Hendrik Degener
Institut für Informatik

Freie Universität Berlin

Unterstützung des Programmierers bei der Einarbeitung in bestehenden Code

Seminarvortrag „Beiträge zum Softwareengineering“

EINLEITUNG

Problem

- Stolpersteine bei der Einarbeitung in Code
  - Schlechte Benennung von Variablen
  - Verschachtelung von Funktionen
  - Mangelnde Kommentare und Beschreibungen
  - „Die Nadel liegt im Heuhaufen“ – Welche Funktionen sind für mich relevant?
- Vermeidung erhöht Komfort und Arbeitsgeschwindigkeit, reduziert Frustration

Lösungsansätze

- Zwei Ansätze werden vorgestellt:
  - Verständnis: Funktionen mit High Level Actions beschreiben
  - Navigation: Bei der Code-Suche relevante Funktionen finden

BESTEHENDE FUNKTIONEN VERSTEHEN

Grundlegendes Problem

- Interne Dokumentation fehlt
- Schlechte Benennung von Methodennamen
- Mangelnde Aufteilung bläht Methoden auf
- Bestimmte Pattern kehren immer wieder
  - Können erkannt und „übersprungen“ werden
  - Aber: erhöhen kognitive Last

Ansatz

- Beschreibung von Methoden mit High Level Actions

Ansatz

- Beschreibung von Methoden mit High Level Actions
- Beispiel:

```
for (int x = 0; x<vAttacks.size(); x++){
        WeaponAttackAction waa = vAttacks.elementAt(x);
        float fDanger = getExpectedDamage(g, waa);
        if (fDanger > fHighest) {
                fHighest = fDanger;
                waaHighest = waa;
        }
}
return waaHighest;
```

Ansatz

- Beschreibung von Methoden mit High Level Actions
- Beispiel:

```
for (int x = 0; x<vAttacks.size(); x++){
        WeaponAttackAction waa = vAttacks.elementAt(x);
        float fDanger = getExpectedDamage(g, waa);
        if (fDanger > fHighest) {
                fHighest = fDanger;
                waaHighest = waa;
        }
}
return waaHighest;
```

```
Get weapon attack action object (in vAttacks) with
highest expected damage.
```

Ziel

- High Level Actions finden
- Entsprechende Beschreibung generieren

Was wird gemacht?

- Analyse von:
  - Statement-Sequenzen

```
contentPane.add(endedPanel);
contentPane.add(bidPanel);
contentPane.add(endingPanel);
```

```
buildGameMenu();
buildViewMenu();
buildOrdersMenu();
buildReportMenu();
```

Was wird gemacht?

- Analyse von:
  - Statement-Sequenzen
  - Konditionalen Blöcken

```
switch(type){
case A: ...
case B: ...
}
```

```
if (type == PT_HTTP){
        tester = new NetworkAdminHTTPTester(core, l);
}else if (type == PT_TCP){
        tester = new NetworkAdminTCPTester(core, l);
}else{
        tester = new NetworkAdminUDPTester(core, l);
}
```

Was wird gemacht?

- Analyse von:
  - Statement-Sequenzen
  - Konditionalen Blöcken
  - Schleifen

```
for (DrawingView v : views){
        if (v.getComponent() == c)
                return v;
}
```

Wie wirds gemacht?

- Beschreibungsgenerierung auf Basis von:
  - Syntaxanalyse
  - Semantikanalyse
  - Felderbenennung

Wie wirds gemacht?

- Beschreibungsgenerierung auf Basis von:
  - Syntaxanalyse
  - Semantikanalyse
  - Felderbenennung

1. Synthetisierung der Statements
2. Integration synthetisierter Statements
   - Zusammenfassen von Statements
   - Generierung von Beschreibungen

Synthetisierung eines Statements

- = Generierung einer Nominalphrase

1. Erweiterung von Abkürzungen in Benennung
2. Camel Case Splitting
3. Statements synthetisieren
   - SWUM (Software Word Usage Model)
     - Erfasst Aufkommen sowie linguistische und strukturelle Informationen der Wörter im Code
   - Bsp: list.add(item) -> add item to list
   - Bsp: list.remove(bigItem) -> remove big item from list

Integration von Statements

- Zusammenfassung von Statements zu einem Block
- Beschreibung des Blocks

Integration von Statements

- Wann sind zwei Statements integrierbar?

```
boolean areIntegrable(Statement S1,Statement S2)
{
 VP1 = generateVerbPhraseFor(S1);
 VP2 = generateVerbPhraseFor(S2);
 // A verb phrase is of the form,
 // verb noun-phrase [prepositional-phrase]
 // V NP [PP], with PP optional

 if !sameVerb(VP1, VP2) return false;

 if !sameHeadWord(NP1, NP2) AND
  !fieldsOfSameClass(NP1, NP2)
    return false;

 if present(PP1, PP2) AND !samePreposition(PP1, PP2)
  return false;

 return true;
}
```

V HW PP

add item to list
remove big item from list

NP

1.

2.

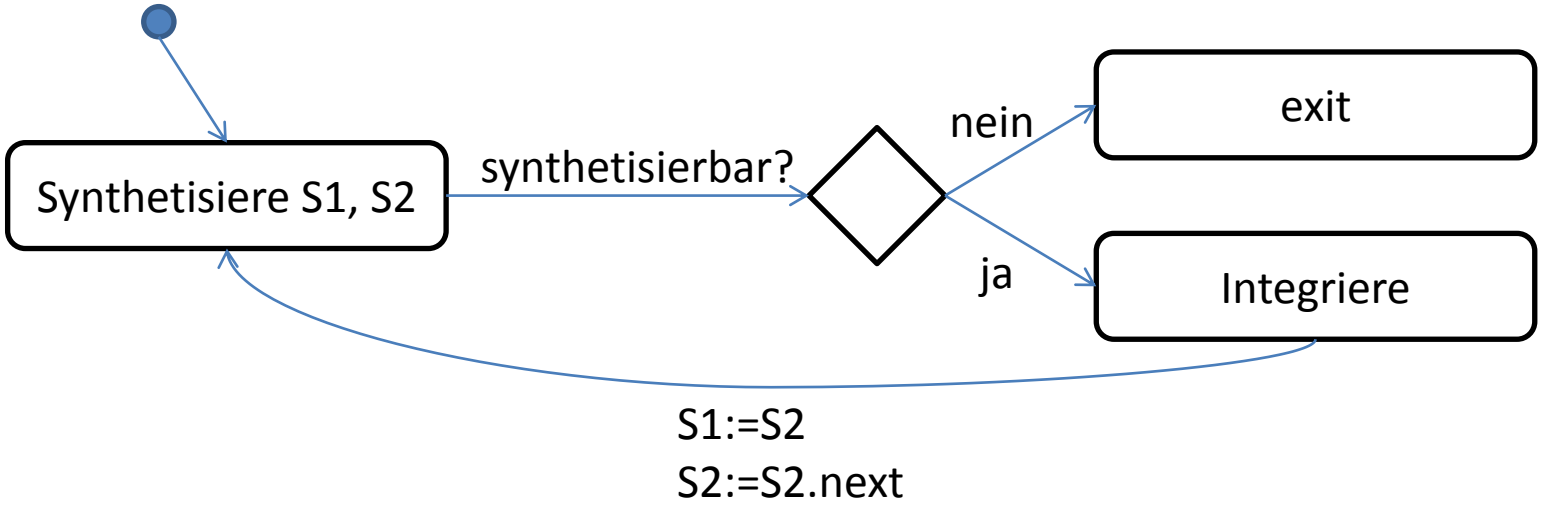
3.

Was wird gemacht?

- Analyse von:
  - Statement-Sequenzen
  - Konditionalen Blöcken
  - Schleifen

Sequenzen

- Betrachte nur Statements mit einem oder mehreren Methodenaufrufen
- Für jedes aufeinanderfolgende Statement S1, S2 dieser Form:



Sequenzen

```
buildGameMenu()
```

```
buildViewMenu()
```

Sequenzen

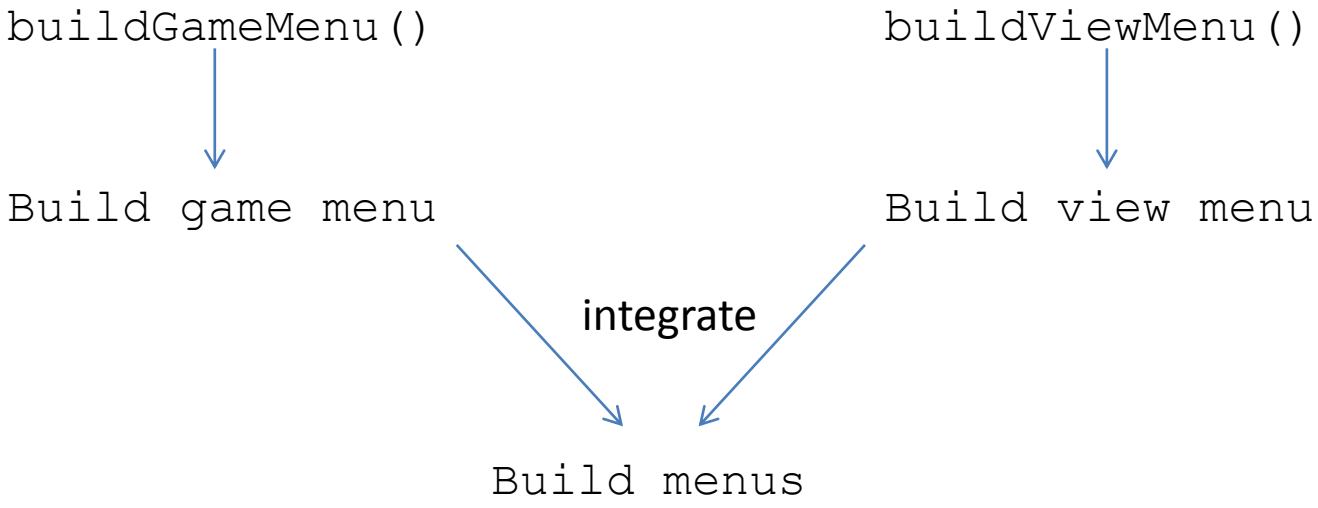
`buildGameMenu()`

`Build game menu`

`buildViewMenu()`

`Build view menu`

Sequenzen


buildGameMenu()
buildViewMenu()
Build game menu
Build view menu
integrate
Build menus

Was wird gemacht?

- Analyse von:
  - Statement-Sequenzen 
  - Konditionalen Blöcken
  - Schleifen

Konditionale

IF (…)
    {THEN-Block}
ELSE IF (…)
    {THEN-Block}
ELSE (…)
    {THEN-Block}

switch (…)
    case: {THEN-Block}
    case: {THEN-Block}
    case: {THEN-Block}

→integriere über THEN-Blöcke!

- Zunächst nur Betrachtung von Blöcken mit Größe 1

Konditionale

Methodenaufrufe

```
37    if (nothingJrb.isSelected()){
38        ConfigurationManager.setProperty(...);
39    } else if (lastJrb.isSelected()){
41        ConfigurationManager.setProperty(...);
42    } else if (LastKeepPosJrb.isSelected()){
44        ConfigurationManager.setProperty(...);
45    }
```

Variablenzuweisungen

```
3 if ( type == PT_HTTP ){
4   tester = new NetworkAdminHTTPTester(core, 1);
5 } else if ( type == PT_TCP ){
6   tester = new NetworkAdminTCPTester(core, 1);
7 }else{
8   tester = new NetworkAdminUDPTester(core, 1);
9 }
```

Return statements

```
6   switch(movementType) {
7   case IEntityMovementType.MOVE_NONE: return "N";
8   case IEntityMovementType.MOVE_WALK: return "W";
9   case IEntityMovementType.MOVE_RUN: return "R";
10  }
```

Variablenzuweisungen

Variablenzuweisungen

```
3 if ( type == PT_HTTP ){
4   tester = new NetworkAdminHTTPTester(core, l);
5 } else if ( type == PT_TCP ){
6   tester = new NetworkAdminTCPTester(core, l);
7 }else{
8   tester = new NetworkAdminUDPTester(core, l);
9 }
```

- Zuweisung zur selben Variable v

1. Synthetisierung
   - Zuweisung durch create- oder get-Methode
     → „[create OR get] lexicalize(v)“
   - Zuweisung durch sonstigen Methodenaufruf
     → „update lexicalize(v)“
2. Integration

Variablenzuweisungen

Variablenzuweisungen

```
3 if ( type == PT_HTTP ){
4   tester = new NetworkAdminHTTPTester(core, l);
5 } else if ( type == PT_TCP ){
6   tester = new NetworkAdminTCPTester(core, l);
7 }else{
8   tester = new NetworkAdminUDPTester(core, l);
9 }
```

„create network administration protocol tester based on type“

- Zuweisung zur selben Variable v

1. Synthetisierung
   - Zuweisung durch create- oder get-Methode
     → „[create OR get] lexicalize(v)“
   - Zuweisung durch sonstigen Methodenaufruf
     → „update lexicalize(v)“
2. Integration

Return statements

Return statements

```
6   switch(movementType) {
7   case IEntityMovementType.MOVE_NONE: return "N";
8   case IEntityMovementType.MOVE_WALK: return "W";
9   case IEntityMovementType.MOVE_RUN: return "R";
10  }
```

1. Synthetisierung

   →„return lexicalize(v)“

2. Integration

Return statements

Return statements

```
6   switch(movementType) {
7   case IEntityMovementType.MOVE_NONE: return "N";
8   case IEntityMovementType.MOVE_WALK: return "W";
9   case IEntityMovementType.MOVE_RUN: return "R";
10  }
```

"Return movement abbreviation based on movement type"

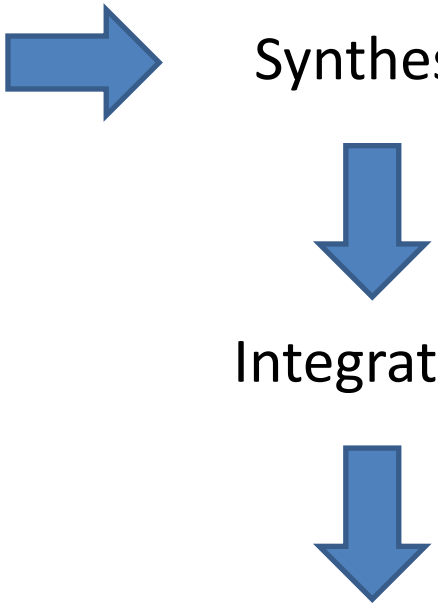
1. Synthetisierung

   →„return lexicalize(v)“

2. Integration

Kombination von Konditionalen

```
If (...)
      Variablenzuweisung
      Variablenzuweisung
      Methodenaufruf
      Return-Statement
else if (...)
      Variablenzuweisung
      Variablenzuweisung
      Methodenaufruf
      Return-Statement
else (...)
      Variablenzuweisung
      Variablenzuweisung
      Methodenaufruf
      Return-Statement
```




```
Variablenzuweisung
Variablenzuweisung
Methodenaufruf
Return-Statement
```

Was wird gemacht?

- Analyse von:
  - Statement-Sequenzen ✓
  - Konditionalen Blöcken ✓
  - Schleifen

Schleifen

1. Finde Templates für übliche Schleifenaufgaben
   - „zählen“, „kopieren“, „suchen“, „Maximum“, „Minimum“, „ist enthalten?“
2. Entwickle Beschreibung

Schleifen finden

Beispiel: Finde Objekt mit maximalem Wert

```
ALGORITHM TEMPLATE:

1  initialize variable, max to 0
2  Intialize variable, maxItem to null
3
4  loop over each item i in a Collection c
5   {
6     set variable, current, with the return value
      from a method-call which uses i as a parameter
8     if (current > max) //also: max < current
9     {
10     update max to current
11     update maxItem to i
12    }
13  }
```

Schleifen beschreiben

- Beschreibe gefundene Schleife durch zugehöriges Beschreibungs-Template
- Beispiel:

```
get <item> (in <collection>) with <adjective> <criteria>
```

```
ALGORITHM TEMPLATE:

1  initialize variable, max to 0
2  Intialize variable, maxItem to null
3
4  loop over each item i in a Collection c
5   {
6     set variable, current, with the return value
      from a method-call which uses i as a parameter
8     if (current > max) //also: max < current
9     {
10     update max to current
11     update maxItem to i
12    }
13  }
```

Schleifen beschreiben

get <item> (in <collection>) with <adjective> <criteria>

```
for (int x = 0; x<vAttacks.size(); x++){
        WeaponAttackAction waa = vAttacks.elementAt(x);
        float fDanger = getExpectedDamage(g, waa);
        if (fDanger > fHighest) {
                fHighest = fDanger;
                waaHighest = was;
        }
}
return waaHighest;
```

- <item>: weapon attack action object
- <collection>: vAttacks
- <adjective>: highest
- <criteria>: expected damage

Schleifen beschreiben

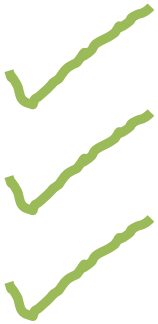
get <item> (in <collection>) with <adjective> <criteria>

```
for (int x = 0; x<vAttacks.size(); x++){
        WeaponAttackAction waa = vAttacks.elementAt(x);
        float fDanger = getExpectedDamage(g, waa);
        if (fDanger > fHighest) {
                fHighest = fDanger;
                waaHighest = was;
        }
}
return waaHighest;
```

get weapon attack action object (in vAttacks)
with highest expected damage

Was wird gemacht?

- Analyse von:
  - Statement-Sequenzen
  - Konditionalen Blöcken
  - Schleifen



Zwischenbilanz

- Bestimmte Statements können in Gruppen zusammengefasst werden
  - Sequenzen, Konditionale, Schleifen
  - Beschreibung in natürlicher Sprache
- Anwendung
  - Generierung von Kommentaren
  - Auslagerung der Gruppen in Methoden
  - Vorschläge für bessere Methodennamen

Zwischenbilanz

- Nachteile
  - Allzu schlechte Benennung behindert Algorithmus
  - Nur „Standard“-Code kann verwendet werden
    - Mit Erfahrung sowieso einfach zu lesen

Portfolio - Finde die Nadel im Heuhaufen

RELEVANTE FUNKTIONEN FINDEN

Problem

- Was macht der Code? Wie benutze ich ihn?
  - Verschlingt 50% - 80% der Zeit!
- Suche nach Einstiegspunkten
  - Wie werden Funktionen im Kontext anderer Funktionen genutzt?
  - Gesamtlösung muss oft aus Teilbeispielen zusammengefügt werden

Problembeispiel

- Suchanfrage: „mip map dithering texture image graphics“
- Es existieren:
  - Funktionen mit mip map-Techniken
  - Funktionen für Textur-Rendering
  - Funktionen um Grafiken zu manipulieren
- Wir wollen aber alles auf einmal!

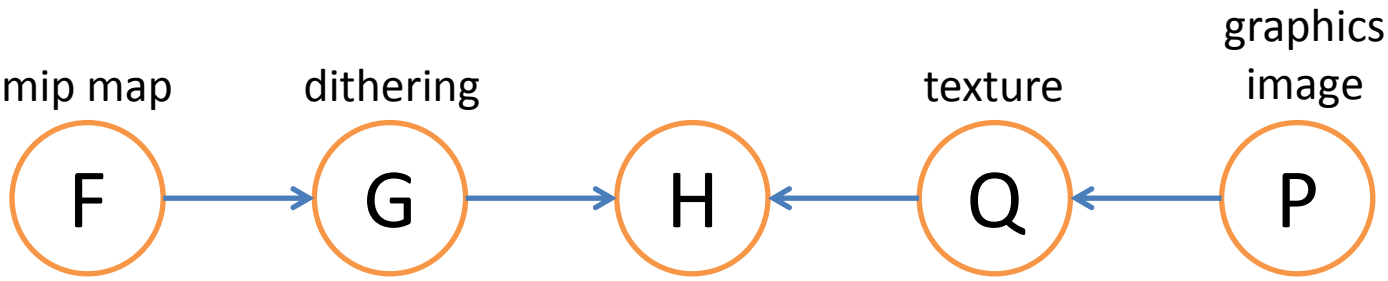
Problembeispiel

- Anfrage (Aufgabe): „mip map dithering texture image graphics“
- Welche Funktionen sind für die Aufgabe relevant?
- Wie setze ich die Funktionen zusammen, um die Aufgabe zu lösen?

Ansätze

- Schlüsselwortsuche
- Statische Analyse von Funktionen
  - Z.B. PageRank

Problem der Schlüsselwortsuche



- Funktion H ist relevant
  - Enthält aber kein Schlüsselwort
- Funktion G ist relevant
  - Generisches Interface -> häufig benutzt

→ Sowohl G als auch H werden nicht explizit als relevant erkannt

Ansatz von Portfolio

1. Erweiterte Schlüsselwortsuche
2. Statische Analyse (PageRank)
3. Kombiniertes Modell relevanter Funktionen bezüglich 1 und 2
4. Visualisiere relevante Funktionen

Modell

- PageRank
  - Je häufiger die Funktion aufgerufen wird, desto höher die Relevanz
  - Unabhängig von der eigentlichen Suche
- Modifiziertes Spreading Activation Network (SAN)
  - Basiert auf relevanten Funktionen bezüglich Schlüsselwortsuche (WOS)
  - Knoten = Funktionen
  - Kante = Funktionale Abhängigkeit
  - Knotengewicht = Relevanz

Modell

- **PageRank**
  - Je häufiger die Funktion aufgerufen wird, desto höher die Relevanz
  - Unabhängig von der eigentlichen Suche
- Modifiziertes Spreading Activation Network (**SAN**)
  - Basiert auf relevanten Funktionen bezüglich Schlüsselwortsuche (**WOS**)
  - Knoten = Funktionen
  - Kante = Funktionale Abhängigkeit
  - Knotengewicht = Relevanz

Modell - Details

- PageRank

$$r(F_i) = \sum_{F_j \in B_{F_i}} \frac{r(F_j)}{|F_j|}$$

Pagerank der aufrufenden Funktion

Anzahl der Funktionen, die von der aufrufenden Funktion aufgerufen werden

  - B_{F_i} ist die Menge der Funktionen, die *F_i* aufrufen
  - $|F_i|$ ist die Anzahl der Funktionen, die *F_i* aufruft

Modell - Details

- WOS
  - Suchterme werden folgendermaßen gewichtet

$$tf = \frac{n}{\sum_k n_k}$$

Anzahl der Vorkommnisse des Terms in der Funktion

Anzahl der Vorkommnisse des Terms in allen Funktionen

  - Alle Funktionen werden miteinander verglichen und die Ähnlichkeit bestimmt

$$d(f_i, f_j) = \frac{f_i \cdot f_j}{\| f_i \| \cdot \| f_j \|}$$

Modell - Details

- SAN
  - Ausgehend von WOS werden neue Knotengewichte *N_j* berechnet

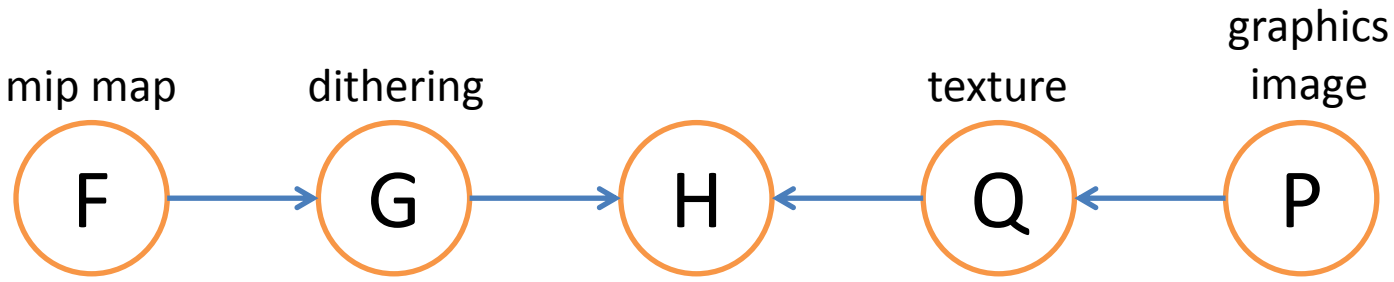
$$N_j = \sum_i f(N_i w_{ij})$$

Dämpfungsfaktor, Konstante zwischen 0 und 1

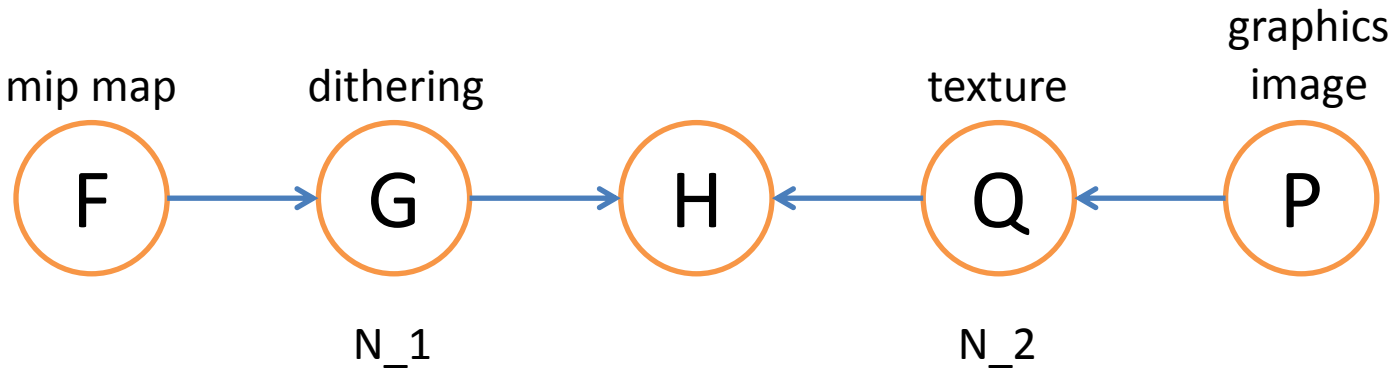
Zu *N_j* inzidente Knoten

Schwellenfunktion, liefert entweder 0 oder 1
Terminierung, wenn 0

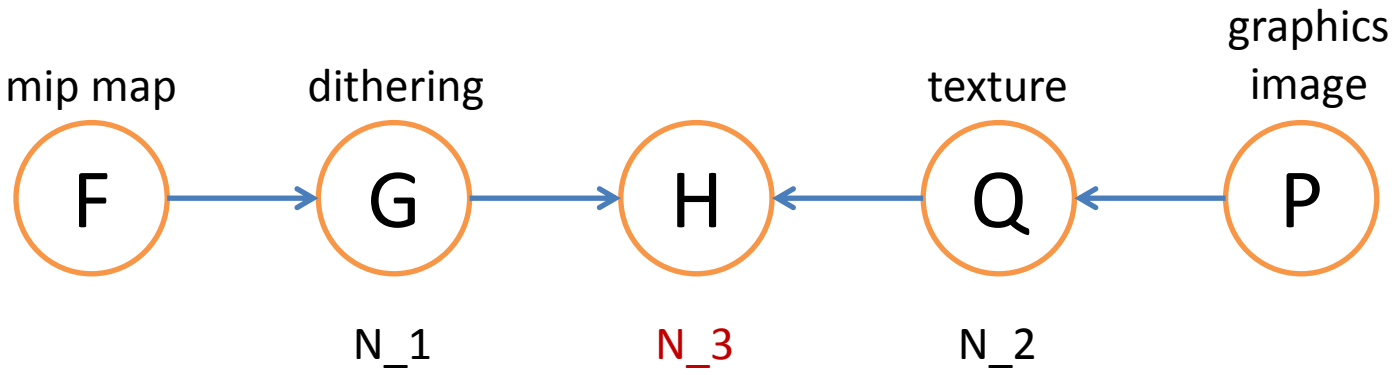
SAN



SAN



SAN



$$N_j = \sum_i f(N_i w_{ij})$$

- H bekommt relativ hohes Knotengewicht!

Kombiniertes Modell

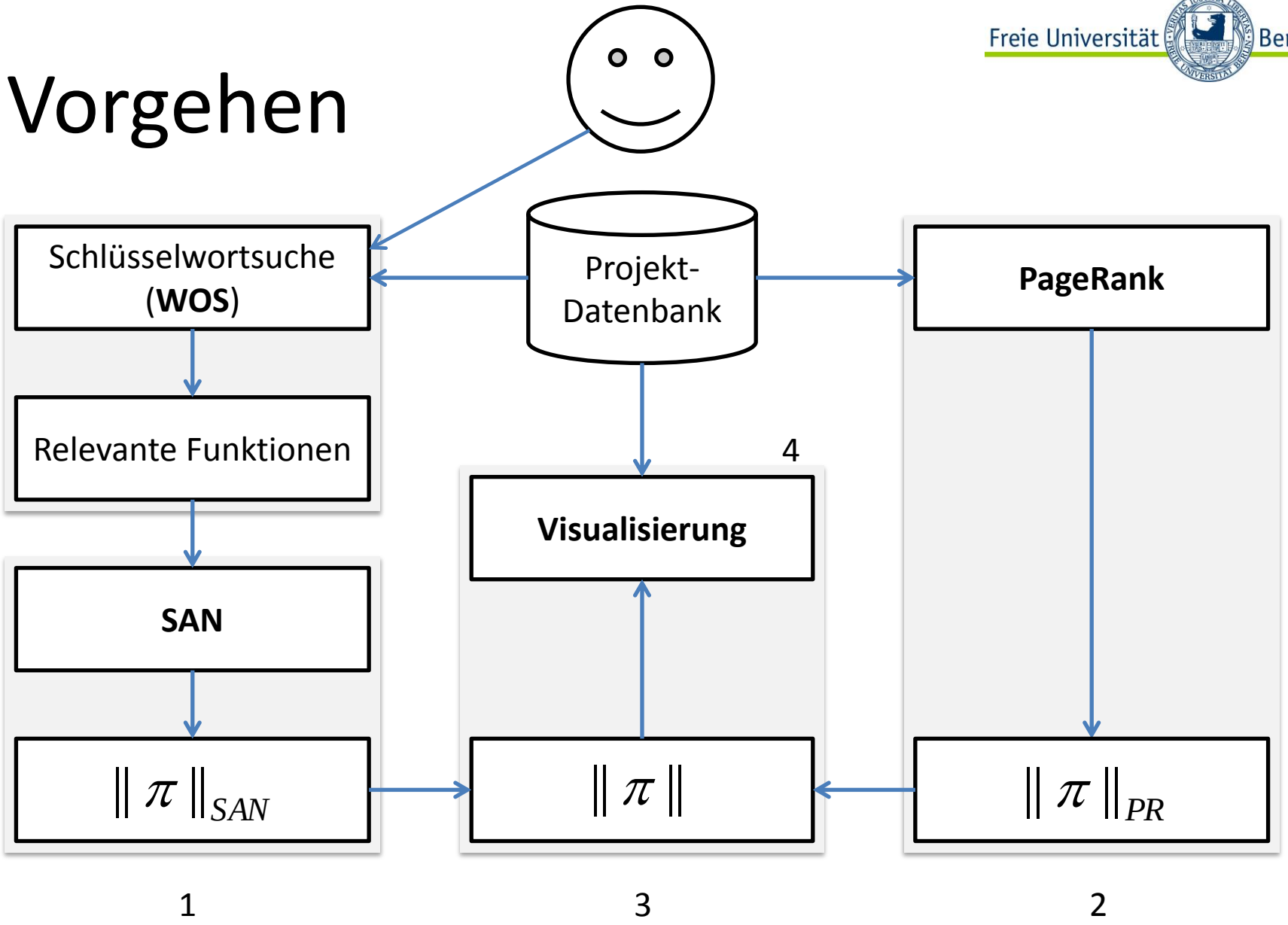
- PageRank liefert ranking-Vektor $\| \pi \|_{PR}$
- SAN liefert spreading activation $\| \pi \|_{SAN}$
  - Beinhaltet die Anzahl geteilter Suchbegriffe

➔ Lineare Kombination $\| \pi \| = f(\| \pi \|_{PR}, \| \pi \|_{SAN})$

Ansatz (von Portfolio)

1. Erweiterte Schlüsselwortsuche
2. Statische Analyse (PageRank)
3. Kombiniertes Modell relevanter Funktionen bezüglich 1 und 2
4. Visualisiere relevante Funktionen

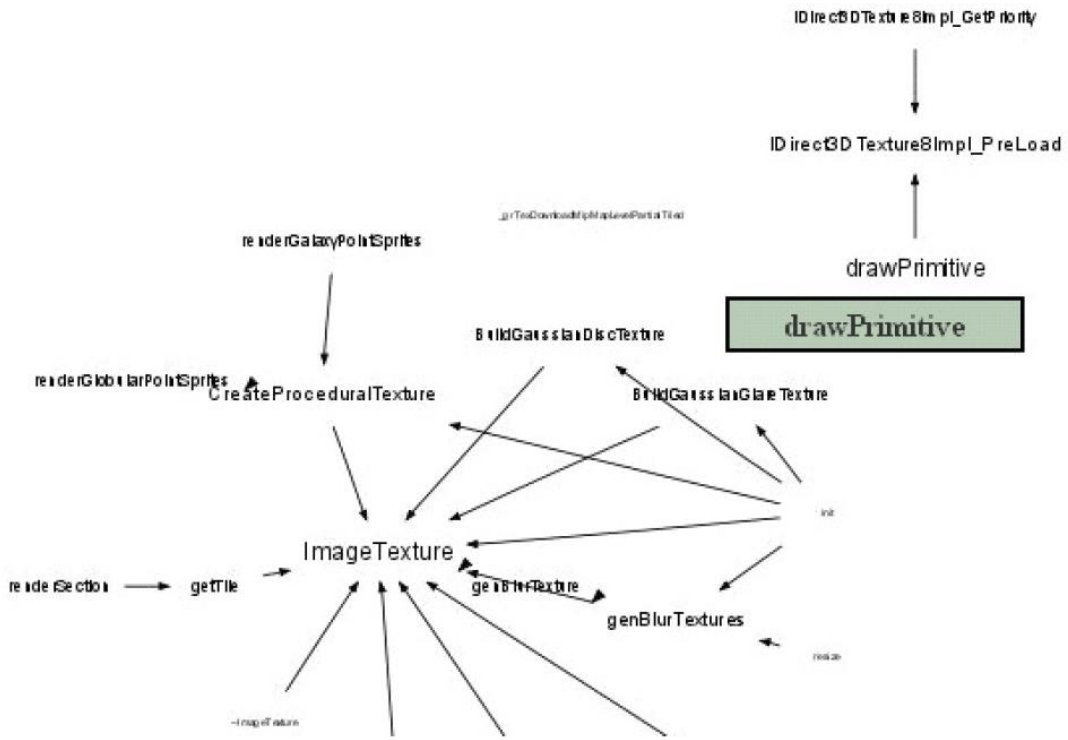
Vorgehen


Schlüsselwortsuche (**WOS**)
Relevante Funktionen
SAN
$\| \pi \|_{SAN}$
Projekt-Datenbank
4
Visualisierung
$\| \pi \|$
PageRank
$\| \pi \|_{PR}$
1
3
2

Visualisierung

mip map dithering texture image graphics

Function Name	Project
ImageTexture	celestia
drawPrimitive	Wine
CreateProceduralTexture	celestia
IDirect3DTexture8Impl_PreLoad	Wine
TiledTexture	celestia
CreateTextureFromImage	celestia
genBlurTextures	celestia
genBlurTexture	celestia
BuildGaussianGlareTexture	celestia
BuildGaussianDiscTexture	celestia
loadTileTexture	celestia
getTile	celestia
renderGalaxyPointSprites	celestia
renderSection	celestia
renderGlobularPointSprites	celestia
IDirect3DTexture8Impl_GetPriority	Wine
_grTexDownloadMipMapLevelPartialTiled	driglide



Zusammenfassung

- Zwei Ansätze um Entwickler bei Einarbeitung in Code zu unterstützen:
- Navigation: Finden relevanter Funktionen
  - Verwendung von Schlüsselwörtern, PageRank und SAN
  - Visualisierung über Graphen
- Verständnis: Beschreibung von Funktionssegmenten
  - Sequenzen, Konditionale, Schleifen
  - mögliche Methodenausgliederung, -umbenennung

Vielen Dank für Ihre Aufmerksamkeit!

References:

• Sridhara et al.: „Automatically Detecting and Describing High Level Actions within Methods“
• McMillan et al.: „Portfolio: Finding Relevant Functions and Their Usages“
• E. Hill: „Integrating Natural Language and Program Structure Information to Improve Software Search and Exploration“