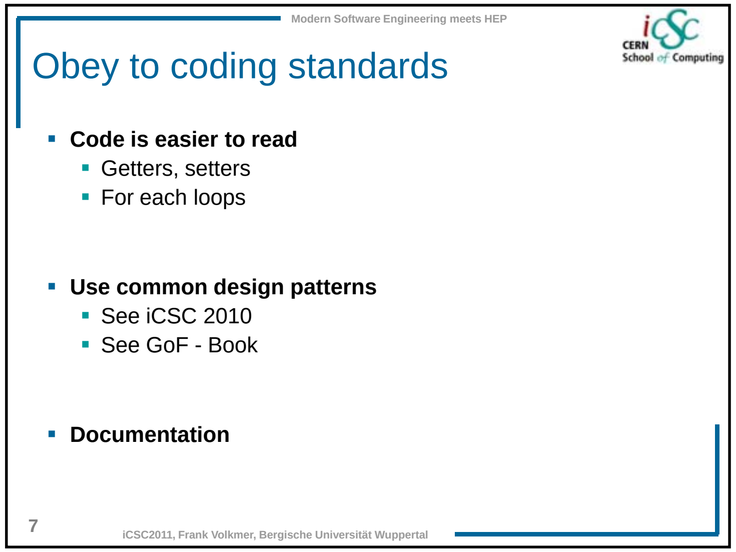
Why refactor?

- **Source code ages**
  - Becomes ugly
    - messy, cluttered, unstructured
  - Coding conventions change
- **Requirements change**
  - Always during development and maintenance
  - Performance issues
  - Design problems / extensibility
- **New programming techniques**
  - Transform Java Collections to Generics



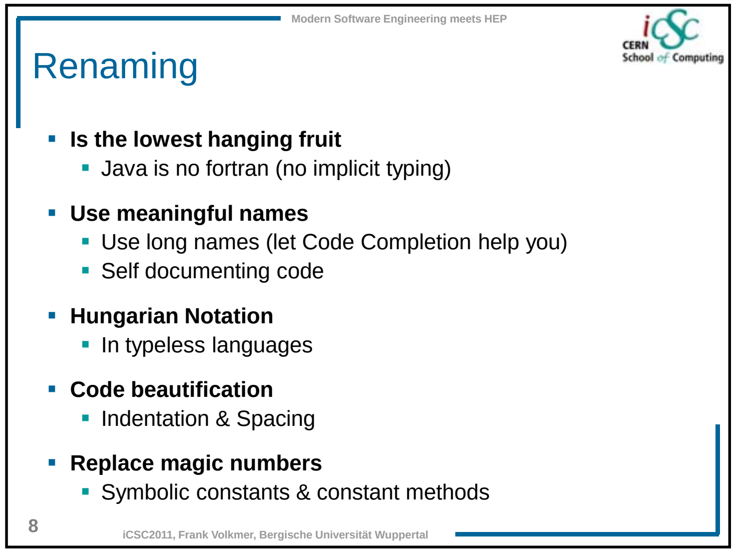
What is Refactoring?

- **"A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior."** *(M. Fowler)*
- **Disciplined way to restructure code without changing functional requirements**
  - "Deaging of software"
  - Series of small changes
- **See: http://refactoring.com**



Obey to coding standards

- **Code is easier to read**
  - Getters, setters
  - For each loops
- **Use common design patterns**
  - See iCSC 2010
  - See GoF - Book
- **Documentation**



Renaming

- **Is the lowest hanging fruit**
  - Java is no fortran (no implicit typing)
- **Use meaningful names**
  - Use long names (let Code Completion help you)
  - Self documenting code
- **Hungarian Notation**
  - In typeless languages
- **Code beautification**
  - Indentation & Spacing
- **Replace magic numbers**
  - Symbolic constants & constant methods

Technique: Extract / Inline method

```
void printOwing() {
   printBanner();
   printf("name: %s" + name);
   printf("age: %d" + age);
}

//becomes:

void printOwing() {
   printBanner();
   printDetails(getAge());
}
void printDetails (int age) {
   printf("name: " + name);
   printf("amount: " + age);
}
```

Technique: Temp variables

```
double basePrice = anOrder.basePrice();
return (basePrice > 1000);

//becomes:

return (anOrder.basePrice() > 1000);
```

Technique: Substitute Algorithm

```
String foundPerson(String[] people){
    for (int i = 0; i < people.length; i++) {
         if (people[i].equals ("Don")){
              return "Don";
         }
         if (people[i].equals ("John")){
              return "John";
         }
         if (people[i].equals ("Kent")){
              return "Kent";
         } }
    return "";
}
```

```
//becomes:

String foundPerson(String[] people){
    List candidates = Arrays.asList(new
         String[] {"Don", "John", "Kent"});
         for (int i=0; i<people.length; i++)
              if (candidates.contains(people[i]))
                   return people[i];
         return "";
    }
```

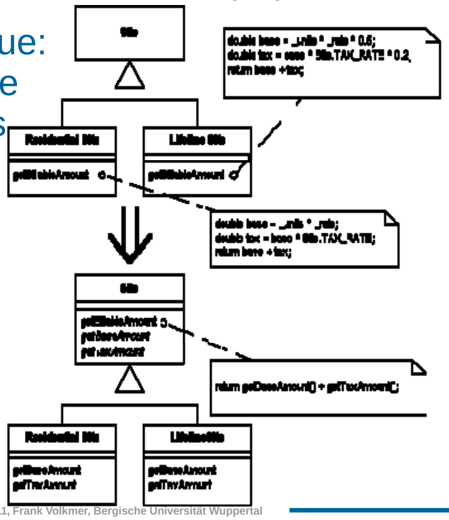
Data Encapsulation

- **Use encapsulation**
  - for member fields
- **Make own collections immutable**
- **Separation of concern**

Technique: Encapsulate field

```
public String _name

//becomes:

private String _name;

public String getName() {
    return _name;
}

public void setName(String name) {
    _name = name;
}
```

Technique: Template methods



Method calls

- **Change method signatures**
  - Parameters
  - Return values
  - Exceptions
- **Separate Query from Modifier**
  - getTotalAndSub5()
    - getTotal()
    - Sub5()
- **Use explicit methods**
- **Preserve object**

Technique: Use Explicit Methods

```
void setValue(String name, int value)
{
    if (name.equals("height")) {
        _height = value;
        return;
    }
    if (name.equals("width")) {
        _width = value;
        return;
    }
    Assert.shouldNeverReachHere();
}
```

```
//becomes :

void setHeight(int arg)
{
    _height = arg;
}

void setWidth (int arg)
{
    _width = arg;
}
```

Technique: Preserve Object

```
int low = daysTempRange().getLow();
int high = daysTempRange().getHigh();
withinPlan = plan.withinRange(low, high);

// becomes:

withinPlan = plan.withinRange(daysTempRange());
```

Technique: Replace constructor with Factory Method

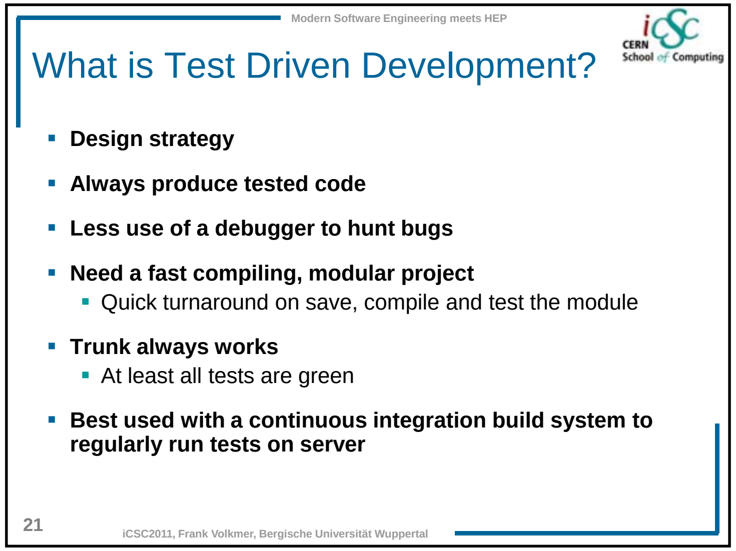
```
Employee (int type) {
    _type = type;
}
//becomes:
static Employee create(int type) {
    return new Employee(type);
}
```

Tooling

- **Automated refactoring**
  - Context aware
  - Parameterizing option
  - Menu driven assistents & wizards
- **IDE support**
  - Eclipse
    - Photran
  - Xcode
  - NETBEANS
  - IntelliJ IDEA
  - Visual Studio .net
  - …

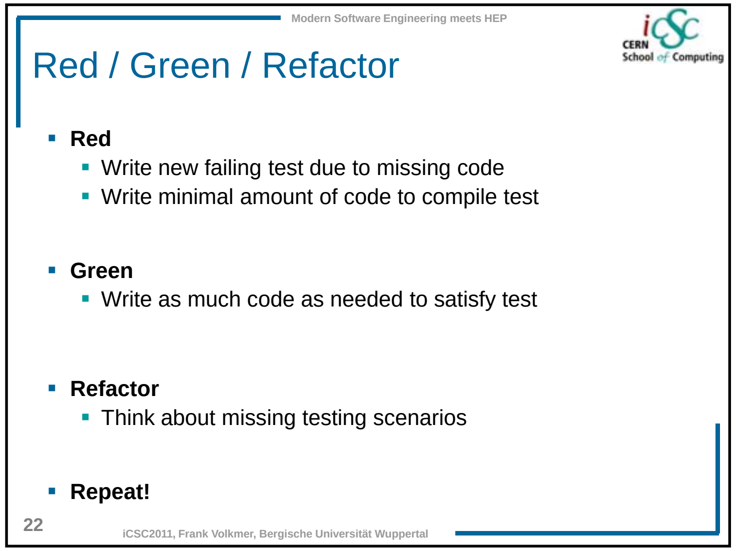
Test driven development - Outline

- **What is Test Driven Development?**
- **Red / Green / Refactor**
- **Unit Tests**
- **Tools**
- **Possible Problems**



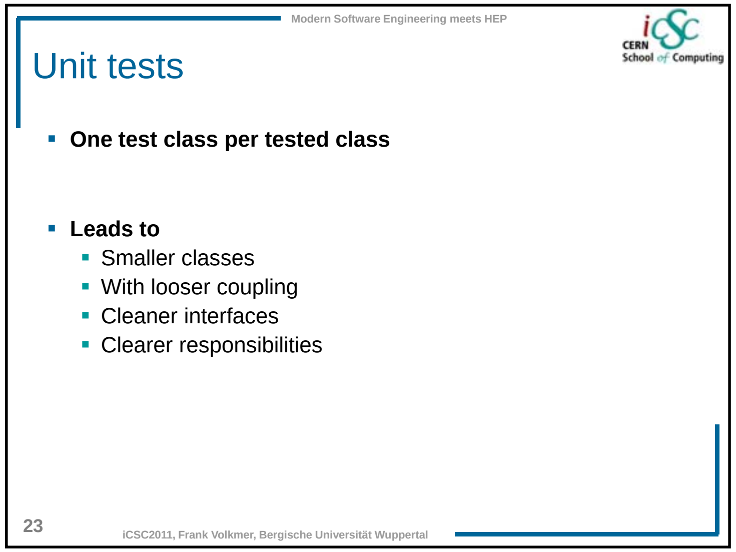
What is Test Driven Development?

- **Design strategy**
- **Always produce tested code**
- **Less use of a debugger to hunt bugs**
- **Need a fast compiling, modular project**
  - Quick turnaround on save, compile and test the module
- **Trunk always works**
  - At least all tests are green
- **Best used with a continuous integration build system to regularly run tests on server**



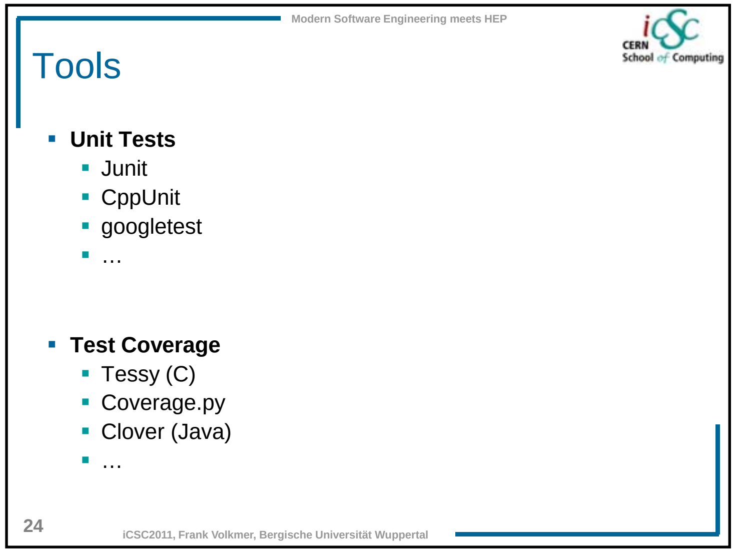
Red / Green / Refactor

- **Red**
  - Write new failing test due to missing code
  - Write minimal amount of code to compile test
- **Green**
  - Write as much code as needed to satisfy test
- **Refactor**
  - Think about missing testing scenarios
- **Repeat!**



Unit tests

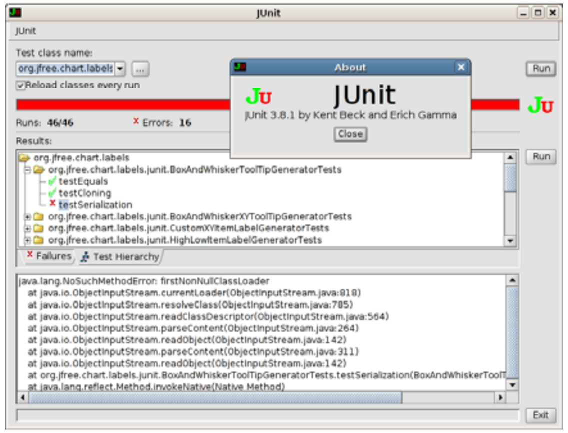
- **One test class per tested class**
- **Leads to**
  - Smaller classes
  - With looser coupling
  - Cleaner interfaces
  - Clearer responsibilities



Tools

- **Unit Tests**
  - Junit
  - CppUnit
  - googletest
  - …
- **Test Coverage**
  - Tessy (C)
  - Coverage.py
  - Clover (Java)
  - …

JUnit



JUnit 3.8.1 by Kent Beck and Erich Gamma

Runs: 46/46 Errors: 16

```
org.jfree.chart.labels
  org.jfree.chart.labels.junit.BoxAndWhiskerToolTipGeneratorTests
    testEquals
    testCloning
    testSerialization
  org.jfree.chart.labels.junit.BoxAndWhiskerXYToolTipGeneratorTests
  org.jfree.chart.labels.junit.CustomXYItemLabelGeneratorTests
  org.jfree.chart.labels.junit.HighLowItemLabelGeneratorTests
```

```
java.lang.NoSuchMethodError: firstNonNullClassLoader
  at java.io.ObjectInputStream.currentLoader(ObjectInputStream.java:818)
  at java.io.ObjectInputStream.resolveClass(ObjectInputStream.java:785)
  at java.io.ObjectInputStream.readClassDescriptor(ObjectInputStream.java:564)
  at java.io.ObjectInputStream.parseContent(ObjectInputStream.java:264)
  at java.io.ObjectInputStream.readObject(ObjectInputStream.java:142)
  at java.io.ObjectInputStream.parseContent(ObjectInputStream.java:311)
  at java.io.ObjectInputStream.readObject(ObjectInputStream.java:142)
  at org.jfree.chart.labels.junit.BoxAndWhiskerToolTipGeneratorTests.testSerialization(BoxAndWhiskerToolT
  at java.lang.reflect.Method.invokeNative(Native Method)
```

Integration Tests

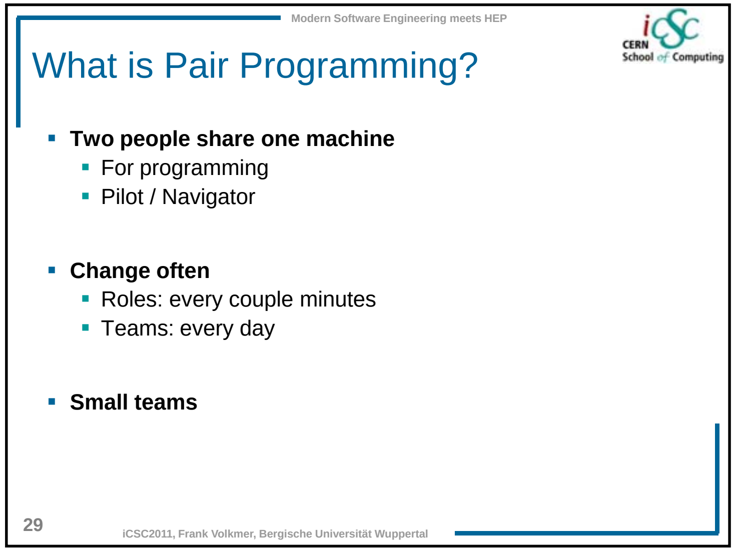
- **Happen after unit testing but before system tests**
  - Unit tests cover single modules, without interaction
  - Unit tests are often run against mock objects
- **Use interfaces of modules, use them as black boxes**
- **Group several tested modules and test them in integrated concert**
- **Test**
  - Proper integration of module associations
  - Layers of modules
  - Inter process communication

Problems with TDD

- **Can lock your API quite early**
- **Developers do have blind spots**
- **Psychological mindset: plan to fail**
- **If you prototype and experiment, TDD can be a lot of extra effort**

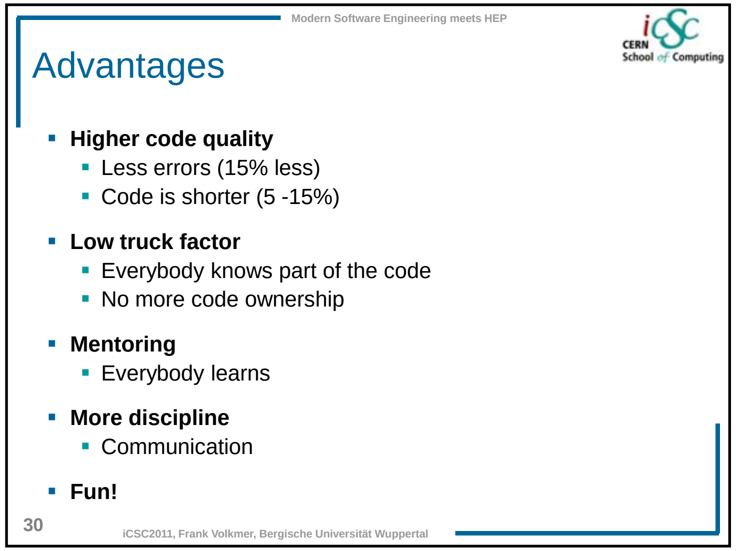
Pair Programming - Outline

- **What is Pair Programming?**
- **Advantages**
- **Possible Problems**



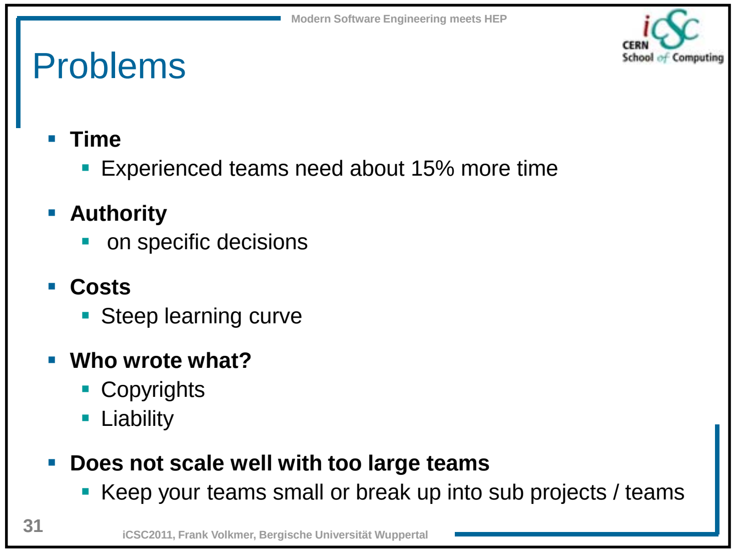
What is Pair Programming?

- **Two people share one machine**
  - For programming
  - Pilot / Navigator
- **Change often**
  - Roles: every couple minutes
  - Teams: every day
- **Small teams**



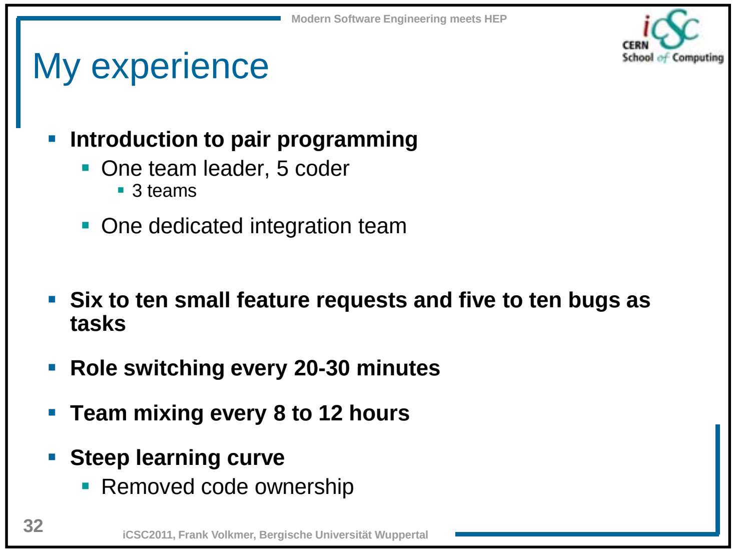
Advantages

- **Higher code quality**
  - Less errors (15% less)
  - Code is shorter (5 -15%)
- **Low truck factor**
  - Everybody knows part of the code
  - No more code ownership
- **Mentoring**
  - Everybody learns
- **More discipline**
  - Communication
- **Fun!**



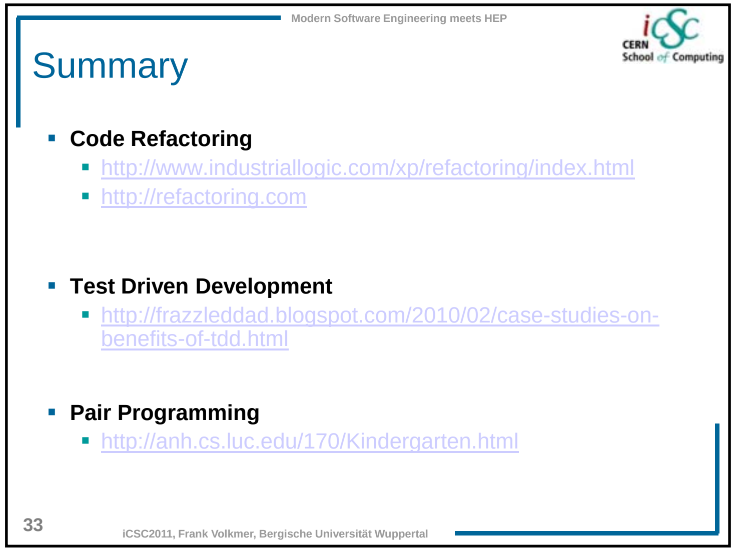
Problems

- **Time**
  - Experienced teams need about 15% more time
- **Authority**
  - on specific decisions
- **Costs**
  - Steep learning curve
- **Who wrote what?**
  - Copyrights
  - Liability
- **Does not scale well with too large teams**
  - Keep your teams small or break up into sub projects / teams



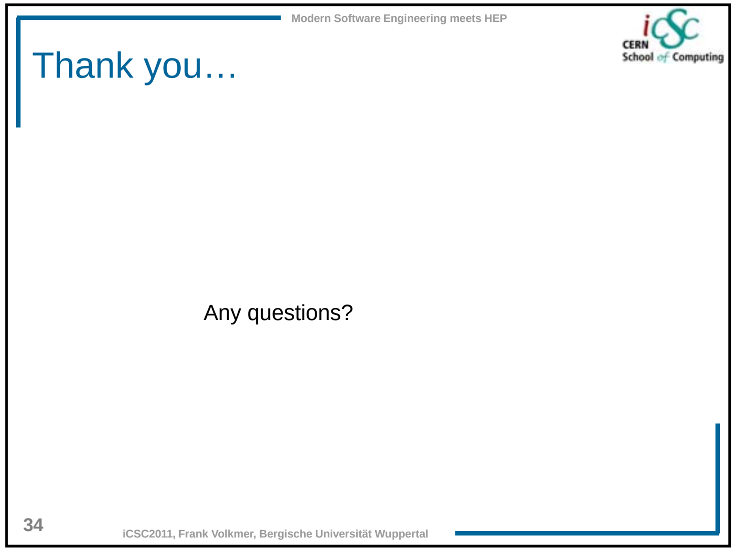
My experience

- **Introduction to pair programming**
  - One team leader, 5 coder
    - 3 teams
  - One dedicated integration team
- **Six to ten small feature requests and five to ten bugs as tasks**
- **Role switching every 20-30 minutes**
- **Team mixing every 8 to 12 hours**
- **Steep learning curve**
  - Removed code ownership



Summary

- **Code Refactoring**
  - http://www.industriallogic.com/xp/refactoring/index.html
  - http://refactoring.com
- **Test Driven Development**
  - http://frazzleddad.blogspot.com/2010/02/case-studies-on-benefits-of-tdd.html
- **Pair Programming**
  - http://anh.cs.luc.edu/170/Kindergarten.html



Thank you…

Any questions?