Identifier Splitting for SBSE
Software Maintenance

Kevin Graham

Supervisors:
Prof. Edmund Burke
Dr. Jerry Swan
SBSE: Search-based Software Engineering

- Formulating common software engineering problems in terms of a search-based optimisation problem
  - Many competing and complex objectives
  - Large search spaces

- Search based software engineering research has gained much interest in recent years [10]
  - Improvements to software engineering processes
  - Quality of code produced
Software Maintenance

- **60/60 rule exists in software development**
  - Maintenance costs dominate development costs at around 60% [8]
  - Further, 60% of maintenance costs consumed by enhancement alone [8]

- **Steps therefore must be taken to reduce maintenance costs over the lifetime of a project**
Identifiers

- Identifier names in source code are natural language tokens used to label program entities
  - Including; variables/constants, methods/functions, classes and objects in OOP etc.

- Notationally, they can be described as a character sequence ‘C{c₀, ..., cₙ}’ [11]
  - Where a character cᵢ can be a letter, digit or special character
Identifiers

- The natural language of identifier names captures:
  - Domain-specific knowledge
  - Clues to programming intention

- Much research exists exploring the role identifiers and the language used for the human comprehension of source code
  - E.g. concise and consistent naming [4]
Identifiers: Composition

- Programming languages and programming conventions place constraints on form and content of identifier names [1]
  - Programming languages impose hard constraints
    - You can be sure constraints are followed
  - Programming conventions impose soft constraints
    - Not applied universally

- Fully Autonomous techniques should be able to handle both conventionally and unconventionally constructed identifiers [1]
Identifier Splitting Problem

- Defined as the identification of the component words in identifiers, where:
  - No explicit boundary
  - Present, but open to misinterpretation

- Involves the solution to two sub-problems
  - Mixed-case tokenisation [5,11]
  - Same-case tokenisation [5,11]

- Often extended to include abbreviation expansion
  - E.g. HTTP = \{hyper, text, transfer, protocol\}
Mixed-case

- Involves the tokenisation of identifiers containing internal capitalisation boundaries \([1,5]\)
  - Lower to upper case (LCUC) e.g. `getIdentifier`
  - Upper to lower case (UCLC) e.g. `XMLParser`

- **LCUC transitions readily tokenised** \([1, 5]\)
  - Well defined boundary between words i.e. easily identified
  - Conventional

- **UCLC has two possible boundaries** \([1, 5]\)
  - `GPSstate` → `{GP, Sstate}` according to camel-casing
  - `GPSstate` → `{GPS, state}` according to alternative split
Mixed-case: UCLC

- Has been addressed (somewhat) using dictionary-based scoring metrics
  - Samurai Identifier Splitter [enslen]
  - Identifier Name Tokeniser Tool (INTT) [1]

- But what about…
  - Unconventional acronyms e.g. ‘connectDnDServer’
  - Acronyms/words with bounding digits e.g. ‘POP3Server’
    - *Pop3Server* doesn’t solve this problem
    - What about ‘*Cad3DModel*’?
  - Ambiguous boundaries e.g. ‘*GPRSend*’
Sub-problem addresses the tokenisation of same-case multi word identifiers [1, 5]

- E.g. MAXVALUE or thenewestone

Existing techniques mainly rely on words being found in a dictionary

- Sometimes more than one dictionary e.g. abbreviations
- But no dictionary can ever be complete
  - Invented words for some new concept
  - Neologisms e.g. ‘devoidify’, ‘detokenated’, ‘pathinate’, ‘precisify’ [1]
Motivation

- **Tools used to aid program comprehension**
  - Particularly during maintenance tasks

- **Many existing analysis tools don’t leverage natural language**
  - Arguably, domain knowledge and programming intention is more informative to a programmer

- **Tools have emerged that exploit this natural language information** [12]
  - Natural Language Program Analysis (NLPA)
Motivation

- To process natural language, accurate tokenisation is a necessary first step
  - Words have individual meanings and probably have a different or no meaning while un-tokenised

- Incorrect tokenisation at this stage can propagate errors to later processes during program analysis [1]
Tokenisation for Humans

- For us the notion of splitting identifiers seems easy:
  - E.g. tokenise: ‘yankeedoodlewenttolondonridingonapony’

- We have the advantage of higher-level strategies on our side
  - We have many a mental dictionary, an idea of probable word frequencies, n-gram frequencies, letters never seen together, context checking etc. etc.
  - We can apply all of these or only some of these to give evidence for a split point
  - We can even invent/analogise new methods from old ones
  - We can adapt to new information and change our conclusions and approach accordingly

- For identifier splitting, we need our approaches to do the same
Proposed Solution: Blackboard Architecture

- A multi-agent problem solving architecture
  - Opportunistic Problem Solving Model

- First developed as a means to recognise vocal speech
  - Hearsay II [6]

- Involves 3 main components:
  - Agents/Knowledge Sources
  - Blackboard/Global Workspace
  - Control Mechanism
Blackboard Architecture: Metaphor

The first week of the brainstorming session went slowly.

Courtesy of www.gograph.com
Blackboard Architecture: Characteristics

- Modularity
- Diversity of Problem Solving Techniques
- Flexible Representation of Information
- Common Interaction Language
- Event-driven Triggers
- Need for a Control Mechanism
Blackboard Architecture: Benefits to Tokenisation

- Components can be applied at the most opportune time
  - Some operations best performed before others
    - E.g. for ‘connectDnDServer’
- Heterogeneous components and representations can easily collaborate in a blackboard system
  - E.g. Neural net splitter [7] and graph-based tokenisation & abbreviation expansion [9, 3]
- Supports concurrency of problem solving processes
  - On multiple levels of speciality and granularity
End of the Line

THANKS FOR LISTENING
References


References 2


References 2 ²/₅


Example

blackboard

Controller

Add()  Subtract()  Multiply()

agents
Example: Iteration 1

blackboard

solution 0  target 120

Random # (1-10)
6

Add()  Subtract()  Multiply()

agents

Controller
Example: Iteration 1

blackboard

solution 0
target 120

Random # (1-10)
6

Add()
Subtract()
Multiply()

s:0 t:120 r:6

agents

Controller
Example: Iteration 1

- **blackboard**
  - **solution**: 0
  - **target**: 120
  - Random # (1-10): 6

- **Controller**

- **agents**
  - Add()
    - s:0 t:120 r:6
  - Subtract()
    - s:0 t:120 r:6
  - Multiply()
Example: Iteration 1

blackboard

solution
0

target
120

Random # (1-10)
6

Controller

Add()

Subtract()

Multiply()

s:0 t:120 r:6

agents

s:0 t:120 r:6

s:0 t:120 r:6
Example: Iteration 1

blackboard

solution 0
target 120

Random # (1-10)
6

Controller

add(): score = 114
mul(): score = 120

Add()

s:0 t:120 r:6

Subtract()

s:0 t:120 r:6

Multiply()

s:0 t:120 r:6

agents
Example: Iteration 1

Controller

Random # (1-10)
6

solution
6

target
120

blackboard

add(): score = 114
mul(): score = 120

call()
Example: Iteration 2

- **blackboard**
  - solution: 6
  - target: 120
  - Random # (1-10): 9

- **Controller**

- **agents**
  - Add()
  - Subtract()
  - Multiply()
Example: Iteration 2

- **blackboard**
  - **solution** 6
  - **target** 120
  - **Random # (1-10)** 9

- **agents**
  - **Add()**
  - **Subtract()**
  - **Multiply()**

- **Controller**
  - add(): score = 105
  - mul(): score = 66
Example: Iteration 2

- **blackboard**
  - solution: 66
  - target: 120
  - Random # (1-10): 9

- **agents**
  - Add()
  - Subtract()
  - Multiply()

- **Controller**
  - add(): score = 105
  - mul(): score = 66

- **Call()**
Example: Iteration 3

Blackboard:
- Solution: 66
- Target: 120
- Random # (1-10): 2

Add(): score = 52
Mul(): score = 12

Agents:
- Add()
- Subtract()
- Multiply()

Controller
Example: Iteration 3

blackboard

solution 132
target 120

Random # (1-10) 2

add(): score = 52
mul(): score = 12

Controller

agents

Add() Subtract() Multiply()
Example: Iteration 4

blackboard

solution 132
target 120

Random # (1-10)
10

sub(): score = 2

Controller

Add()
Subtract()
Multiply()

agents
Example: Iteration 4

blackboard

solution 122  
solution target

122 120

Random # (1-10)

10

Controller

sub(): score = 2

Call()

Add() Subtract() Multiply()
Example: Iteration 5

blackboard

solution 122

Random # (1-10)

3

target 120

sub(): score = 1

Controller

agents

Add()

Subtract()

Multiply()
Example: Iteration 5

- **Solution**: 119
- **Target**: 120
- **Random #:** (1-10) 3
- **Controller**: Random # (1-10) 3
- **Agents**: Add(), Subtract(), Multiply()
- **Call()**: sub(): score = 1
Example: Iteration 6

- **Blackboard**
  - Solution: 119
  - Target: 120
  - Random #: 1

- **Agents**
  - Add()
  - Subtract()
  - Multiply()

- **Controller**
  - add(): score = 0
  - mul(): score = 1
Example: Iteration 6

**blackboard**

- solution: 120
- target: 120
- Random # (1-10): 1

**Controller**
- add(): score = 0
- mul(): score = 1

**agents**
- Add()
- Subtract()
- Multiply()