JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm published by ETH Zurich, Chair of Software Engineering ©**JOT, 2006**

Vol. 5, No. 6, July - August 2006

Contents

Page

Editorial 5

COLUMNS

Java at Large

The Initium RJS ScreenSaver: Part 2, UNIX 7-15
By Douglas A. Lyon and Francisco Castellanos

We are motivated to study screen-savers because they represent a minimally invasive technology for volunteering CPU services. Typically, computers are used between 40 and 60 hours out of a 168-hour week. This represents approximately 35% utilization. Our theory is that a screen-saver based *cycle scavenging* will improve this number dramatically.

Strategic Software Engineering

Professional development 17-22
By John McGregor

Nothing is more strategic than having the personnel with the appropriate skills to execute whatever strategy executives devise. In our business this is made particularly difficult because of the rapid changes in available technologies and techniques. In this month's issue of Strategic Software Engineering.

The OOP Scene

The API Field of Dreams – Too Much Stuff! *It's Time to Reduce and Simplify APIs!* 23-27
By Dave Thomas

The reality of frameworks and class libraries is far removed from the promise. With very few exceptions, OO libraries are bloated, poorly documented, very difficult to use and modify, and have disappointing performance. This accidental complexity makes using some of the

popular frameworks challenging even for experts, let alone for normal application developers.

Business Objects

Patterns of Anti-Patterns 29-33
By Mahesh Dodani

Antipatterns identify common problems with solutions, and then show how to refactor the solution to get rid of the problem. Therefore, developing a antipattern is a top down process, where problems with a recurring solution are identified, and then a best practice refactoring of the solution is developed to address the problems. The problem, solution, and refactored solution then get put together into an antipattern.

Cyber Databases

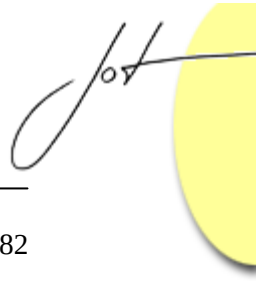
On Assuring Software Quality and Curbing Software Development Cost 35-42
By Won Kim

To assure software quality, a few things have to be done right. The development team has to be properly staffed and organized. A development process has to be in place and followed by members of a development team. To curb the cost of developing software, a few things have to be done. Beyond possibly outsourcing development to where the cost is much lower, the in-house development team has to work as a cohesive team, and the caliber of each member of the development team needs to be continually and appropriately upgraded.

REFEREED ARTICLES

A Classification Framework for Software Reuse 43-61
By Vitaly Khusidman and David M. Bridgeland

Software reuse is commonly used to leverage existing assets and to reduce development cost and time. Reuse can be accomplished by several different mechanisms. This paper describes these mechanisms and proposes a classification framework for them. The framework has two dimensions: retest scope—how the reuse impacts the need for testing—and binding time—when the reuse is realized. By examining these two dimensions, we define a matrix of reuse scenarios.



Removing Redundant Boundary Checks in Contextual Composition Frameworks 63-82
By Mireca Trofin and John Murphy

Contextual composition frameworks allow components to specify boundary conditionsdescribing properties that the runtime context must meet. Based on theseconditions, component platforms execute services when the control is passed from a component to another one. The goal of these services is to ensure that the boundary conditions are met.

ABS++ : Assertion Based Subtyping in C++ 83-105
By Herbert Toth

As the software community has learned during the last two decades, the object-oriented approach is very powerful for developing large software systems. Much of this power is due to the key concept of inheritance. However, the static checks enforced by C++ or Java compilers upon derived classes test for such syntactic and typing restrictions that guarantee the lack of runtime type errors. Obviously, however, this is not enough to prevent surpising and often disastrous behavior of programs.

Enriching the Object-Oriented Paradigm via Shadows in the Context of Mathematics 107-126
By Marc Conrad, Tim French, Marianne Huchard, Carsten Maple and Sandra Pott

Whereas the pure object-oriented paradigm is closely linked to mathematical structures, the actual implementation of mathematics within software artifacts often does not follow a strict object-oriented design process. We do not advocate the mass re-education of mathematicians to adopt more rigorous and formal object oriented design processes such as exemplified in the UML (Unified Modelling Process). We prefer to simply accept the de facto situation as a given, and go on later to suggest ways in which an extension to object orientation, namely shadows can be used to actively support the development of mathematical software artifacts, within informal and rapid software development contexts of use.

Improving the Use of Multiplicity in UML Association 127-132
By Hee Beng Kuan, Yong Yang and Lei Bian

We propose a derived class and a derived association to improve the use of multiplicity for specifying business rules. In the use of n-ary association, if there is any business rule that can be expressed in terms of the number of instances of the association that associate each instance of an associated class or each combination of instances of k associated classes (1 < k < n-1), one from each class, we propose the inclusion of a derived class and/or a derived association.

OUTLOOK

A brief outlook to the next issue 133