Aspektorientierte Programmierung in C++

Sören Frey | Florian Walker

Vortrag im Rahmen des Master Seminars

hochschule mannheim

Übersicht

1. AOP Grundlagen

- **Einordnung:**
  - **Assembler**
  - **Prozedurale Sprache (PASCAL, FORTRAN)**

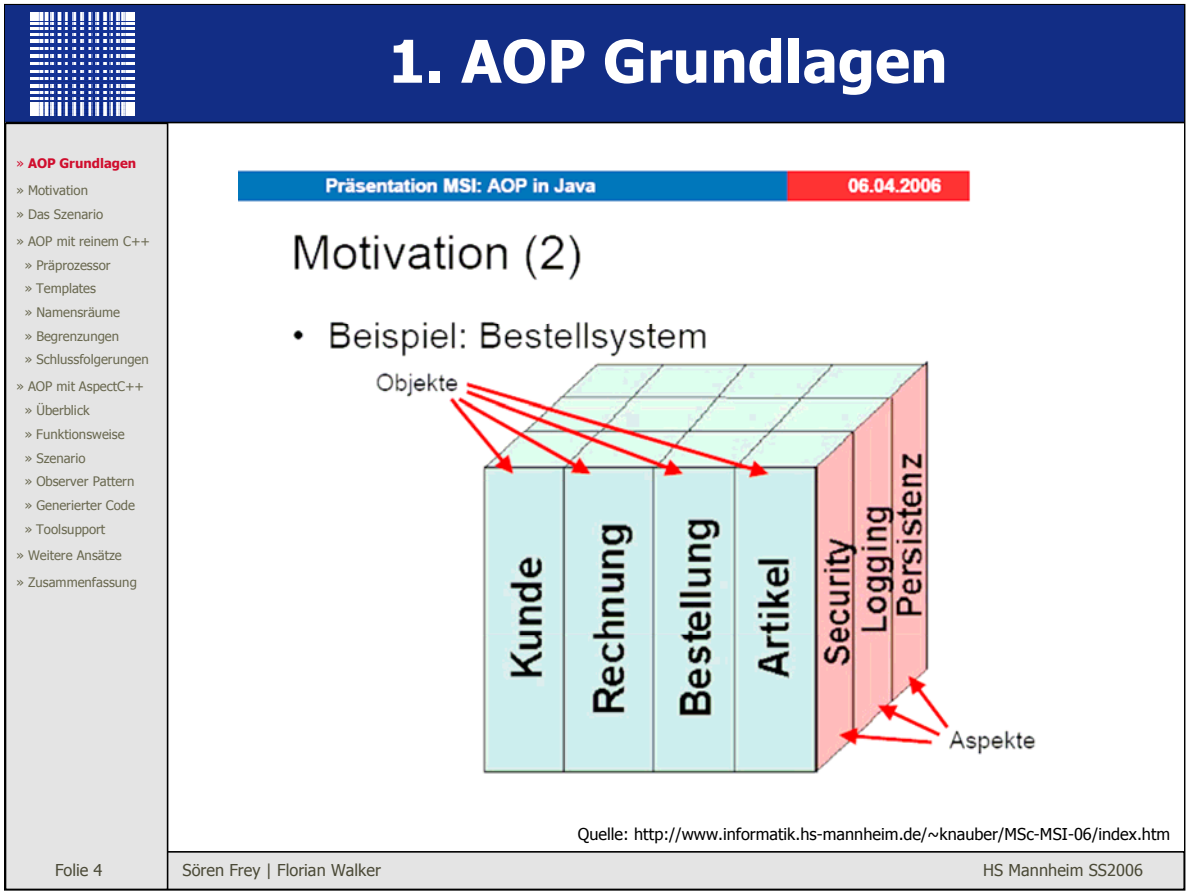
```
kuchen = backen(mehl,
  zucker, butter)
```

  - **OO-Sprachen (C++, JAVA)**

```
kuchen = new
Kuchen(mehl, eier,
  zucker);
Kuchen.backe();
```

  - **Aspektorientierte Sprachen (AspectJ, AspectC++)**

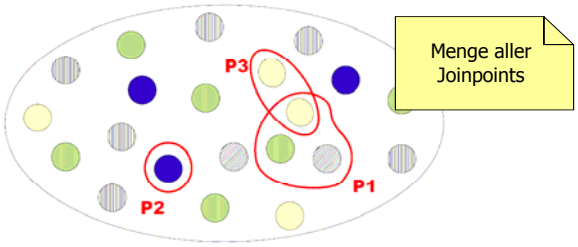
```
OOPS um Aspekte
  angereichert
```



- Kernfunktionalitäten (eng. *core concerns*) können durch Objektorientierung sauber in Module getrennt werden. Es gibt jedoch Belange wie Sicherheit, Datenerfassung und Fehlerbehandlung in jedem System die die Kernfunktionalitäten quer schneiden (eng. *crosscut*) und sich deshalb nicht eindeutig in Softwaremodule zuordnen lassen. Dies führt zu einer Verstreuung der *crosssutting concens* über den gesamten Code einer Anwendung und beeinflusst so die Wiederverwendbarkeit, Wartung und Verständlichkeit.

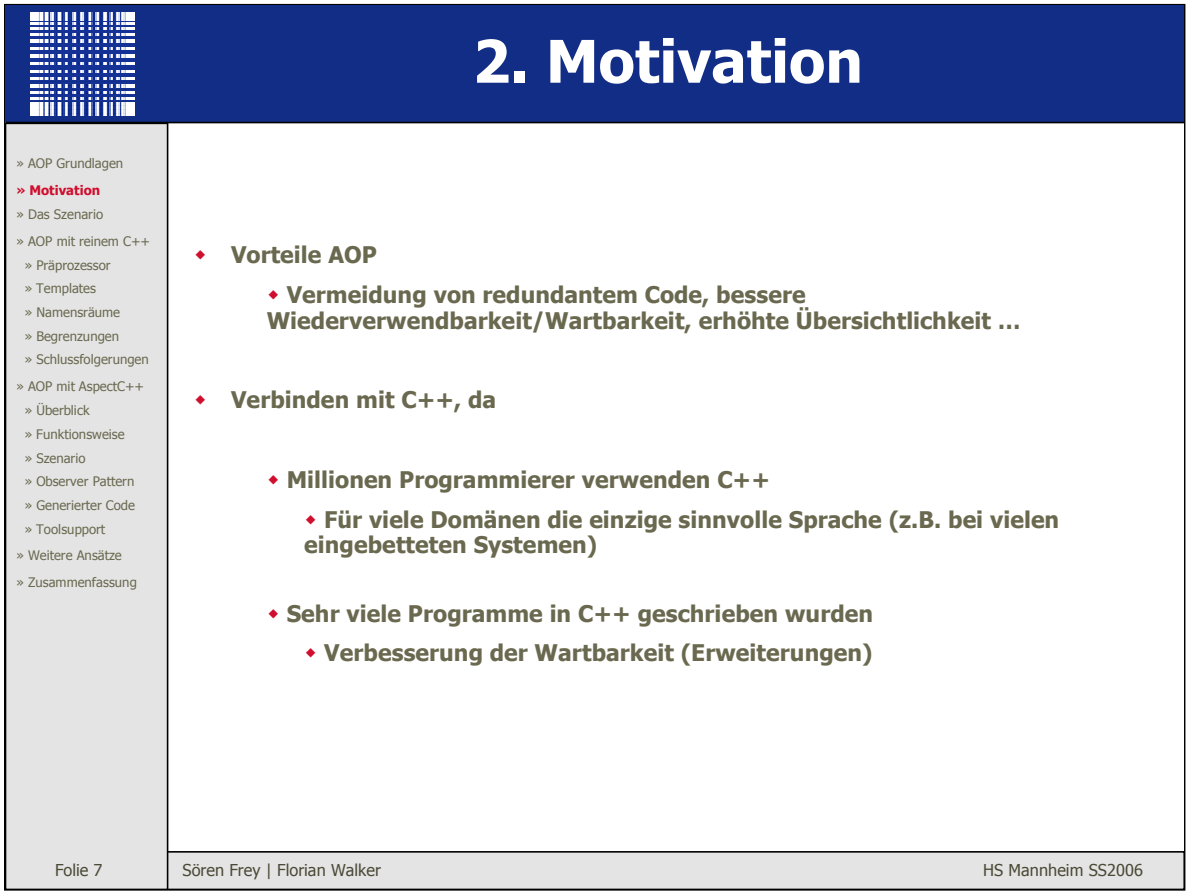
1. AOP Grundlagen - Begriffe

- **Joinpoint**
  - **Wohldefinierter Punkt im Programmfluss**
  - **Können sein: Aufruf, Ausführung einer Methode, Zugriff auf Variable, Behandlung einer Exception ...**
- **Pointcut**
  - **Die Auswahl eines oder mehrerer Joinpoints**
  - **Bei erreichen des Pointcuts wird der Aspekt-Code ausgeführt**



1. AOP Grundlagen - Begriffe

- **Advice**
  - **Der Code des Aspekts**
  - **Ausführung bei erreichen des Pointcuts**
  - **Kann eingewebt werden: z.B. before, after, around**
- **Introduction**
  - **Erweiterung anderer Klassen, Interfaces um zusätzliche Funktionalität**
  - **Beispielsweise hinzufügen neuer Funktionen oder Attribute**
- **Aspect**
  - **„Container für neue Konzepte"**
  - **können Advices und Pointcuts enthalten**
  - **sowie Methoden und Variablen**

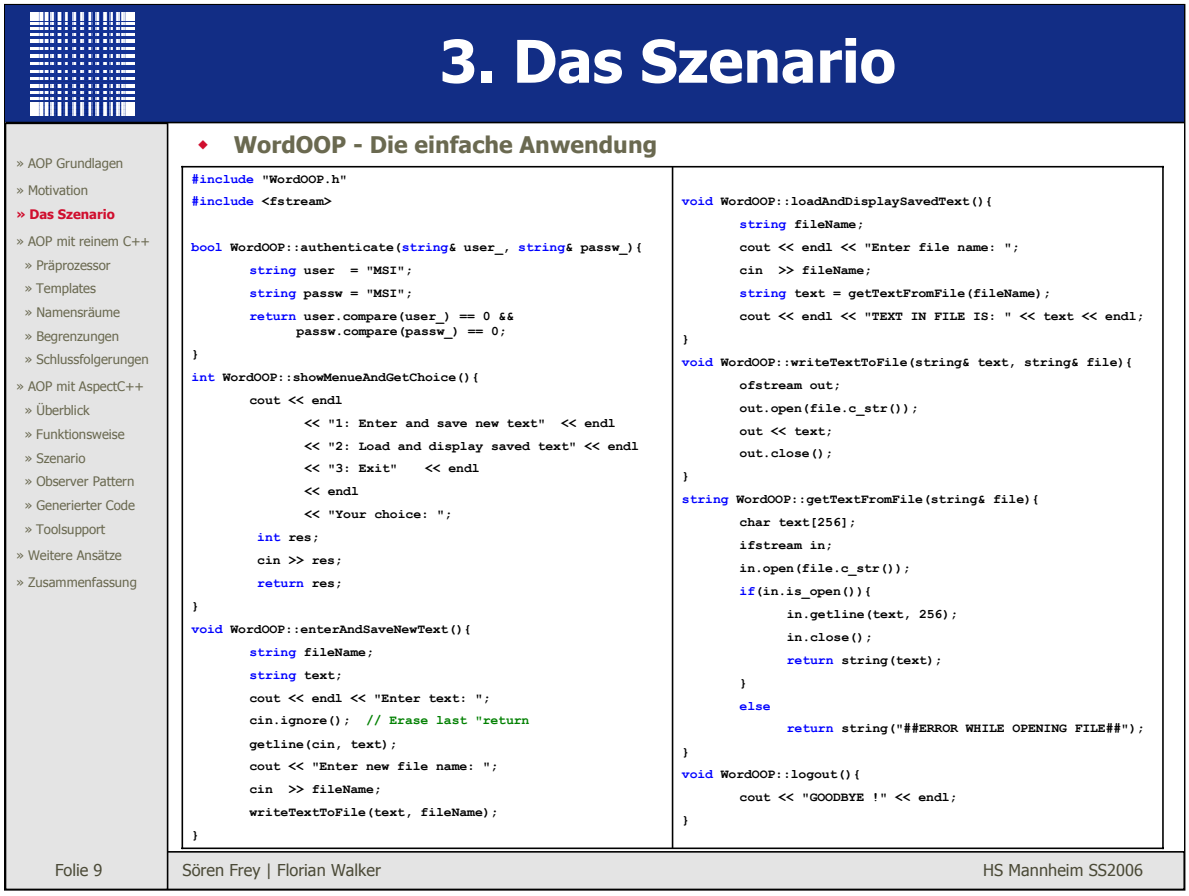


- Verbesserung der Wartbarkeit da Erweiterungen oft die *crosssutting concens* sind.

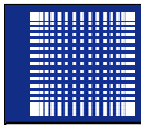
3. Das Szenario

- **Anwendung: WordOOP**
  - **Einfache „Textverarbeitung"**
  - **Funktionen**
    - **Authentifikation über Benutzernamen/Passwort**
    - **Texteingabe und anschließende Speicherung**
    - **Gespeicherten Text laden und anzeigen**

WordOOP
+authenticate(string& user_, string& passw_):bool +showMenueAndGetChoice():int +enterAndSaveNewText():void +loadAndDisplaySavedText():void +writeTextToFile(string& text, string& file):void +getTextFromFile(string& file):string +logout():void



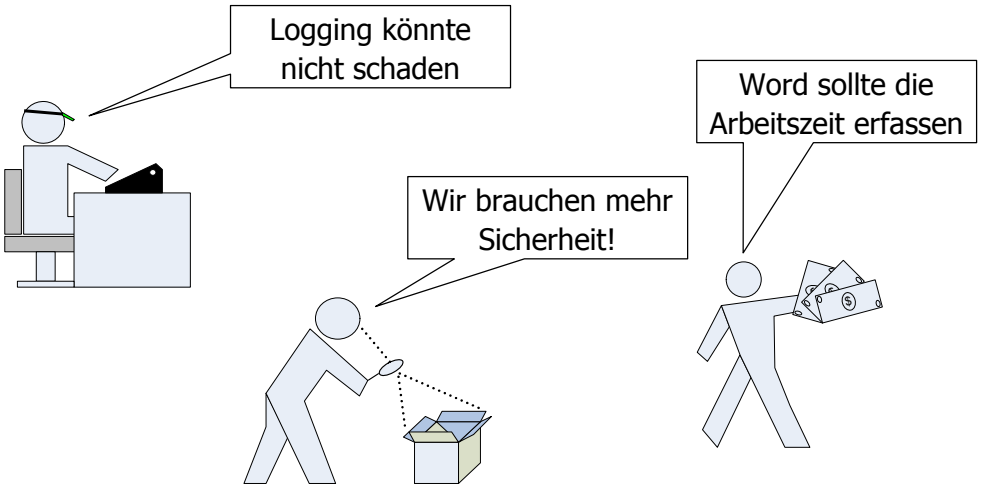
- Inhalt der Implementierungsdatei WordOOP.cpp
  - Definition der Klasse
  - Fachlicher Code
- Klassendeklaration in der Header-Datei WordOOP.h
- Verwendung durch Anwendungscode (app.cpp)

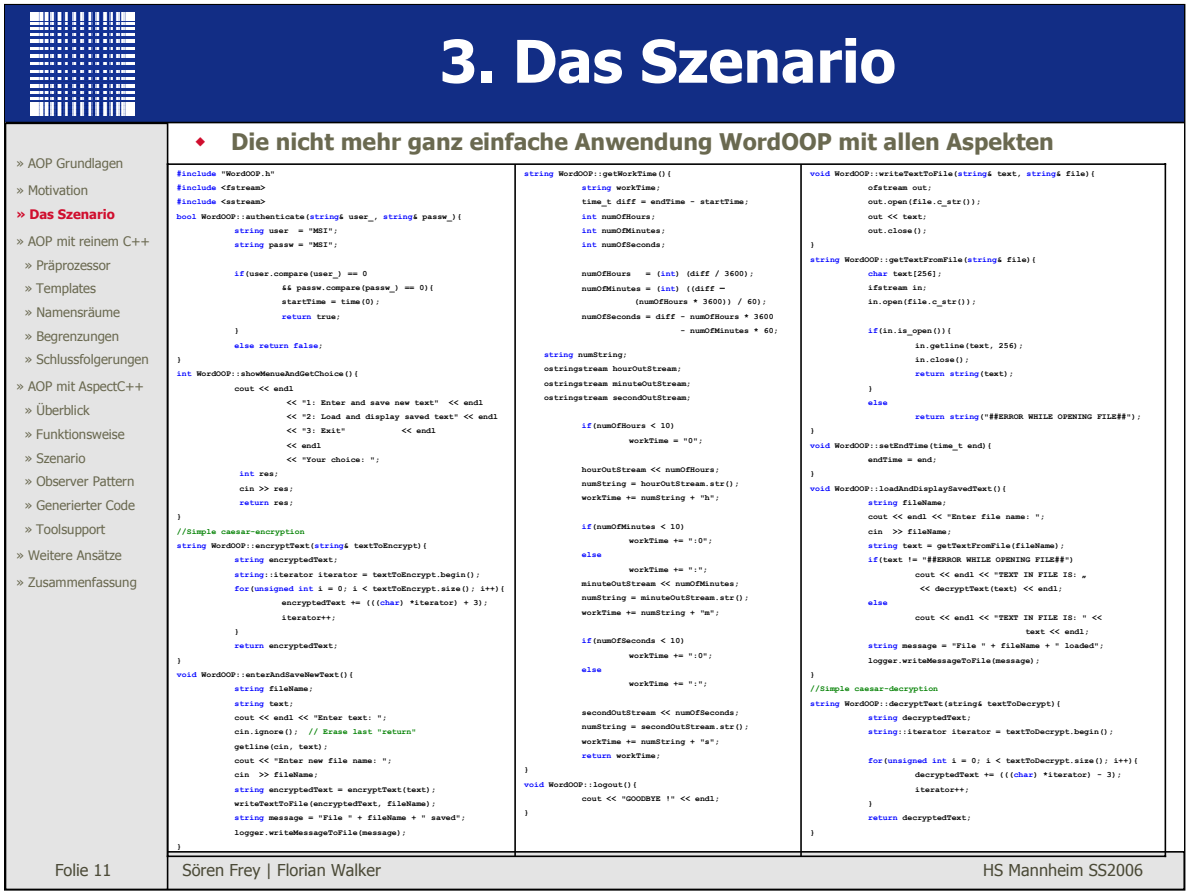


3. Das Szenario

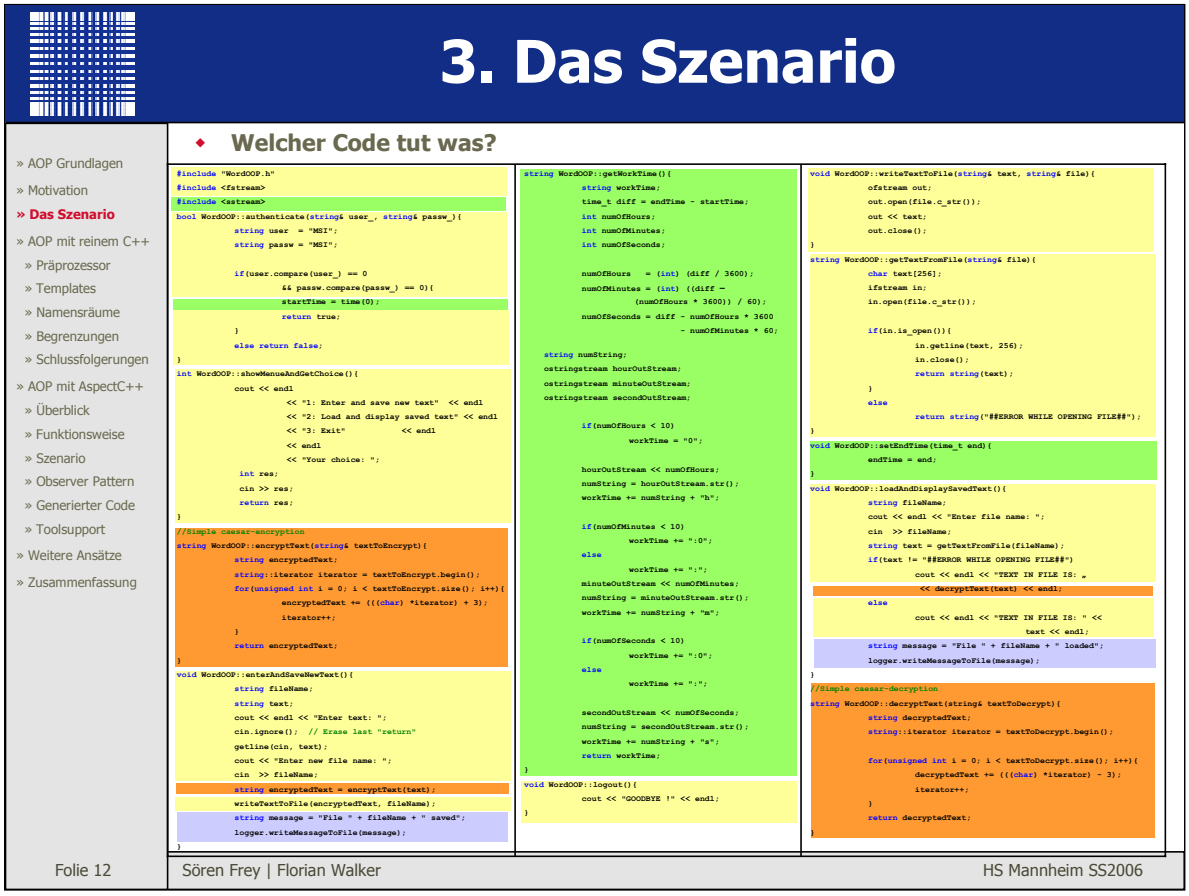
Typisches Problem:

- **Verschiedene Stakeholder fordern Erweiterungen von WordOOP**





- Inhalt der Implementierungsdatei nach Realisierung aller geforderten Aspekte direkt in der Fachklasse.

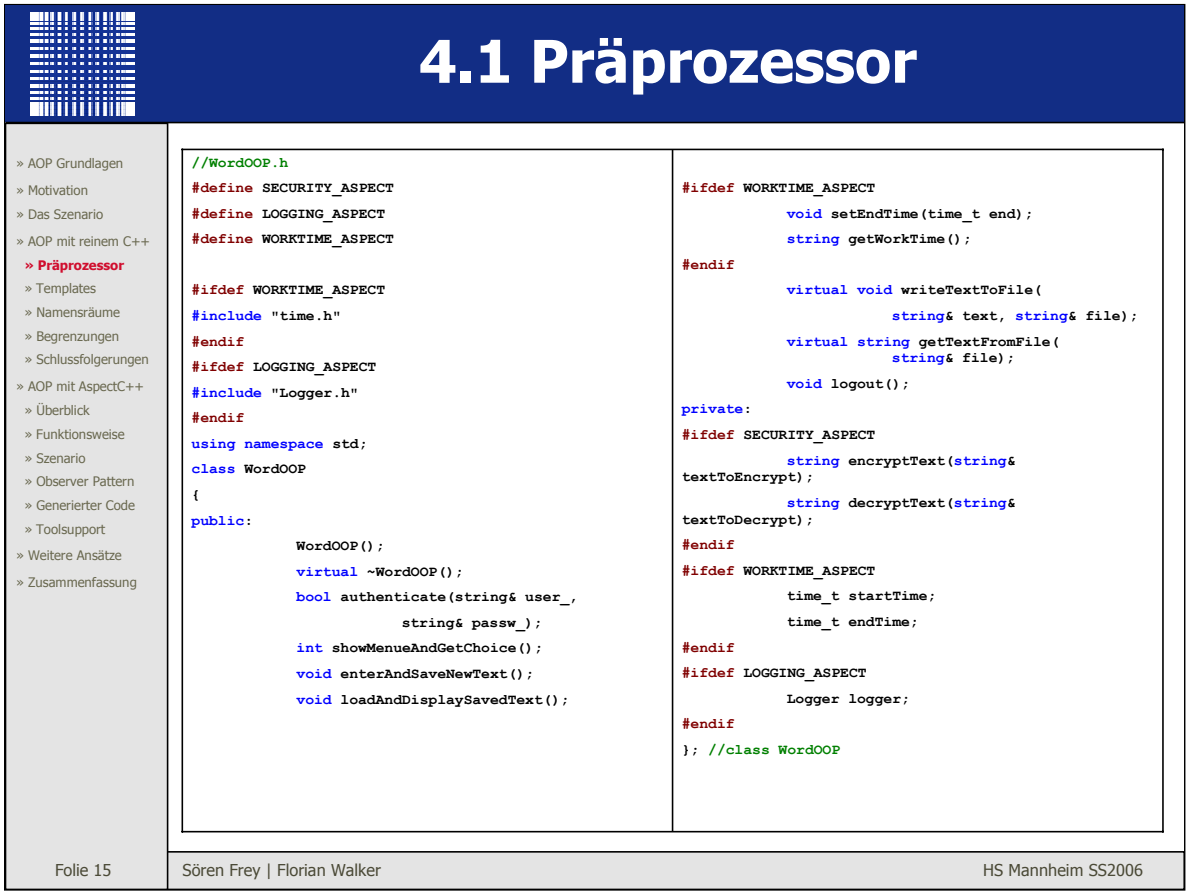

3. Das Szenario
» AOP Grundlagen
» Motivation
» Das Szenario
» AOP mit reinem C++
» Präprozessor
» Templates
» Namensräume
» Begrenzungen
» Schlussfolgerungen
» AOP mit AspectC++
» Überblick
» Funktionsweise
» Szenario
» Observer Pattern
» Generierter Code
» Toolsupport
» Weitere Ansätze
» Zusammenfassung
Welcher Code tut was?
#include "WordOOP.h"
#include <fstream>
#include <sstream>
bool WordOOP::authenticate(string& user_, string& passw_){
string user = "MSI";
string passw = "MSI";
if(user.compare(user_) == 0
&& passw.compare(passw_) == 0){
startTime = time(0);
return true;
}
else return false;
}
int WordOOP::showMenueAndGetChoice(){
cout << endl
<< "1: Enter and save new text" << endl
<< "2: Load and display saved text" << endl
<< "3: Exit" << endl
<< endl
<< "Your choice: ";
int res;
cin >> res;
return res;
}
//Simple caesar-encryption
string WordOOP::encryptText(string& textToEncrypt){
string encryptedText;
string::iterator iterator = textToEncrypt.begin();
for(unsigned int i = 0; i < textToEncrypt.size(); i++){
encryptedText += (((char) *iterator) + 3);
iterator++;
}
return encryptedText;
}
void WordOOP::enterAndSaveNewText(){
string fileName;
string text;
cout << endl << "Enter text: ";
cin.ignore(); // Erase last "return"
getline(cin, text);
cout << "Enter new file name: ";
cin >> fileName;
string encryptedText = encryptText(text);
writeTextToFile(encryptedText, fileName);
string message = "File " + fileName + " saved";
logger.writeMessageToFile(message);
}
string WordOOP::getWorkTime(){
string workTime;
time_t diff = endTime - startTime;
int numOfHours;
int numOfMinutes;
int numOfSeconds;
numOfHours = (int) (diff / 3600);
numOfMinutes = (int) ((diff -
(numOfHours * 3600)) / 60);
numOfSeconds = diff - numOfHours * 3600
- numOfMinutes * 60;
string numString;
ostringstream hourOutStream;
ostringstream minuteOutStream;
ostringstream secondOutStream;
if(numOfHours < 10)
workTime = "0";
hourOutStream << numOfHours;
numString = hourOutStream.str();
workTime += numString + "h";
if(numOfMinutes < 10)
workTime += ":0";
else
workTime += ":";
minuteOutStream << numOfMinutes;
numString = minuteOutStream.str();
workTime += numString + "m";
if(numOfSeconds < 10)
workTime += ":0";
else
workTime += ":";
secondOutStream << numOfSeconds;
numString = secondOutStream.str();
workTime += numString + "s";
return workTime;
}
void WordOOP::logout(){
cout << "GOODBYE !" << endl;
}
void WordOOP::writeTextToFile(string& text, string& file){
ofstream out;
out.open(file.c_str());
out << text;
out.close();
}
string WordOOP::getTextFromFile(string& file){
char text[256];
ifstream in;
in.open(file.c_str());
if(in.is_open()){
in.getline(text, 256);
in.close();
return string(text);
}
else
return string("##ERROR WHILE OPENING FILE##");
}
void WordOOP::setEndTime(time_t end){
endTime = end;
}
void WordOOP::loadAndDisplaySavedText(){
string fileName;
cout << endl << "Enter file name: ";
cin >> fileName;
string text = getTextFromFile(fileName);
if(text != "##ERROR WHILE OPENING FILE##")
cout << endl << "TEXT IN FILE IS: „
<< decryptText(text) << endl;
else
cout << endl << "TEXT IN FILE IS: " <<
text << endl;
string message = "File " + fileName + " loaded";
logger.writeMessageToFile(message);
}
//Simple caesar-decryption
string WordOOP::decryptText(string& textToDecrypt){
string decryptedText;
string::iterator iterator = textToDecrypt.begin();
for(unsigned int i = 0; i < textToDecrypt.size(); i++){
decryptedText += (((char) *iterator) - 3);
iterator++;
}
return decryptedText;
}
Folie 12
Sören Frey | Florian Walker
HS Mannheim SS2006

3. Das Szenario - Probleme

- Vermischung des Codes mit verschiedenen logisch getrennten Aspekten
- Code ist schwer zu schreiben
  - gleichzeitige Beachtung aller Aspekte
- Code ist schwer zu lesen
- Weiterentwicklung und Wartung sind umständlich
  - Verschiedene Belange sind über den gesamten Code verstreut
- Skalierbarkeit ist schwer
  - Anwender bekommt eine Klasse für alles

4. AOP mit reinem C++

- **Überblick**
  - **Erweiterung des vorgestellten Beispiels um verschiedene Aspekte**
    - **Sicherheit – Verschlüsselung der Textdateien**
    - **Arbeitszeiterfassung**
    - **Protokollierung der Dateizugriffe**
  - **Umsetzung verschiedener AOP Konzepte mittels**
    - **Präprozessor**
    - **Templates**
    - **Namensräumen**
  - **Ziel:**
    - **Vorteile und Beschränkungen von reinem C++ aufzeigen**
    - **Notwendigkeit von Spracherweiterungen einschätzen**



- Der Präprozessor führt Änderungen am Quellcode durch, bevor der eigentliche Compiler das Programm übersetzt.
- #ifdef, #endif ermöglichen die bedingte Übersetzung bestimmter Programmteile ⇔ „compile-time-Schalter"
  - Häufige Anwendung: Konfiguration von Debug-Code

4.1 Präprozessor

- **Vorteil:**

  + Funktionen können aktiviert/deaktiviert werden

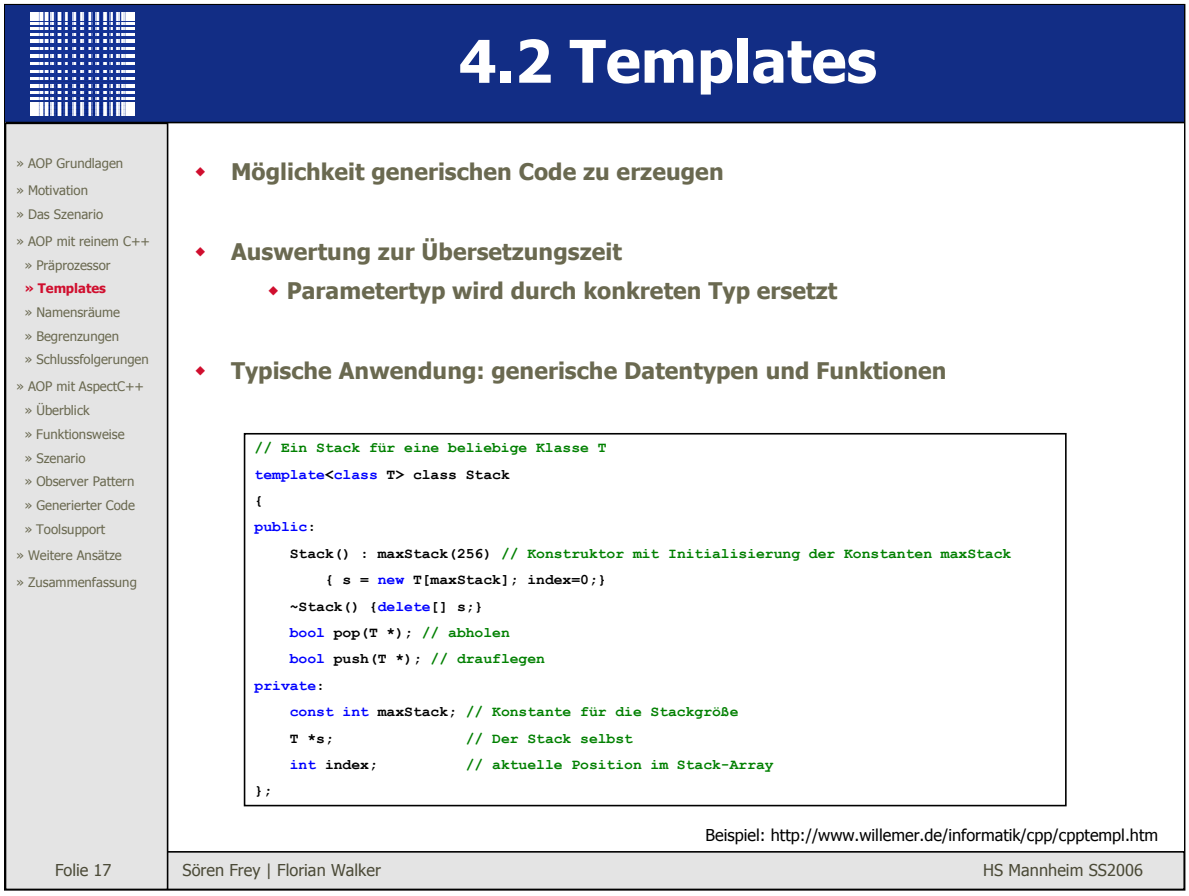
- **Nachteile:**

  - Keine Modularisierung der Aspekte

  - Verteilung des Aspekts über den gesamten Quellcode

  - Quellcode ist sehr schwer zu lesen

➔ Eleganter: C++ - Templates



- Generische Programmierung ist ein Paradigma, bei dem einzelne Funktionen oder Klassen möglichst allgemein beschrieben werden, so dass sie für verschiedene Datentypen anwendbar sind.
- Wie beim Präprozessor erfolgt die Auswertung zur Übersetzungszeit.
- Templates auch für Funktionen möglich. Beispiel:

```
template <class T>
T Algorithmus1 (T a, T b, T c){
     T d = a+b+c;
     return d;
}
```

  - Aufruf:

```
int A = 5, B = -26, C = -48, D = 0;
D = Algorithmus1<int>(A, B, C);
```

  - Wenn die Datentypen der Argumente gleichartig sind, kann die Angabe des speziellen Datentyps weggelassen werden. Der Compiler findet selbst den Typ heraus:

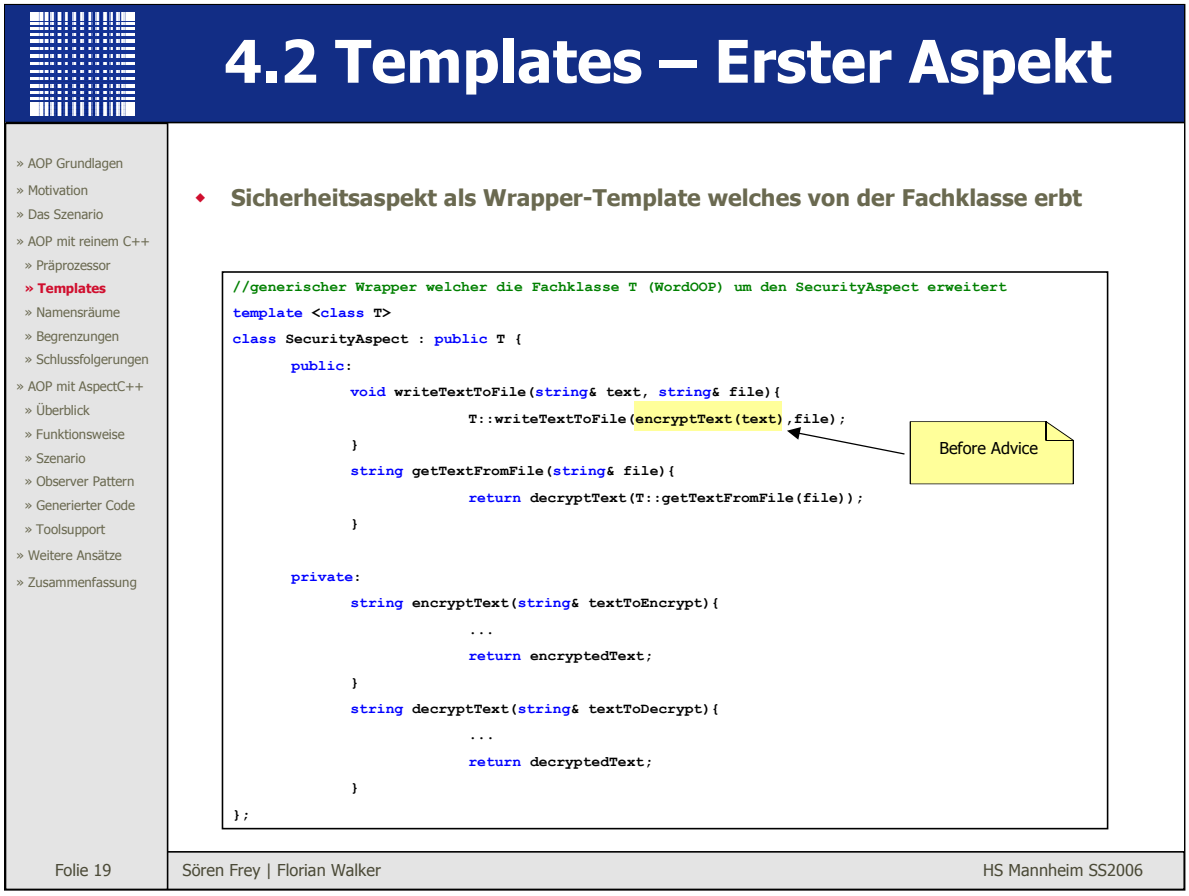
```
int A = 5, B = -26, C = -48, D = 0;
D = Algorithmus1 (A, B, C);
```

4.2 Templates

- **Templates erlauben die Kapselung des Aspektcodes unabhängig vom fachlichen Code**

```
class WordOOP {
...
        void writeTextToFile(string& text, string& file){
                ofstream out;
                out.open(file.c_str());
                out << text;
                out.close();
        }
        string getTextFromFile(string& file){
                char text[256];
                ifstream in;
                in.open(file.c_str());
                if(in.is_open()){
                                in.getline(text, 256);
                                in.close();
                                return string(text);
                }
                else return string("##ERROR WHILE OPENING FILE##");
        }
... };
```

Fachlicher Code des Szenarios



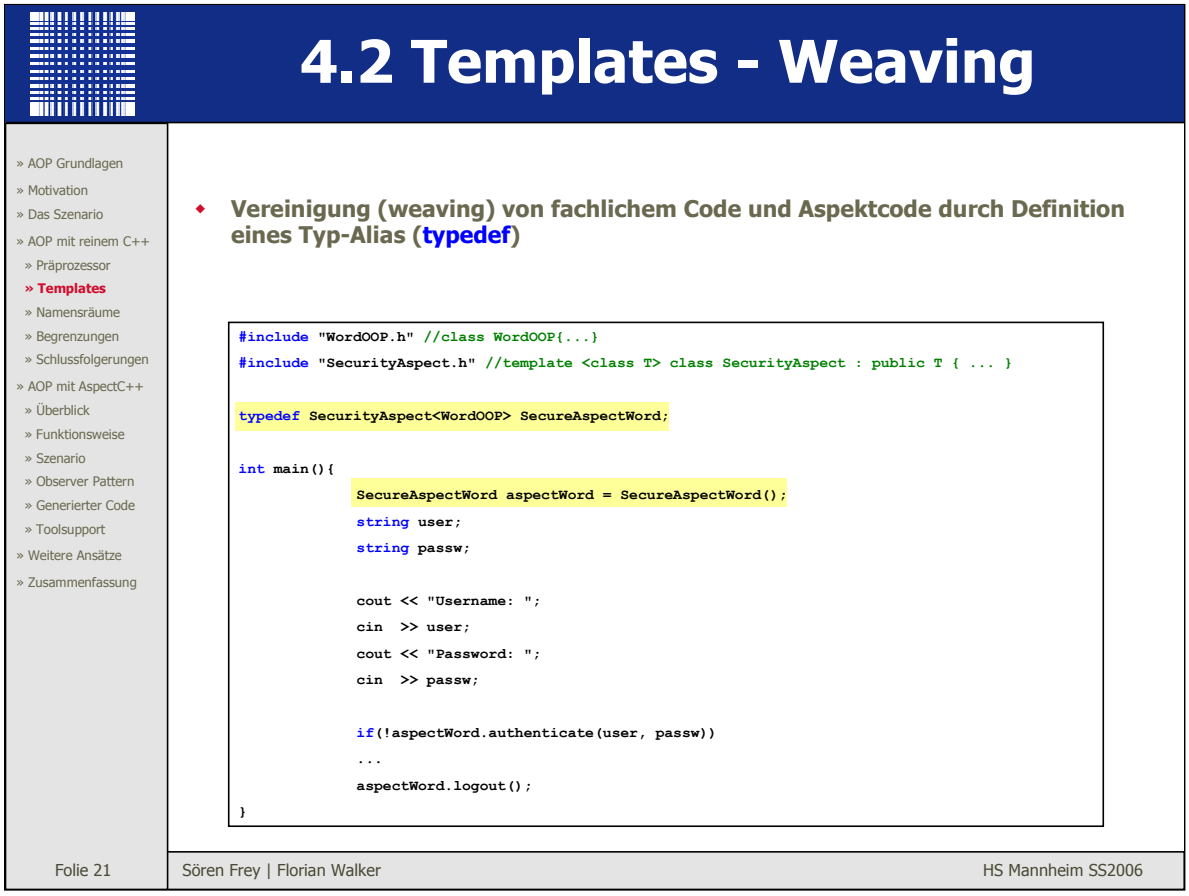
- (eng.) Wrapper = Hülle
- Wrapper-Template erbt von der fachlichen Klasse und überschreibt die für den Aspekt relevanten Methoden.
- Before Advice: Aspekt-Code wird vor fachlichem Code ausgeführt
- After Advice: Aspekt-Code wird nach fachlichem Code ausgeführt

4.2 Templates – Erster Aspekt

- **Sicherheitsaspekt als Wrapper-Template welches von der Fachklasse erbt**

```
//generischer Wrapper welcher die Fachklasse T (WordOOP) um den SecurityAspect erweitert
template <class T>
class SecurityAspect : public T {
      public:
            void writeTextToFile(string& text, string& file){
                  T::writeTextToFile(encryptText(text),file);
            }
            string getTextFromFile(string& file){
                  return decryptText(T::getTextFromFile(file));
            }

      private:
            string encryptText(string& textToEncrypt){
                  ...
                  return encryptedText;
            }
            string decryptText(string& textToDecrypt){
                  ...
                  return decryptedText;
            }
};
```

After Advice



- Mithilfe der typedef-Anweisung können Synonyme (andere Namen) für bestehende Datentypen gebildet werden
- Syntax:

```
typedef Datentyp Synonym;
```

- Beispiele:

```
typedef unsigned short WORD;
typedef unsigned long DWORD;
```

4.2 Templates – Zwischenstand

- **Implementierung von Aspekten als Wrapper-Templates**
  - **Aspekt erbt von der Fachklasse und überschreibt die notwendigen Methoden**
  - **Weaving erfolgt durch Definition von Typ-Alias**
- **Generischer Aspektcode**
  - **Aspekt kann auf jede Fachklasse angewendet werden, welche die gleiche Schnittstelle besitzt**
- **Trennung des Aspektcodes**
  - **Der gesamte für den SecurityAspect relevante Code ist im Template gekapselt**
  - **Fachklasse und Aspekt könnten separat weiterentwickelt werden**

4.2 Templates – Zweiter Aspekt

- **Umsetzung wie erster Aspekt ➔ Wrapper-Template**

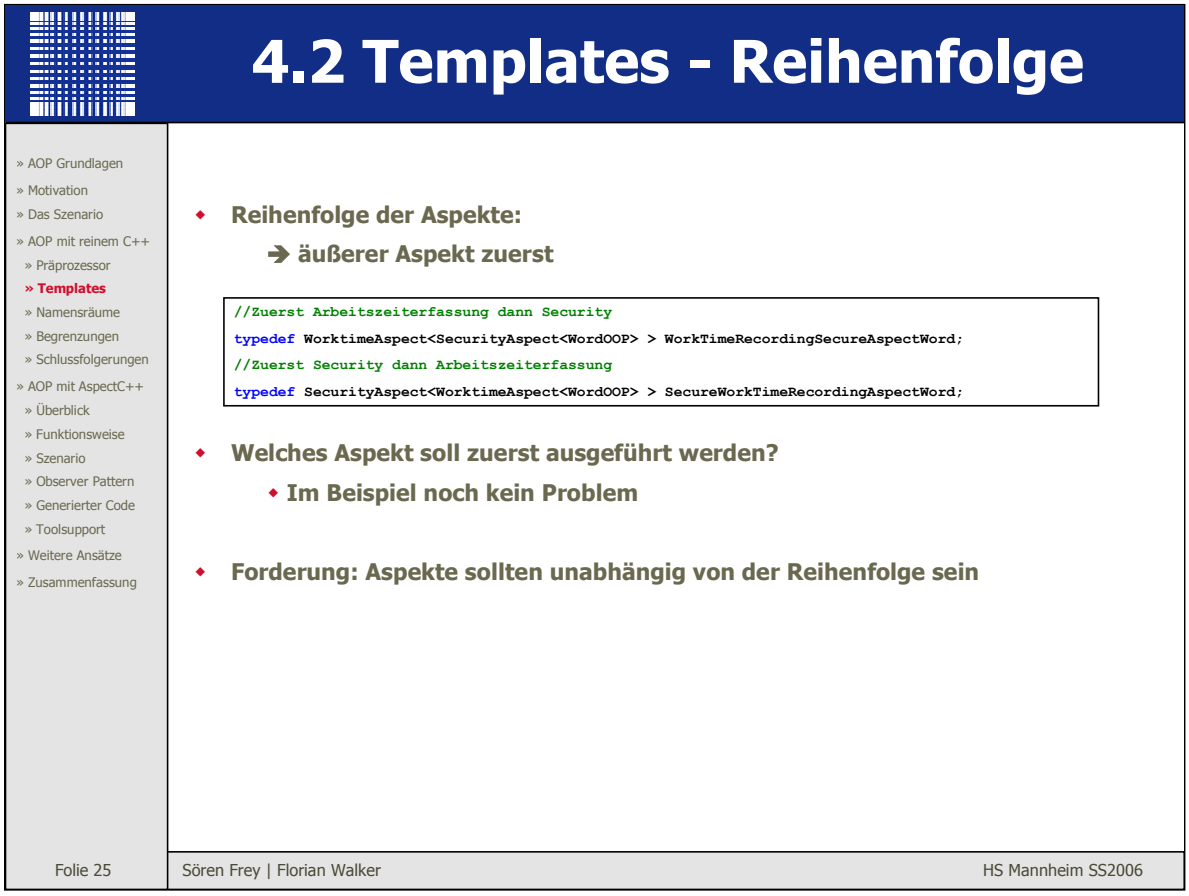
```
template <class T>//WorkTimeAspect als Wrapper
class WorktimeAspect : public T {
        private:
            time_t startTime;
            time_t endTime;
        public:
            bool authenticate(string& user_, string& passw_){
                        startTime = time(0);
                        return T::authenticate(user_,passw_); //afer advice
            }
        ^   void logout(){
                        endTime = time(0);
                        cout << "You have been working for " << getWorkTime() << " ";
                        T::logout(); //after advice
            }
            string getWorkTime(){
                        ...
                        return workTime;
            }
};
```

Einführen einer neuen Methode (Introduction)

4.2 Templates - Weaving

- **Bereits bekannte Verwendung von typdef**

```
#include "WordOOP.h" //class WordOOP{...}
#include "SecurityAspect.h" //template <class T> class SecurityAspect : public T { ... }
#include "WorktimeAspect.h" //template <class T> class WorktimeAspect : public T { ... }

typedef WorktimeAspect<SecurityAspect<WordOOP> > WorkTimeRecordingSecureAspectWord;

int main(){
            WorkTimeRecordingSecureAspectWord aspectWord =
                        WorkTimeRecordingSecureAspectWord();

            string user;
            string passw;
            cout << "Username: ";
            cin  >> user;
            cout << "Password: ";
            cin  >> passw;

            if(!aspectWord.authenticate(user, passw))
            ...
            aspectWord.logout();
}
```



- Reihenfolge der Aspekte im Beispiel bisher nicht relevant, da die Aspekte die Fachklasse nicht an der gleichen Stelle erweitern/modifizieren.

4.2 Templates – Dritter Aspekt

- **Vorgehen wie bei den zwei vorangegangenen Aspekten**

```
template <class T>
class LoggingAspect : public T {
public:
     void writeTextToFile(string& text, string& file)
     {
          string beforeUsageMessage =
                    getBeforeUsageMessage();
          writeMessageToFile(beforeUsageMessage);
          T::writeTextToFile(text,file);
          string afterUsageMessage =
                    getAfterUsageMessage();
          writeMessageToFile(afterUsageMessage);
     }

     string getTextFromFile(string& file)
     {
          string beforeUsageMessage =
                    getBeforeUsageMessage();
          writeMessageToFile(beforeUsageMessage);
          string res = T::getTextFromFile(file);
          string afterUsageMessage =
                    getAfterUsageMessage();
          writeMessageToFile(afterUsageMessage);
          return res;
     }
```

```
private:
     void writeMessageToFile(string& message){
          //Make message persistent
     }

     string getBeforeUsageMessage(){
                    return "";
     }

     string getAfterUsageMessage(){
          return "File accessed";
     }
};
```

4.2 Templates – Dritter Aspekt

- **Aspekt verwendet „around advice" welcher fachlichen Code innerhalb des Aspektcodes ausführt**

```
template <class T>
class LoggingAspect : public T {
public:
      void writeTextToFile(string& text, string& file)
      {
            string beforeUsageMessage =
                        getBeforeUsageMessage();
            writeMessageToFile(beforeUsageMessage);
            T::writeTextToFile(text,file);
            string afterUsageMessage =
                        getAfterUsageMessage();
            writeMessageToFile(afterUsageMessage);
      }

      string getTextFromFile(string& file)
      {
            string beforeUsageMessage =
                        getBeforeUsageMessage();
            writeMessageToFile(beforeUsageMessage);
            string res = T::getTextFromFile(file);
            string afterUsageMessage =
                        getAfterUsageMessage();
            writeMessageToFile(afterUsageMessage);
            return res;
      }
```

```
private:
      void writeMessageToFile(string& message){
            //Make message persistent
      }

      string getBeforeUsageMessage(){
                        return "";
      }

      string getAfterUsageMessage(){
            return "File accessed";
      }
};
```

4.2 Templates – Dritter Aspekt

- **Mehrfacher Code durch Definition des gleichen Advices für mehrere Joinpoints**

```
template <class T>
class LoggingAspect : public T {
public:
      void writeTextToFile(string& text, string& file)
      {
            string beforeUsageMessage =
                        getBeforeUsageMessage();
            writeMessageToFile(beforeUsageMessage);
            T::writeTextToFile(text,file);
            string afterUsageMessage =
                        getAfterUsageMessage();
            writeMessageToFile(afterUsageMessage);
      }

      string getTextFromFile(string& file)
      {
            string beforeUsageMessage =
                        getBeforeUsageMessage();
            writeMessageToFile(beforeUsageMessage);
            string res = T::getTextFromFile(file);
            string afterUsageMessage =
                        getAfterUsageMessage();
            writeMessageToFile(afterUsageMessage);
            return res;
      }
```

```
private:
      void writeMessageToFile(string& message){
            //Make message persistent
      }

      string getBeforeUsageMessage(){
                        return "";
      }

      string getAfterUsageMessage(){
            return "File accessed";
      }
};
```

4.2 Templates – Wiederverwendung

- **Anwendung eines Advice auf mehrere Joinpoints erfordert einheitliche Schnittstelle der Joinpoints**

```
template <class T>
class LoggingAspect : public T {
     public:
            struct WriteAction
            {
                  string message;
                  string file;
                  T t;
                  void proceed(){t.writeTextToFile(message,file);}
            };
            struct ReadAction
            {
                  string file;
                  string res;
                  T t;
                  void proceed(){res = t.getTextFromFile(file);}
            };
...
```

4.2 Templates – Wiederverwendung

- **Advice als Template-Funktion**
  - **Action-Klasse parametrisiert die Funktion**

```
template <class T>
class LoggingAspect : public T {
     public:
            ...
            template<class action>
            void myAdvice(action* a)
            {
                  string beforeUsageMessage = getBeforeUsageMessage();
                  writeMessageToFile(beforeUsageMessage);
                  a->proceed();
                  string afterUsageMessage = getAfterUsageMessage();
                  writeMessageToFile(afterUsageMessage);
            }
            ...
```

4.2 Templates – Wiederverwendung

- **Advice an Joinpoints binden**
  - **Durch die Action-Klassen konnte der Advice-Code in einer Funktion gekapselt werden**

```
template <class T>
class LoggingAspect : public T {
     public:
            ...
            void writeTextToFile(string& text, string& file)
            {
                  WriteAction tjp = {text,file};
                  myAdvice(&tjp);
            }

            string getTextFromFile(string& file)
            {
                  ReadAction tjp = {file};
                  myAdvice(&tjp);
                  return tjp.res;
            }
...
```

4.2 Templates – Zwischenstand

- **Vermeidung von doppelten Codes durch**
  - **Delegierung an Action-Klassen**
  - **Implementierung des Advices als Template-Funktion**
  - **Parametrisieren der Template-Funktionen mit den Action-Klassen**
  - **Overhead durch Speicherung zusätzlicher Argumente / Rückgabewerte**

4.2 Templates

- **Kombination der Fachklasse mit allen Aspekten**

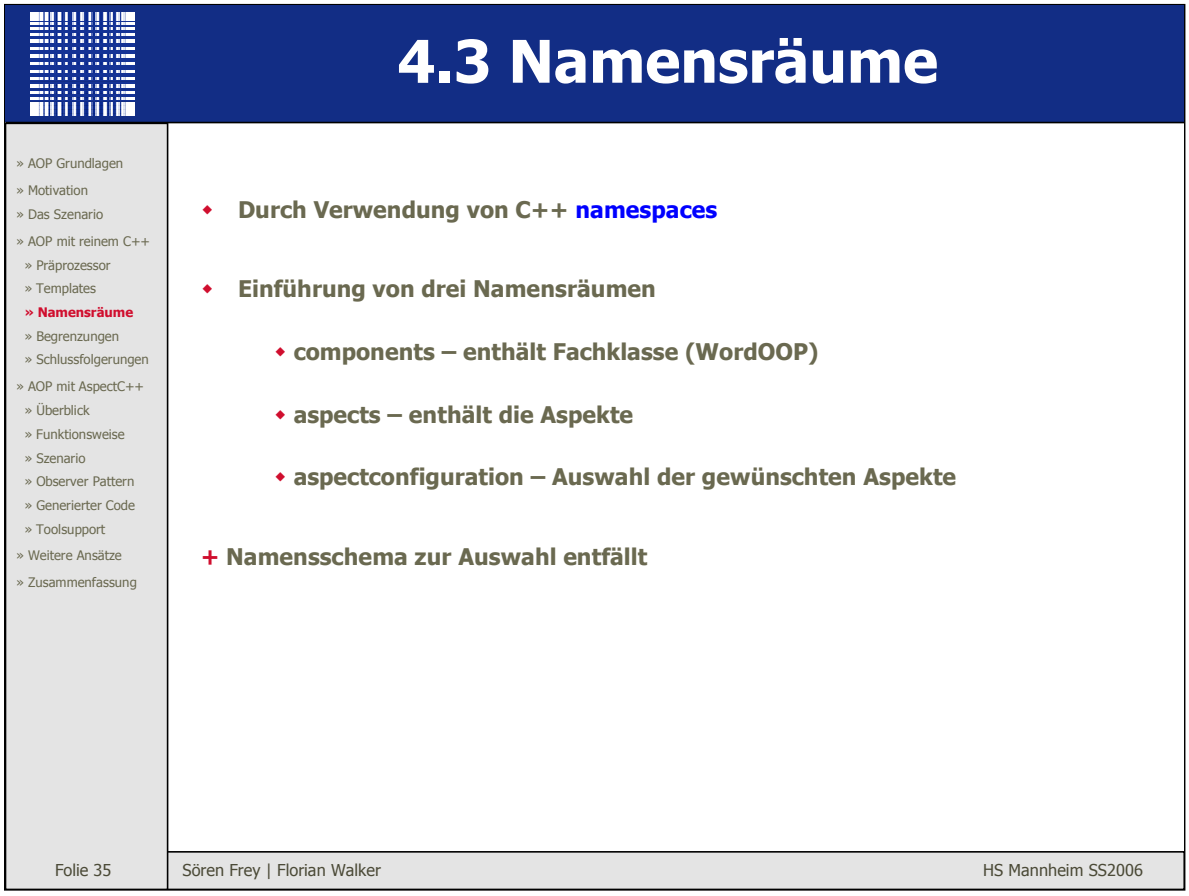
```
typedef LoggingAspect<WorktimeAspect<SecurityAspect<WordOOP> > >
                                                    LoggingAndWorkTimeRecordingSecureAspectWord;
```

- **Vielleicht etwas besser zu lesen:**

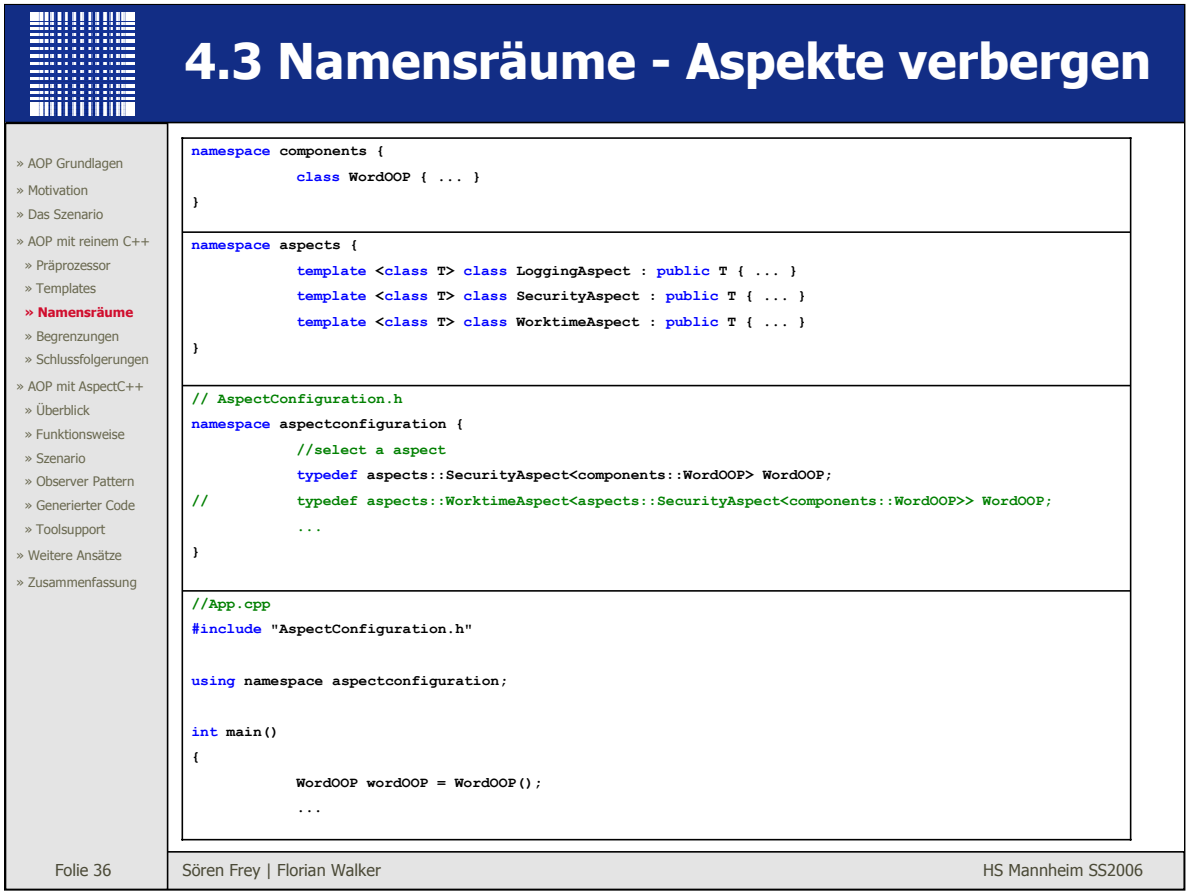
```
typedef SecurityAspect<WordOOP> SecureAspectWord;
typedef WorktimeAspect<SecureAspectWord> WorkTimeRecordingSecureAspectWord;
typedef LoggingAspect<WorkTimeRecordingSecureAspectWord> LoggingAndWorkTimeRecordingSecureAspectWord;
```

4.2 Templates - Probleme

- **Eine Kerneigenschaft von AOP:**
  Fachlicher Code ist sich *nicht* der Modifikation durch den Aspekt bewusst
- **Erweiterte Klasse kann nicht den gleichen Namen tragen wie Fachklasse**
  - **Auswahl erfolgt durch das gewählte Namensschema**
    (z.B. LoggingAndWorkTimeRecordingSecureAspectWord)
- **➔ Verletzung der Eigenschaft**
- **➔ verbergen der Aspekte im Client-Code**



- Namespaces (= namensräume) kennzeichnen Einheiten die zusammengehören. Sie verhindern Namenskonflikte und teilen den Code in logische Einheiten auf.
- Importieren aller Namen eines Namensraums mittels `using` Direktive (`using namespace myNS`)
- Zugriff auf Element in anderem Namensraum mittels Bereich-Auflösungs-Operator (`myNS::ElementA`)



- Vermeidung von Namenskonflikten
- Fachlicher Code muss nur die Verwendung des aspectconfiguration-Namensraumes enthalten

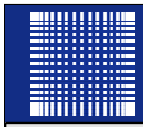
4.4 Begrenzungen

- **Einsatz der vorgestellten Methoden nur sinnvoll für einfache Aspekte**
- **Anwendung eines Aspekts auf Klassen mit unterschiedlicher Schnittstelle ist nicht möglich**
- **Kein Advice auf nicht virtuelle private Methoden möglich**
- **Starke Verwendung von Templates und Namensräumen macht den Code schwer zu verstehen und erschwert die Fehlerbeseitigung**
- **Oft: mehr Code für den Wrapper als für den Aspekt**

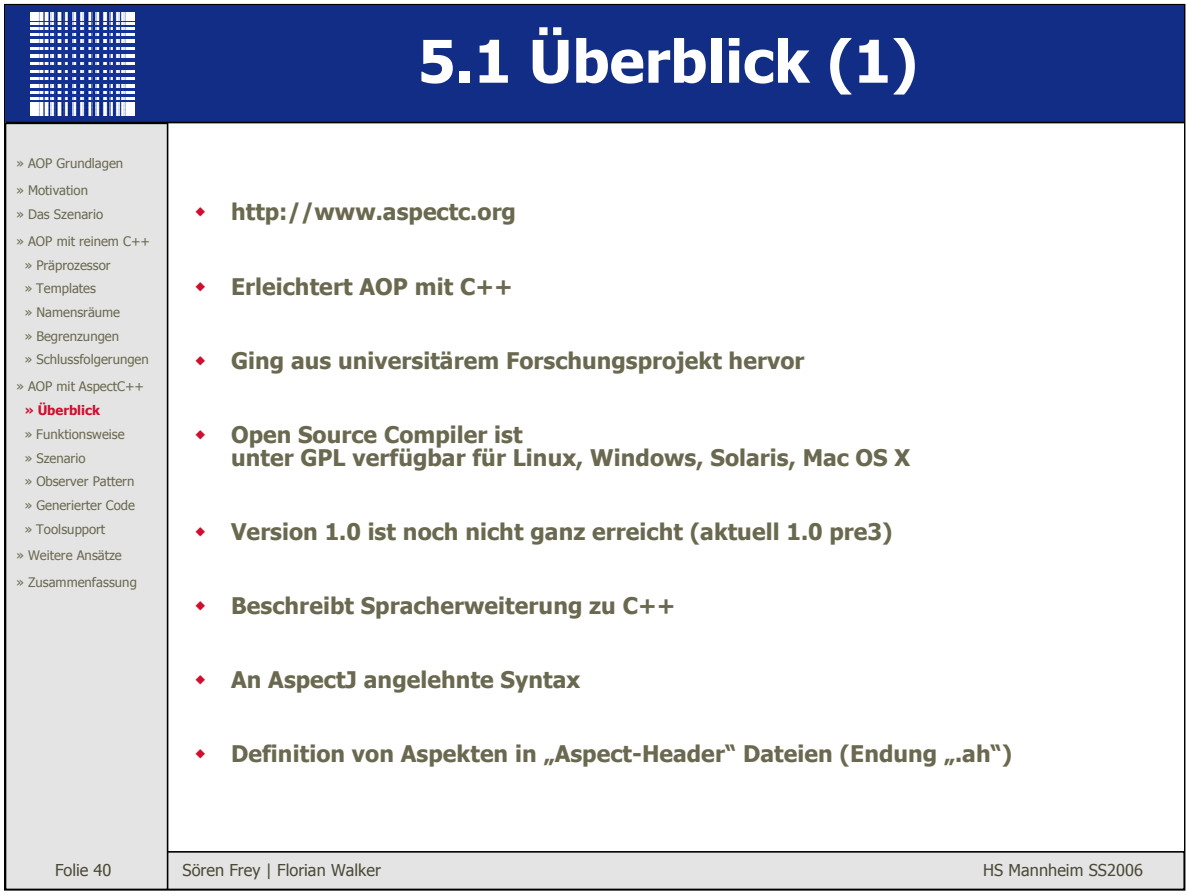
4.5 Schlussfolgerungen

- „separation of concerns" ist mit C++ Templates ohne zusätzliche Tools und Spracherweiterungen möglich
- Jedoch nur sinnvoll wenn
  - eine geringe Anzahl Aspekte umzusetzen ist
  - die Aspekte einfacher Natur sind
- „AOP mit reinem C++ ist wie OOP mit C"
- ➔ Notwenigkeit von
  - Spracherweiterungen
  - Werkzeugunterstützung

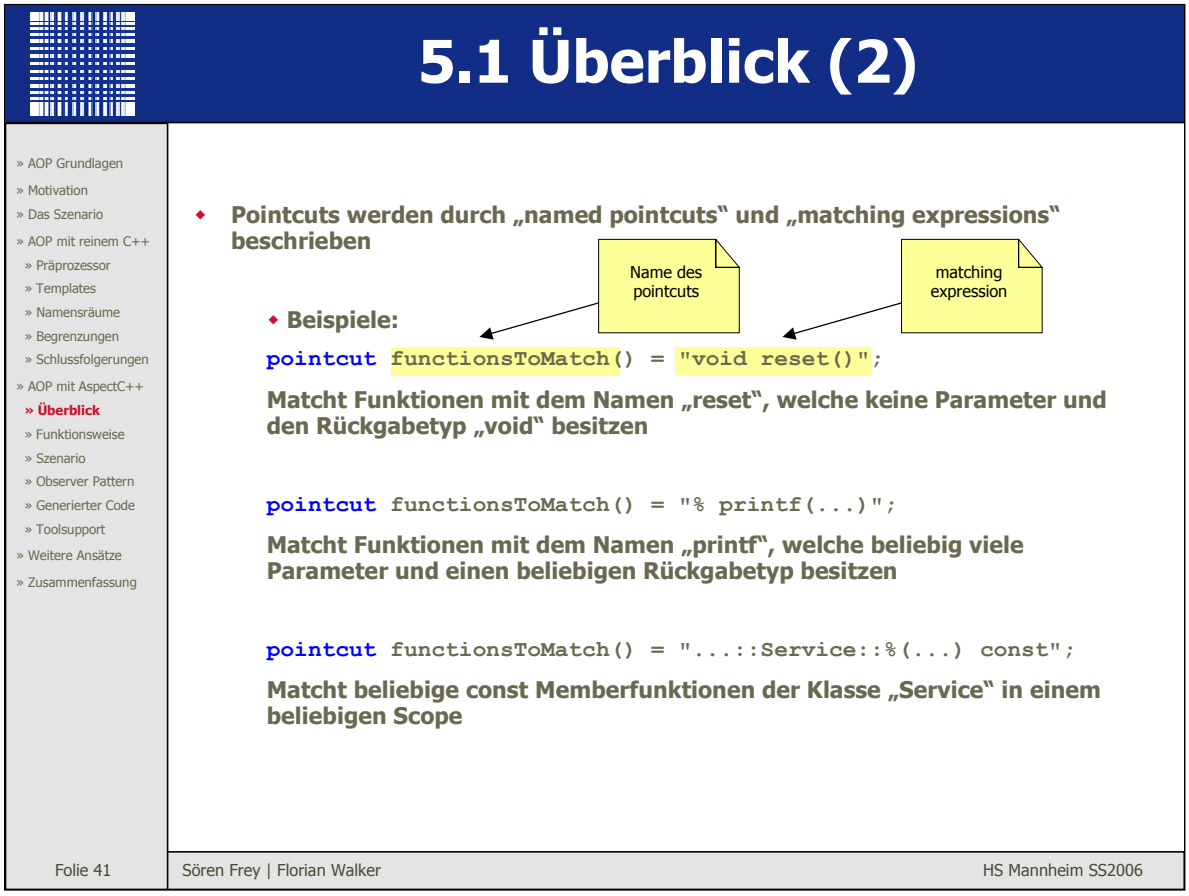
  für größere Projekte sinnvoll.



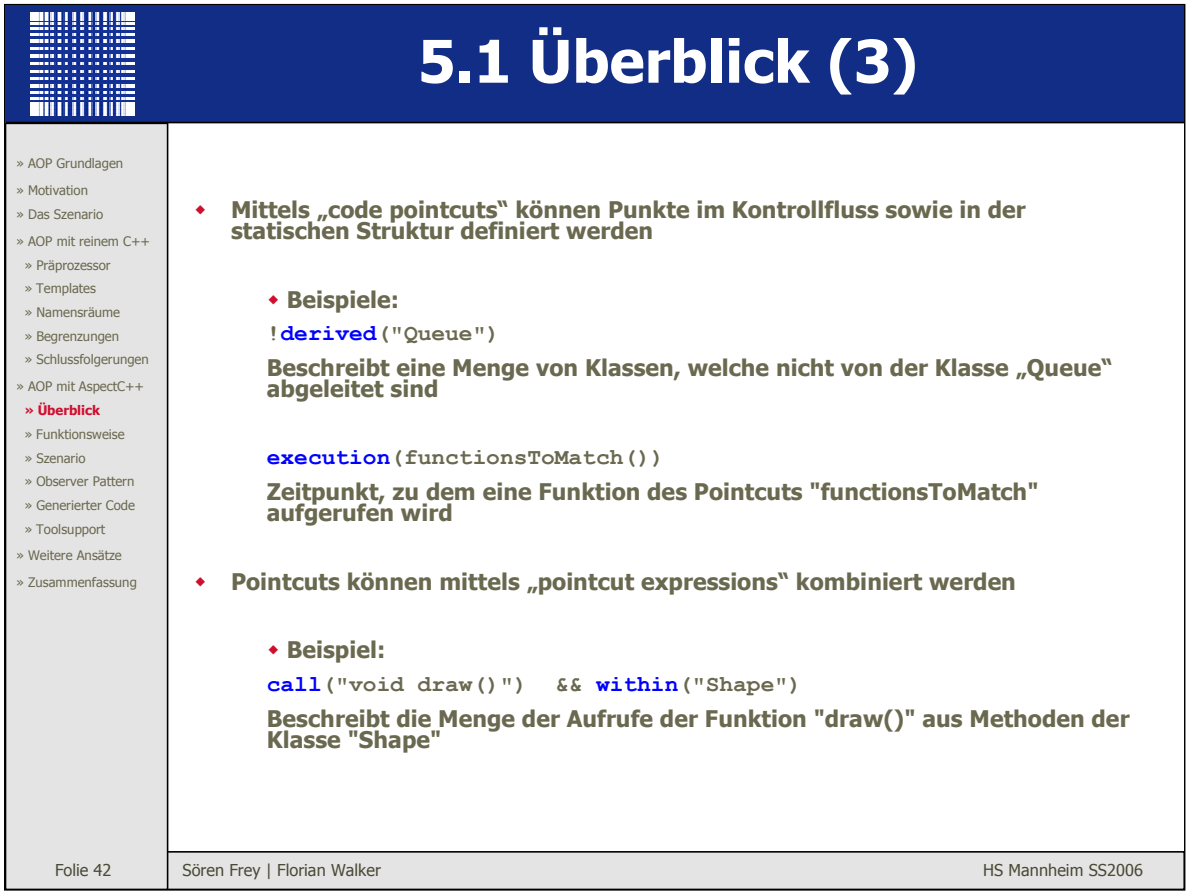
5. AOP mit AspectC++



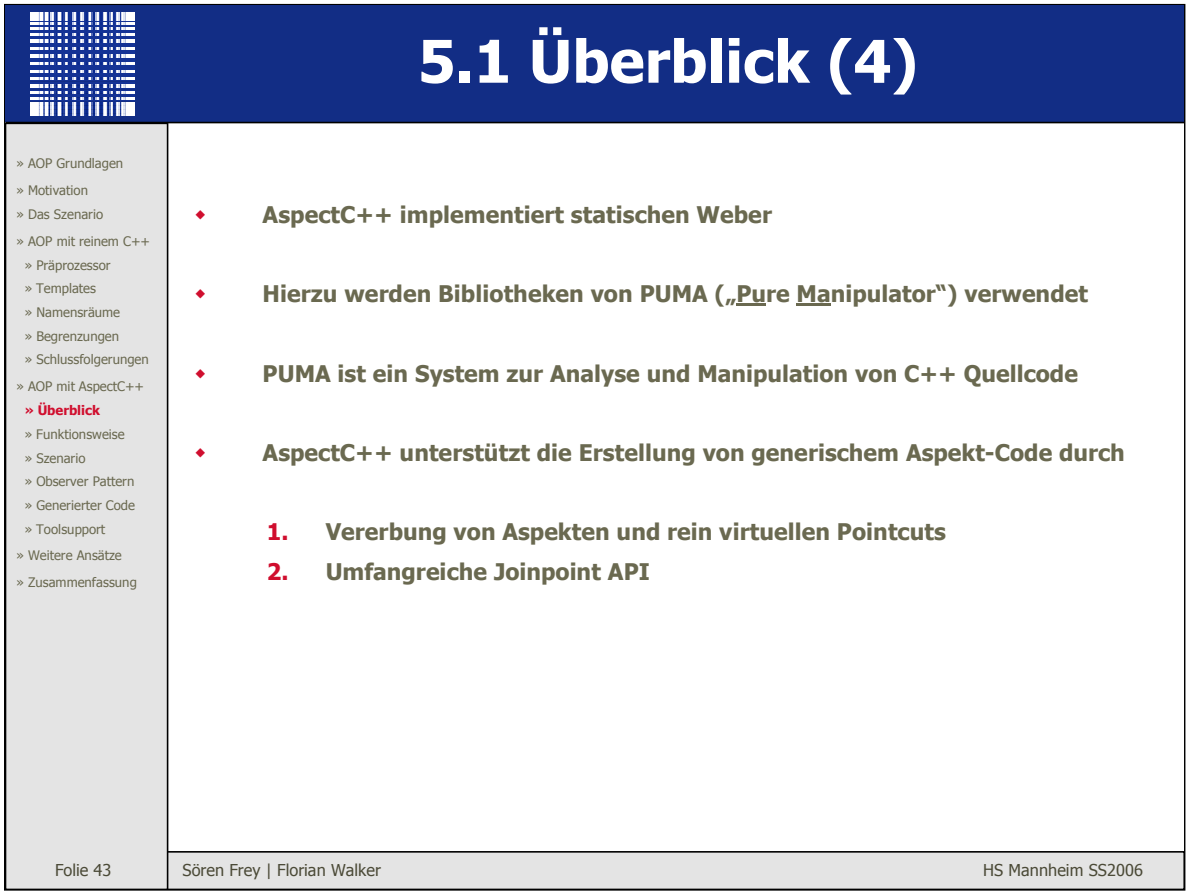
- Ursprünge an der Universität Magdeburg und der University of California, Irvine



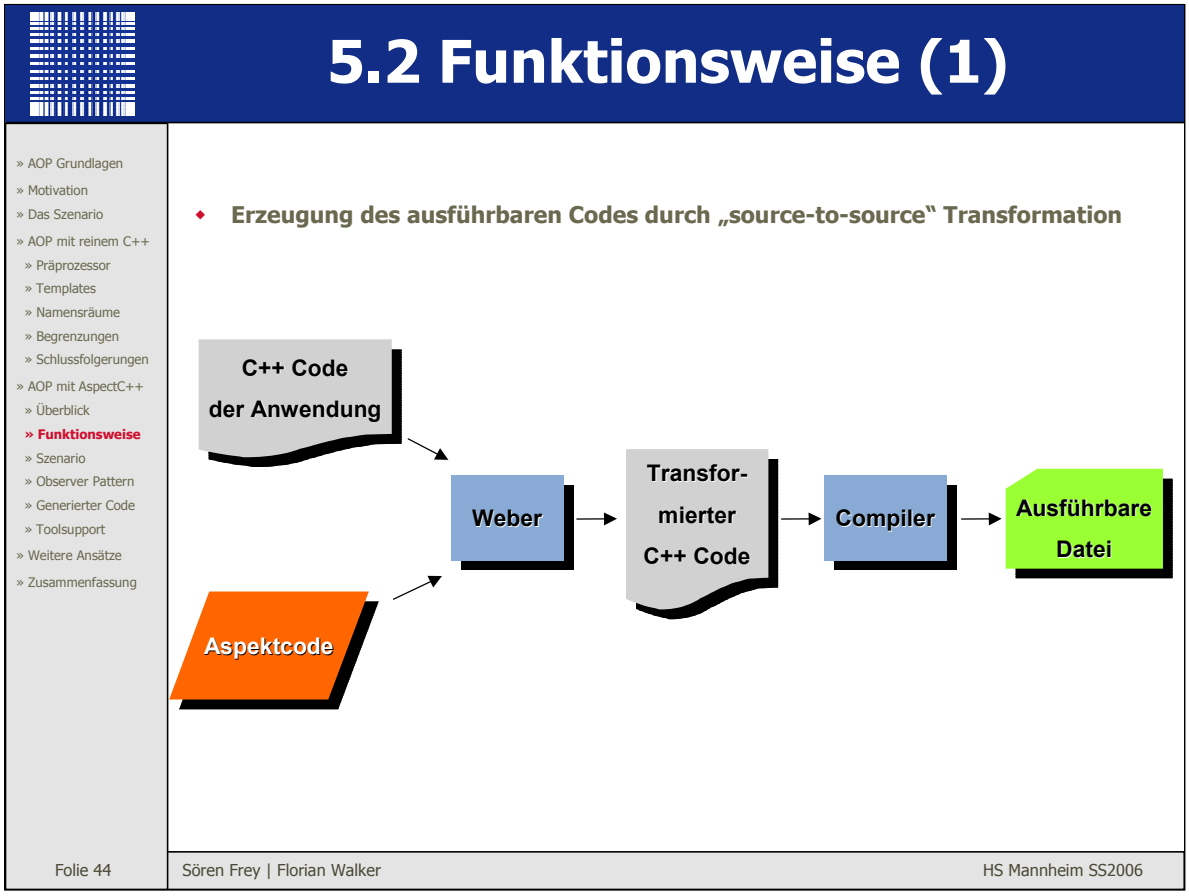
- In „matching expressions“ kann auch nach Templates und deren Ausprägungen gesucht werden
- Auch die Suche nach virtuellen Methoden ist möglich



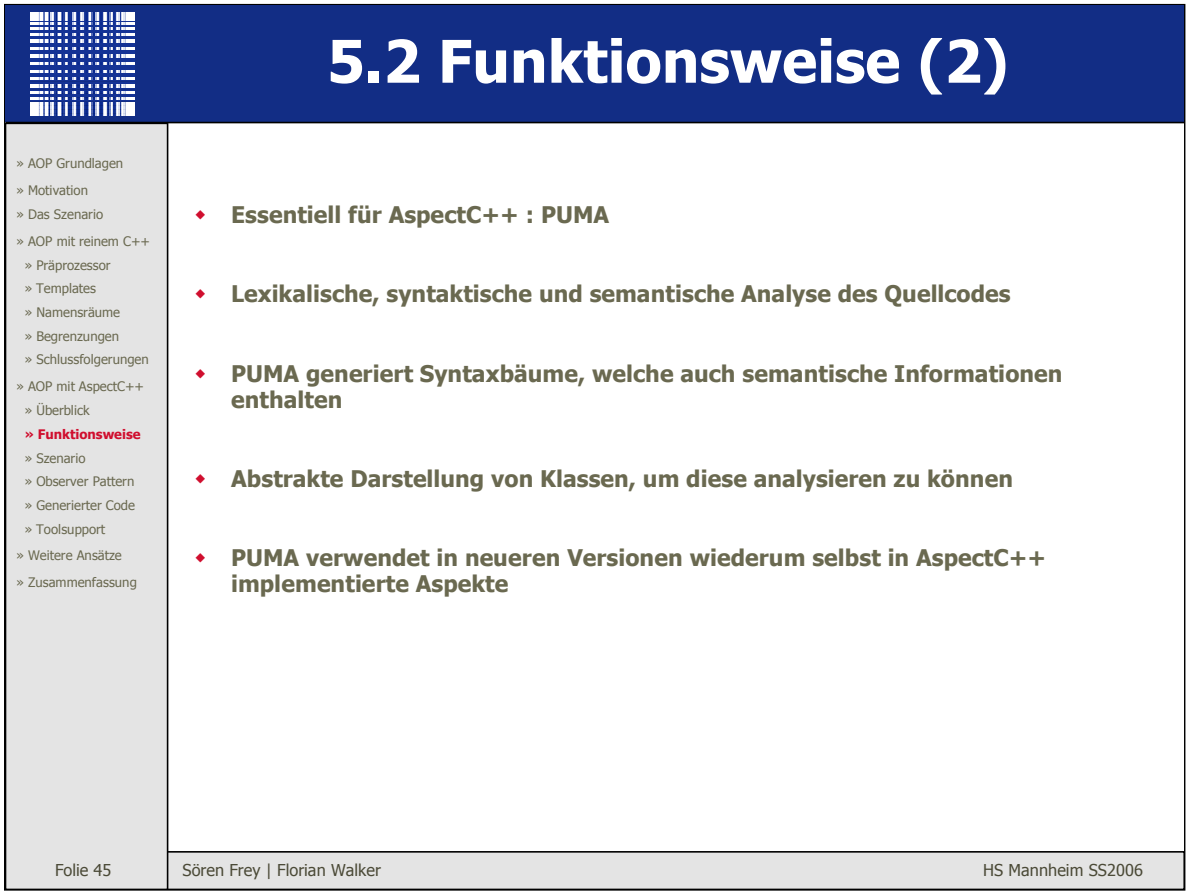
- Weitere Beispiele für zur Definition von „code pointcuts" wären die Pointcut Funktionen call(), construction(), destruction(), base()



- PUMA wurde an der Universität Magdeburg entwickelt



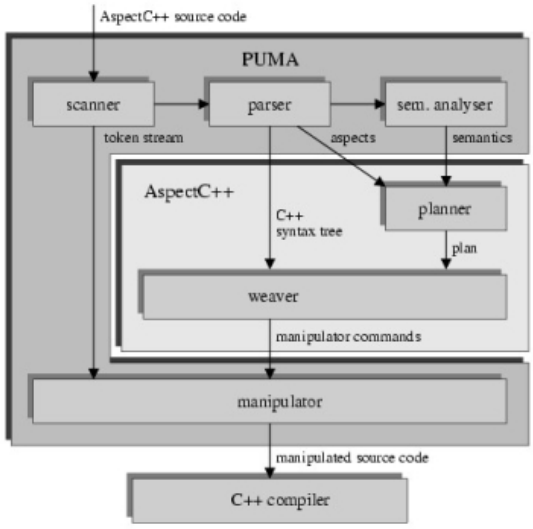
- Der Weber wird in diesem Fall durch einen Präprozessor implementiert



- Semantische Informationen betreffen beispielsweise die Sichtbarkeit von Variablen. Angenommen eine Klasse besitzt ein Attribut und Methoden, welche mit diesem Attribut arbeiten. Will man das Attribut umbenennen, so würde eine simple Textersetzung auch lokale Variablen mit dem selben Namen in den Methoden ersetzen. PUMA ermöglicht hierbei eine feinere Steuerung.

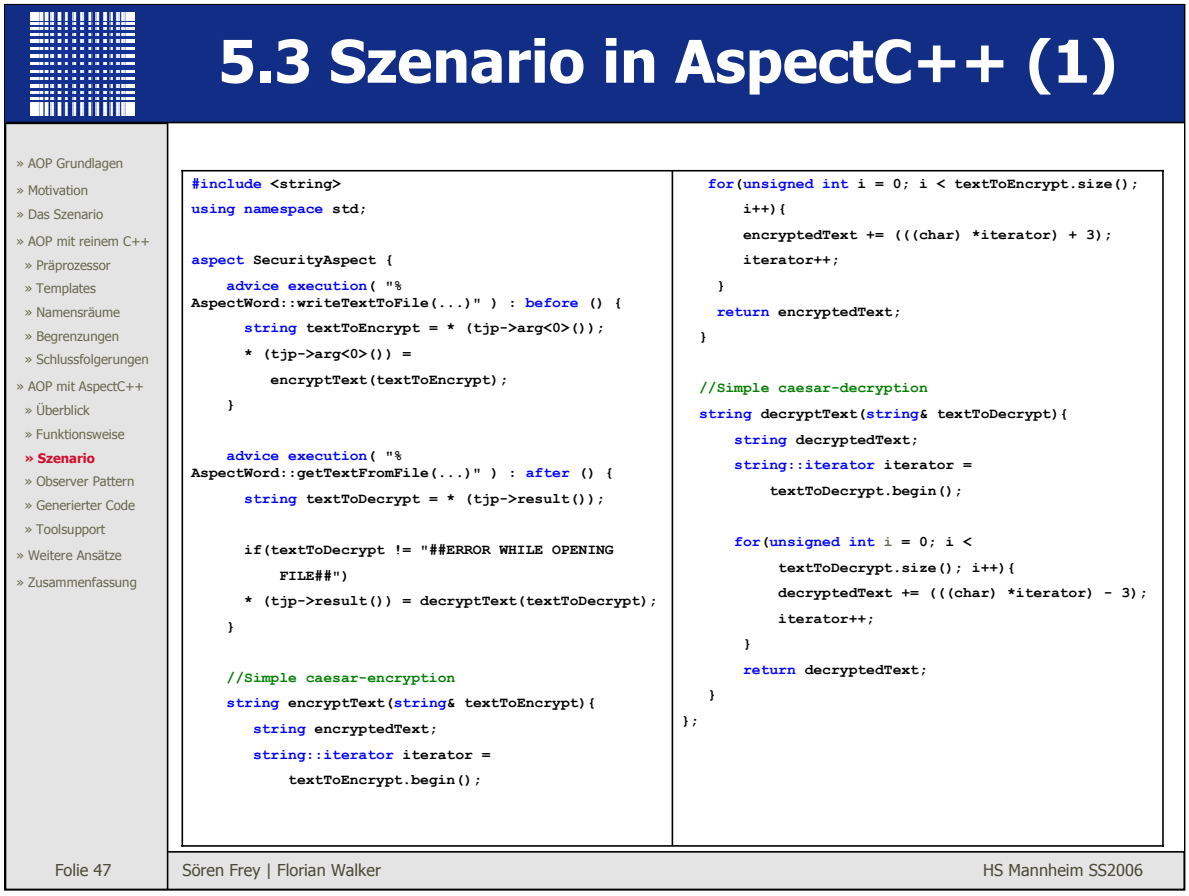
5.2 Funktionsweise (3)

- **Übersicht der Architektur**



Quelle: http://www.aspectc.org/fileadmin/publications/tools2002.ps.gz

- In der Planungsphase werden die „pointcut expressions" ausgewertet und die Joinpoint Mengen berechnet. Es wird ein Plan für den Weber erstellt, welcher die Joinpoints und die dazu auszuführenden Operationen enthält. Der Weber überführt den Plan in konkrete Manipulationskommandos, welche auf dem C++ Syntaxbaum von PUMA basieren. Die eigentliche Manipulation wird von der „manipulation engine" (manipulator) von PUMA ausgeführt. Hieraus resultiert reiner C++ Code, welcher die eingewobenen Aspekte enthält.



- Nachfolgend wird der Aspekt zur Definition der Sicherheitsfunktionalitäten beschrieben

5.3 Szenario in AspectC++ (2)

Definition eines neuen Aspektes

```
#include <string>
using namespace std;

aspect SecurityAspect {
    advice execution( "%
AspectWord::writeTextToFile(...)" ) : before () {
      string textToEncrypt = * (tjp->arg<0>());
      * (tjp->arg<0>()) =
         encryptText(textToEncrypt);
    }

    advice execution( "%
AspectWord::getTextFromFile(...)" ) : after () {
      string textToDecrypt = * (tjp->result());

      if(textToDecrypt != "##ERROR WHILE OPENING
          FILE##")
      * (tjp->result()) = decryptText(textToDecrypt);
    }

    //Simple caesar-encryption
    string encryptText(string& textToEncrypt){
       string encryptedText;
       string::iterator iterator =
           textToEncrypt.begin();
  for(unsigned int i = 0; i < textToEncrypt.size();
      i++){
      encryptedText += (((char) *iterator) + 3);
      iterator++;
   }
   return encryptedText;
 }

 //Simple caesar-decryption
 string decryptText(string& textToDecrypt){
     string decryptedText;
     string::iterator iterator =
         textToDecrypt.begin();

     for(unsigned int i = 0; i <
          textToDecrypt.size(); i++){
          decryptedText += (((char) *iterator) - 3);
          iterator++;
      }
      return decryptedText;
  }
};
```

5.3 Szenario in AspectC++ (3)


```
#include <string>
using namespace std;

aspect SecurityAspect {
    advice execution( "%
AspectWord::writeTextToFile(...)" ) : before () {
      string textToEncrypt = * (tjp->arg<0>());
      * (tjp->arg<0>()) =
         encryptText(textToEncrypt);
    }

    advice execution( "%
AspectWord::getTextFromFile(...)" ) : after () {
      string textToDecrypt = * (tjp->result());

      if(textToDecrypt != "##ERROR WHILE OPENING
          FILE##")
      * (tjp->result()) = decryptText(textToDecrypt);
    }

    //Simple caesar-encryption
    string encryptText(string& textToEncrypt){
       string encryptedText;
       string::iterator iterator =
           textToEncrypt.begin();
```

```
ncrypt.size();

) + 3);
}

 //Simple caesar-decryption
 string decryptText(string& textToDecrypt){
     string decryptedText;
     string::iterator iterator =
         textToDecrypt.begin();

     for(unsigned int i = 0; i <
          textToDecrypt.size(); i++){
          decryptedText += (((char) *iterator) - 3);
          iterator++;
      }
      return decryptedText;
  }
};
```

Advice, welches die Daten vor dem persistenten Schreiben verschlüsselt. Zugriff auf Daten des betreffenden Joinpoints über Variable „tjp". Manipulation des Arguments, mit welchem „writeTextToFile()" aufgerufen wird

5.3 Szenario in AspectC++ (4)

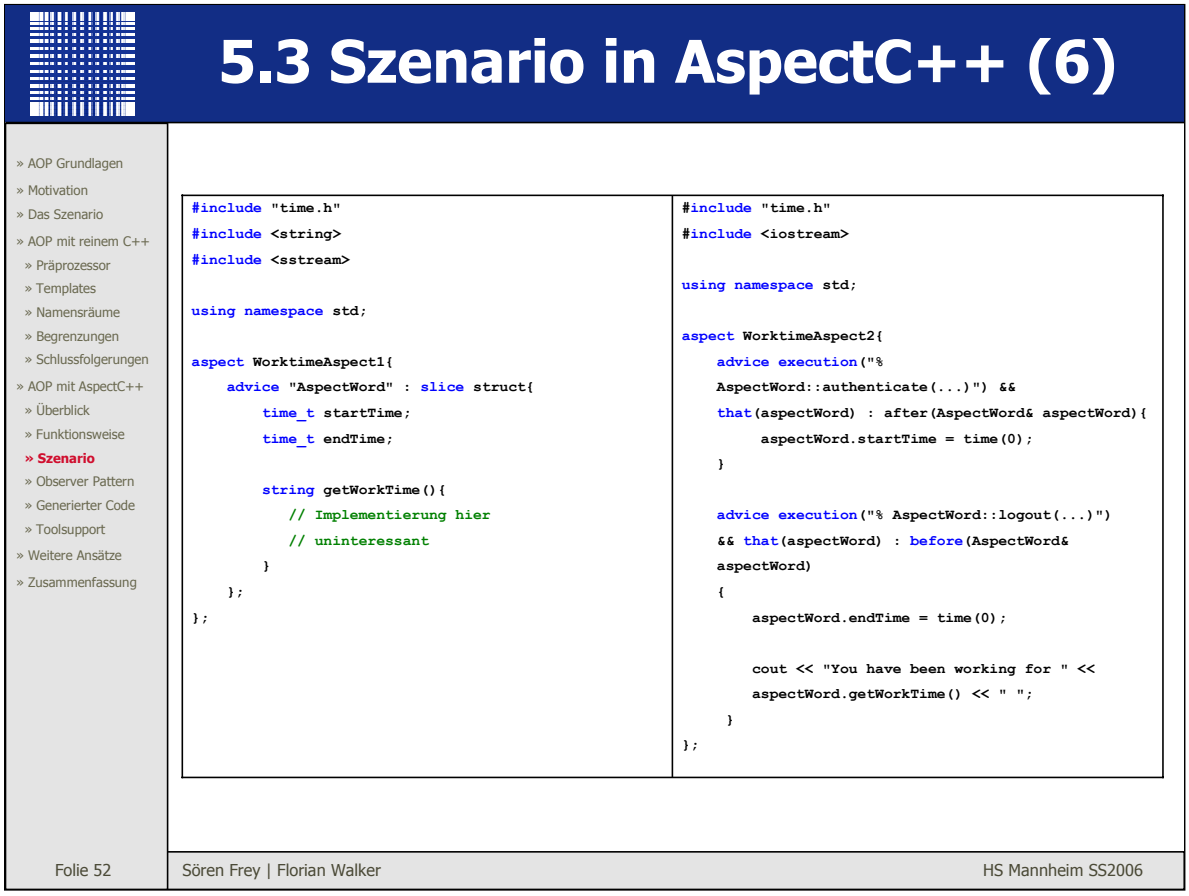

```
#include <string>
using namespace std;

aspect SecurityAspect {
    advice execution( "%
AspectWord::writeTextToFile(...)" ) : before () {
      string textToEncrypt = * (tjp->arg<0>());
      * (tjp->arg<0>()) =
         encryptText(textToEncrypt);
    }

    advice execution( "%
AspectWord::getTextFromFile(...)" ) : after () {
      string textToDecrypt = * (tjp->result());

      if(textToDecrypt != "##ERROR WHILE OPENING
          FILE##")
      * (tjp->result()) = decryptText(textToDecrypt);
    }

    //Simple caesar-encryption
    string encryptText(string& textToEncrypt){
       string encryptedText;
       string::iterator iterator =
           textToEncrypt.begin();
   for(unsigned int i = 0; i < textToEncrypt.size();
       i++){
       encryptedText += (((char) *iterator) + 3);
       iterator++;
    }
    return encryptedText;
  }

    for(unsigned int i = 0; i <
         textToDecrypt.size(); i++){
         decryptedText += (((char) *iterator) - 3);
         iterator++;
      }
      return decryptedText;
   }
};
```

Advice, welches die Daten nach dem Lesen von der Festplatte entschlüsselt. Der Rückgabewert der Funktion „getTextFromFile()" wird geändert, so dass der entschlüsselte Wert zurück gegeben wird

5.3 Szenario in AspectC++ (5)


```
#include <string>
using namespace std;

aspect SecurityAspect {
    advice execution( "%
AspectWo                          re () {
      st
      *
    }

    advice execution( "%
AspectWord::getTextFromFile(...)" ) : after () {
      string textToDecrypt = * (tjp->result());

      if(textToDecrypt != "##ERROR WHILE OPENING
          FILE##")
      * (tjp->result()) = decryptText(textToDecrypt);
    }

    //Simple caesar-encryption
    string encryptText(string& textToEncrypt){
       string encryptedText;
       string::iterator iterator =
           textToEncrypt.begin();
  for(unsigned int i = 0; i < textToEncrypt.size();
      i++){
      encryptedText += (((char) *iterator) + 3);
      iterator++;
   }
   return encryptedText;
 }

 //Simple caesar-decryption
 string decryptText(string& textToDecrypt){
     string decryptedText;
     string::iterator iterator =
         textToDecrypt.begin();

     for(unsigned int i = 0; i <
          textToDecrypt.size(); i++){
          decryptedText += (((char) *iterator) - 3);
          iterator++;
      }
      return decryptedText;
  }
};
```

Einfache Funktionen zum Ver- und Entschlüsseln der Daten mittels Cäsar-Code



- Nachfolgend werden die Aspekte zur Erfassung der Arbeitszeit beschrieben

5.3 Szenario in AspectC++ (7)

Bug in AspectC++ erzwingt Aufteilung in 2 Aspekte für Worktime-Komponente

```
#include "time.h"
#include <string>
#include <sstream>

using namespace std;

aspect WorktimeAspect1{
    advice "AspectWord" : slice struct{
        time_t startTime;
        time_t endTime;

        string getWorkTime(){
           // Implementierung hier
           // uninteressant
        }
    };
};
```

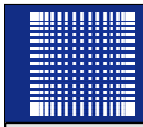
```
#include "time.h"
#include <iostream>

using namespace std;

aspect WorktimeAspect2{
    advice execution("%
    AspectWord::authenticate(...)") &&
    that(aspectWord) : after(AspectWord& aspectWord){
         aspectWord.startTime = time(0);
    }

    advice execution("% AspectWord::logout(...)")
    && that(aspectWord) : before(AspectWord&
    aspectWord)
    {
        aspectWord.endTime = time(0);

        cout << "You have been working for " <<
        aspectWord.getWorkTime() << " ";
     }
};
```

5.3 Szenario in AspectC++ (8)


```
#include "time.h"
#include <string>
#include <sstream>

using namespace std;

aspect WorktimeAspect1{
    advice "AspectWord" : slice struct{
        time_t startTime;
        time_t endTime;

        string getWorkTime(){
           // Implementierung hier
           // uninteressant
        }
    };
};
```

```
#include "time.h"
#include <iostream>

using namespace std;

                                    &
                                rd& aspectWord){
        aspectWord.startTime = time(0);
    }

    advice execution("% AspectWord::logout(...)")
    && that(aspectWord) : before(AspectWord&
    aspectWord)
    {
        aspectWord.endTime = time(0);

        cout << "You have been working for " <<
        aspectWord.getWorkTime() << " ";
     }
};
```

Introduction von 2 Variablen und einer Methode in Applikation. Definition der Variablen im Aspekt wäre semantisch nicht gleichwertig !



5.3 Szenario in AspectC++ (9)


```
#include "time.h"
#include <string>
#include <sstream>

using

aspect
    ad
        time_t endTime;

        string getWorkTime(){
           // Implementierung hier
           // uninteressant
        }
    };
};
```

Advice zum Starten der Zeitmessung. Eine Kontextvariable „aspectWord" wird an „that()" (die aufrufende Instanz) gebunden. Hierüber kann im Advice auf die neu eingeführte Variable zugegriffen werden

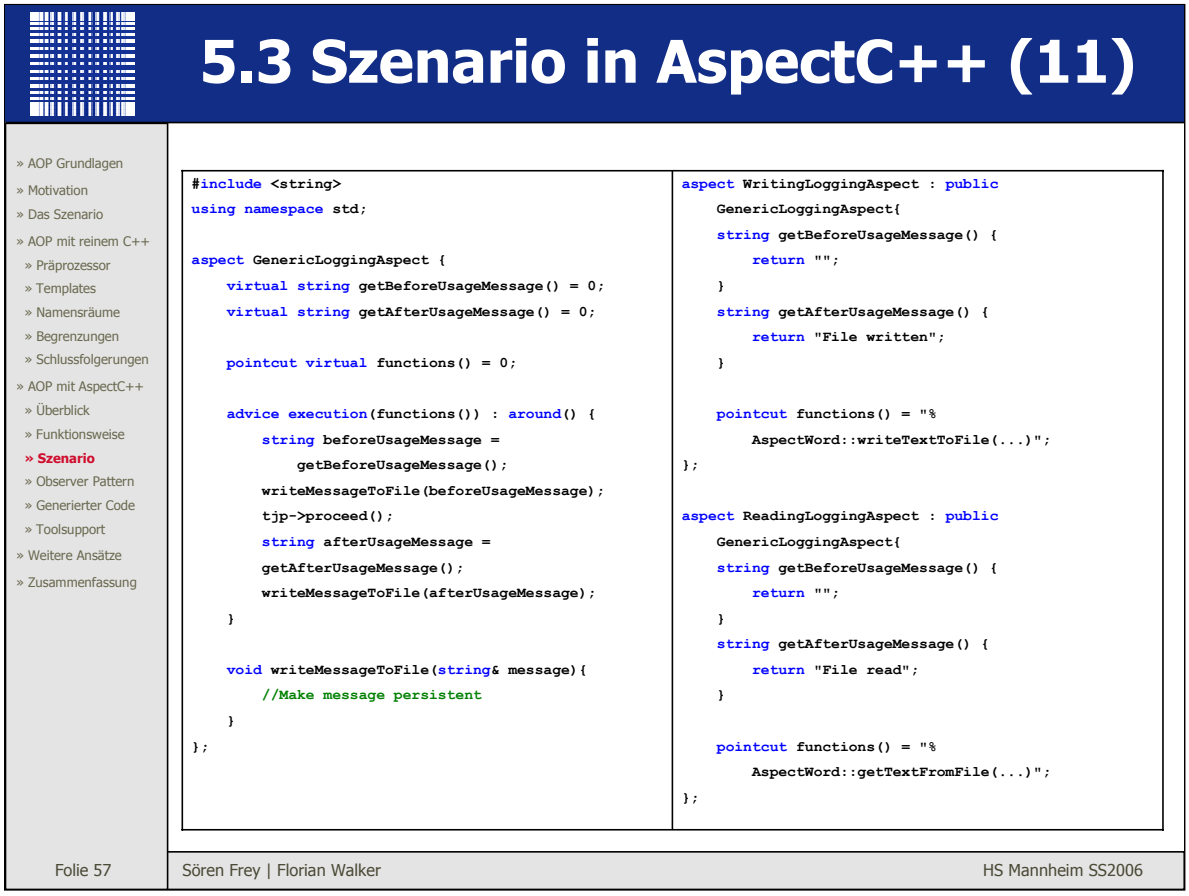
```
#include "time.h"
#include <iostream>

using namespace std;

aspect WorktimeAspect2{
    advice execution("%
    AspectWord::authenticate(...)") &&
    that(aspectWord) : after(AspectWord& aspectWord){
         aspectWord.startTime = time(0);
    }

    advice execution("% AspectWord::logout(...)")
    && that(aspectWord) : before(AspectWord&
    aspectWord)
    {
        aspectWord.endTime = time(0);

        cout << "You have been working for " <<
        aspectWord.getWorkTime() << " ";
     }
};
```

5.3 Szenario in AspectC++ (10)


```
#include "time.h"
#include <string>
#include <sstream>

using namespace std;

aspect WorktimeAspect1{
    advice "AspectWord" : slice struct{
        time_t startTime;
        time_t endTime;
    }
};
```

Advice zum Beenden der Zeitmessung und Ausgabe der geleisteten Arbeitszeit. Wiederum wird über eine Kontextvariable auf die per Introduction neu eingeführten Klassenmember zugegriffen

```
#include "time.h"
#include <iostream>

using namespace std;

aspect WorktimeAspect2{
    advice execution("%
    AspectWord::authenticate(...)") &&
    that(aspectWord) : after(AspectWord& aspectWord){
         aspectWord.startTime = time(0);
    }

    advice execution("% AspectWord::logout(...)")
    && that(aspectWord) : before(AspectWord&
    aspectWord)
    {
        aspectWord.endTime = time(0);

        cout << "You have been working for " <<
        aspectWord.getWorkTime() << " ";
     }
};
```



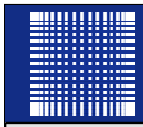
- Nachfolgend wird der generische Aspekt zum Loggen von Daten und dessen reale Ausgestaltungen beschrieben

5.3 Szenario in AspectC++ (12)


```
#include <string>
using namespace std;

aspect GenericLoggingAspect {
    virtual string getBeforeUsageMessage() = 0;
    virtual string getAfterUsageMessage() = 0;

    pointcut virtual functions

    advice execution(functions
        string beforeUsageMessage =
            getBeforeUsageMessage();
        writeMessageToFile(beforeUsageMessage);
        tjp->proceed();
        string afterUsageMessage =
        getAfterUsageMessage();
        writeMessageToFile(afterUsageMessage);
    }

    void writeMessageToFile(string& message){
        //Make message persistent
    }
};
```

Definition eines generischen Aspektes. Hiervon werden 2 konkrete Aspekte abgeleitet

```
aspect WritingLoggingAspect : public
    GenericLoggingAspect{
    string getBeforeUsageMessage() {
        return "";
    }
    string getAfterUsageMessage() {
        return "File written";
    }

    nctions() = "%
        Word::writeTextToFile(...)";
};

aspect ReadingLoggingAspect : public
    GenericLoggingAspect{
    string getBeforeUsageMessage() {
        return "";
    }
    string getAfterUsageMessage() {
        return "File read";
    }

    pointcut functions() = "%
        AspectWord::getTextFromFile(...)";
};
```



5.3 Szenario in AspectC++ (13)


```
#include <string>
using namespace std;

aspect GenericLoggingAspect {
    virtual string getBeforeUsageMessage() = 0;
    virtual string getAfterUsageMessage() = 0;

    pointcut virtual functions() = 0;

    advice execution(functions()) : around() {
        string beforeUsageMessage =
            getBeforeUsageMessage();
        writeMessageToFile(beforeUsageMessage);
        tjp->proceed();
        string afterUsageMessage =
        getAfterUsageMessage();
        writeMessageToFile(afterUsageMessage);
    }

    void writeMessageToFile(string& message){
        //Make message persistent
    }
};
```

```
aspect WritingLoggingAspect : public
    GenericLoggingAspect{
        return "File written";
    }

    pointcut functions() = "%
        AspectWord::writeTextToFile(...)";
};

aspect ReadingLoggingAspect : public
    GenericLoggingAspect{
    string getBeforeUsageMessage() {
        return "";
    }
    string getAfterUsageMessage() {
        return "File read";
    }

    pointcut functions() = "%
        AspectWord::getTextFromFile(...)";
};
```

Der Aspekt erhält durch rein virtuelle Methoden und einen rein virtuellen Pointcut eine generische Ausgestaltung

5.3 Szenario in AspectC++ (14)


```
#include <string>
using namespace std;

aspect GenericLoggingAspect {
    virtual string getBeforeUsageMessage() = 0;
    virtual string getAfterUsageMessage() = 0;

    pointcut virtual functions() = 0;

    advice execution(functions()) : around() {
        string beforeUsageMessage =
            getBeforeUsageMessage();
        writeMessageToFile(beforeUsageMessage);
        tjp->proceed();
        string afterUsageMessage =
        getAfterUsageMessage();
        writeMessageToFile(afterUsageMessage);
    }

    void writeMessageToFile(string& message){
        //Make message persistent
    }
};
```

```
aspect WritingLoggingAspect : public
    GenericLoggingAspect{
    string getBeforeUsageMessage() {
        return "";
    }
    string getAfterUsageMessage() {
        return "File written";
    }

    GenericLoggingAspect{
    string getBeforeUsageMessage() {
        return "";
    }
    string getAfterUsageMessage() {
        return "File read";
    }

    pointcut functions() = "%
        AspectWord::getTextFromFile(...)";
};
```

Im „around-Advice" des generischen Aspektes werden die jeweiligen Meldungen (vor und nach einem Methoden-Aufruf) ausgegeben. Durch „tjp->proceed()" wird der eigentliche Methodenrumpf ausgeführt

5.3 Szenario in AspectC++ (15)


```
#include <string>
using namespace std;

aspect GenericLoggingAspect {
    virtual string getBeforeUsageMessage() = 0;
    virtual string getAfterUsageMessage() = 0;

    pointcut virtual functions() = 0;

    a
        string afterUsageMessage =
        getAfterUsageMessage();
        writeMessageToFile(afterUsageMessage);
    }

    void writeMessageToFile(string& message){
        //Make message persistent
    }
};
```

In den konkreten Aspekten werden die rein virtuellen Methoden überschrieben. Somit wird festgelegt, was bei Schreib- und Lesevorgängen jeweils vor und nach dem Methodenaufruf ausgegeben werden soll

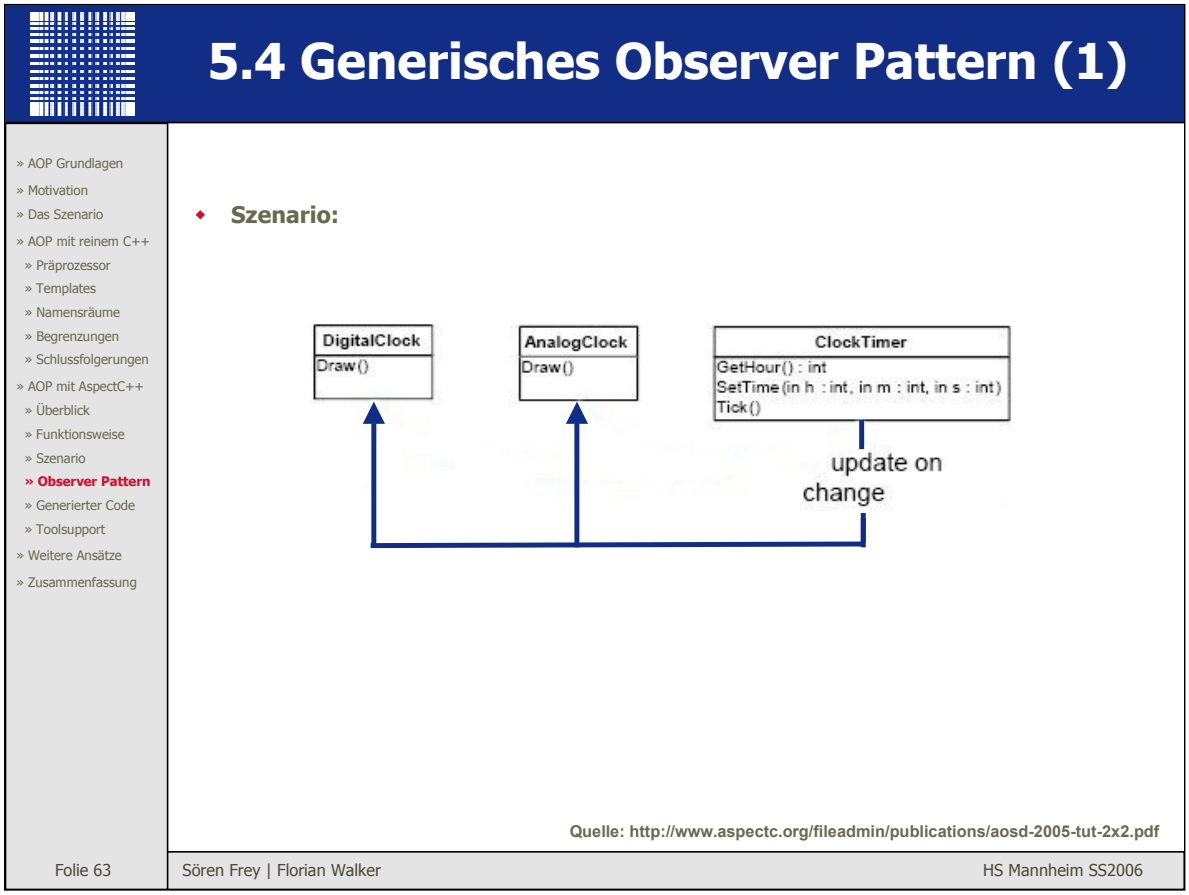
```
aspect WritingLoggingAspect : public
    GenericLoggingAspect{
    string getBeforeUsageMessage() {
        return "";
    }
    string getAfterUsageMessage() {
        return "File written";
    }

    pointcut functions() = "%
        AspectWord::writeTextToFile(...)";
};

aspect ReadingLoggingAspect : public
    GenericLoggingAspect{
    string getBeforeUsageMessage() {
        return "";
    }
    string getAfterUsageMessage() {
        return "File read";
    }

    pointcut functions() = "%
        AspectWord::getTextFromFile(...)";
};
```

5.3 Szenario in AspectC++ (16)


```
#include <string>
using namespace std;

aspect GenericLoggingAspect {
    virtual string getBeforeUsageMessage() = 0;
    virtual string getAfterUsageMessage() = 0;

    pointcut virtual functions() = 0;

    advice execution(functions()) : around() {
        string beforeUsageMessage =
    }

    void writeMessageToFile(string& message){
        //Make message persistent
    }
};
```

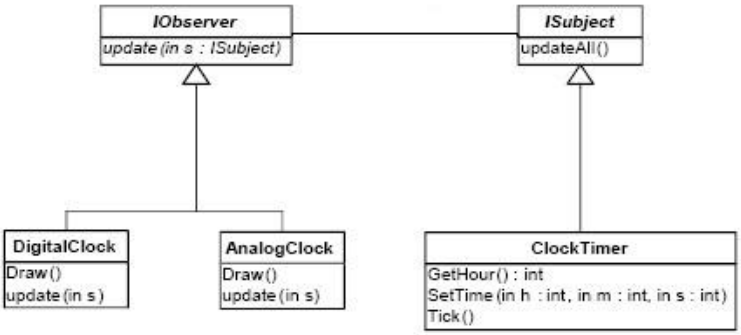
Durch die Realisierung des formals rein virtuellen Pointcuts in den konkreten Aspekten wird festgelegt, bei welchen Methoden der Logging-Vorgang durchgeführt werden soll

```
aspect WritingLoggingAspect : public
    GenericLoggingAspect{
    string getBeforeUsageMessage() {
        return "";
    }
    string getAfterUsageMessage() {
        return "File written";
    }

    pointcut functions() = "%
        AspectWord::writeTextToFile(...)";
};

aspect ReadingLoggingAspect : public
    GenericLoggingAspect{
    string getBeforeUsageMessage() {
        return "";
    }
    string getAfterUsageMessage() {
        return "File read";
    }

    pointcut functions() = "%
        AspectWord::getTextFromFile(...)";
};
```



- Das Beispiel zeigt, dass auch bei der Implementierung von Design Patterns durch AOP eine weitergehende Modularisierung und eine „separation of concerns" möglich ist

5.4 Generisches Observer Pattern (2)

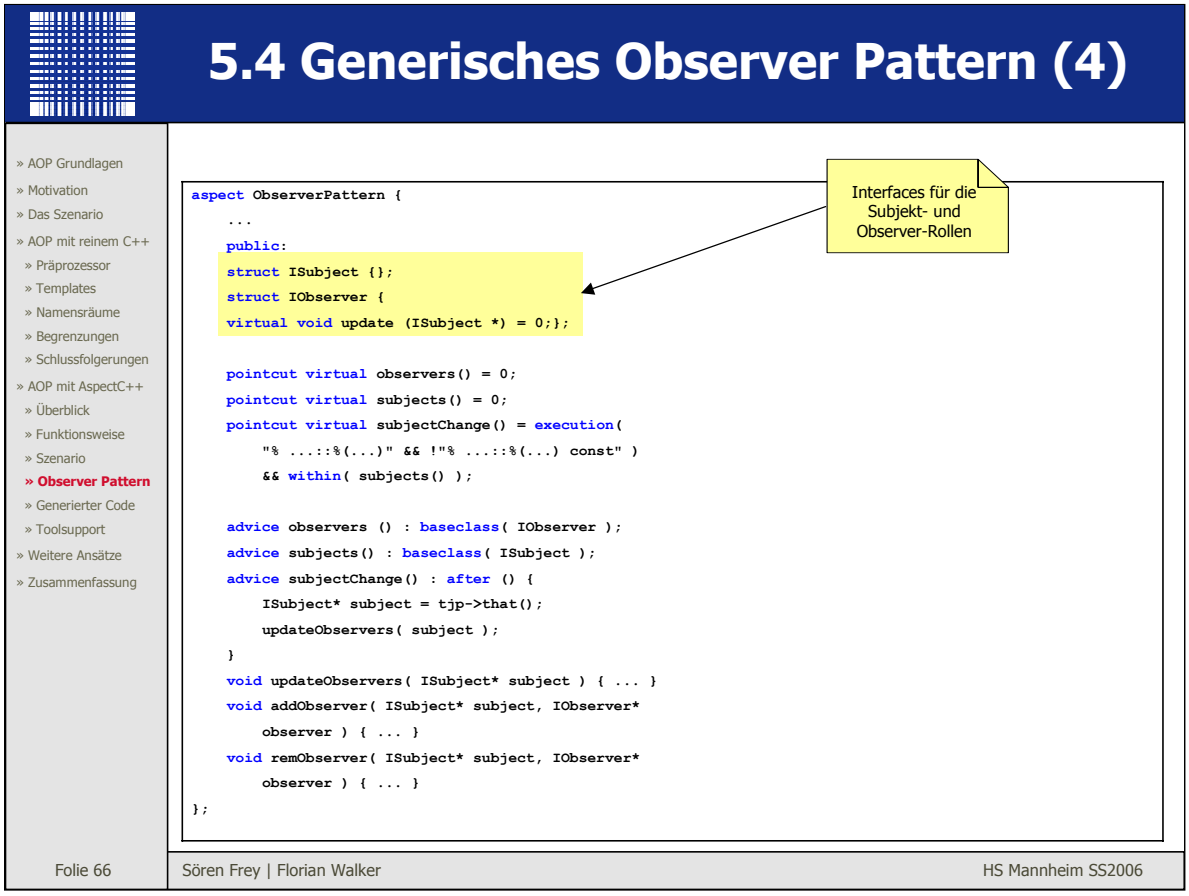


Quelle: http://www.aspectc.org/fileadmin/publications/aosd-2005-tut-2x2.pdf

- Ein Beobachter möchte über Geschehnisse in einer anderen Klasse in Kenntnis gesetzt werden, um darauf angemessen reagieren zu können. Hierzu kann er sich bei dem so genannten „Subject" registrieren und deregistrieren. Im Beispiel soll ein Beobachter die Methode „Draw()" aufrufen, wenn er ein Update-Ereignis erhält.

5.4 Generisches Observer Pattern (3)


```
aspect ObserverPattern {
    ...
    public:
    struct ISubject {};
    struct IObserver {
    virtual void update (ISubject *) = 0;};

    pointcut virtual observers() = 0;
    pointcut virtual subjects() = 0;
    pointcut virtual subjectChange() = execution(
        "% ...::%(...)" && !"% ...::%(...) const" )
        && within( subjects() );

    advice observers () : baseclass( IObserver );
    advice subjects() : baseclass( ISubject );
    advice subjectChange() : after () {
        ISubject* subject = tjp->that();
        updateObservers( subject );
    }
    void updateObservers( ISubject* subject ) { ... }
    void addObserver( ISubject* subject, IObserver*
        observer ) { ... }
    void remObserver( ISubject* subject, IObserver*
        observer ) { ... }
};
```

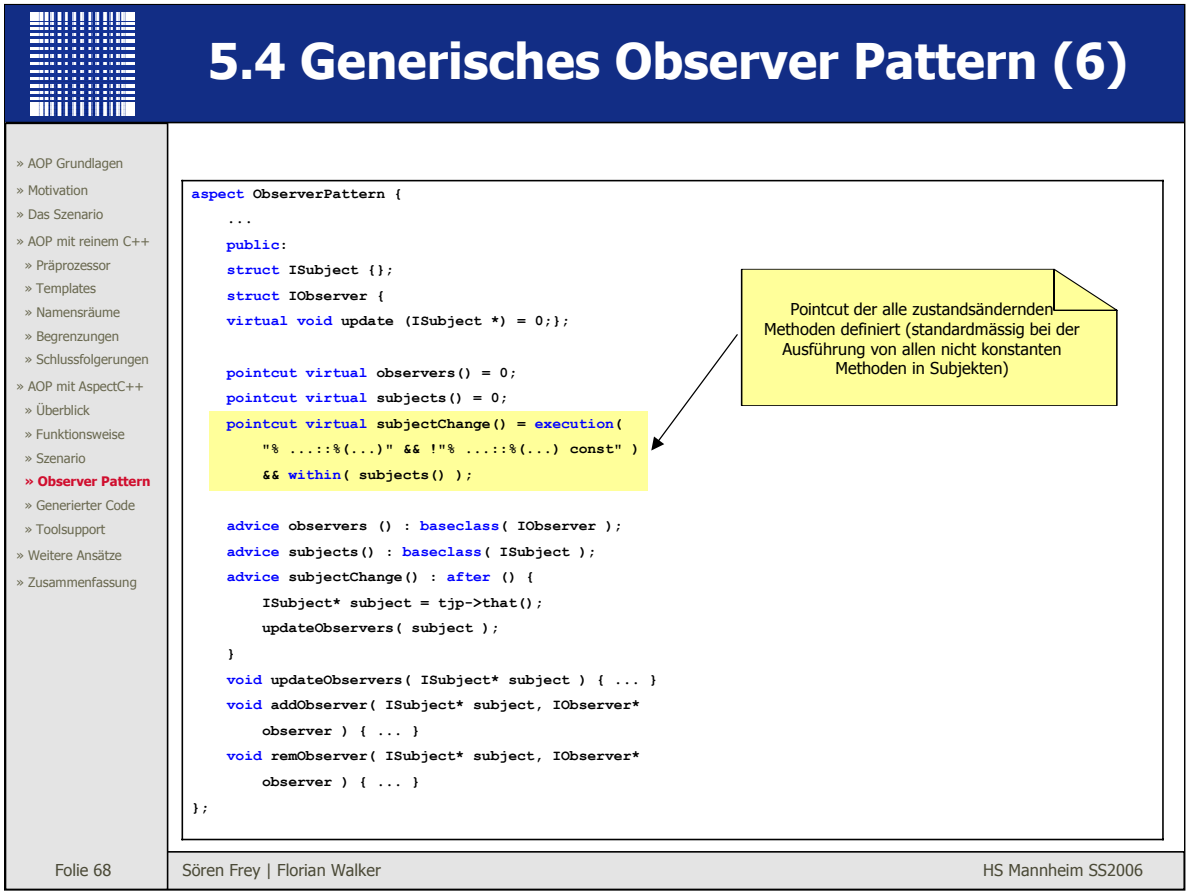


- Jeder Beobachter soll eine update-Methode erhalten

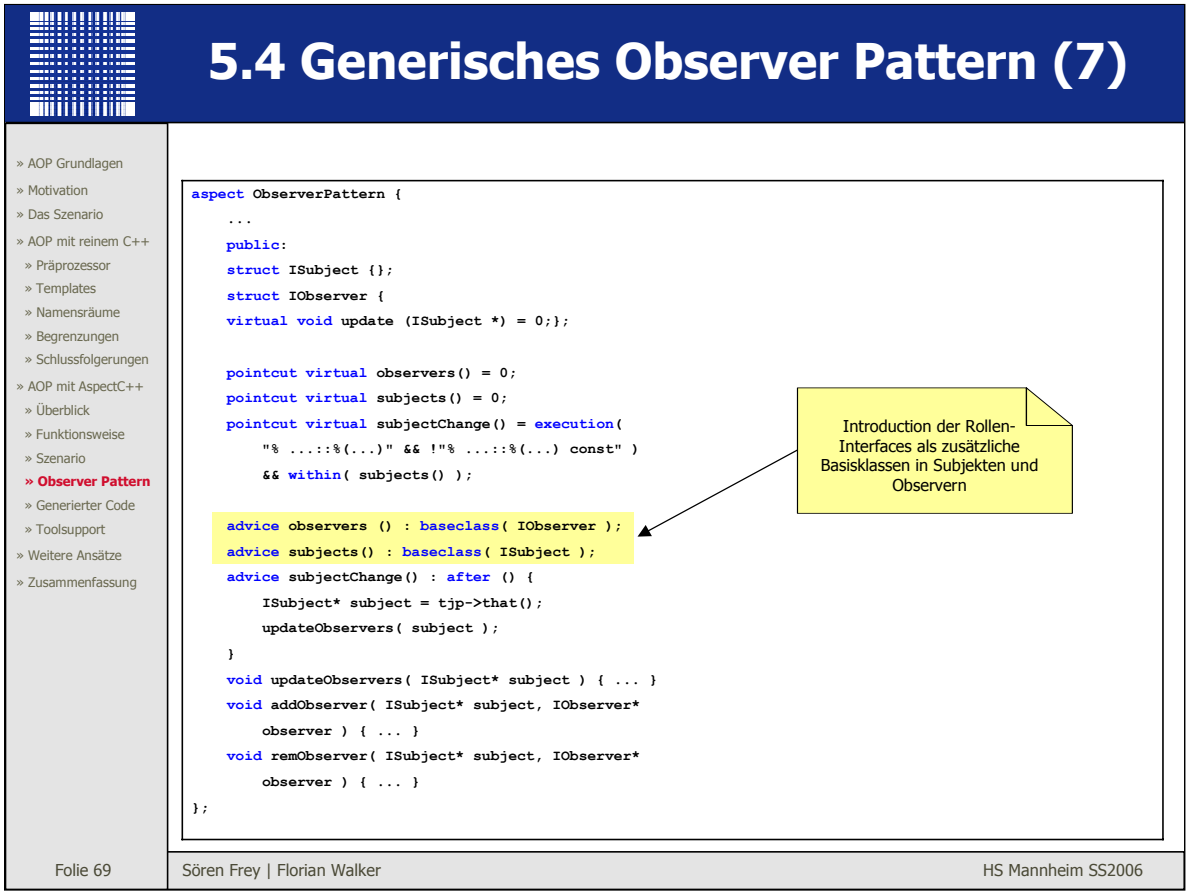
5.4 Generisches Observer Pattern (5)


```
aspect ObserverPattern {
    ...
    public:
    struct ISubject {};
    struct IObserver {
    virtual void update (ISubject *) = 0;};

    pointcut virtual observers() = 0;
    pointcut virtual subjects() = 0;
    pointcut virtual subjectChange() = execution(
        "% ...::%(...)" && !"% ...::%(...) const" )
        && within( subjects() );

    advice observers () : baseclass( IObserver );
    advice subjects() : baseclass( ISubject );
    advice subjectChange() : after () {
        ISubject* subject = tjp->that();
        updateObservers( subject );
    }
    void updateObservers( ISubject* subject ) { ... }
    void addObserver( ISubject* subject, IObserver*
        observer ) { ... }
    void remObserver( ISubject* subject, IObserver*
        observer ) { ... }
};
```

Abstrakte Pointcuts, welche Subjekte und Observer definieren (müssen überschrieben werden)



- Selbstverständlich könnte auch eine spezifischere Festlegung auf bestimmte Funktionen erfolgen, die „matching expressions“ sind hierbei mächtige Werkzeuge

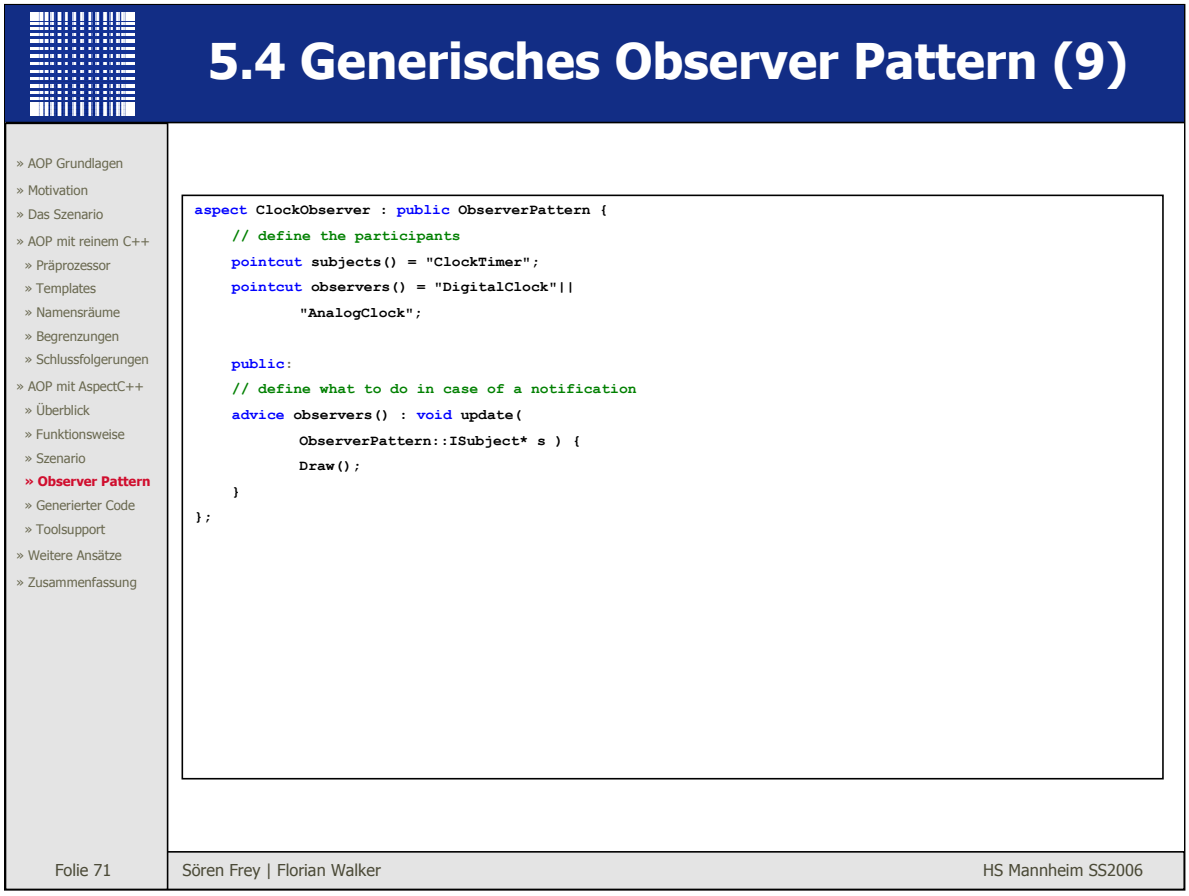


- Matcht eine Klasse mit den angegeben Subjekten oder Beobachtern, so wird ihr automatisch über Vererbung die jeweilige Rolle zugewiesen

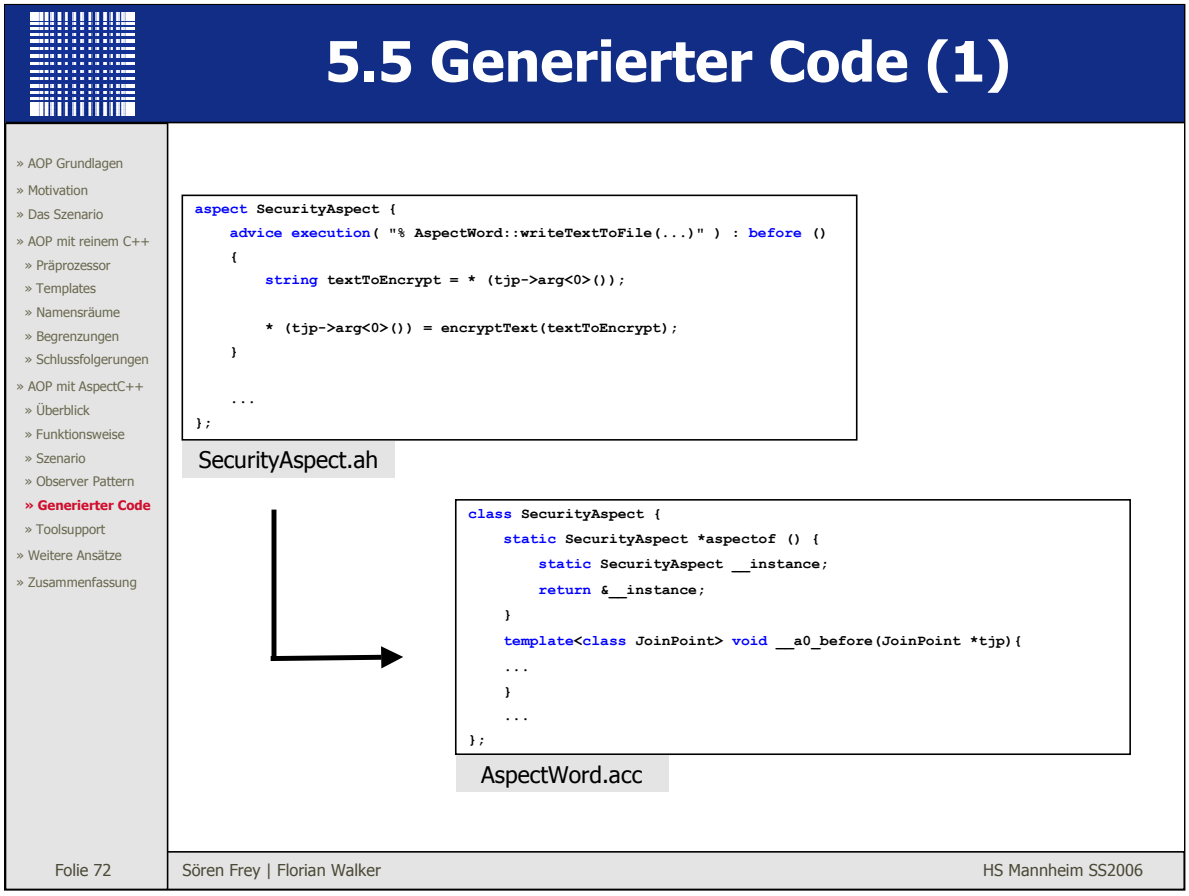
5.4 Generisches Observer Pattern (8)


```
aspect ObserverPattern {
    ...
    public:
    struct ISubject {};
    struct IObserver {
    virtual void update (ISubject *) = 0;};

    pointcut virtual observers() = 0;
    pointcut virtual subjects() = 0;
    pointcut virtual subjectChange() = execution(
        "% ...::%(...)" && !"% ...::%(...) const" )
        && within( subjects() );

    advice observers () : baseclass( IObserver );
    advice subjects() : baseclass( ISubject );
    advice subjectChange() : after () {
        ISubject* subject = tjp->that();
        updateObservers( subject );
    }
    void updateObservers( ISubject* subject ) { ... }
    void addObserver( ISubject* subject, IObserver*
        observer ) { ... }
    void remObserver( ISubject* subject, IObserver*
        observer ) { ... }
};
```

Advice, der die Observer nach dem Ausführen einer zustandsändernden Methode updated



- Zur Anwendung des Observer Patterns reicht es nun aus, lediglich die gewünschten Subjekte und Beobachter mittels Pointcuts zu spezifizieren und den update-Advice zu überschreiben



- Der Beispielcode stellt lediglich einen kleinen Ausschnitt aus dem unfangreichen generierten Code dar

5.5 Generierter Code (2)


```
aspect SecurityAspect
    advice execution( "% AspectWord::writeTextToFile(...)" ) : before ()
    {
        string textToEncrypt = * (tjp->arg<0>());

        * (tjp->arg<0>()) = encryptText(textToEncrypt);
    }

    ...
};
```

SecurityAspect.ah

Aspekte werden zu Klassen transformiert

```
class SecurityAspect {
    static SecurityAspect *aspectof () {
        static SecurityAspect __instance;
        return &__instance;
    }
    template<class JoinPoint> void __a0_before(JoinPoint *tjp){
    ...
    }
    ...
};
```

AspectWord.acc

5.5 Generierter Code (3)


```
aspect SecurityAspect {
    advice execution( "% AspectWord::writeTextToFile(...)" ) : before ()
    {
        string textToEncrypt = * (tjp->arg<0>());

        * (tjp->arg<0>()) = encryptText(textToEncrypt);
    }

    ...
};
```

SecurityAspect.ah

Eine globale Instanz des Aspektes wird standardmässig erstellt

```
class SecurityAspect {
    static SecurityAspect *aspectof () {
        static SecurityAspect __instance;
        return &__instance;
    }
    template<class JoinPoint> void __a0_before(JoinPoint *tjp){
    ...
    }
    ...
};
```

AspectWord.acc

5.5 Generierter Code (4)

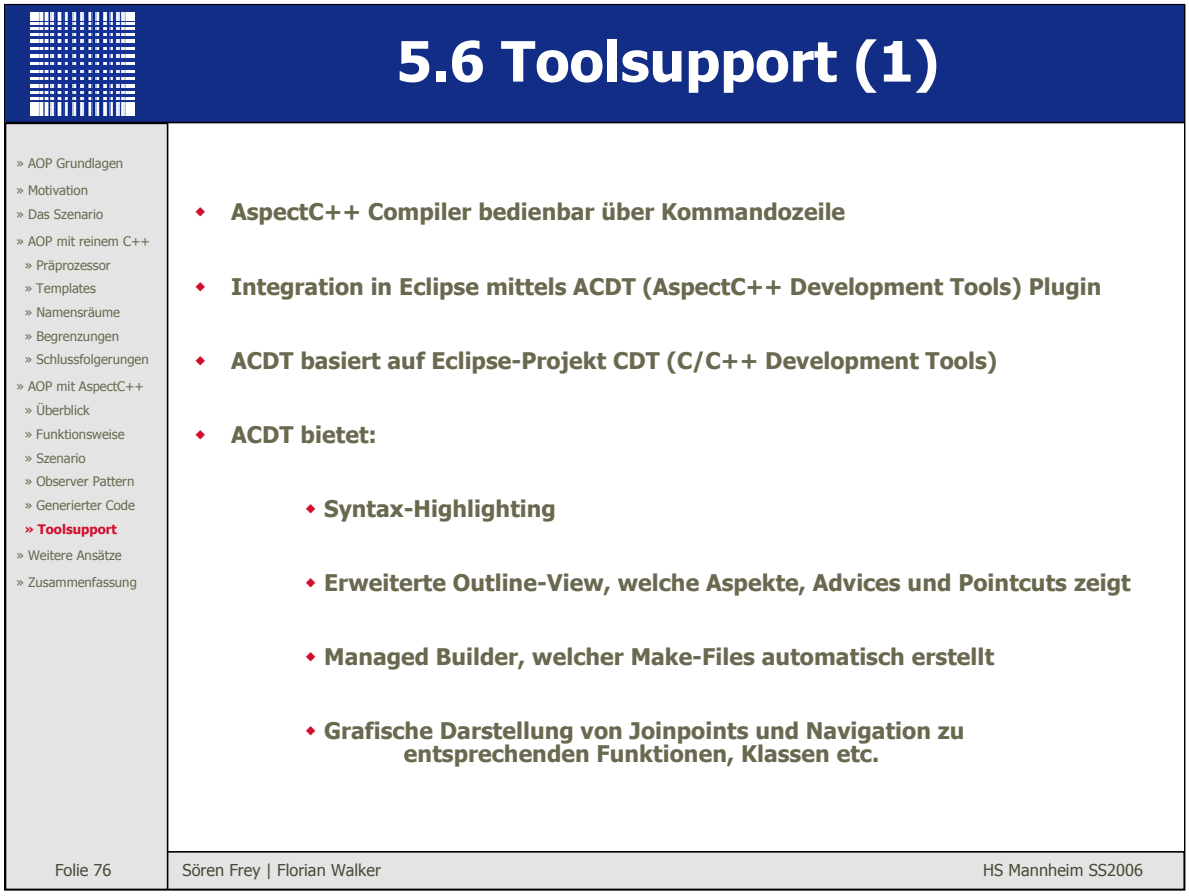

```
aspect SecurityAspect {
    advice execution( "% AspectWord::writeTextToFile(...)" ) : before ()
    {
        string textToEncrypt = * (tjp->arg<0>());

        * (tjp->arg<0>()) = encryptText(textToEncrypt);
    }

    ...
};
```

SecurityAspect.ah

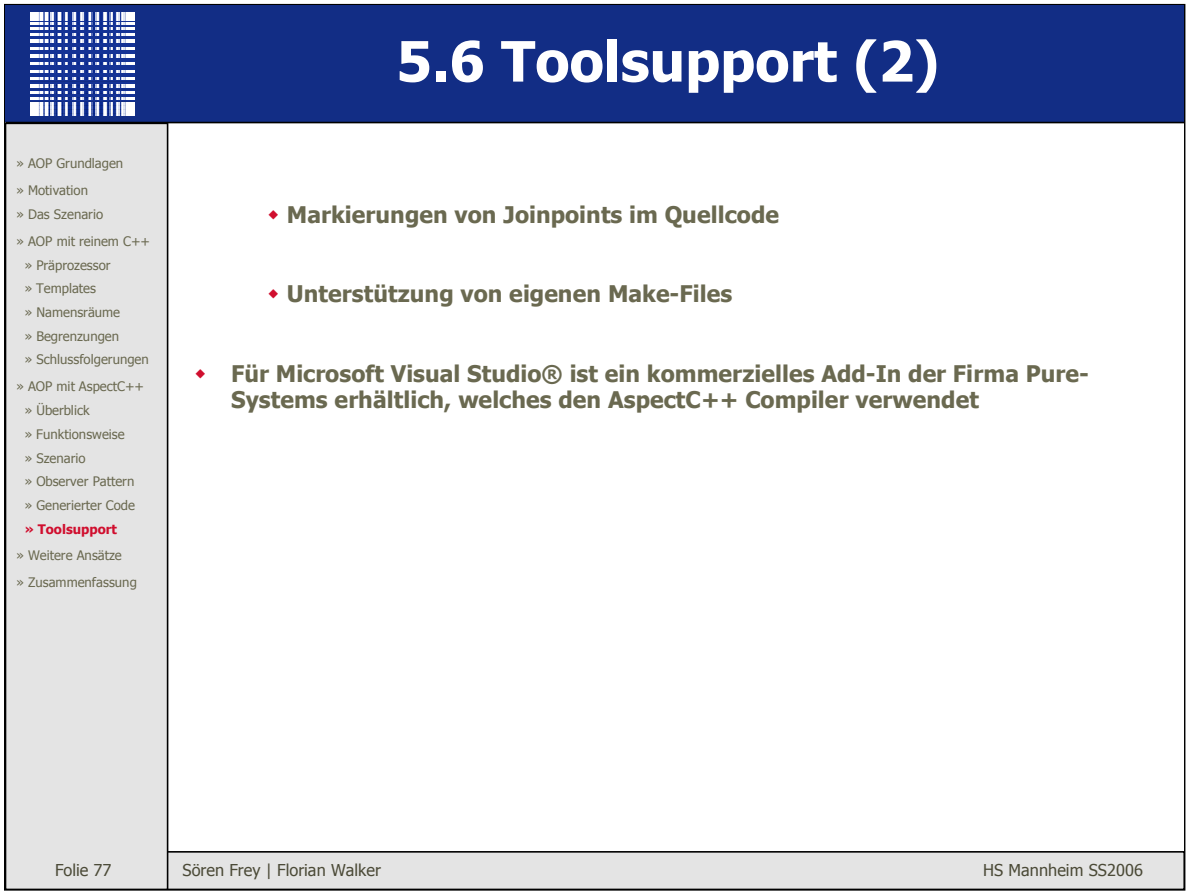
Advices werden zu Memberfunktionen umgeformt

```
class SecurityAspect {
    static SecurityAspect *aspectof () {
        static SecurityAspect __instance;
        return &__instance;
    }
    template<class JoinPoint> void __a0_before(JoinPoint *tjp){
    ...
    }
    ...
};
```

AspectWord.acc



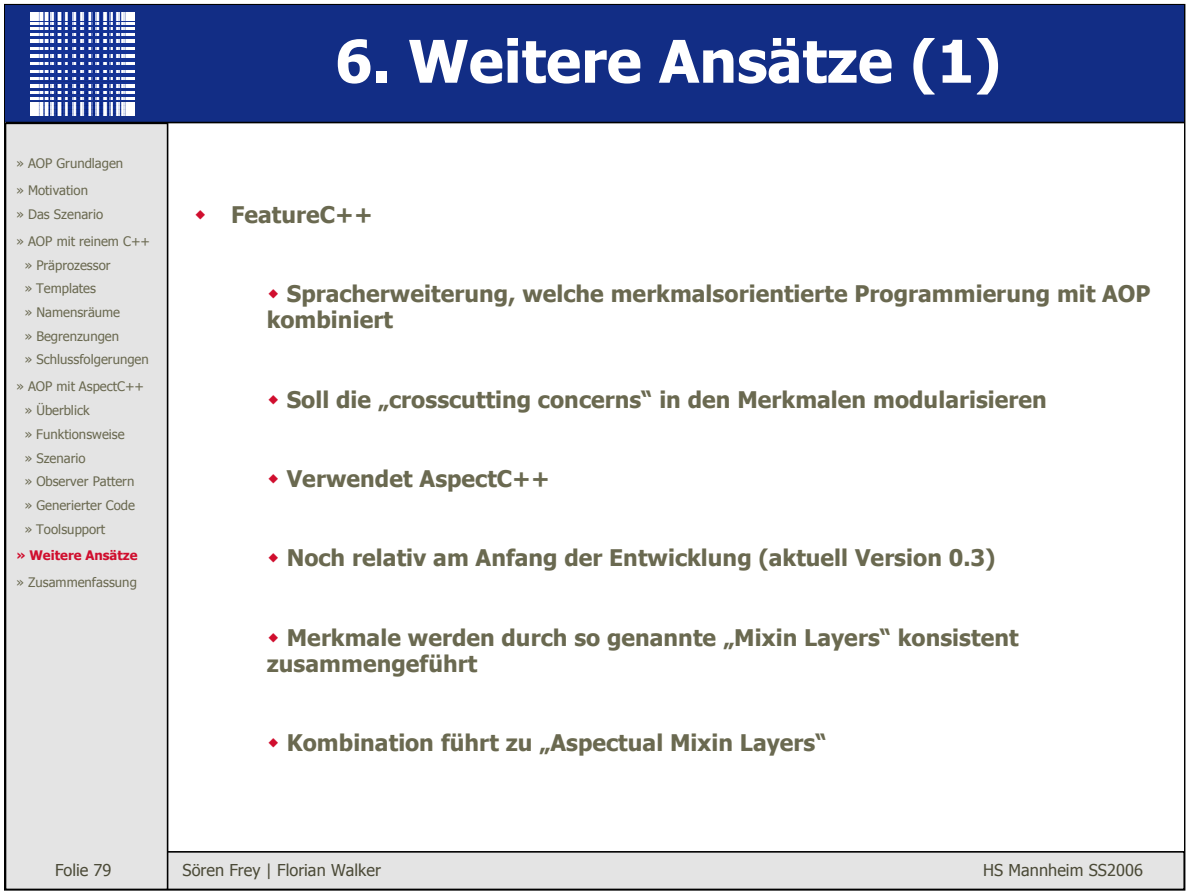
- Homepage von ACDT: http://acdt.aspectc.org
- Homepage von CDT: http://www.eclipse.org/cdt



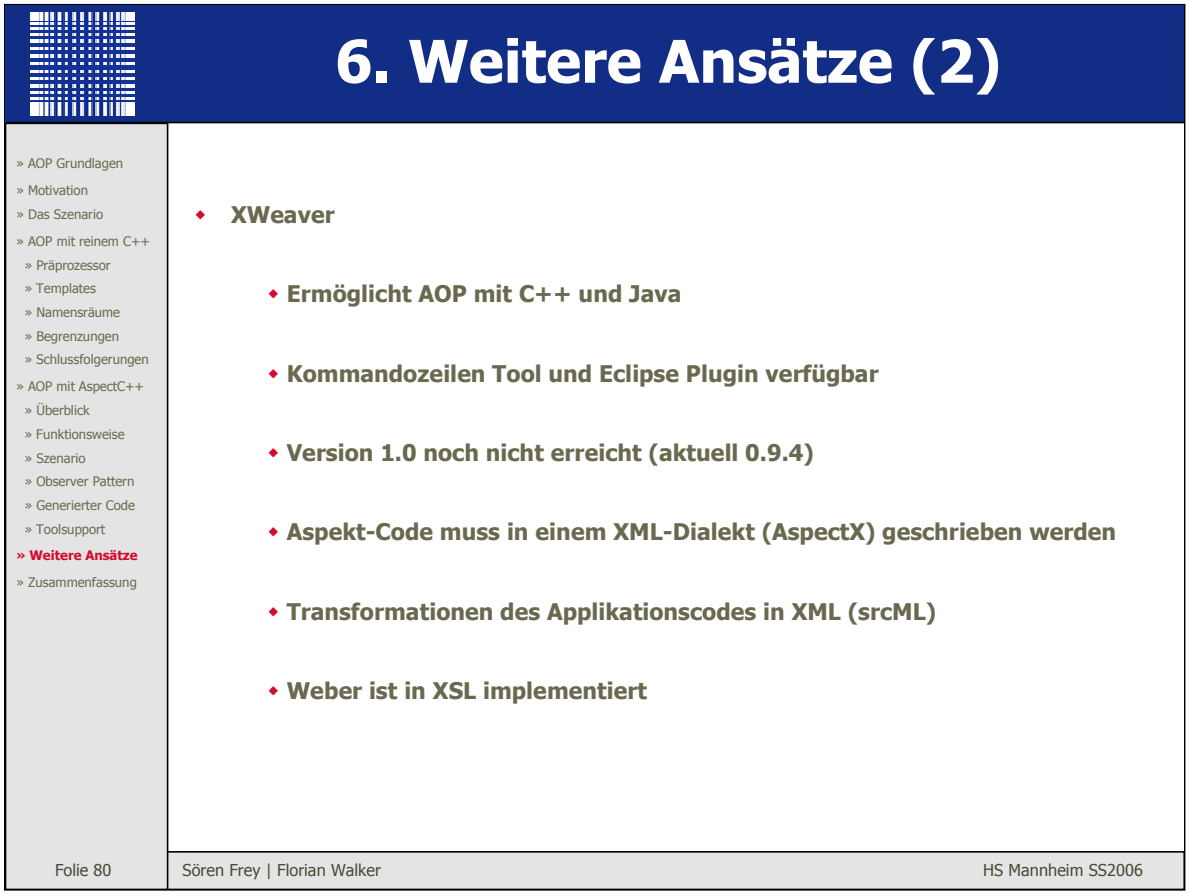
- Homepage der Firma Pure-Systems: http://www.pure-systems.de

5.6 Toolsupport (3)

Demo

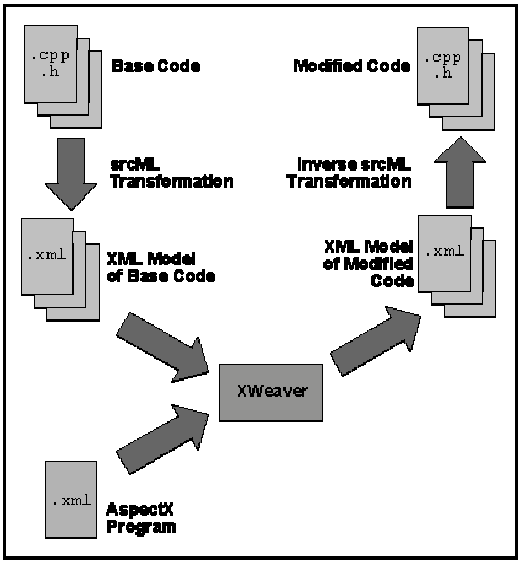


- Homepage von FeatureC++: http://wwwiti.cs.uni-magdeburg.de/iti_db/forschung/fop/featurec/



- Homepage von XWeaver: http://www.xweaver.org/
- Homepage von srcML: http://www.sdml.info/projects/srcml/

6. Weitere Ansätze (3)



Quelle: http://www.xweaver.org/

- Hierbei handelt es sich ebenso um eine „source-to-source" Transformation

7. Zusammenfassung (1)

- C++ Templates können verwendet werden, um die verschiedenen „concerns" mit reinem C++ zu modularisieren
- Die fehlende Ausdrucksfähigkeit und schlechte Skalierung schränkt diese Technik ein auf
  - eine geringe Anzahl an Aspekten
  - wenige Interaktionen zwischen Aspekten
  - Anwendungen, die für die Verwendung von Aspekten „gut geeignet" sind
- AspectC++ am weitesten verbreitet für AOP mit C++
- AspectC++ skaliert besser und ist weniger abhängig von „zweckdienlichen" Anwendungen

7. Zusammenfassung (2)

- **AspectC++ bietet eine gute Unterstützung für wieder verwendbaren und generischen Code**
- **Unterstützung für wichtige Tools ist vorhanden**
- **Verwobener Quellcode kann von bevorzugtem Compiler kompiliert werden**
- **Wenige Alternativen**

Quellen (1)

- **PUMA – The PURE Manipulator:**
  http://ivs.cs.uni-magdeburg.de/~puma/
- **Homepage von AspectC++**
  http://www.aspectc.org/
- **C/C++ Development Tools**
  http://www.eclipse.org/cdt/
- **AspectC++ Development Tools**
  http://acdt.aspectc.org
- **Pure-Systems GmbH**
  http://www.pure-systems.de

Quellen (2)

- **Aspect-Oriented Software Development Community & Conference**
  http://aosd.net/
- **Homepage von FeatureC++**
  http://wwwiti.cs.uni-magdeburg.de/iti_db/forschung/fop/featurec/
- **Seminar Aspektorientierte Programmierung**
  http://myhpi.de/~nicolai/aspectc++.pdf
- **iX 8/2001, S. 143: C++-Know-how - Teil 1-3**
  http://www.heise.de/ix/artikel/2001/08/143/
- **Aspect Oriented Software Delopment – Vorlesung FH-Ulm**
  http://www.aosd.de/Vorlesung/SS06.html
- **Aspekt-Orientierte Programmierung mit C++**
  http://www4.informatik.uni-erlangen.de/Lehre/SS03/HS_AspectOS/Ergebnisse/Cpp-Ausarbeitung.pdf

Danke für die Aufmerksamkeit

hochschule mannheim

HS Mannheim SS2006