SEN Q&A

Matthias Schett

5. Juni 2013

1. Gemeinsames Erarbeiten einer SEN UE
2. Q&A
3. (Optional)Schreiben der Doku

Inhalt

Inhalt

Erste Analyse

Erste Analyse

- Wie viele Aufgaben sind enthalten

Erste Analyse

- Wie viele Aufgaben sind enthalten
- Haben die Aufgaben eine Abhängigkeit zueinander

Erste Analyse

- Wie viele Aufgaben sind enthalten
- Haben die Aufgaben eine Abhängigkeit zueinander
- Ist eine Schnittstelle gegeben?

Erste Analyse

- Wie viele Aufgaben sind enthalten
- Haben die Aufgaben eine Abhängigkeit zueinander
- Ist eine Schnittstelle gegeben?
- Gibt es Vorgaben zur Formtierung des Inputs oder Outputs

Inhalt

Vorgaben

Vorgaben

- Erste Lösungsgedanken für die Problemstellung machen

Vorgaben

- Erste Lösungsgedanken für die Problemstellung machen
- Wenn möglich alles in kleinere Teile aufteilen[1]

[1] Erhöht in großen Projekten auch die Testbarkeit

Vorgaben

- Erste Lösungsgedanken für die Problemstellung machen
- Wenn möglich alles in kleinere Teile aufteilen[1]
- Klassenbasiert oder rein Funktionsbasiert?

[1] Erhöht in großen Projekten auch die Testbarkeit

Vorgaben

- Erste Lösungsgedanken für die Problemstellung machen
- Wenn möglich alles in kleinere Teile aufteilen[1]
- Klassenbasiert oder rein Funktionsbasiert?
- Erste Gedanken über die Wahl der möglichen STL Algorithmen

[1] Erhöht in großen Projekten auch die Testbarkeit

Inhalt

Methodiken

Methodiken

- Test Driven Development

Methodiken

- Test Driven Development
- Behavoir Driven Development (eine Abwandlung von TDD)

Methodiken

- Test Driven Development
- Behavoir Driven Development (eine Abwandlung von TDD)
- Feature Driven Development

Methodiken

- Test Driven Development
- Behavoir Driven Development (eine Abwandlung von TDD)
- Feature Driven Development
- etc.

Methodiken

- Test Driven Development
- Behavoir Driven Development (eine Abwandlung von TDD)
- Feature Driven Development
- etc.

Methodiken

- Test Driven Development
- Behavoir Driven Development (eine Abwandlung von TDD)
- Feature Driven Development
- etc.

Welche also wählen?

Wahl der Methodik

Wahl der Methodik

Abhängig von:

- Persönlicher Preferenz

Wahl der Methodik

Abhängig von:

- Persönlicher Preferenz
- Vorhanden Werkzeugen

Wahl der Methodik

Abhängig von:

- Persönlicher Preferenz
- Vorhanden Werkzeugen
- evtl. Firmeninternen Vorgaben

Wahl der Methodik

Abhängig von:

- Persönlicher Preferenz
- Vorhanden Werkzeugen
- evtl. Firmeninternen Vorgaben

Wahl der Methodik

Abhängig von:

- Persönlicher Preferenz
- Vorhanden Werkzeugen
- evtl. Firmeninternen Vorgaben

Für unser Beispiel entscheinden wir uns für TDD

Besonderheiten TDD

Vorgehensweise:

Besonderheiten TDD

Vorgehensweise:

1. Erstellen des Tests

Besonderheiten TDD

Vorgehensweise:

1. Erstellen des Tests
2. Erste Implementierung bis erster Test erfolgreich

Besonderheiten TDD

Vorgehensweise:

1. Erstellen des Tests
2. Erste Implementierung bis erster Test erfolgreich
3. Refactoring (Verbesserungen usw.)

Besonderheiten TDD

Vorgehensweise:

1. Erstellen des Tests
2. Erste Implementierung bis erster Test erfolgreich
3. Refactoring (Verbesserungen usw.)
4. Testen bis wieder erfolgreich

Besonderheiten TDD

Vorgehensweise:

1. Erstellen des Tests
2. Erste Implementierung bis erster Test erfolgreich
3. Refactoring (Verbesserungen usw.)
4. Testen bis wieder erfolgreich
5. Wieder zu Schritt 3 bis mit dem Algorihmus zufrieden

Inhalt

Klassendesign

Klassendesign

- Was will man beschreiben?

Klassendesign

- Was will man beschreiben?
- Aus welchen Teilen besteht es?

Klassendesign

- Was will man beschreiben?
- Aus welchen Teilen besteht es?

Klassendesign

- Was will man beschreiben?
- Aus welchen Teilen besteht es?

Tipp 1:

Gutes Klassendesign kann viel Arbeit sparen!

Klassendesign

- Was will man beschreiben?
- Aus welchen Teilen besteht es?

Tipp 1:

Gutes Klassendesign kann viel Arbeit sparen!

Klassendesign

- Was will man beschreiben?
- Aus welchen Teilen besteht es?

Tipp 1:

Gutes Klassendesign kann viel Arbeit sparen!

Tipp 2:

Funktions- und Membernamen dürfen ruhig lang sein
Schlecht lesbar: BkAccDepBal
Besser: BankAccountDepositBalance

KISS and DRY

KISS and DRY

Drei wichtige Prinipien in der Softwareentwicklung:

KISS and DRY

Drei wichtige Prinipien in der Softwareentwicklung:

- KISS - Keep it simple stupid

KISS and DRY

Drei wichtige Prinipien in der Softwareentwicklung:

- KISS - Keep it simple stupid
- DRY - Don't repeat yourself

KISS

KISS

- Nichts implementieren, dass nicht gefordert ist.

KISS

- Nichts implementieren, dass nicht gefordert ist.
- Nichts implementieren, mit dem Hintergedanken, dass man das irgendwann mal brauchen könnte.

DRY

DRY

- Kleine Wiederverwendbare Methoden/Klassens schreiben. (bzw. Alles in Module aufteilen)

Quellcodeverwaltung

Quellcodeverwaltung

Hat Quellcodeverwaltung bei so kleinen Projekten Sinn?

Quellcodeverwaltung

Hat Quellcodeverwaltung bei so kleinen Projekten Sinn?

Quellcodeverwaltung mach immer Sinn

Quellcodeverwaltung

Quellcodeverwaltung

Vorteile von Quellcodeverwaltung

Quellcodeverwaltung

Vorteile von Quellcodeverwaltung

- Änderungen ohne Sorgen

Quellcodeverwaltung

Vorteile von Quellcodeverwaltung

- Änderungen ohne Sorgen
- Bei fehlerhaften Änderungen Problemfreies rückgängig machen.

Quellcodeverwaltung

Vorteile von Quellcodeverwaltung

- Änderungen ohne Sorgen
- Bei fehlerhaften Änderungen Problemfreies rückgängig machen.
- Leichteres Arbeiten im Team

Hands-on Code Lab

Hands-on Code Lab

Jetzt folgt der Praktische Teil! Vorgehenweise:

Hands-on Code Lab

Jetzt folgt der Praktische Teil! Vorgehenweise:

- Implemtierung des Testreibers

Hands-on Code Lab

Jetzt folgt der Praktische Teil! Vorgehenweise:

- Implemtierung des Testreibers
- Klassen Header schreiben

Hands-on Code Lab

Jetzt folgt der Praktische Teil! Vorgehenweise:

- Implemtierung des Testreibers
- Klassen Header schreiben
- Klasse implementieren

Hands-on Code Lab

Jetzt folgt der Praktische Teil! Vorgehenweise:

- Implemtierung des Testreibers
- Klassen Header schreiben
- Klasse implementieren
- Testen

Inhalt

Klassentemplates

```
1  template <typename T>
2  class Calculator{
3      T value;
4
5      T getValue() const;
6      setValue(const &T value);
7
8      add(const &T value2);
9      sub(const &T value2);
10 };
```

Funktionspointer

Implementierung:

```
1 template <typename T>
2 bool compare(const &T a, const &T b){
3     return a == b;
4 }
```

Funktionspointer

Implementierung:

```
1 template <typename T>
2 bool compare(const &T a, const &T b){
3     return a == b;
4 }
```

Verwendung:

```
1 max_element(myVec.begin(), myVec.end(),
      compare);
```

Funktionsobjekte

Implementierung:

```
1 template <typename T>
2 class compare{
3     bool operator() (const &T a, const &T b){
4         return a == b;
5     }
6 };
```

Funktionsobjekte

Implementierung:

```
1 template <typename T>
2 class compare{
3     bool operator() (const &T a, const &T b){
4         return a == b;
5     }
6 };
```

Verwendung:

```
1 max_element(myVec.begin(), myVec.end(),
      compare());
```

Funktionsobjekte

Implementierung:

```
1 template <typename T>
2 class compare{
3     bool operator() (const &T a, const &T b){
4         return a == b;
5     }
6 };
```

Verwendung:

```
1 max_element(myVec.begin(), myVec.end(),
      compare());
```

Info:

Es muss nicht unbedingt eine Klasse sein, es kann genauso gut eine Struktur verwendet werden!

Template Spezialisierung

Template Spezialisierung

Wann benötigt?

Template Spezialisierung

Wann benötigt?

- Wenn der Compiler nicht von den Parametern bestimmen kann welcher Typ benötigt wird

Beispiele Template Spezialiserung
Keine Spezialisierung notwendig

Beispiele Template Spezialiserung

Keine Spezialisierung notwendig

Hier weiß der Compiler welcher Typ erwartet wird.

```
1 template <typename T>
2 void setValue(T const & a){
3 value = a;
4 }
```

Beispiele Template Spezialiserung

Keine Spezialisierung notwendig

Hier weiß der Compiler welcher Typ erwartet wird.

```
1 template <typename T>
2 void setValue(T const & a){
3 value = a;
4 }
```

Aufruf:

```
1 int a, b;
2 setValue(a, b);
```

Beispiele Template Spezialiserung

Spezialisierung notwendig

Beispiele Template Spezialiserung

Spezialisierung notwendig

Hier ist das ganze nicht klar

```
1 tempate <typename T>
2 T getValue(){
3 return ( a * pow(c, 2) );
4 }
```

Beispiele Template Spezialiserung

Spezialisierung notwendig

Hier ist das ganze nicht klar

```
1 tempate <typename T>
2 T getValue(){
3 return ( a * pow(c, 2) );
4 }
```

Aufruf:

```
1 int val = getValue<int>();
```

Verwendung Template Spezialiserung

- Funktoren

Verwendung Template Spezialiserung

- Funktoren
- Funktionsobjekte

Verwendung Template Spezialiserung

- Funktoren
- Funktionsobjekte
- Metohden bei denen der Compiler den Typ nicht kennt.

iterator_traits

iterator_traits

- Klasse die Eigenschaften von Iteratoren definiert

iterator_traits

- Klasse die Eigenschaften von Iteratoren definiert
- STL Algorithmen arbeiten häufig mit Traits um bestimmte Zustände (range, etc.) zu prüfen

iterator_traits

- Klasse die Eigenschaften von Iteratoren definiert
- STL Algorithmen arbeiten häufig mit Traits um bestimmte Zustände (range, etc.) zu prüfen

iterator_traits

- Klasse die Eigenschaften von Iteratoren definiert
- STL Algorithmen arbeiten häufig mit Traits um bestimmte Zustände (range, etc.) zu prüfen

Membername	Bedeutung
difference_type	Resultat einer Substraktion des Iterators mit einem anderen
value_type	Welchen Typ beihnaltet der Iterator
pointer	Pointer Typ auf den der Iterator zeigen kann
reference	Referenz Typ auf den der Pointer zeigen kann
iterator_category	Um welcher Iterator Typ handelt es sich (Output, Forward, Random-Access)

Tabelle : Member der Traitsklasse[cpp]

Verwendung iterator_traits

Verwendung iterator_traits

Beispiel: Herausfinden welchen Typ man bekommt für Output Iterator

Verwendung iterator_traits

Beispiel: Herausfinden welchen Typ man bekommt für Output Iterator

```
1  template <typename InputItor>
2  void superPrintFunction(InputItor begin,
       InputItor end){
3        std::ostream_iterator<InputItor::
           value_type> out_it(std::cout, ",");
4
5      std::copy(begin, end, out_it);
6  }
```

STL Fehler lesen

Es ist nicht ganz einfach STL Fehler zu lesen

Aber:

Wenn man im Fehler einfach nach seinen eigenen Filenamen sucht, dann lässt sich das ganze recht leicht auflösen.

Inhalt

Dokumentation

Möglichkeiten:

- WYSIWYG[2] (Microsoft Word)

[2] What you see is what you get

Dokumentation

Möglichkeiten:

- WYSIWYG[2] (Microsoft Word)
- WYSIWYM[3] (Latex)

[2] What you see is what you get
[3] What you see is what you mean

Inhalt

Vorteile

Vorteile

- Quellcode lässt sich leicht einbinden.

Vorteile

- Quellcode lässt sich leicht einbinden.
- Keine Probleme bei Änderungen in der Fromatierungen

Vorteile

- Quellcode lässt sich leicht einbinden.
- Keine Probleme bei Änderungen in der Fromatierungen
- Komplexe Mathematische Formeln relativ einfach möglich

Vorteile

- Quellcode lässt sich leicht einbinden.
- Keine Probleme bei Änderungen in der Fromatierungen
- Komplexe Mathematische Formeln relativ einfach möglich
- Sehr gute Satzqualität (vor allem bzgl. Wort und Zeilentrennung)

Vorteile

- Große Dokumente lassen sich auf kleine aufteilen und anschließend sehr einfach zusammenfügen

Vorteile

- Große Dokumente lassen sich auf kleine aufteilen und anschließend sehr einfach zusammenfügen
- Ausgabeformate sind leicht austauschbar (PDF, DVI oder HTML Ausgabe)

Vorteile

- Große Dokumente lassen sich auf kleine aufteilen und anschließend sehr einfach zusammenfügen
- Ausgabeformate sind leicht austauschbar (PDF, DVI oder HTML Ausgabe)
- Portabilität (Das Ergebnis sieht auf allen Systemen gleich aus.)

Vorteile

- Große Dokumente lassen sich auf kleine aufteilen und anschließend sehr einfach zusammenfügen
- Ausgabeformate sind leicht austauschbar (PDF, DVI oder HTML Ausgabe)
- Portabilität (Das Ergebnis sieht auf allen Systemen gleich aus.)
- Läuft sowohl auf Windows, Linux und Mac

Nachteile

Nachteile

- Extrem steile Lernkurve

Nachteile

- Extrem steile Lernkurve
- Für das Ergebnis muss das Dokument zuerst durch den Compiler

Nachteile

- Extrem steile Lernkurve
- Für das Ergebnis muss das Dokument zuerst durch den Compiler
- Tabellen umständlich

Nachteile

- Extrem steile Lernkurve
- Für das Ergebnis muss das Dokument zuerst durch den Compiler
- Tabellen umständlich
- Kein offizieller Support[4]

[4] Es gibt aber eine sehr große und starke Community hinter Latex

Wie sieht ein Latex Dokument aus

Wie sieht ein Latex Dokument aus

- Besteht aus Präambel (quasi Header mit den Includes)

Wie sieht ein Latex Dokument aus

- Besteht aus Präambel (quasi Header mit den Includes)
- Dem Document Body

Beispiel Dokument

Quellcodeanalyse Latex

Quellen I

 *Cpp-reference (http://www.cplusplus.com/)*, zuletzt besucht 03.06.2013 21:57.

 R.C. Martin, *Clean code: A handbook of agile software craftsmanship*, Pearson Education, 2008.

 Scott Meyers, *Effektiv c++ programmieren*, 3 ed., Addison-Wesley, 2011.

 *Stackoverflow (http://www.stackoverflow.com/)*, zuletzt besucht 03.06.2013 20:53.