

Contents

	Page
Editorial	5

COLUMNS

Classification Theory

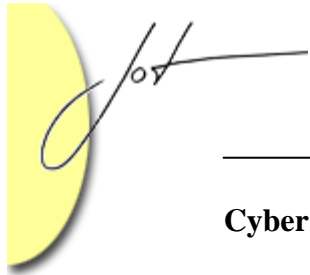
The Theory of Classification Part 20: Modular Checking of Classtypes	7
<i>By Anthony J H Simons</i>	

A first-order type system has two things to commend it. Firstly, it is quite simple to implement a type-checker that can check types for exact correspondence, or for subtype compatibility with a given type. The type of the source object can be compared with that of the target variable to see if the former can be converted up to the latter, using subtyping rules like those we discussed in [1]. Secondly, code that has been checked once need never be checked again, or recompiled in new contexts. This is because the type system can never reveal more specific information about an object that is passed into a more general variable (which we have called the “type loss problem”), so the code need only be checked once over the most general type that it can accept.

Business Objects

Confessions of a Service-Oriented Abuser	19
<i>By Mahesh H. Dodani</i>	

Hello, my name is Pat, and I resemble the person that Erich is talking about in his quote above. I introduced my pattern abusing story to you in 1999 [Dodani, M., "Rules are for Fools, Patterns are for Cool Fools", Journal of Object-Oriented Programming, Vol. 12, No. 6, pp. 21-23, SIGS Publications, October 1999], and asked for help in getting over my abuse.

**Cyber Databases****The JBoss Integration Plug-in for IntelliJ IDEA, Part 2.** 25*By Douglas Lyon, Martin Fuhrer and Thomas Rowland*

This paper describes how to add a stateless session bean. It is stateless because we are not concerned with remembering values of attributes between successive calls from the client. This paper also demonstrates that our session bean can be either remote (i.e., a bean with a remote interface) or it can be local (a bean with a local interface).

Strategic Software Engineering**Context** 35*By John McGregor*

Context is the setting in which any statement will, or perhaps should, be interpreted. We can think of context as a set of constraints, or limitations, that set boundaries and a set of assertions that establish certain facts. The boundaries may be beliefs, or logical propositions, or any definition upon which our work depends. The “best” design for a given solution is context dependent. Change the context and the evaluation of the design may change.

Cyber Databases**On Issues with Component-Based Software Reuse** 45*By Won Kim*

As the size and complexity of software products, and the compression of software product development cycle seem ready to go out of control, large software development organizations are taking a hard look at the allure of the substantial productivity enhancements envisioned by early proponents of component-based software reuse.

REFEREED ARTICLES

Using Reflection to Reduce the Size of .NET Executables 51*By Vasian Cepa*

Current binary compressors cannot be used to pack .NET executables because .NET makes use of a specially modified PE file format. We will rely on reflection capabilities supported by .NET to pack .NET binaries using pure C# code. The solution is quite general and can be used with any .NET executable, no matter in what front-end language it was written.

Support for Design by Contract™ in the C# Programming Language

By Rachel Henne-Wu, Dr. William Mitchell and Dr. Cui Zhang

There is evidence that “contracts,” or assertion techniques involving preconditions, postconditions, and invariants, have a positive effect on overall software quality. Regrettably, very few programming languages support these techniques. Since the advent of Bertrand Meyer’s Design by Contract™ method, introduced in the language Eiffel, a number of systems have been built to implement support for contracts in more commonly-used languages. Such support has not been satisfactorily implemented in C#. In this paper, we compare the different approaches of existing systems and introduce Contract Sharp, a tool that provides support for contracts in C#.

Connecting Powertypes and Stereotypes

By Brian Henderson-Sellers and Cesar Gonzalez-Perez

Powertypes constitute an advanced OO modelling mechanism that is usually utilized in the form of a specific pattern. Stereotypes comprise the basic customization and extension mechanism in UML, and are also used following a certain pattern. Although different in purpose, these two patterns present some interesting similarities and are shown here to become structurally identical in specific circumstances. This fact can help reduce the apparent complexity of UML and may be of special importance for tools that store and transform models that use powertypes and stereotypes.

E-Tester: a Contract-Aware and Agent-Based Unit Testing Framework for Eiffel

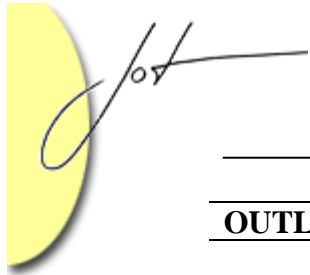
By Jonathan S. Ostroff, Richard F. Paige, David Makalsky and Phillip J. Brooke

We describe a contract-aware unit testing framework, E-Tester, for the Eiffel programming language. The framework differs from JUnit in its first-class support for lightweight formal methods, through test support for contracts and assertions. As well, it supports a form of negative test, called **violation cases**, which aim at validating contracts. It also differs based on its use of **agents** for expressing tests and test cases.

Statically Qualified Types in Timor

By J. Leslie Keedy, Klaus Espenlaub, Christian Heinlein, Gisela Menger, and Mark Evered

Instances of qualifying types (“qualifiers”) have a role analogous to adjectives in natural language which are used to qualify nouns (the “targets”), e.g. a “synchronised person” or a “protected list”. Such adjectival qualification occurs in two forms in natural language.



OUTLOOK

A brief outlook to the next issue

117