ACM SIGPLAN Forth

Forth Report

Editor: *Paul Frenger, P.O. Box 820506, Houston, TX 77282; pfrenger@ix.netcom.com*

EXTREME Forth

Paul Frenger

1 EXTREME Introduction

In November, 1999 I attended ACM's OOPSLA Conference in Denver, Colorado. OOPSLA'99 was a well organized, well-attended meeting set in a breathtaking locale. Incidentally, I presented a poster [1] on object oriented programming using the Forth language (the only Forth paper there, as I recall). Between presentations I ran across a book on "extreme programming" at one of the bookseller's exhibits [2]. Thumbing through this book quickly gave me the gist of the concept, which I found interesting but (to me as a Forth programmer) not extraordinary. Now, over a year later, I realize that my non-Forth colleagues might like to hear about extreme programming (it is usually abbreviated XP) and why a Forth user would not find it all so new and exciting.

To begin, you might want to hear about XP directly from those who espouse its use. Go to the official website [3] and browse around. Take it all in. Cogitate on the differences between XP (which is a programming methodology, not a programming language) and more traditional requirement-oriented system design techniques.

(While the compulsive programmers are browsing their way through the Web hypertext mentioned above, the rest of us hang-loose code hackers will stay here and talk informally about XP. They will rejoin us later).

XP is described as a "lightweight" software development methodology, which means it has very few, easy-to-follow rules. Writing code and testing it starts right away. In contrast, "heavyweight" methods involve many (precise) rules, mostly centered around extensive planning and the production of volumes of system documentation; the actual coding comes in later almost like an afterthought.

2 EXTREME Personalities

The following short tale encompasses my understanding of the history of XP. XP is an outgrowth of the object oriented programming movement. It is a "work in progress", so tomorrow it may look different in various details. Three persons are described [4] as its creators (they are informally known as the "Three Extremos"): Ward Cunningham, Kent Beck and Ron Jeffries.

Ward Cunningham is a computer consultant well-known for his work in object oriented programming practices. Kent Beck is also a computer consultant; he is credited with bringing the XP core practices together and giving it its name. Starting in 1996, Beck worked with Ron Jeffries on a payroll project which is now referred to as "C3" (the Chrysler Comprehensive Compensation project), the first XP opportunity. What we now know as XP was put together "ad hoc" during this C3 project. The first thing Beck and Jeffries did after looking over the C3 mess was to throw out everything which had been done on the project up to that time and start over anew.

"Extreme Programming" is a misnomer: there is nothing really extreme about it. You do not see gangs of gun-toting hackers in black leather apparel, blasting into office buildings in a hail of bullets, killing business-suited programmers and taking over their commercial projects (a la "The Matrix"). The term "Extreme Programming" is a typical (somewhat *juvenile*, I might add, Mr. Beck) computing hyperbole intended as an attention-getting device, like: "my program BOMBED" (oh did it, was anyone hurt?) and the term HACKER itself (hmm, has anyone seen my meat cleaver lately?). To Beck's credit, though, I must admit that the phrase "Extreme Programming" printed on the spine of a computer book did grab my attention easily at OOPSLA'99.

3 EXTREME Practices

Now (getting to the point) what does XP methodology entail? Instead of huge mounds of (probably off-base) plans and documents, you start with a few "user stories" on Ronald Reagan sized 3-by-5 index cards. A user story describes in not more than three sentences something a

part of the program has to do, like "check the customer name against the customer database and if not found assign a new customer number, otherwise show the customer number which was found". Simple enough. Very modular. Hardly mysterious. Eminently codable in most programming languages. Start collecting these user stories and the program you need to craft will become apparent.

Next, start writing programs (ignore the fact that you don't have the user's entire requirements in hand yet ... loosen up a little!). Programs are to be written by pairs of programmers working together at one workstation. The purpose here is to let one programmer "coach" the other as they work, helping to create a language to describe the problem at hand. XP adherents feel that each *duo* will be more productive together than if they worked separately. Conversely, every programmer "owns" the entire project and may browse through the work of other team members.

Preparing to write the code fragment embedded in their user story, the programming pair first creates "unit tests". A unit test is an OOP message or data or whatever, which will enable immediate checking of the code which the team will now write. The intent is to find mistakes early in the game, when they can be fixed quickly and cheaply.

The user is given the "current release" of the project software on a frequent basis (not more than two weeks apart). By so doing, the programmers keep the users appraised of current progress on the project, and allow the users to make changes and catch conceptual or procedural errors (repeat after me) early in the game, when they can be fixed quickly and cheaply. This avoids the terrible moment often experienced with heavyweight programming methodologies, where the programming team gives the user the "deliverables", only to have the user reject the product angrily.

The XP "prime directive" is to *do the simplest thing which could possibly work.* No frills. No grand ideas. Just working code. This concept pushes the project toward completion at the fastest possible rate. If the user wants to extend the program capabilities, that's another (user) story.

A corollary of this rule is the use of "spike solution" or just "spike". A spike is a minimal program solution intended to get the programmer through a difficult area in the project, such as a new algorithm. The spike does not use the existing framework, classes or code; it takes the team past all distractions and hangups and focuses on getting a code fragment to work. Once the code runs, the essence of the spike solution is assimilated into the "official" program and the spike is discarded.

XP methodology involves a lot more, but the above is a useful thumbnail sketch of the essentials.

Well, I see that the website browsers are back again, just in time for me to mention some of my own XP-like experiences and then to bring up the subject of the Forth language.

4 A Semi-EXTREME Personal Experience

Twenty years ago, I found myself hired to design a computer system to allow patients to obtain health care by telephone [5].

To describe it briefly, the (preregistered) patient would call the service, where a receptionist would verify membership and put the patient into the virtual "Waiting Room". A Nurse would initiate a new chart entry by taking the patient's complaint and assigning a priority to the call. The Doctor would take the next call in the Waiting Room and speak with the patient, further extending the chart and making the diagnosis. Treatment would ensue: give advice, call prescriptions to the pharmacy or refer the patient to a Specialist or to an Emergency Room. We called this service a "Telephone HMO" and spent three years bringing it up on an IBM S370 mainframe, in COBOL, with Intecolor 80-by-48 character color displays.

But in the first week of the project in late 1979, the Doctor who wanted to commission the design asked me to "show him something he could relate to" as far as the layout of the system was concerned (he was very visually-oriented). He gave me two calendar weeks to accomplish this. I decided to forego giving him the usual flowcharts and planning documents and to go straight to a computerized model he could browse through and play with, hands-on, himself. I asked him to purchase a "Compucolor II" 8080- based personal computer by Intecolor (I'm not giving you a reference here because you *really don't* want to know about it) for my use and went into seclusion for two weeks. Using the Compucolor's BASIC interpreter I literally "threw together" a 32 Kbyte program which modeled the core system function screens in living color: Registration, Waiting Room, Charting (with pop-up Help Screens for the Doctor), Prescriptions, Advice, Referral and Reports. When I presented this simulation program (on time, mind you), the Doctor and his

staff were thrilled and the initial project funding was approved.

I will never forget the impact this little "throw-away" simulator had. In addition to its use as a "proof of concept" and a funding tool, its little 64-by-32 character displays were expanded to the full 80-by-48 Intecolor displays (I extended the original BASIC code to drive the big monitor via the Compucolor II's serial port). This setup was later used to clue-in the COBOL programmers who wrote the "real" system for us, and it was a surprisingly effective design and verification tool [6].

Was this an XP-like experience? In some ways, it was (remember, this was circa 1980, before the IBM PC, before the Internet, before Windows ... heck, even before Bill Gates made his first billion dollars!).

For example, when building the expanded 80-by-48 character simulator, the user (and his staff) made written suggestions regarding its format and content which I put into a small binder for later coding. These notes were functionally similar to "user stories". These suggestions were quickly converted into "spikes" and then into fleshed-out functional programs, at first by myself alone, but later with an assistant. The updated simulator served as the "current release" of the system software. The big simulator's BASIC code was made as modular as possible; this facilitated our updates. Much of the BASIC screen/keyboard handling code (ie: cursor X-Y locations, input/output field sizes, pop-up screen details) later were converted into their COBOL equivalents for the final mainframe system. Indeed, the COBOL system was created via a "heavyweight" technique, but largely by copying my simulator's information into the final specifications and documentation. We avoided a number of traps and problems by building and honing that system simulator. Eventually the little Intecolor II and its BASIC code was discarded, but not before it had saved us tens of thousands of development dollars by our estimates. The "heavyweight" programming company we hired to write the final COBOL code eventually adopted this simulator technique for their other client jobs.

I just wish we had Forth available for this project instead of BASIC!

5 EXTREME Forth

Once you've used Forth for awhile, XP seems pretty hohum. Why? One answer is that recommended Forth programming style and techniques accomplish much of what XP espouses. Another is that the forces which created XP had also created Forth a generation earlier.

Forth had its own "Extremo" in the form of Chuck Moore, inventor of the language. Moore claims he *discovered* the language rather than *creating* it. You have to talk to the man for awhile, as I have, to understand the wry humor in this statement.

Moore's goal in creating the Forth language is reminiscent of the later XP "prime directive" (to do the simplest thing which could possibly work). Even after twenty years of using Forth, climbing though its innards and even writing my own compiler, I often still wonder how it works. Internally, the magic is in its user-accessible two-stack design, user-extensible vocabularies, incremental compiler and virtualized disk handling system. Some examples to consider.

First, Forth is a modular language. Writing modules and sub-modules (called "words") is encouraged. Complex programs are designed by repeatedly factoring the problem top-down into subunits, then writing the code bottom-up in modular fashion. XP "user stories" just beg for this kind of approach, since they are modular by their nature.

Forth encourages code-writing with a minimum of timewasting, up-front documentation writing. Each and every Forth "word" can be a spike solution if that's what you need: write it, run it, test it, modify it. Working in Forth's interpretive mode you can repeat this cycle over and over in mere minutes (no separate compiling step, or linking, or "making" to slow you down). When you're done with the code, assimilate it or throw it away.

Next, Forth encourages testing of each "word" as soon as it is written. You may pass the new word its test parameters on the stack, and receive its results likewise on the stack. The Forth word **.STACK** will nondestructively show you what's on the stack while debugging in interpretive mode. You may also put test data onto mass storage and either **LOAD** it or access it by 1024 byte **BLOCK**s . You may **DUMP** areas of RAM memory for examination before and after you execute a word. You may write Forth words with debugging code embedded in it, and compile the final programs "clean" (bypassing the debug code with conditional compilation).

Oh, Forth isn't an object oriented language, so how can XP even be applied to it? Well, if you want objects, you can have them. Most Forth vendors now provide object oriented extensions you can easily add to their software. You can add a "C++ like" OOP syntax to most Forths in only 12 lines of code [7], thanks to Bernd Paysan. You are

free to modify or extend the compiler as you will. Indeed, Forth is so plastic in its capabilities that it is often used to create other languages and compilers. Forth never tells me "no". I really hate "no", so now you see why I really like Forth.

6 EXTREME Conclusion

The Forth language has a number of built-in features which make it highly compatible with the XP concept. These have been outlined above. A skilled Forth programmer will likely already be using XP-like practices in his or her programming style. Few conventional compilers facilitate this relationship the way Forth does (maybe Lisp and Scheme; probably not C++ or Java; certainly not BASIC, COBOL or Fortran, in my opinion). ANS Forth code is highly transportable; Forth is equally at home in mainframe applications, PC software, networks and tiny embedded systems.

You try XP, try Forth, then decide for yourself. As Dennis Miller always says, I could be wrong.

7 EXTREME References

1. Frenger, P., "Objects in ANDROID.FORTH", OOPSLA'99, Denver, *Conference Supplement*, pg.59-60.

2. I don't recall that title, but a quick keyword search with either Amazon.com or Barnes & Noble will reveal the usual suspects.

3. `http://www.extremeprogramming.org`.

4. Waters, J., "Extreme method simplifies development puzzle", *Application Development Trends*, Vol.7 No.7, July 2000, pg.20-26.

5. Frenger, P., "Advanced Techniques Used to Create a Telephone Medical Consultation Service", *Proc Rocky Mountain Bioeng Sympos*, Mayo Clinic, April 1983, pg.103-107.

6. Frenger, P., "Using Microcomputers for Medical Software Development, Modeling and Test Marketing", *Proc ISMM Intl Sympos on Microcomputer Applic in Med and Bioeng*, New York City, Oct 1984, pg.71-75.

7. `http://www.jwdt.com/~paysan/screenful.html`.

Paul Frenger is a medical doctor who has been professionally involved with computers since 1976. He has worked as a computer consultant, published over one hundred articles in the bioengineering and computer literature, edited the ACM SIGForth Newsletter for four years and acquired three computer patents along the way. Paul was bitten by the reverse Polish bug in 1981 and has used Forth ever since. Being both a physician and a computer programmer, Paul believes that the term 'hacker' is doubly appropriate in his case.