Simplicity in Forth

Federico de Ceballos
Universidad de Cantabria

federico.ceballos@unican.es

October, 2005

Abstract

In his book "Simplicity" [1], celebrated author Edward de Bono (famous for concepts such as "Lateral Thinking" or "Po") put forward ten rules that should be used in every system that tries to define itself as simple. This paper studies how the Forth language meets these rules.

1 Advantages of Simplicity

Simplicity is a nice word and, therefore, one we would like to have at our side. Simplicity can ease our lives and our actions. Learning a simple system saves us time, money and energy.

Simplicity is both elegant and powerful.

A complex system may have *the user's illusion*. This means that the user believes he or she commands the system and is in charge or everything. However, this can be very far away from the truth.

2 Disadvantages of Simplicity

By travelling the simplicity way, we may end being simplistic. By doing so, we may lose the usefulness of the original idea.

De Bono mentions the following advantages of a very complex book:

> *If you don't have anything to say, it is better if you say it in the most complicated way possible, otherwise the other people will notice the lack of content.*
>
> *Critics will love your book because they will feel as privileged, as they believe to be the only ones who can understand it.*
>
> *Critics will use rivers of ink describing your work, something that doesn't happen with a simple book.*
>
> *University professors will appreciate the book, because their science will be needed in order to explain it to the common people.*
>
> *Nobody will dare criticize it, as nobody will feel sure about having understood it.*
>
> *Philosophers may read in the book whatever they like, as its complexity justifies any interpretation.*
>
> *The public will buy the book to show off their own culture, even if they don't ever read it.*
>
> *The book will become a cult object.*

It will be natural to think that the author is a profound thinker studying very complex concepts.

A lot of fake intellectuals will have a good time, enjoying the complexity of the book.

These don't really apply to software, as the view from the end-user will not usually include the view from the inside. It may happen that a really complex system may be thought of as a marvel of simplicity.

3 How to Search for Simplicity

The following strategies can be used in order to advance in the quest for simplicity.

The historical analysis

Cut

Listen

Combine

Find out the concepts

The « mass » and the exceptions

Restructure

Begin again from square one

Make modules and smaller units

The provocative amputation

Wishful thinking

The energy transfer

The ladder approach

The perfume approach

Even if they are general concepts, a programmer will find that these phrases evoke methods that he or she has used in the past.

The idea of modular programming with short words should be close to any Forth programmer's heart. On the other hand, the idea of *combination* to generate the answer to several problems at the same time will probably bring forward the over-generalized solution criticized in Thinking Forth [2].

4 The Ten Rules of Simplicity

In this section, we shall take a detailed look to each of the rules prescribed by de Bono, not from his point of view, but rather from their usefulness to a Forth programmer.

You must attribute a high value to simplicity

Forth is a useful language. However, other computer languages can be regarded as equally useful. It may even be argued that the usefulness of mainstream languages is greater than that of "niche" ones.

A Forth programmer should therefore be conscious about the different among different paradigms and the reason behind the choice.

You must pursue simplicity with determination

It is not often the case that there is only one way of solving a particular task. According to these rules, the simplicity of the solution should be one of the factors taken into account when comparing different approaches.

You must thoroughly know the soil you are treading

One thing many Forth programmers have in common is the intimate knowledge (the phrase "carnal knowledge" comes to mind) they have about the programming environment they are using. They also extend this intimate knowledge to the hardware they are using and sometimes even to the external system controlled by the hardware.

All this should give this kind of programmer a head start when approaching a new problem.

You must project alternatives and possibilities

The Forth language encourages the programmer towards an incremental development. This way, the programmer improves his or her knowledge of the problem as he or she advances in the solution.

It should be noted that the best way is often unknown until the end of the project is getting closer.

A language that allows the programmer to easily compare different alternatives using a level of development that can be shown to work correctly is an interesting choice.

You must discuss and eliminate some of the existing elements

In a normal language, the resources available are by and large fixed. What you can do is extend the language with some given packages. Sometimes these packages are just black boxes ready to be used and other times the user can customize them.

In Forth the compiler can be enhanced, changed and sometimes even built again from scratch.

The additional packages are nearly always given as source code, divided into minute words that can be included only when needed and can also be changed easily.

Because of this, the author has chosen Forth when he tried to develop a simple environment that could be mastered by the end user in all its details [4].

This is a clear advantage, but it can transform itself in a problem and the programmer spends a lot of time "improving the tools" and no much is left to really solve the problem at hand.

Another problem is that we can be carried over by the sheer beauty of elimination. However, when we eliminate everything, nothing remains [6].

You must be ready for a fresh start

In his book "*The Mythical Man-Month: Essays on Software Engineering"* [3], Frederick P. Brooks argues that in many cases a full system should be developed with the only aim of using it to learn how that sort of system should be properly developed. Once this first version is ready, it should be thrown away and a second one should be started from the beginning.

However, this statement should be taken with a grain of salt. If the system is really complex, too much expense would already have gone into it so it would be difficult to justify that time and money just for experience. (As the dictum goes, experience is what you get when you don't get anything else.) If the system is simple enough, a good programmer should be able to write at least some useful code from the very beginning.

In Forth, the programmer should have a nice toolset, and this should improve with the time spent in the project. If some part of these tools couldn't be used, we would think that something has gone terrible wrong.

You must use concepts

Forth is made of words. It could even be argued that Forth is nothing but words. The name of the word is an important part of any definition.

Choosing good names is an art in itself. This is more important in Forth than in other languages.

As Dijkstra put it: *Besides a mathematical inclination, an exceptionally good mastery of one's native tongue is the most vital asset of a competent programmer.*

It may be necessary to divide things in smaller units

The human mind is really good at analysing problems, far above that any machine developed so far. On the other hand, it is widely acknowledge that this same mind has some basic limitations when trying combine several different ideas.

According to Miller [5], seven is the magic number that matches the different concepts what can be kept in our heads at the same time. Forth allows the programmer to code using tiny definitions. It has been said that the correct length of a definition should be one or two lines long. (Maybe using seven other words, once the stack noise has been removed.)

You must be prepared to sacrifice other values in favour of simplicity

It would be naïve to think that Forth is the only language that can be used to solve a given problem. Furthermore, it would be simplistic to assume that other languages are in a different league and only we are using Forth because of some sort of arcane knowledge.

We must assume that other languages have some important advantages over Forth:

> The compiler is readily available when buying the machine or installing the operating system, so that it can be used without any especial action.
>
> A great number of book, articles, examples, tutorial and web pages are available.
>
> It has a great user base and knowledge can be shared.

Maybe these reasons are not very important to us. However, they are there, and we should know what sacrifices we are doing if we decide to use other way.

You must know in whose name you are projecting simplicity

This is a really difficult subject. It can be easily forgotten what we are looking for and which advantages we shall gain with it. It is easy to let yourself go in the beauty of problem solving and forget that our code and our computer are only tools in the way and not the final objective.

We could be tempted to describe Forth metaphorically, as Mike Ham did: *Forth is like Tao: it is a way, and is realized when followed. Its fragility is its strength, its simplicity its direction*. This sounds nice, but a language is also a tool. At the end of the day, we would like to be able to enjoy the result of our work and not only how much we enjoyed working.

5 Conclusions

The author believes that the language Forth should be presented to other programmers as an example of a simple language capable of meeting most if not all of their need in quite an elegant manner. Even if they keep using their favourite language, the insight given by Forth will be priceless.

On the other hand, Forth programmers should also ponder what the rest of the programming community is doing and how it is solving day-to-day problems more or less successfully. A Forth programmer should not be blinded by the shine of his or her tools and should restrict him- or herself to carry out the current task in the most efficient manner.

References

[1] Edward de Bono. *Semplicità*. Sperling & Kupfer Editori, 1998.

[2] Leo Brodie. *Thinking Forth*. Fig Leaf Press, 1984.

[3] Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley Professional, 1995.

[4] Federico de Ceballos. *A Minimal Development Environment for the AVR Processor*. 17th EuroForth Conference. Schloss Dagstuhl, 2001.

[5] George A. Miller. *The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information*. The Psychological Review, 1956.

[6] C.H. Ting. *Tao of Forth*. Twenty-First Forml Conference. Asilomar, 1999.