Abstract

Many authors consider that product family development, instead of one-off development, is a decisive step towards the systematic software reuse and economy of scope. This thesis is aligned with this movement and proposes a new process to build product families, named EDD (*Exemplar Driven Development*), which takes advantage of the similarity between family products to make them by analogy.

The first EDD activity is to build a specific product of a family. Next, this exemplar is flexibilized to satisfy the remaining products requirements. That is, an analogy relation is defined in a formal way to derive products automatically from the exemplar. Finally, the family products are obtained parametrizing the exemplar flexibilization.

Among EDD contributions should be mentioned the following:

- Facing the development and maintenance of a product family with an iterative strategy. An exemplar which satisfies the family fixed requirements is built as soon as possible. Flexibilization layers, which implement family variable requirements, are added to the exemplar in successive development cycles.

- Family fixed requirements are usually more stable than variable requirements. EDD separates the fixed requirements implementation (located in the exemplar) from the variable requirements implementation (located in the modules which make the exemplar flexible).

- Building a product family is usually decided when repetitive work is detected in the development of specific products in the same domain or when business opportunities are expected if a successful product is extended. EDD recognizes this situation and tries to take advantage of it by reusing the exemplar completely.

This thesis explores different ways to flexibilize an exemplar by applying the most common techniques of code generalization (inheritance, genericity, code templates…). Unfortunately, the thesis demonstrates that these techniques suffer limitations which hinder flexibilizations with important qualities like modularity, non-invasiveness, applicability to any software product… In order to avoid these limitations, the thesis

proposes a new language called ETL (*Exemplar Transformation Language*) which is implemented in Ruby.

With the purpose of illustrating the power and versatility of EDD and ETL, the thesis includes examples of the development of programs written in Java and C++; stored procedures written in TRANSACT SQL; tests suites written in Java and Modula-2; and documentation written in HTML and Javadoc.