

Re: Obstacles for Tcl/Tk commercial application development ?

Source: <http://coding.derkeiler.com/Archive/Tcl/comp.lang.tcl/2008-02/msg00079.html>

- *From:* Cesar Rabak <csrabak@xxxxxxxxxxxxx>
 - *Date:* Sun, 03 Feb 2008 02:55:48 -0300
-

Óscar Fuentes escreveu:

Darren New <dnew@xxxxxxxxxxxxx> writes:

[snip]

Static typing doesn't give you **any** guarantees of correctness. Maybe it helps good programmers be better, but it doesn't give you any guarantees. That's exactly 100% what I'm saying. Why are you arguing with me? ;-)

Are you teasing me? I guess not, as you write too much for this discussion being a joke.

```
proc foo { a b } { return [expr { a + b }] ;# may bomb! }

int foo(int a, int b) { return a + b; // just fine }
```

It depends on what kind of comfort you're after. The may bomb above at run time against 'just fine' after a compilation step, right?

The distance for 'correctness' may be smaller than you seem to believe, because in the Tcl example being a dynamic and interpreted language just after the definition of foo you may run some other code and see the 'bomb' still an error to be corrected in the development. Whereas the C equivalent would require compile and link before you can even think of using the function foo.

[snip]

Yah. I'm just discussing in general. I like calling people on things when they make absolute statements that I think are inaccurate, because occasionally I learn a new absolute truth! :-)

Re: Obstacles for Tcl/Tk commercial application development ?

Please quote the text where I made an absolute statement or claimed a "new absolute truth".

It seems that we are on a different wavelength. As I insist on claiming a *limited* but quite real effectiveness of static typing on software quality, you insist on an "all or nothing" view.

If you admit that software testing is necessary, how can you dismiss the value of a feature that guarantees that *some* bugs does not happen? On a dynamic language, you need lots of test cases for checking something that a statically typed language checks while compiling.

Because most of the bugs due type system are not due errors on the internal use of the types but due the conversions we as programmers have to do in order our programs be useful.

In your oversimplified example above the most probable error will not feeding something different than two ints in the foo function but overflows due limited number of bits, misconversion from user entry to the program, etc.

Assymptotically the type system becomes less and less useful until its contribution to the robustness of the systems is marginal and the only attempt of having a program with less errors is thorough testing and and a lot of walkthroughs.

All this discussion is summarized, from my part, saying that you have some opinions wrong (see the first paragraph of this message) but I keep failing to change your mind, partly because you ignore my warnings about some things falling outside the coverage of static typing. If you are interested on pursuing further a clarification about the usefulness of static typing, ask somebody with a better English skills than me :-)

Fair enough. From my part it seems that these discussions about 'advantages' of static type system in a forum for a non typed language¹ tend to over advertise their 'advantages' not taking in account the different approaches to development brought by the paradigms.

Last, but not least, in the case I need to implement some bizarre feature on my datetime or string handling, be assured that the compiler will point me to the "interesting" places that possibly need to be changed and, after the change, it will refuse to compile code that does not obey the checks that describes the new feature. (BTW, this is something C++ does not do well, the first part due to

Re: Obstacles for Tcl/Tk commercial application development ?

automatic type casts, and the second part due to poor expressiveness for type constraints, something that the C++ community is trying to address on the next standard).

Yes exactly due the fact that for languages being reasonable useful you need to get rid of an annoying 'type cop' otherwise you end up programming by demonstrating theorems (like Haskell) that shines in Computer Science but are not capable of producing a GUI type system so you have to use the syntax to call external libraries without any advantage and, worse, get crippled by the false expectation of the automation of the language for catching errors.

[1] slightly simplified as shown elsethread.

.