OOPSLA'92
Vancouver, British Columbia, Canada
5 – 10 October 1992

Addendum
to the
Proceedings

Experience Report—
The WyCash Portfolio Management System

Report by:

Ward Cunningham
Cunningham & Cunningham, Inc.

U.S. pension funds, corporations, and banks invest billions of dollars in the "cash" markets. Cash securities are generally considered those with a remaining term to maturity of less than one year, but can include those with maturities as long as five years. Incredibly diverse in nature, cash securities are actually negotiated between issuer and buyer, and new security types are frequently introduced into the market. WyCASH+ is a portfolio management system which provides basic accounting, record-keeping and reporting, as well as analytical computations to assist the manager of cash portfolios.

For the development of WyCASH+, Wyatt Software chose to employ object technology in order to quickly and effectively address the diversity present in the market. Objects help in two ways. First, many security types fit nicely into an inheritance hierarchy which is directly supported by our language (Smalltalk) saving us considerable effort in coding. Second, changing market demands often require massive revisions which we have been able to accommodate because of the modularity intrinsic in a totally object-oriented implementation. Our customers value our responsiveness as much as, if not more than, our product's fit to their current needs.

We developed the product by incremental growth from a working prototype. Each member of our small engineering team maintains at least general knowledge of all aspects of the roughly four megabytes of source code. This includes some libraries provided by the vendor and others written to our specification by third-party contractors. Mature sections of the program have been revised or rewritten many times providing the consolidation that is key to understanding and continued incremental development.

We found that some key implementation ideas were slow to emerge in the course of WyCash's development. Although many of our objects are derived directly from things or concepts that appear in documentation, and others were easily drawn from the collective experience of our programmers, a third, and more tantalizing, category of objects only surfaced through a process we could call Incremental Design Repair. We found these highly leveraged abstractions only because we were willing to reconsider architectural decisions in the light of recent experience. The same flexibility that allowed us to tackle diversity in our problem domain, specifically the flexibility afforded by the universal use of polymorphic messages sends (i.e. pure object-oriented programming), allowed us to include architectural revisions in the production program that would be judged too dangerous for inclusion under any other circumstance. For example, it was not uncommon for a new feature to fit poorly into an existing object architecture. Polymorphism gave us the option of revising the architecture for only some of the program features. Our newly designed objects

could coexist with objects of an earlier design allowing us to defer conversion work until later releases. This had the added advantage of allowing our programmers to familiarize themselves with the new architecture as time became available. The ultimate removal of the immature architecture would leave us with a program that had been simplified in the course of adding features—a truly enviable situation.

We believe this process leads to the most appropriate product in the shortest possible time. Although immature code may work fine and be completely acceptable to the customer, excess quantities will make a program unmasterable, leading to extreme specialization of programmers and finally an inflexible product. Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite. Objects make the cost of this transaction tolerable. The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation, object-oriented or otherwise.

The traditional waterfall development cycle has endeavored to avoid programming catastrophe by working out a program in detail before programming begins. We watch with some interest as the community attempts to apply these techniques to objects. However, using our debt analogy, we recognize this amounts to preserving the concept of payment up-front and in-full. The modularity offered by objects and the practice of consolidation make the alternative, incremental growth, both feasible and desirable in the competitive financial software market.

Contact information:

Ward Cunningham
Cunningham & Cunningham, Inc.
7830 S.W. 40th Ave.
Portland, Oregon 97219
(503) 245-5633