

TABASCO: a Taxonomy-based Domain Engineering Method

Loek Cleophas

Bruce W. Watson
Derrick G. Kourie
Andrew Boake

SoC Software
Construction

Software Construction Group
Technische Universiteit Eindhoven
The Netherlands
<http://www.win.tue.nl/soc>



espresso
Espresso Research Group
University of Pretoria, South Africa
<http://www.espresso-soft.org>

FASTAR

FASTAR Research Group
University of Pretoria, South Africa
<http://www.fastar.org>

Overview

- Context
 - Application & Domain Engineering
 - Generative Programming
- TABASCO
 - Outline
 - Related Work
- Taxonomy Construction
 - Taxonomies vs. Feature Models
- Toolkit Design
 - Example: SPARE TIME
- DSL Design & Implementation
 - Examples: Sorting components, FUEL & SPARE FUEL
- Final Remarks

TABASCO: a Taxonomy-based Domain Engineering Method
SAICSIT 2005, White River, South Africa, September 20 – 22, 2005

Application vs. Domain Engineering

- Most Software Engineering is Application Engineering
 - focus on single-system development
 - not well-suited for reuse
- Domain Engineering
 - emphasizes reuse by developing generic software
 - instantiation into concrete systems
 - reuse of parts in different systems

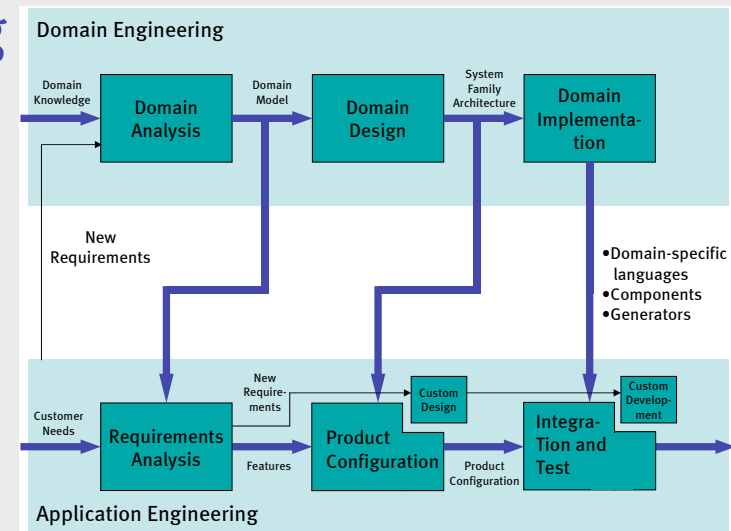
“Domain Engineering is the activity of collecting, organizing and storing past experience in building systems or parts of systems in a particular domain in the form of reusable assets, as well as providing an adequate means for reusing these assets when building new systems”

[Czarnecki & Eisenecker 2000]

Main idea: capture *domain knowledge* and have means for reuse, to reduce production cost and increase quality

Domain Engineering

- Domain analysis
 - scoping
 - modeling
 - based on commonalities/variabilities analysis
- Domain design
 - system family architecture
 - should be flexible
 - based on experience and creativity
 - production plan: how to build concrete systems
 - manual, tool-supported, fully automated
- Domain implementation



(based on [Czarnecki & Eisenecker 2000, SEI 1997])

Generative Programming & Domains

- Seeks to automate creation of products
- Domain model must be *generative*
 - capturing *configuration knowledge*:
 - specifies validity of feature combinations, defaults, construction rules, optimizations, ...
- Domain design includes mapping from user requirements to implementation components
 - allows automatic generation based on user requirements
- Domain implementation phase implements configuration knowledge and mapping in *generator*

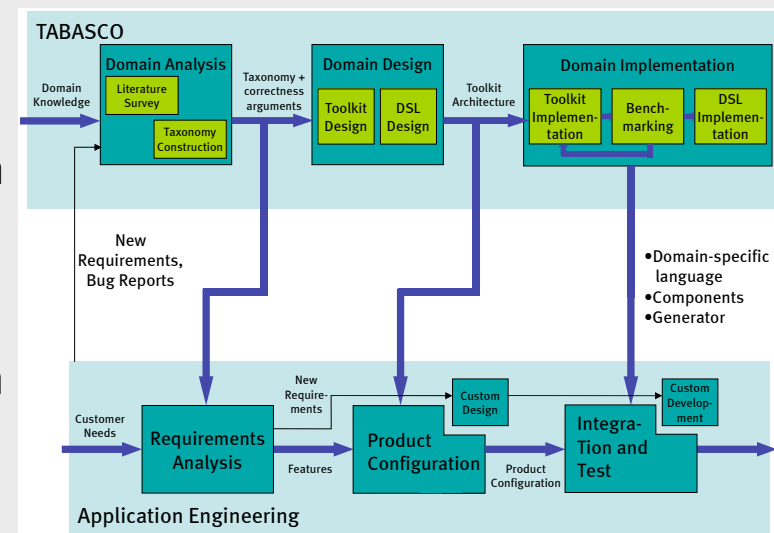
TABASCO – I

- Method for developing reusable software components in form of *Domain Specific Toolkit* (DST)
- Domain engineering: often large and complex business software families, third-party developers involved → high risk
- Reduce risk → restrict form of domains
 - family of related algorithms or data structures
 - little third-party involvement
- Similar to DEMRAL [Czarnecki 1998], but
 - domain model not as formal there
 - no direct pedagogic aim in DEMRAL
 - not as correctness-oriented as TABASCO

TABASCO – II

Process consists of multiple steps:

1. Selection of domain
2. Literature survey
3. Taxonomy construction
4. Toolkit design
5. DSL design
6. Toolkit implementation
7. Benchmarking
8. DSL implementation



Main goals: create taxonomy to bring order to domain/
improve understanding, create toolkit of reusable
software components

TABASCO — Related Work

- Taxonomy development and algorithm derivation
 - garbage collection [Jonkers 1983]
 - attribute evaluation [Marcelis 1990]
- TABASCO
 - *Taxonomies and Toolkits of Regular Language Algorithms* [Watson 1995]
 - *TABASCO of SPARE TIME: a Case Study* [Cleophas & Watson 2005]
- (Partially) applied to various domains, e.g.
 - keyword pattern matching: exact & approximate
 - finite automata: construction, minimization
 - Lempel-Ziv compression
 - graph representations

Taxonomy Construction — I

A taxonomy is “[an] orderly classification of plants and animals according to their presumed natural relationships”
[Mish 1993]

- Classification/taxonomy based on essential details of algorithms (or data structures) is possible as well
- Main goal: improve understanding of algorithms and their relations, i.e. commonalities and variabilities
- Root is high-level algorithm solving the problem
 - formal requirements associated with it: pre- and postcondition, invariant, specification of running time/ memory usage
 - other algorithms derived from root by adding *details*

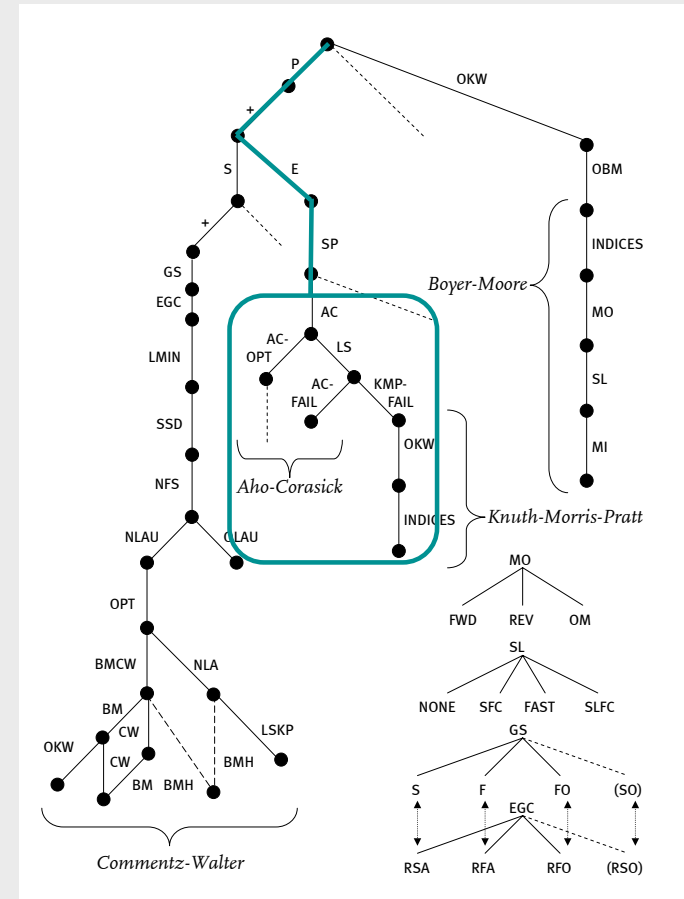
Taxonomy Construction — II

- Addition of detail
 - results in refined algorithm for same/similar problem
 - goal: arrive at algorithms from literature
 - may lead to new or improved algorithms
 - accompanied by correctness arguments
- Detail categories
 - *problem details*; change pre/post
 - *algorithm details*; change algorithm structure
 - *representation details*
 - *performance details*
- First three categories most important in taxonomy
- Fourth very important for toolkit user → mapping needed: *Domain-Specific Language (DSL)*

Taxonomy Construction — III

- Detail choice and order depend on personal preference & domain understanding
- Inclusion of different orders for single algorithm leads to DAG vs. tree
 - sequence of details leading to algorithm used to indicate algorithm

Example: *keyword pattern matching (kpm)* taxonomy



Taxonomies & Feature Models

- In [Czarnecki & Eisenecker 2000], feature models are used as domain models
- Feature model consists of
 - feature diagrams, depicting variable and common features
 - configuration knowledge, consisting of constraints, defaults, optimizations
- Choosing taxonomy construction or feature modeling depends on domain
 - working of algorithms not too complex, relationships well known e.g. *sorting algorithms* → feature model
 - algorithms more complex, relationships not clear → taxonomy construction
 - feature modeling not well-suited for describing correctness arguments, algorithm relationships

Advantages of Taxonomy Construction

- Makes algorithm comparison easier
- Gives clear and correct algorithm presentation
- Brings order to a field, result usable as teaching aid
- Well suited for discovering new algorithms
- Gives formal specification of algorithms
- Availability of taxonomy aids in construction of DST

Disadvantages (vs. feature modeling)

- Takes more time and effort
- Might be overkill for some domains

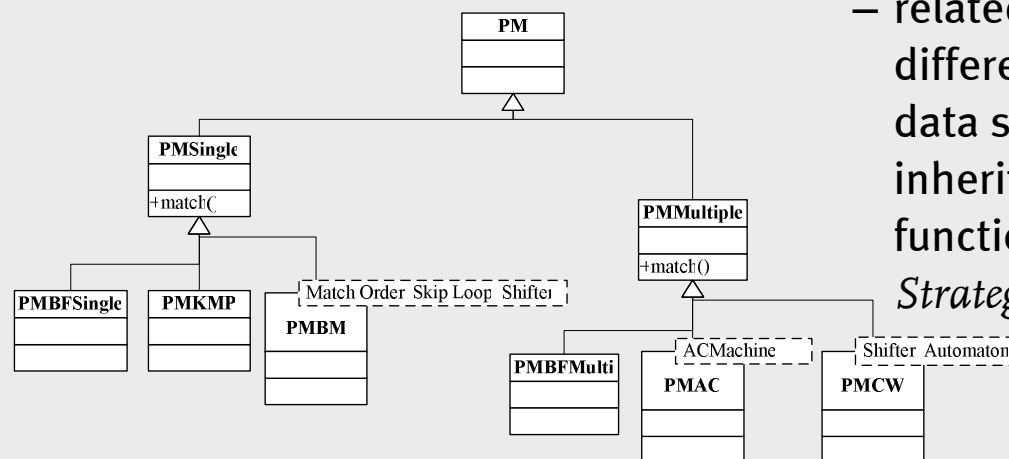
Toolkit Design

- Domain design not straightforward, requires experience and creativity
- TABASCO makes domain design more straightforward:
 - domains consist of closely related algorithms
 - taxonomy relates algorithms → makes commonalities and variabilities explicit
- Get to high-level design using *multi-paradigm design* [Coplien 1998]
 - maps commonality/variability types (*name, behavior, data structure, fine or gross algorithm, defaults, state values*) to (C++) language constructs (*inheritance, overloading, virtual functions, containment, structs, enumerations*) or to design patterns (*Strategy, Template Method, Singleton, Bridge, Adapter*) [Gamma et al. 1995]

The Design of SPARE TIME — I

Example: toolkit based on *kpm* taxonomy

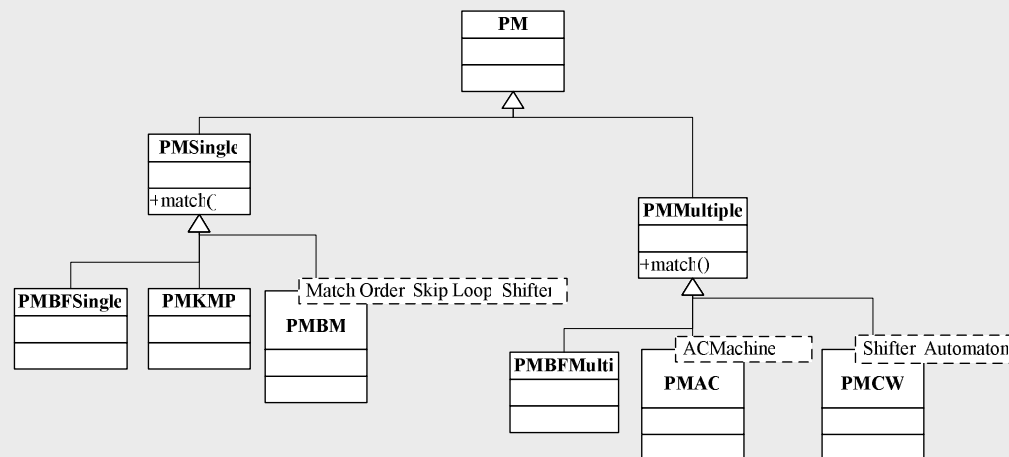
- Algorithms for similar problems, multiple and single *kpm*
- Slightly different interfaces → classes *PMSingle* and *PMMultiple* deriving from abstract class *PM*
- Approaches for multiple *kpm* (brute-force, Aho-Corasick, Commentz-Walter)



– related operations,
difference in algorithm,
data structure, state →
inheritance with virtual
functions; corresponds to
Strategy design pattern

The Design of SPARE TIME — II

- Three variants of Aho-Corasick (failure function, goto function, multiple-keyword Knuth-Morris-Pratt)
 - share algorithm, differ in state update/automaton → template class with automaton parameter; similar to *Template Method* design pattern
- Implementation straightforward, given algorithm derivations in common, abstract format



Advantages of Taxonomy-based Toolkit Design

- Taxonomy availability simplifies design task
- Taxonomy homogeneity leads to homogeneity of design, interface → easier to understand
- Taxonomy leads to easier to understand/debug code
- Uniformity of style gives confidence in accuracy of implementations' relative performance
- Taxonomy and toolkit can serve as example of toolkit design & implementation techniques
- Correctness arguments in taxonomy give confidence in toolkit correctness and safety
- Taxonomy gives formal specification of requirements satisfied; helps in understanding components, creating mapping from user requirements to components

DSL Design & Implementation – I

- Difference between user requirements and what toolkit components provide
 - makes effective and easy use hard for average user
- Overcome by development of DSL or *‘little language’*
- DSL tailored towards (domain) concepts average user is interested in or familiar with
 - Allows user to specify requirements to be satisfied by requested component, or information usable in selection of toolkit component
 - Usually focused on *performance* and *problem details*, toolkit components focused on *algorithm* and *representation details*
- *Mapping* from DSL expression to toolkit component
 - based on configuration knowledge

DSL Design & Implementation – II

- Information on valid component combinations, construction rules from domain model (taxonomy)
- Information on defaults, efficiency from benchmarking
- Different kinds of DSL
 - implicit: textual advise on which components to use when
 - explicit: allows user to compose DSL expression, automatically processed to select/generate component
- Generator may be generic: taking DSL description, DSL expression, generating appropriate component

Example: DSL for specifying sorting components

- Allows specification of array size, variance in values, worst case running time, ...

DSL Design & Implementation – III

Example (*cont'd*): DSL for specifying sorting components

- DSL and mapping implemented using *C++ template metaprogramming*
 - compares parameters to decision table at compile time
 - selects component based on first row matching
 - other techniques possible (*scripting, makefiles*)

Example: generic DSL tool (FUEL) & DSL for obtaining *kpm* components (SPARE FUEL)

- FUEL experimental, takes DSL description and expression, selects/generates component
 - Implemented in *Ruby, XML* for DSL description files
- SPARE FUEL based on SPARE TIME toolkit, *kpm* taxonomy
 - Allows specification of alphabet, keyword set size, minimal keyword length, ...

DSL Design & Implementation – IV

```
...
<codefiles>
  <file src="sparefuel.src" target="sparefuel.h"/>
</codefiles>
<algorithms>
  <alg id="CWNFS" name="CW-[NFS, Factoracle]"
      classname="PMCW<CWShiftNFS, RFactoracle">">
    <param>cw/cws.hpp</param>
  </alg>
</algorithms>
<rules>
  <rule name="alphabet" value="English" default="CWNFS">
    <rule name="memory" value="high" default="CWNFS">
      <rule name="length" value="3" default="ACOpt">
        <rule name="setsize" value="1" default="CWNFS"/>
      </rule>
    </rule>
  </rule>
</rules>
...
```

```
./fuel.rb sparefuel alphabet=English memory=high length=3 \
setsize=1
```

```
/* Chosen algorithm: CW-[NFS, Factoracle]
   Parameters: alphabet=English, memory=high, length=3, setsize=1 */
#include "cw/cws.hpp"
typedef PMCW<CWShiftNFS, RFactoracle> PMAlg;
```

Concluding Remarks

- We presented TABASCO in the context of Domain Engineering and Generative Programming
- Important TABASCO steps were considered in detail
 - giving examples of their application
 - highlighting the benefits of such steps and TABASCO
- TABASCO is domain engineering method
 - for a restricted form of domain, with little third-party involvement
 - aimed at constructing a taxonomy (to bring order and improve understanding) and a DST (to enable reuse of components)
 - taxonomy makes toolkit design easier
 - DSL helps toolkit users

Future Work

- Application of TABASCO to other algorithmic or data structure fields
 - should lead to more explicit and general guidelines on getting from domain model to domain-specific toolkit
 - should provide feedback on usability and effectiveness of TABASCO for domain engineering
- Apply experimental tool FUEL to other domains
 - should indicate whether FUEL is general enough for any algorithmic or data structure field, or needs to be amended