

Intelligentes Programmieren: CodeBroker

im Rahmen des Proseminars
„Künstliche Intelligenz“ (SS '04)

Thilo Schmitt

Einführung – Grundgedanken

- Was ist Programmieren?
- Was macht ein Entwickler?

- Unterstützung durch ein System möglich?
→ Wie?

- verschiedene Anforderungen an das System

Methode 1: „Naives Entwickeln“

- Entwicklung einer neuen Lösung
(Erstellung *from scratch*)

- Vorteil: „engste“ Lösung
- Nachteil: mitunter horrender Zeitaufwand, Fehleranfälligkeit, „man erfindet das Rad neu“

Methode 2: Adaption

- ähnliche Teilprobleme mit bereits existenten „Programmfragmenten“ lösen
- „Fragmente“: Komponenten

- Vorteil: „Baukastenprinzip“
- „Wiederverwendung von Software“:
software reuse

- Zusammenstellung von Komponenten in Bibliotheken, Zugriff per API

Software Reuse – Probleme (1)

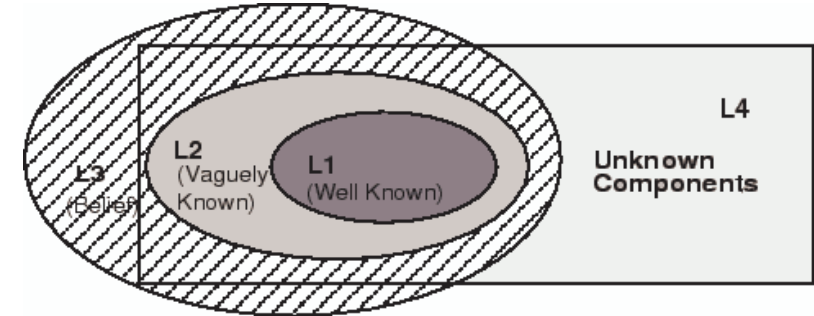
- rasantes Wachstum der APIs
- Beispiel Java-Core-API:

Version	Packages	Klassen
Java 1.0	8	211
Java 1.1	23	503
Java 1.2	59	1525
Java 2	> 70	> 2100

→ API sehr unübersichtlich geworden

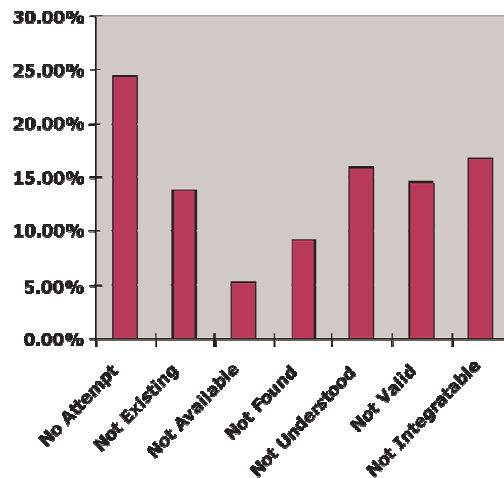
Software Reuse – Probleme (2)

- „Das Wissen eines Programmierers“:



Software Reuse – Probleme (3)

- Fehlermodi
- Ansatzpunkte für ein unterstützendes System



Software Reuse – Zusammenfassung

- Gerade bei objektorientierten Sprachen die bevorzugte Wahl
- zeiteffizient
- vorhandene APIs sehr umfangreich
- Viele Fehlerquellen die zum Scheitern der Methode führen
- Unterstützende Systeme wünschenswert

Systeme zur Unterstützung

- Benötigt werden Informationen ...
- ... zum richtigen Thema
- ... zum richtigen Kontext
- ... zur richtigen Zeit

- Und das automatisch (aktiv) → AIS
Hier: „*component repository*“ → ACRS
- Außerdem: Gute Beispiele zum Einsatz der Komponenten

CodeBroker – Allgemeines

- Entwicklung von Java-Anwendung
- für Emacs mit JDE

- (theoretisch) geeignet für jede objektorientierte Programmiersprache
- (theoretisch) geeignet für jede Entwicklungsumgebung mit passender Schnittstelle

CodeBroker – Ansätze

- Untersuchung der Eingabe *on the fly*
- Extraktion von Informationen über ...
- ... die zu lösende Teilaufgabe
→ Programm-Kommentare
- ... den bedingenden Kontext
→ Methoden-Signaturen
- ... das Wissen des Programmierers (annäherungsweise)
→ gewisse Komponenten filtern
→ Personalisierung

CodeBroker in Aktion (1)

Beispielaufgabe:

Simulating the card dealing process.

The program has to simulate the process of card dealing.

Each card is represented with a number from 0 to 51. The program should produce a list of 52 cards, as if it is resulted from a human card dealer.

CodeBroker in Aktion (1)

```
emacs@buddy.cs.colorado.edu
Buffers Files Tools Edit Search Mule Java Help

/** This class simulate the process of card dealing. Each card is
  represented with a number from 0 to 51. And the program should produce
  a list of 52 cards, at it is resulted from a human card dealer */
public class CardDealer1 {
  static int [] cards=new int[52];
  static {
    for (int i=0; i<52; i++) cards[i]=i;
  }
  /** Create a random number between two limits */
}

--:** CardDealer1.java 09-10 12:05 Mail (Java)--L9--All-----
1 0.40 setMaxRow The maxRows limit is set to limit the number of ro
2 0.38 nextBytes Generates a user specified number of random bytes.
3 0.38 getInt Generate a random number using the default generat
4 0.38 getInt Generate a random number using the default generat
--:** *RCI-display* 09-10 12:05 Mail (ReusableComponentInfo)--L1--Top-----
com.objectspace.jgl.util.Randomizer::static int getInt(int hi)
```

CodeBroker in Aktion (2)

```
emacs@buddy.cs.colorado.edu
Buffers Files Tools Edit Search Mule Help

/** This class simulate the process of card dealing. Each card is
  represented with a number from 0 to 51. And the program should produce
  a list of 52 cards, at it is resulted from a human card dealer */
public class CardDealer1 {
  static int [] cards=new int[52];
  static {
    for (int i=0; i<52; i++) cards[i]=i;
  }
  /** Create a random number between two limits */
  public static int getRandomNumber (int from, int to) {
}

--:** CardDealer1.java 09-10 12:13 0.07 Mail (JDE)--L10--All-----
1 0.69 getInt Generate a random number using the default generat
2 0.64 getLong Generate a random number using the default generat
3 0.59 getFloat Generate a random number using the default generat
4 0.59 getDouble Generate a random number using the default generat
--:** *RCI-display* 09-10 12:13 0.07 Mail (ReusableComponentInfo)--L1--Top-----
com.objectspace.jgl.util.Randomizer::static int getInt(int lo, int hi)
```

CodeBroker in Aktion (3)

```
emacs@buddy.cs.colorado.edu
Buffers Files Tools Edit Search Mule JDE Java Help

/** This class simulate the process of card dealing. Each card is
  represented with a number from 0 to 51. And the program should produce
  a list of 52 cards, at it is resulted from a human card dealer */
public class CardDealer2 {
  static int [] cards=new int[52];
  static {
    for (i=0; i<52; i++) cards[i]=i;
  }
  /** Shuffle the cards to an arbitrary order */
}

--:** CardDealer2.java 09-09 10:58 Mail (JDE)--L9--All-----
1 0.42 next Flips to the next card of the specified container.
2 0.41 previous Flips to the previous card of the specified contai
3 0.37 first
4 0.37 last
Skip Components Menu
void next(java.awt.Container parent) > tInfo)--L1--Top-----
CardLayout > This Session Only
java.awt > All Sessions
Query Refinement >
```

CodeBroker in Aktion (4)

```
emacs@buddy.cs.colorado.edu
Buffers Files Tools Edit Search Mule JDE Java Help

/** This class simulate the process of card dealing. Each card is
  represented with a number from 0 to 51. And the program should produce
  a list of 52 cards, at it is resulted from a human card dealer */
public class CardDealer2 {
  static int [] cards=new int[52];
  static {
    for (i=0; i<52; i++) cards[i]=i;
  }
  /** Shuffle the cards to an arbitrary order */
}

--:** CardDealer2.java 09-10 12:17 0.05 Mail (JDE)--L9--All-----
1 0.24 primaryOr Get the primary order of a collation order.
2 0.24 secondary Get the secondary order of a collation order.
3 0.24 tertiary0 Get the tertiary order of a collation order.
4 0.22 randomShu Shuffle a sequence with uniform distribution by pe
--:** *RCI-display* 09-10 12:17 0.05 Mail (ReusableComponentInfo)--L1--Top-----
com.objectspace.jgl.algorithms.Shuffling::static void randomShuffle(com.objectsp
```

CodeBroker – Evaluierung

- 5 Testpersonen mit unterschiedlichen Erfahrungsgraden
- Wertungen von 1 (nutzlos) bis 10 (äußerst nützlich)

wenig ← *Programmier-Erfahrung* → viel

Testperson	1	2	3	4	5
Note	7	4	8.5	7	8

∅ = 6,9 ≈ 7

CodeBroker – Verbesserungen

- Informationsvorrat erweitern
- Algorithmen zur Kommentar-Textanalyse ausbauen
- CodeBroker „im *Grid*“
- Abstraktionsebene höher setzen
→ Designs/Pattern

Fazit (Was kann CodeBroker?)

- Unterstützung des Entwicklers durch permanente aktive Analyse der Eingabe
- aufgabenrelevante Komponenten filtern und anbieten
- CodeBroker ist geeignetes Mittel um Fehlerraten minimieren
- Aber: Konzept ist eher glücklich gewählt, als das System „künstlich intelligent“