12. Workshop Software-Reengineering

Consistent Consideration of Naming Consistency

Daniel Speicher, Jan Nonnen
{dsp,nonnen}@iai.uni-bonn.de

Institut für Informatik III
Universität Bonn
0. Programs as Texts

Terms in a Program

http://www.flickr.com/photos/pcalcado/3485582094/ created by Phillip Calçado

To access the URLs, you need to replace %e2%80%90 with a normal hyphen.

https://sewiki.iai.uni-bonn.de/research/cultivate/tutorial_exploring_concepts
0. Programs as Texts

Fontsize $\sim$ tf-idf

term frequency – inverse document frequency

- $tf(\text{term, doc}) = \frac{\#\text{occurrences}(\text{term, doc})}{\#\text{occurrences}(\*, \text{doc})}$
- $idf(\text{term}) = \log_2(\frac{\#\text{documents-with}(\*)}{\#\text{documents-with}(\text{term})})$
- If a term is in 2 of 16 documents, it needs $\log_2(16/2)=3$ bit information to find it.
- $tf$-idf$(\text{term, doc}) = tf(\text{term, doc}) \times idf(\text{term})$
- Idea: Salton, Buckley 1988
„Ich hätt´ da mal gern `ne Frage...“
Regional phrase to introduce a question: „I’d like to have a question...“

„Ja, also im Augenblick ... habe ich leider keine dabei.“
Konrad Beikircher
Ironic answer, ignoring the intention of the phrase: „Unfortunately I have none at the moment.“

What are the right research questions to ask?
There are some answers and even questions, but they look rather unrelated to us.
Goal: Quality criteria for naming

• **Example analysis: Naming Dependencies**
  – Detecting where a term is introduced
  – Detecting naming dependencies

• **Preliminary Simple Unified Meta Model of Naming**
  – Derivation of the PreSUMMoN\(^1\)
  – Semi-automatically produces hypotheses
  – Compatibility with current approaches

• *Ihr Kommentar zum Thema und PreSUMMoN*

---

1) Ken Beck on twitter, 26.03.10: „If you can’t find a really good name for something, pick a really terrible name (I like “fred”). Then you are motivated to fix it.” [Minor typos fixed.]
1. Example analysis: Naming Dependencies - Motivation

Refactoring goal: Reuse Planner
Refactoring Strategy: Extract Interface

1. Example analysis: Naming Dependencies - Motivation

Planner

Planner Library
1. Example analysis: Naming Dependencies - Motivation

Names reveal low Design Quality

---

**Planner**
- void schedule(IExhibition[] exhibitions)

**IExhibition**
- CoordinatesList getPreferredCoordinates()
- setAssignedCoordinates(Coordinates)

---

**Exhibition**
- CoordinatesList getPreferredCoordinates()
- setAssignedCoordinates(Coordinates)

---

**Show**
- CoordinatesList getPreferredCoordinates()
- setAssignedCoordinates(Coordinates)
- CoordinatesList getPreferredTimes()
- setAssignedTime(Time)

---

**Lecture**
- CoordinatesList getPreferredCoordinates()
- setAssignedCoordinates(Coordinates)
- List <Room> getPreferredRooms()
- setAssignedRoom (Room)
1. Example analysis: Naming Dependencies - Motivation

What does “Exhibition” mean?

- **Planner**
  - `void schedule(IExhibition[] exhibitions)`
- **Exhibition**
  - `CoordinatesList getPreferredCoordinates()`
  - `setAssignedCoordinates(Coordinates)`
- **Show**
  - `CoordinatesList getPreferredCoordinates()`
  - `setAssignedCoordinates(Coordinates)`
  - `TimesList getPreferredTimes()`
  - `setAssignedTime(Time)`
- **Lecture**
  - `CoordinatesList getPreferredCoordinates()`
  - `setAssignedCoordinates(Coordinates)`
  - `List <Room> getPreferredRooms()`
  - `setAssignedRoom (Room)`

---

universität bonn
1. Example analysis: Naming Dependencies - Motivation

**Term Visualisation (plain)**

- **Planner Library**
  - assigned coordinate
  - exhibition get i list
  - planner preferred
  - schedule set

- **Exhibition**
  - assigned coordinate
  - exhibition get i list
  - preferred set

- **Show**
  - assigned coordinate
  - exhibition get i list
  - preferred set
  - show
  - time

- **Lecture**
  - assigned coordinate
  - exhibition get i
  - lecture
  - list preferred room
  - set
1. Example analysis: Naming Dependencies - Motivation

Term Visualisation (tf)

Planner Library
- assigned coordinate
- exhibition get list planner
- preferred schedule set

Exhibition
- assigned coordinate
- exhibition get list preferred set

Show
- assigned coordinate
- exhibition get list preferred set show
- time

Lecture
- assigned coordinate
- exhibition get list preferred
- room set
1. Example analysis: Naming Dependencies - Motivation

Term Visualisation (tf-idf)
Detecting Naming Dependencies

1. Example analysis: Naming Dependencies – The Detection Rules

Detecting Naming Dependencies

• A type \( p \) named \( n \) introduces a non compound lexical item \( l \):
  – if \( n=l \), or
  – if \( l \) is contained in \( n \) and all other lexical items in \( n \) are already introduced.

[Rule should be extended]

• A program element \( p_1 \) is naming dependent on a program element \( p_2 \), if \( p_2 \) introduces a lexical item that is part of the name of \( p_1 \).
1. Example analysis: Naming Dependencies – Consistency Criteria

=> Consistency Criteria

• *introduces* should be unambiguous: We demand that for each introduced lexical item there is exactly one program element, that introduces it.

• *naming dependencies* should be compatible with the static dependencies: A naming dependency from p1 to p2 is *compatible with* the static dependencies, if there exists a path of static dependencies from p1 to p2.
Good Code and Concepts 1/3

Ron Jeffries: Kent Beck offered the following "rules" for simple design. In priority order, the code must:

1. Run all the tests
2. Contain no duplicate code
3. Express all the ideas the author wants to express
4. Minimize classes and methods

Don Wells: **Refactor mercilessly** to keep the design simple as you go and to avoid needless clutter and complexity. Keep your code clean and concise so it is easier to understand, modify, and extend. **Make sure everything is expressed once and only once.** In the end it takes less time to produce a system that is well groomed.

(Don Wells, http://www.extremeprogramming.org/rules/refactor.html)
Good Code and Concepts 3/3

Adrian Colyer: When you are writing a piece of software it's generally a good idea if you can design the program in such a way that each unique idea, concept, requirement, etc. addressed by the program has a direct and clear representation in the source code. ... Design elements that have a 1-to-1 mapping to an implementation are easy to add, remove, and maintain.

http://www.aspectprogrammer.org/blogs/adrian/2004/05/the_ted_neward.html
Elements AND Relations
PreSUMMoN

2. Preliminary Simple Unified Meta Model of Naming - Derivation of the PreSUMMoN

PreSUMMoN

Static Relation

Lexical Relation

Program Element

Lexical Item

Concept

Conceptual Relation

name

{ordered}
2. Preliminary Simple Unified Meta Model of Naming - Derivation of the PreSUMMoN

Program Space

Elements and relations describing static code dependencies

- **Elements:**
  - packages, types, fields, methods, statements, expressions, ...

- **Relations:**
  - calls, uses, extends, implements, depends on, control flow, data flow, ...
Lexical space:
Program element names and parts of these names

- **Elements:**
  - nouns, verbs, adverbs, adjectives, abbreviations.

- **Relations:**
  - Rules for building compounds, fragments of sentences, sentences, abbreviations, derivations, and textual length.
Concept space
Model of the programmer's mental concepts

- **Elements:**
  - domain, domain entities, design pattern, roles in design pattern, architectural pattern, roles in architectural pattern, functional and non-functional requirements, metaphors...

- **Relations:**
  - is a, is part of, is antonym of, is similar to, belongs to domain...
Generic Consistency 1/3

Unambiguous mappings  As we saw it is sometimes required that a mapping from one space to another is unambiguous, i.e. that there is exactly one image for each source.
Generic Consistency 2/3

Compatible relations  Given relations $R_1$ and $R_2$ in the program, lexical, or concept space. With a mapping $\varphi$ between two spaces, we call the relation $R_1$ compatible (under $\varphi$) with the relation $R_2$ on the elements $e_1$ and $e_2$, iff $e_1 R_1 e_2 \implies \varphi(e_1) R_2 \varphi(e_2)$. 
Correlated characteristics Considering partial orders $\succeq$ or equivalence relations $\approx$, we can find mappings $\psi$ into the real numbers, such that $e_1 \succeq e_2 \iff \psi(e_1) \geq \psi(e_2)$ or $e_1 \approx e_2 \iff \psi(e_1) = \psi(e_2)$ respectively. In these cases we can use correlation measures to assess the degree of compatibility between two relations.
Semi-automatically hypotheses production

- Hard: Find candidates for the mappings

- Easy: Take our generic consistency criteria as a template and create a specific criterion.

- Considering the amount of element and relation types this should yield many hypotheses.
Examples

• K. Beck: Same Level of Abstraction Principle
• R.C. Martin: Conciseness corresponding to scope
• J. Singer, C. Kirkham: Type postfixes - Correlation between type name and micro patterns.
• Correlation between „information“ in method names and method bodies.
Examples

• Florian Deißenböck, Daniel Ratiu. How programs represent reality (and how they don’t). *WCORE’06: Proceedings of the 13th Working Conference on Reverse Engineering 2006*

  Uses graph matching to map concepts in an ontology to program elements.
Ihr Kommentar zum Thema und PreSUMMon?

• Typische Verstehensprobleme beim Reengineering?
• Untersuchenswerte Codebasen?
• Kriterien für gute Benennung?
• Anregungen durch unseren Ansatz?