COMPLETE LANGUAGE INSTANCE — EXECUTIVE SUMMARY

Lawrence J. Dickson, PhD
2757 Chaffee Street
National City, CA 91950

25 March 2006

(619) 470-2355
tjoccam@tjoccam.com

I. Synopsis

Despite immense increases in computer processing power, storage, and memory, and raw volume of development tools, technical progress threatens to stall out. Complexity defeats creativity on a wide front, leaving only chip speed and miniaturization to drive progress. This threatens morale, the economy, and the sheer usability of computer-based technology.

I trace the problem to software paradigms that have become well-nigh universal. Designs based on them are incapable of enforcing or explaining correctness. The difficulty increases with multiple layers of complexity, often resulting in vast time and cost overruns or outright project failure.

A design and language paradigm exists that corrects these problems, if it is consistently applied to all parts of a computer's operations, from power-up to power-down. I propose a low-cost "stealth" development, targeted at a single computer architecture and set of peripherals, but designed to be extendable to practically any architecture using straightforward principles.

II. The stallout

Major software projects costing hundreds of millions of dollars often fail outright, although the hardware is clearly sufficient for the job. Thoroughly tested firmware, even on space probes, frequently glitches for unknown reasons. Commercial technical advances backed by thousands of developers slow to a crawl, announced features are delayed or abandoned, and perceived utility stalls or even declines. The impossibility of predicting code behavior makes security problematic.

Worse yet, the creativity and morale of the nation's "garage" developers have practically disappeared. Code stiffness prevents soldering together new gadgets. Refactoring of old features now dresses up as innovation. A blizzard of acronyms disguises this defensive coding, and boxes developers into ever-narrowing knowledge spaces.

III. Analogy and metaphor

Analogy equates similar things, while metaphor equates dissimilar things. If a term is applied metaphorically, conclusions based on analogy will not necessarily hold. Standard software paradigms are full of hidden metaphor.

My own historical metaphor is epicycles in Renaissance astronomy. The wrong turn began around 1970 with stack-based C and uncontrolled dynamic resource allocation. This first epicycle, the infinite resource metaphor, succeeded static assembly and Fortran. It saves a few percent of resources which are now cheap (CPU power and memory), and pays a price which is now huge (complex development cost and incorrectness).

By the 1990s, the misnamed "object-oriented" paradigm was built on the dynamic structures, abandoning the hard design to microcorrectness checks. It now is also nearly universal (the second epicycle). Current paradigms advance toward matrix management of the objects (Java interfaces and "aspect-oriented"), the third epicycle. Each fix creates new deep-seated problems. I propose here a return to elliptical orbits.

IV. The conceptual breakthrough

One principle serves as the foundation: Terms which have been applied metaphorically are required to be applied analogically. That is, usages designed around these terms must derive only from those features that are similar among the real items denoted by the terms.

This approach has been used in the past by "process-oriented" languages and related hardware. But the scope of application was always limited, omitting boot-up, operating systems, drivers and the like. Nobody had a financial motive to extend it to an entire computing system, and it was swept aside in the early 1990s.

There is no great difficulty in extending known technology, including some developed by the author, to embrace an entire exemplary computing system. This proposal is to do just that. What is proposed is a Transparent Analogical Complete Language Instance of the Resource-Oriented Paradigm.

Firstly, because it is now fully analogical, the design description of any software component built using this tool enforces the behavior designed for, and does not require any hidden, arcane coding or usage features for correctness. Thus — like car parts — components can be combined to any degree of complexity without surprise conflicts based on lack of reciprocal knowledge.

Secondly, the system and language design will itself be analogical, and thus applicable to other hardware using well-defined design principles. This value feature resembles the extendability of the C programming language to practically all CPUs.

Thirdly, code security is provable. Power-up to power-down design means there is no access for exploits except within the paradigm. And the paradigm not only forbids buffer overflows, but allows software activation, data passing and resource sharing only through predefined channels. Using protected mode or its compiled equivalent, software modules provably become as secure as a robust physical firewall.

V. Practical plans

I propose employing myself, and using as part-time consultant an expert in compiler syntax and writing, plus someone with aptitude and wide knowledge of current hardware and tools. Using a syntax that looks familiar, we will define the language, consistent both with a legacy platform (DOS/B008/TRAMs) used as a development target, and with modern unsymmetric multiprocessors (e.g. the IBM Cell and Sony PlayStation 3). We will apply it to all levels, from bootup and configuration to scripting and compilation. I will then write and test it.

Minimal, scriptable command-line support for all programs and peripherals must include program-based, not operating-system-based, multitasking and multiprocessing capability. These concepts have been fully proven by my work and I expect a usable system and language within a calendar year after first run of the target hardware.

VI. Details

For a detailed discussion of the technology and plans, including a list of nine specific factors auguring success, see my white paper "Transparent Analogy as a Foundation for Language." For technical and personal references and details of ownership and cost, please inquire.