Hypothesis-Based Concept Assignment to Support Software Maintenance

Nicolas Gold
Department of Computation
UMIST, UK
N.E.Gold@co.umist.ac.uk

Keywords: Software Maintenance, Program Comprehension, Concept Assignment

Abstract

Software maintenance is typically the most expensive part of the software lifecycle, with program comprehension forming the most costly part of software maintenance. This paper outlines a method for assisting program comprehension by addressing the concept assignment problem. The method, termed Hypothesis-Based Concept Assignment, uses informal information contained within source code to reason plausibly about the concepts contained within the code. An extensive evaluation has shown that the method can accurately recognise concepts in a range of real-world programs.

1. Introduction

Software maintenance typically accounts for at least 50 percent of the total lifetime cost of a software system [7]. It is desirable to reduce the cost of this activity whilst preserving the quality of the software system, maintenance process, and maintainer’s understanding. Software understanding is one of the largest cost factors in software maintenance and a method is presented by which the cost of understanding may be reduced by assisting the maintainer.

The method, Hypothesis-Based Concept Assignment (HB-CA), addresses the concept assignment problem: the process of assigning descriptive terms to their implementation in source code, the terms being nominated by a maintainer and usually relating to computational intent [2]. By assigning descriptive terms to regions of source code, the maintainer can expend less effort in gaining an understanding of a particular source program.

The approach is computationally inexpensive having linear computational growth in the length of the source under analysis. It performs concept assignment primarily using informal information such as identifier names and comments, and adopts a novel approach to analysing monolithic and poorly structured code, using a self-organising neural network to associate similar and nearby concept hypotheses rather than relying entirely on their syntactic position.

2. The Concept Assignment Problem

The term concept assignment was introduced by Biggerstaff et al. to describe the problem of assigning terms regarding computational intent to appropriate regions of source code [1]. A concept is defined as a descriptive term at a higher level of abstraction than the source code. Concepts are nominated by the maintainer and can be broadly divided into those related to the software engineering domain (e.g. Read or Write:File), or those related to the application domain (e.g. Pay:Fees or Produce:Cheque). The method presented here does not make any distinction between these classes of concept, allowing the maintainer to work at a level of abstraction suitable for their needs.

Two major research issues can be identified within the overall concept assignment problem:

- **Segmentation**: finding the location and extent of concepts in the source code.
- **Concept Binding**: determining which concepts are implemented at these locations.

Segmenting a program involves grouping pieces of conceptual information generated from the source code.

Concept binding involves analysing these groups for the most plausible concept assignment for each.

The emphasis of this work is on automatic concept assignment with minimal user involvement, although the activity can also be performed semi-automatically or manually. HB-CA is an example of a plausible reasoning concept assignment system. This approach was adopted as plausible reasoning systems tend to have linear computational growth in the length of the source code under analysis [1]. Two other examples of plausible reasoning concept assignment systems are DM-TAO (part of the DESIRE toolset) [1], and IRENE [4]. These
systems adopt different approaches to HB-CA; DM-TAO has a complex knowledge base and inference engine driven by a connectionist network, IRENE uses rule-based concept acquisition techniques to retrieve business knowledge from COBOL programs.

3. Hypothesis-Based Concept Assignment

The Hypothesis-Based Concept Assignment (HB-CA) method (see [2]) is a three-part non-interactive process. It operates on the procedure division of IBM COBOL II programs (although a complete program is provided as input). The procedure division provides enough information to obtain useful results although the data division may be included in future work.

The three stages of HB-CA are Hypothesis Generation, Segmentation, and Concept Binding. The flow of control and data is sequential.

The process begins with hypothesis generation from source code. This is followed by segmentation of the hypotheses to determine regions of conceptual focus in the program. Finally, concept binding finds the dominant concept in each segment.

Each stage uses a knowledge base termed the library.

3.1 Knowledge Base (Library)

It is anticipated that the maintainer, or some other person with domain and software engineering knowledge will construct the library initially. The knowledge base content would then be improved as HB-CA is used.

There are two entities in the library: concepts, and indicators. Concepts are the terms nominated by the maintainer to describe items or activities in the domain. Indicators are evidence for concepts expressed in the implementation language, in this case IBM COBOL II.

The library encodes two types of relationship: Indicator-Concept, and Concept-Concept. The indicator-concept relationship maps evidence for a concept to that concept. Concept-concept relationships map evidence for a concept to that concept. Concept-concept relationships map concepts to others to form composites and specialisations.

3.2 Hypothesis Generation

The hypothesis generation stage takes source code as its input. Using information contained in the knowledge base, the source code is analysed for indicators of various concepts. When an instance is found and matched, a hypothesis for the appropriate concept is generated. Matching is performed using a variety of flexible criteria. The resulting collection of hypotheses is ordered by the position of the indicators in the source code.

3.3 Segmentation

The segmentation stage takes the sorted hypotheses and attempts to break them into segments. Initially, this is performed using hypotheses for primary segmentation points (COBOL II section boundaries). Each of the initial segments is analysed to determine whether it has the potential to contain a number of smaller segments. If this is the case, a self-organising map is used to establish areas of conceptual focus within the segment (see [2,3]).

Self-organising maps (SOMs) (see [6]) are a form of neural network employing unsupervised, competitive learning. They perform a topological mapping of high-dimensional input data to a low-dimensional output space. If required, the hypotheses generated in the first stage of HB-CA are translated into a high dimensional vector format and clustered on the output nodes of a SOM. These clusters are analysed and, if sufficiently large (valid), then form the basis for new, smaller segments. Clusters that are not large enough (invalid) are merged with those that are valid. The output of the stage is a collection of segments, each containing a number of hypotheses.

3.4 Concept Binding

Concept binding analyses each segment’s hypotheses to determine which concept has the most evidence. It exploits relationships in the knowledge base to generate conclusions, and scores these on the basis of concept occurrence. A number of disambiguation rules can be applied to choose between equally strong concepts. When a concept has been selected, the segment is labelled with the name of that concept. After all segments have been analysed and labelled, the results form the overall output of the method.

3.5 Implementation

HB-CA has been implemented using Borland’s Delphi environment in a system termed HB-CAS. Self-organising map processing is provided by the SOM_PAK suite [5]. HB-CAS has separate modules for each part of the process. Data is passed between them using Windows .INI files as these provide a structured, multi-attribute data representation.

4. Evaluation

HB-CA has been evaluated on a wide range of criteria (see [2]), using real-world COBOL II source programs. 22 programs were selected from a set of 150 drawn from a real-world financial services payment system. The only
criteria were to include a range of sizes (about 20-1500 lines). A library of 23 concepts was used, each with 1-6 indicators. HB-CAS was used for analyses where application of the method was required but the theoretical aspects are based on the description in [2]. The results are summarised below.

4.1 Scalability

Scalability refers to the ability of HB-CA to maintain its accuracy on any size of program. In theory, if HB-CA can be accurate on one segment, it can be accurate on any number and thus any size of program. The evaluation showed however, that as program size increases to 1000 lines or more, HB-CA’s accuracy falls. Investigation showed this to be the result of the segmentation algorithms merging valid and invalid clusters and thus reducing the quality of evidence. The situation only arose when a self-organising map was employed for segmentation. A full description of the investigation can be found in [2,3]. HB-CA’s mean accuracy was found to be 84-88% depending on the options used.

4.2 Segmentation and Concept Binding Issues

On the whole, flexible segmentation using the self-organising map is successful, however there are occasions when too much segmentation takes place and a routine that should be analysed as one segment becomes separated into several, each with the same assigned concept. This occurs when the map learns an area of difference as well as similarity (i.e. an area is different to its surroundings because it contains different hypotheses, not similar ones). This problem could be corrected by aggregating adjacent segments with the same concept binding.

Concept binding was evaluated with respect to the effectiveness of the disambiguation rules. In most cases the rules proved to be extremely effective, requiring an arbitrary choice in only about 10% of cases.

4.3 Computational and Spatial Cost

HB-CA’s computational cost was investigated using module execution times from HB-CAS to represent the relative complexity of the method stages. The results indicated a linear increase in computational cost with the length of the source code being analysed. Detailed analysis revealed that although this is true for the hypothesis generation stage, segmentation and concept binding can be non-linear. The non-linear parts are dominated by the hypothesis generation and thus the overall cost appears linear. Spatial costs are very similar, hypothesis generation causing linear growth with the length of the source being analysed, and the remainder of the method being non-linear.

4.4 Library Content and Representational Power

Generally, it was found that HB-CA performed best when the indicators in the library were not shared between concepts. This reduces confusion in the segmentation and concept binding stages.

The library’s representational power can be considered in terms of its ability to represent concepts and indicators. It was originally designed to store textual tokens and would require significant modification to represent other types of information e.g. structured plans. This allows it to represent a wide variety of concepts and indicators but constraints (e.g. “only one person can reserve one seat”) and domain specific relationships cannot be modelled.

4.5 Expandability

HB-CA was designed to incorporate many types of indicator in its analysis. Since the main data structure is an ordered list of hypotheses, any indicator recognition method that can create a hypothesis in the correct form can incorporate its information in the list and thus be considered in segmentation and concept binding.

4.6 Domain Independence

HB-CA’s algorithms are not specific to a particular domain, however, its reliance on a domain model means that a library developed for one domain does not transfer well to another. Investigation showed that accuracy was roughly halved when a library from one domain was applied to programs from another.

4.7 Language Independence

This part of the evaluation looked at three classes of language: Imperative Non Object-Oriented, Imperative Object-Oriented, and Non-Imperative.

Languages in the first class are similar to COBOL II hence little modification would be required to analyse them. Object-oriented languages pose a more difficult problem. The linear analysis approach adopted by HB-CA may not work well with the coding style often adopted for object-oriented code. Finally, non-imperative languages challenge many of the assumptions on which HB-CA rests (e.g. sequence between program statements) hence analysing this type of code would almost certainly be unsuccessful.
4.8 Cognitive Requirements

HB-CA was evaluated against the cognitive requirements framework described in [8]. For all relevant elements it meets the criteria for success but some pertain solely to visualisation tools and these were disregarded.

4.9 Applicability

In addition to the method-based evaluation described in the preceding sections, the applicability of HB-CA in the maintenance process was examined. This identified a number of areas in which the method could be applied: ripple analysis using concepts, module selection driven by concepts, and reuse candidate identification. Finally, it can be applied for the reason it was originally intended: to assist program comprehension.

5. Conclusions and Future Work

Although flexible segmentation is generally successful, the algorithms that post-process the data produced from the SOM can cause problems. Throughout the method, evidence is preserved wherever possible thus invalid clusters resulting from SOM analysis are reintegrated with their neighbouring valid clusters. Indiscriminately adding this unrelated conceptual data to valid clusters causes confusion during concept binding and can reduce the accuracy of concept assignment. This problem could be solved by either disregarding invalid clusters altogether, or using the conceptual information in the invalid cluster to assign it appropriately to a neighbouring valid cluster.

The simplicity of the knowledge base means that a maintainer can update it with minimal effort and there is no requirement for a large domain analysis exercise before HB-CA is employed. It would, however, be interesting to explore a more complex knowledge base structure to determine whether this could improve the accuracy of HB-CA without comprising its low cost.

Finally, it may be possible to translate the entire problem to a SOM, using the most frequently triggering hypothesis as the concept label for a node or segment. This would remove the separate concept binding step and simplify the segmentation processing.

In general, HB-CA is a success. It has high concept recognition accuracy using a simple knowledge base, and linear computational growth in the length of source code being analysed.

Acknowledgements

This work was undertaken during my time at Durham University and I would particularly like to thank Professor Keith Bennett for all his help. I also gratefully acknowledge the support of the UK Engineering and Physical Sciences Research Council and CSC.

References