chi sparks 2014

The Programming Language as Human Interface

Steven Pemberton
CWI
Science Park 123, NL-1098 XG Amsterdam
steven.pemberton@cwi.nl
http://www.cwi.nl/~steven

ABSTRACT

Programming languages are mostly not designed for humans, but for computers. As a result, programming time is increased by the necessity for programmers to translate problem description into a step-wise method of solving the problem. This demonstration shows a step towards producing more human-oriented programming languages, by developing an interactive map application in a language that allows specification of *what* needs to be solved rather than *how* to solve it.

Author Keywords

Programming languages; HCI.

ACM Classification Keywords

F.3.m Studies of Program Constructs; Miscellaneous; H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

INTRODUCTION

Despite all appearances to the contrary (active at night, little-to-no sleep, diet consisting solely of pizza, sugar and caffeine) programmers are humans too. This might lead you to concluse that programming languages must be a human-computer interface, and that therefore they would be designed as a human interface, with requirements analysis, iterative design and user testing. But with notable exceptions, e.g. [1], most programming languages are not designed like this.

HISTORY

In the 1950's, when computing seriously started, computers were expensive, so expensive in fact (in the millions) that you seldom bought a computer but leased one instead. Often you would get a few programmers for free as a sweetener for the deal; in other words, compared to the price of the computer, programmers were essentially free. Partly for that reason, it was essential that programmers made their programs as efficient as possible: within bounds it didn't matter how much time you spent on programming, as long as you reduced the load on the computer. The first programming languages were designed around this time, and consequently they all had the basic premise that you had to tell the computer what to do: the programmer had to turn the problem specification into a step-by-step solution for that problem, with the aim of reducing the time the computer had to spend on it, and not reducing the time the programmer had to spend on it.

Now, 60 years later and the tables are turned: computers are more or less free compared with the cost of the programmer. But whereas computers are now millions of times faster than the computers of the 50's, programmers are still programming with languages that are visible descendants of the original programming languages. Programmers are still telling computers what to do, and as a result are barely more productive than 60 years ago.

HCI AND PROGRAMMING

ISO 9241 [2] defines usability as the effectiveness, efficiency and satisfaction with which users achieve their goals in a particular environment.

In a major study of the costs of programming that eventually led to the design of the Ada programming language, the United States Department of Defense discovered that 90% of the cost of producing software was in debugging. In another piece of research by IBM [3], over a large range of software, both in terms of size, and in terms of programming languages used, it was discovered that the number of bugs in a program did not grow linearly with the length of the program, but as a power function

$$b \propto s^{1.5}$$

In other words, a program 10 times longer has around 30 times more bugs, or alternatively, a program of one-tenth the size costs 3% of the larger program.

What this suggests for language design is that the efficiency part of the HCI equation would best be met by programming languages that are designed to be as compact as possible.

ABC

In earlier work, the author was involved with the design of a programming language for beginners [1]. Although this work was done before the term HCI was in common use, it was designed with what would now be recognised as classical HCI techniques: requirements analysis, iterative design, and user testing. The resulting language ABC turned out to be more powerful than was originally foreseen, being

Published in: van Leeuwen, JP, Stappers, PJ, Lamers, MH, Thissen, MJMR (Eds.) *Creating the Difference: Proceedings of the Chi Sparks 2014 Conference*, April 3, 2014, The Hague, The Netherlands.

useful for 'real' programming as well, and not just for beginners (and in fact went on to become the basis for the programming language Python [4]).

Although the language used mostly classical programming and control structures, such as assignment, procedure and function calls, if and while statements and so on, experience showed that programmers were around an order of magnitude faster at programming than with the classical programming languages it was compared with, such as Pascal, Basic, or C [5]. The main reason behind this was the use in ABC of a small number of high-level data structures. Most programming languages provide a set of low-level data structures, which may then be used to design and build higher-level structures. A key realisation with the third iteration of ABC was that it was the high-level data structures that the programmers needed, and they hardly ever used the low-level structures except to build the higher-level ones. An analysis of programming structures in programming led to a set of 5 data types in ABC that provided as a primitive the essential data structuring that programmers really needed.

XFORMS

XForms [6] is a programming language that investigates another aspect of programming: the control structures. XForms was originally designed (as the name suggests) for Forms applications, but in its second iteration became generalised so that more general applications could be written with it as well.

XForms is unusual in that many of the administrative tasks normally associated with programming are left to the computer to solve. The program is stated far more in terms of *what* needs to be solved than *how* to solve it. As a result, program size, and therefore programming time, is sharply reduced. Experience with several projects shows an order of magnitude reduction in project times.

This demonstration develops an interactive map application in XForms [7], ending up with a functional, working application in around 150 lines of code, an application that would normally require tens of thousands of lines in a traditional language such as Javascript. An interesting property of the resultant program is that it doesn't contain a single while loop (in fact it couldn't, because such a thing doesn't exist in XForms).

A NOTE ON NOTATION

XForms is expressed in XML. This is principally to allow it to be integrated with diverse other XML languages such as XHTML and SVG. However, this is purely a notational issue. Whether you write

```
integer i;
```

or

```
i: integer;
```

or

```
<bind ref="i" type="integer"/>
```

makes no difference to the concepts being described.

A TASTE OF THE PROGRAM

XForms does not normally work in a "first do this, then do that" style of programming that most languages use, but specifies relationships between data that the system autonomously keeps up to date.

For instance, if you have several pieces of data

```
site: http://tiles.osm.org/
zoom: 10
x: 526
y: 336
```

and specify a relationship:

```
url= site+zoom+"/"+x+"/"+y+".png"
```

then the URL will always be kept up to date if the underlying data changes. (Note that this is not an assignment in traditional programming terms, but a unidirectional invariant relation).

Furthermore, if you choose to output the image connected to the URL:

```
output(url, "image/*")
```

then the image on the screen will change as the data does.

If you then take a position on the world's surface in the coordinate system used by the map server:

```
posx: 34477602
posy: 22058667
```

you can then specify the relationship between this position and the map tile that that location is on:

```
x=floor(posx ÷ scale)
y=floor(posy ÷ scale)
scale=2^(26-zoom)
```

(since these are invariants, the ordering doesn't matter; 26 in this case is a function of the tile size and the maximum value of zoom, so that scale represents the number of locations on a tile at any particular level of zoom).

Hence, any time the position gets updated, so does the URL, and so does the display of the corresponding map tile.

Similarly, if the value of site gets changed to another tile server (for instance to a server that serves maps in a different style) as long as the server uses the same coordinate system then it is a trivial issue to display a map in that different style.

CONCLUSION

XForms programming requires a different style of programming than with traditional languages, which can be sensibly compared with how spreadsheets work.

Experience has shown that such a style of programming greatly reduces programming time. One very-large-scale project that tested XForms in a process that a company had done many times before reduced the time and staffing from five years with a team of thirty to one year with a team of ten, in other words from 150 person years to 30; another company that translated a large collection of Javascript programs to XForms reported that the resultant XForm programs were around a quarter of the length of the corresponding Javascript (which plugging into the equation quoted above would lead you to expect a saving of programming time of around a factor of 8).

The program presented here is itself a mere 150 or so lines of code, which is an order or two magnitude less than the equivalent code that would be needed in a traditional language like Javascript.

REFERENCES

1. Geurts, Meertens, Pemberton, The ABC Programmer's Handbook, Prentice-Hall, 1985.
2. ISO 9241-11:1998 Ergonomic requirements for office work with visual display terminals (VDTs) -- Part 11: Guidance on usability. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=16883
3. Brooks, FP, The Mythical Man-Month. Addison-Wesley. ISBN 0-201-00650-2, 1975.
4. Bruce Stewart, An Interview with Guido van Rossum, ONLamp.com, http://www.onlamp.com/lpt/a/2431, 2002.
5. Steven Pemberton, An Alternative Simple Language and Environment for PCs, IEEE Software, Vol 4, No. 1, pp 14-22, Jan 1987.
6. John M. Boyer (ed.), XForms 1.1, W3C 2009, http://www.w3.org/TR/xforms/
7. Steven Pemberton, A Map Application with XForms, http://www.youtube.com/watch?v=2yYY7GJAbOo, 2014.