SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2014; **00**:1–3
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spe

Understanding the Modular Operators of Delta-Oriented Programming

Fausto Carvalho[1], Marcos Oliveira[1], Pedro Costa[1], Alexandre Lucchesi[1], Rodrigo Bonifácio[1], Hilmer Neri[2], Daniel Sandoval[1]

[1]*Computar Science Department, University of Brasília, Brasília, Brazil*
[2]*Faculty of Engineering of Gama, University of Brasília, Brasília, Brazil*

SUMMARY

This paper describes the use of the LaTeX 2ε speauth.cls class file for setting papers for *Software—Practice and Experience*. Copyright © 2014 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: Delta Oriented Programming

1. INTRODUCTION

There is a growing interest in the research and development of variant intensive software systems (such as software product lines []), leading to the design of several techniques for modeling, modularising, implementing, and reasoning about software variability. That is the case of Delta-Oriented programming, a prominent technique for implementing variant systems that has evolved from an experimental, prototypical based approach [**?**] for dealing with software variability composition to a full fledged Java implementation [**?**].

In the context of variant intensive software systems, modularity is recognized as a fundamental property, and the relation between modularity and variant systems (in general) is clearly discussed within the Baldwin and Clark theory [**?**], which states that "*a modular design is a fundamental property that allows a designer to experiment with different alternatives*". Therefore, based on a modular design supported by established design rules, developers should be able to evolve a software product line (SPL) according to software engineering design principles (e.g., the open-closed and principles). Nevertheless, Neves et al. describe that evolving software product lines is a challenging and error prone task (even for small product lines) [**?**], because it requires deeper impact analysis and the involvement of both domain and product engineers to reason about the consistence among SPL assets that represent and relate the variability space (usually described using feature models) to the solution space (i.e., detailed requirements and design, source code, test cases, test scripts, and so on).

It is usually assumed that Delta-Oriented programming improves the modular decomposition of features. For instance, Schaefer et al. compared Delta-Oriented programming with Feature-Oriented programming [**?**], concluding that the first simplifies the evolution of SPLs. In addition, a recent study compared two implementations of a text editor product line: one implementation developed using DeltaJ 1.5 (a Java 5 implementation of Delta-Oriented programming) and another

*Correspondence to: Campus Universitário Darcy Ribeiro – Asa Norte Edifício CIC/EST 70910-900 Braslia – DF – Brasil. Email: rbonifacio@cic.unb.br

center[h]

Fator A			
Tratmento A1		Tratmento A1	
Fator B		Fator B	
Tratamento B1'	Tratamento B2'	Tratamento B1"	Tratamento B2"
Sujeito 1,3	Sujeito 6,2	Sujeito 7,8	Sujeito 5,4

implementation developed using a plugin based architecture on top of the Eclipse Rich Client Platform. The authors conclude that Delta-Oriented programming is more intuitive and simplifies both product line implementation and configurability, by reducing the necessary code base. However, the involved trade-offs of using Delta-Oriented programming are not full understood. In particular because (a) the first study is based on qualitative assessments and lacks the use of a rigorous approach to support its claims, and (b) the second study focuses on building a product line from scratch and investigates redundancy and program comprehension. Although these perspectives relate to modularity, they do not investigate other characteristics of a modular design— which should support independent development and flexibility to evolve.

In this paper we first investigate the use of Delta-Programming to extract a SPL from an existing project. This is a typical approach for software product line engineering that reduces the risks of product line adoption. We then investigate how Delta-Oriented programming accommodate the evolution of a software product line with respect to typical evolutionary scenarios of a product line [**?**]. Therefore, the contributions of this paper are three fold ...

2. DELTA-ORIENTED PROGRAMMING

DOP, Delta Oriented Programming, is a novel programming language approach focused on modularity, expressiveness and flexibility for the development of software product lines []. DOP uses the concept of delta modules, which are units of modularity that specifies changes to be applied to a core module (minimal implementation of the mandatory features of a product) in a partial order. A core module is a valid product build using tradicional single application engenieering techiniques. The concept of core module was conceived in a proof-of-concept implementation of DOP, called Core Delta, in a later version, Pure Delta, the need for a predefined core module was abandoned. [].

2.1. DeltaJava and Core DOP

DOP is not restricted to any specific programming language, and was prototyped in Java, in a tool named DeltaJava. Deltas in DeltaJava can augment, modify or remove code from core deltas modules, on the class level or class structure level [].

Delta modules are similar to features modules in a way it is possible to add or modify the beahvior of a base product, but DOP extends the capabilities of FOP by an operator able to remove code.

2.2. DeltaJ 1.5 and Pure DOP

3. STUDY SETTINGS

The case study is an empirical research method and a well defined approach to investigate a contemporary phenomenon in its real context. [Refs 5.1 do Whohlin] This method was chosen because we wanted to get a trusted and reproducible evaluation for handling modularity in software systems.

Eight researchers were involved in this investigation. In order to reduce researchers bias while performing this study, we decided to separete developers in two groups according to their experiences in software development, as showed in table ??

DOI: 10.1002/spe

ACKNOWLEDGEMENT

This class file was developed by Sunrise Setting Ltd, Torquay, Devon, UK. Website:
www.sunrise-setting.co.uk

REFERENCES

DOI: 10.1002/spe