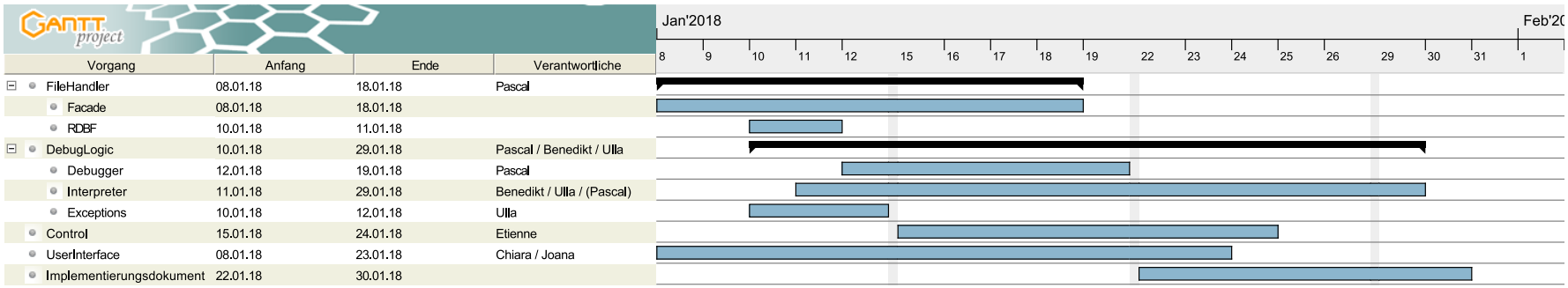
Praxis der Softwareentwicklung
Implementierungsphase

Phasenverantwortlicher: Pascal Zwick

05. Februar 2018

Gantt-Diagramm



Vorgang	Anfang	Ende	Verantwortliche
FileHandler	08.01.18	18.01.18	Pascal
Facade	08.01.18	18.01.18	
RDBF	10.01.18	11.01.18	
DebugLogic	10.01.18	29.01.18	Pascal / Benedikt / Ulla
Debugger	12.01.18	19.01.18	Pascal
Interpreter	11.01.18	29.01.18	Benedikt / Ulla / (Pascal)
Exceptions	10.01.18	12.01.18	Ulla
Control	15.01.18	24.01.18	Etienne
UserInterface	08.01.18	23.01.18	Chiara / Joana
Implementierungsdokument	22.01.18	30.01.18	

Sicherstellung

- Erweiterbarkeit: abstrakte Klassen und Schnittstellen (Interface)
- Wartbarkeit: Javadoc, Kommentare im Quelltext und Nutzung von Checkstyle
- Urheberrecht: GNU GPL 3 (GNU General Public License)
- Benutzerfreundlichkeit: NA

Benutzerfreundlichkeit

- Verständlichkeit: Sprachauswahl (Deutsch, Englisch)
- Übersichtlichkeit: NA
- Erlernbarkeit: Hilfestellungen (Tooltips)
- Modifizierbarkeit: Veränderung der Spaltengröße, ...

Singleton (Einzelstück)

```
private void readBlock(BufferedReader reader, RDBFAdditions ad) throws FileHandlerException, IOException {
    String line;
    while ((line = reader.readLine()) != null) {
        line = line.replace("\t", "");
        int lineType = RDBFParser.getInstance().evaluateLineType(line);
        if (lineType == RDBFParser.LINE_ASSIGNMENT) {
            String name = RDBFParser.getInstance().getVariableName(line);
            String value = RDBFParser.getInstance().getValue(line);
            ad.addData(new RDBFData(name, value));
        } else if (lineType == RDBFParser.LINE_BLOCK) {
            String name = RDBFParser.getInstance().getBlockName(line);
            RDBFBlock block = new RDBFBlock(name);
            ad.addBlock(block);
            readBlock(reader, block);
        } else if (lineType == RDBFParser.LINE_BLOCK_TEXT_LENGTH) {
            int len = RDBFParser.getInstance().getIValue(RDBFParser.getInstance().getValue(line));
            String text = readTextBlock(reader, ad, len);
            ad.addData(new RDBFData("text", text, true));
        } else if (lineType == RDBFParser.LINE_BLOCK_END) {
            return;
        }
    }
}
```

Singleton (Einzelstück)

```
private void readBlock(BufferedReader reader, RDBFAdditions ad) throws FileHandlerException, IOException {
    String line;
    while ((line = reader.readLine()) != null) {
        line = line.replace("\t", "");
        int lineType = RDBFParser.getInstance().evaluateLineType(line);
        if (lineType == RDBFParser.LINE_ASSIGNMENT) {
            String name = RDBFParser.getInstance().getVariableName(line);
            String value = RDBFParser.getInstance().getValue(line);
            ad.addData(new RDBFData(name, value));
        } else if (lineType == RDBFParser.LINE_BLOCK) {
            String name = RDBFParser.getInstance().getBlockName(line);
            RDBFBlock block = new RDBFBlock(name);
            ad.addBlock(block);
            readBlock(reader, block);
        } else if (lineType == RDBFParser.LINE_BLOCK_TEXT_LENGTH) {
            int len = RDBFParser.getInstance().getIValue(RDBFParser.getInstance().getValue(line));
            String text = readTextBlock(reader, ad, len);
            ad.addData(new RDBFData("text", text, true));
        } else if (lineType == RDBFParser.LINE_BLOCK_END) {
            return;
        }
    }
}
```

Strategy (Strategie)

```java
public abstract class DBFileReader {

    public abstract ConfigurationFile loadConfigFile(File f) throws FileHandlerException;

    public abstract LanguageFile loadLanguageFile(File f) throws FileHandlerException;
}
```

```java
@Override
public LanguageFile loadLanguageFile(
    try {
        RDBFFile f0 = reader.loadRDBF

        LanguageFile f = new Language
                getInstance().getSVal

        for (RDBFBlock b : f0.getList
            f.putTranslation(b.getNam
        }

        return f;
    } catch (IOException e) {
        throw new LanguageNotFoundExc
    }
}
```

Visitor (Besucher)

- *WlangBaseVisitor<T>*
  - *CommandGenerationVisitor*
  - *TermGenerationVisitor*

```
public Command visitWhileWithBlock(WhileWithBlockContext ctx) {
    Term condition = this.termGenVisitor.visit(ctx.condition());
    WhileCommand whileCommand = new WhileCommand(this.controller,
    // add child commands
    List<Command> content = this.collectInBlock(ctx.content);
    for (Command c : content) {
        whileCommand.addChild(c);
    }
    return whileCommand;
}
```

- Kontext = Knoten (*While-Knoten*)
- Besuchen der Bedingung
- Sammeln und Hinzufügen der im *while-block* stehenden Anweisungen

```
public Term visitAddition(AdditionContext ctx) {
    Term leftTerm, rightTerm;
    leftTerm = this.visit(ctx.left);
    rightTerm = this.visit(ctx.right);
    return new AdditionTerm(leftTerm, rightTerm);
}
```

- Besuchen beider Terme
- Erstellen eines *Addition* Terms, welcher beide Subterme enthält
- Bsp: $a = (b * c) + d$
  - Linker Term: $b * c$
  - Rechter Term: d

Nutzung von Antlr

```
64  private void createTerm() {
65    CharStream input = CharStreams.fromString(this.specifier);
66    WlangLexer lexer = new WlangLexer(input);
67    CommonTokenStream tokens = new CommonTokenStream(lexer);
68    WlangParser parser = new WlangParser(tokens);
69    // Choose start rule
70    ParseTree tree = parser.webppterm();
71    TermGenerationVisitor visitor = new TermGenerationVisitor();
72    this.expression = visitor.visit(tree);
73  }
```

- Erzeugung eines Ableitungsbaums durch *WlangLexer* und *WlangParser*
- *Visitor* besuchen nun den generierten *ParseTree*

Commands (Befehle)

```
35  public List<TraceState> run() throws DIbuggerLogicException {
36    Scope scope = this.controller.getCurrentScope();
37    // check if condition is of type boolean
38    TermValue value = this.condition.evaluate(scope);
39    if (value.getType() != Type.BOOLEAN) {
40      throw new WrongTypeArgumentException(this.linenumber);
41    }
42    List<TraceState> traceStateList = new ArrayList<TraceState>();
43    traceStateList.add(new TraceState(TraceStatePosition.NOTSPECIAL, this.linenumber, scope));
44    // run the loop
45    while (((BooleanValue) this.condition.evaluate(scope)).getValue()) {
46      for (Command child : this.children) {
47        traceStateList.addAll(child.run());
48      }
49    }
50    return traceStateList;
51  }
```

- Beispiel *While-Command*
- Holen des derzeitigen *Scopes* (Stack-Frame)
- Auswerten der *Condition*
- Befhle innerhalb der while-Schleife ausführen

Exceptions Paket

- Schnittstellen (Interface) können in Java nicht geworfen werden
- *FileHandlerException* wurde als abstrakte Klasse implementiert

Debugger und Interpreter

- *Subject* als Kontrollpunkt des Beobachtermusters
- *Observable* aus Java bietet gleiche Funktionalität
- *TraceIterator* bot Funktionalität zum iterieren über den Trace an
- Trace wird in Form einer Liste gespeichert, also wurde *ListIterator<T>* von Java benutzt

DIbugger