Proposal for paper to be presented at XP 2003 26-29 May, Genoa, Italy

Title	**Extreme Terseness**
Subject	Terse programming languages offer particular advantages to agile processes
Author	Stephen Taylor
Keywords	code, communication, cost of change, refactoring, simplicity, software economics, terseness

Abstract

While XP principles are independent of the languages in which software is developed, we can distinguish properties of programming languages that affect the agility of development. Some languages are inherently more agile than others, and the experience of developing software in these languages reflects this.

A key XP premise is that the cost of change need not rise exponentially or even linearly over time. The history of software development in a group of languages descended from the mathematics notation developed at Harvard in the 1950s by Iverson ("A Programming Language", Kenneth E. Iverson, Wiley, 1962) exhibits this characteristic, primarily through their shared property of extreme terseness.

Code volume affects key XP processes:

> *Communication.* XP development places unprecedented emphasis on communication within a project and the use of code as a medium for it. The terseness of code is a reflection of a language's abstractive or expressive power. Terse code facilitates precise communication about processes and data structures in the same way that university educations and their associated vocabularies enable graduates to communicate precisely about ideas, theories and observations in their fields of study. (See Iverson: "Notation as a tool of thought".)
>
> *Refactoring*. Code volume is always a significant term in the cost of refactoring, and a term independent of complexity. Terseness contributes to XP's virtuous circle of simplicity. While more work, thought and time goes into writing a line of terse code than a line in a verbose language, the code volume ratio between terse and verbose code usually has the terse code finished first. A larger benefit comes from refactoring. The lower the code volume, the less code to review, analyse and rewrite. Terseness benefits refactoring even more than initial development.
>
> *Simplicity, quality, fun.* While it is still possible to write verbose code in a terse language, the abstractive and expressive power of extremely terse languages permits solutions of startling and breath-taking simplicity. (These solutions are often correspondingly fast.) For example, a small spreadsheet, complete with GUI, can be written as 3 lines of primitive K. A single 55-character expression in J calculates and plots a Mandelbrot set. The author recently used sparse-array techniques to optimise a memory-intensive APL process; the refactoring took an hour, writing and testing just four lines of new code.

Systems developed in the languages descended from Iverson's notation typically have code volumes *one to two orders of magnitude* lower than conventional languages. The extreme terseness of these languages emerges from the following properties common to them.

- Rigour and consistency are preferred to ease-of-use
- Syntax and semantics aim to maximise abstraction and generality
- Weak data typing, to permit maximum abstraction of processes
- Latest possible binding of names to objects
- Pervasion wherever possible; abstracting operations item-wise (or atom-wise)

The following languages, descended from Iverson's original notation, share these characteristics: A+, APL, J and K.

The history of software development in these languages (discussed) displays some of the characteristics of XP development, in particular in communication and refactoring.

Of these languages, APL has the longest history of industrial software development. APL software development has typically employed small teams, short planning and release cycles, vague or evolving specifications, and close involvement with end users.

A+ was developed at Morgan Stanley in the 1980s on XP-like principles. The bank had a large and rapidly evolving requirement for software to support traders in financial markets that added new trading instruments each year. Effective support would require the rapid retrieval and analysis of huge data series from market data feeds, a retrieval and analysis no then-existing database could support. A+ started as an implementation of the smallest subset of APL that could possibly work. For a decade and a half Morgan Stanley's Analytical Programming Team plugged its software gap and met changing requirements with rapid development in A+.

The line of language development started with A+ continues today with (the even more obscure) K, a language used primarily to implement the blazingly fast kdb database system. There is a small K commercial programming community, of which the largest European group is centred on Zurich. From this community emerge occasional tales of speed and brevity. A K program implements a small spreadsheet, complete with GUI, in 3-lines of primitive code. A recent test compared code volume and execution speed of K against hand-coded C++, C# and Java on an arbitrarily-chosen benchmark task. (The computing task was inherently scalar and exploited none of K's power in handling arrays.) A solution in K improved on execution speed by one order of magnitude and on code volume by two: the work of 400 lines of C# code was done by four K statements.

Conclusion Extreme terseness in languages makes them particularly suited to agile development processes. Extremely terse languages participate more fully in XP's virtuous circle of simplicity. Some less well-known languages offer order-of-magnitude improvements in terseness. Some of the benefits associated with agile development processes are already exhibited in the history of software development in these languages, as a simple reflection of the impact extreme terseness has had on the cost of change over time. Agile process managers looking for more edge will consider these languages natural candidates for roles in some XP projects.

Stephen Taylor
British APL Association
sjt@netbox.com