Some Software Technology Myths and Trends

Boyko Bantchev

http://www.math.bas.bg/softeng/bantchev

Department of Software Engineering
Institute of Mathematics & Informatics
Bulgarian Academy of Sciences
Sofia, Bulgaria

What Is a Myth?

myth:

2. a widely held but false belief

Compact Oxford English Dictionary

3. a fiction or half-truth, especially one that forms part of an ideology

American Heritage Dictionary

Some “consensus” truths

- Software engineering is engineering

Some “consensus” truths

- Software engineering is engineering
- Object orientation is universally applicable and superior to all other approaches

Some "consensus" truths

- Software engineering is engineering
- Object orientation is universally applicable and superior to all other approaches
- Object orientation is . . .

Some "consensus" truths

- Software engineering is engineering
- Object orientation is universally applicable and superior to all other approaches
- Object orientation is . . .
- Useful programs are necessarily complex. Complex programs are necessarily large

Some "consensus" truths

- Software engineering is engineering
- Object orientation is universally applicable and superior to all other approaches
- Object orientation is . . .
- Useful programs are necessarily complex. Complex programs are necessarily large
- "Academic" programming models are inadequate to real-world problems

Some "consensus" truths

- Software engineering is engineering
- Object orientation is universally applicable and superior to all other approaches
- Object orientation is . . .
- Useful programs are necessarily complex. Complex programs are necessarily large
- "Academic" programming models are inadequate to real-world problems
- Commercial software is more reliable than non-commercial

Some "consensus" truths

- Software engineering is engineering
- Object orientation is universally applicable and superior to all other approaches
- Object orientation is . . .
- Useful programs are necessarily complex. Complex programs are necessarily large
- "Academic" programming models are inadequate to real-world problems
- Commercial software is more reliable than non-commercial
- Interfaces etc.

Engineering?

Software construction is so specific a discipline that we would better not use the word “engineering” – it raises unfulfillable expectations in the wide public (and managers).

We are far from being able to build from component blocks.

Wang W.-L. *Beware the engineering metaphor.* Comm. ACM, Vol. 45 (2002), No. 5, pp. 27-29

Myths about Object Orientation

Many believe OO software construction is:

- universally applicable (the natural way to go);
- superior to all other aproaches;
- universally accepted;
- a way (or *the* way) to achieve reuse;
- about building hierarchies;
- inherently class-based;
- in fact, *C++* (or *Java*, *Eiffel*, *Smalltalk*, . . .);
- etc.

Universal? Superior?

Nothing proven in this respect. Many examples show otherwise (operating systems, DBMSs, . . .).

Not all properties and relations can be (naturally, easily) accomodated in a hierarchy. Attempting to do so often results in bloated code, mostly irrelevant to the problem being solved.

"[. . .] it was disappointing that flatness might beat nested definitions in achieving terseness and clarity." (*Peter Landin*, an early anticipation of the insufficiency of hierarchies.)

Accepted?

In fact, many of the most knowledgeable professionals have strong objections:

- overhierarchicalization;
- inability to express symmetric relations and sharing;
- lack of consistency in modeling techniques;
- inappropriate coupling of data and procedures;
- has to do more with programmers' mediocrity than with system design;
- "cliche-ability" (being convincing but not workable, yet hard to rebut)

Accepted?

- Paul Graham, on OOP in general and on *Java*
- Richard Gabriel, *Objects have failed*. Notes for a debate at OOPSLA 2002
- *An interview with A. Stepanov*. Edizioni Infomedia srl., 2001
- Johnson S.C. *Objecting to Objects* (invited talk). USENIX Tech. Conf. San Francisco, CA, 1994
- Pfister C., Szyperski C. *Why objects are not enough.* Proc. First Intern. Component Users Conference (CUC'96), Munich, Germany, 1996

Accepted?

- Hatton L. *Does OO really match the way we think?* IEEE Software, V. 15 (1998) # 3, 46-54
- *Object Oriented Programming Oversold!*
- *OOP is much better in theory than in practice*

On Reuse

There is no clear confirmation. Many times doubted. Depends on many factors.

Reuse seems to have more to do with genericity (polytypicism, e. g. templatization) than with OO: cf. *C++*'s STL, Boost, TR1, and TR2, and *Java* and *C#* adopting generic constructs – that is where the today's progress in these languages is.

Hierarchies? Classes?

There are other models of object orientation!

Communication vs. internal structuring:

"[. . .] it is not about classes. I'm sorry that I long ago coined the term 'objects' for this topic because it gets many people to focus on the lesser idea. [. . .] The big idea is 'messaging'"

Alan Kay, the inventor of Smalltalk

Prototyping vs. class building

Self, *JavaScript*, *Rebol*, in part *Ruby*

Hierarchies? Classes?

Even staying with hierarchies, there is a better way – ***dynamic classes***.

Benefits: no need for so much subclassing; system's behaviour can change dynamically.

Already long experience with *Smalltalk*, *Objective-C*, *Common Lisp*; now *Ruby*, *Python* go the same way.

Smalltalk is revived. *Objective-C* is considered a major reason for the success of Apple's Cocoa platform.

Usefulness, Complexity and Size

Does a useful program have to be complex?

Does a complex program have to be large?

These questions are important, because, too often, the inverse logic is applied: large hence complex, complex hence useful.

Most large software companies will try to convince us that the answer to both questions is “yes” (and will inevitably add “and expensive, too”).

However . . .

Usefulness, Complexity and Size

There is plenty of counter-evidence.

In *Unix*, *Smalltalk* and other cultures, there is a strong tradition to build small, simple tools that can be glued together in many ways, producing new tools. Utility is achieved through **simplicity** and **combination**.

Kx produces the fastest DBMS in less than 200K, including a full-featured programming system, a simple GUI, and support for all major Internet protocols.

(Not a toy at all – check who their customers are!)

Usefulness, Complexity and Size

Rebol is an “Internet programming language and system” with tens of useful datatypes and a wealth of built-in operations on them, support for graphics programming and for building GUIs, and virtually all Internet protocols. It competes successfully with *Java* in the market of small Internet applications. The size of the insallation file is about 600KB.

Many other very well developed, feature-rich programming systems, with extensive libraries, are no larger than several MB.

Usefulness, Complexity and Size

Experimental evidence shows that:

Given the same problem to code in different languages, and having similar level of experience, programmers can achieve solutions that vary wildly in development time, size and correctness, depending on the language used.

Notably, some functional programming languages seem to do much better than the traditional, including OO ones, in all the three respects. (The ratios, of both time and program length, are in the range 10-20!)

References: [1], [2], [3], [4]

‘Academic’ Programming

A well developed, but still considered by many not suitable for real programming, is the *functional programming* paradigm.

A long list of myths (and some rebuttals) related to FP can be found e. g. here.

Functional Programming

Among the most persistent myths are:

- FP means *Lisp*, and *Lisp* is old
  (neither is true)
- FP is too difficult to learn and use
  (this is mostly a matter of habbit and inertia;
  many universities successfully teach *Haskell* in a first programming course.)
- FP interpreters generate slow programs
  (neither they are only interpreters anymore, nor the code is slow)

Functional Programming

- FP is not used for real-world programming (notable counter-examples: *Yahoo*'s webstore, *Ericsson* (*Erlang* with some 2000 programmers), *Darcs*, *Linspire*, *Pugs*)

Functional Programming

FP is not only FP languages:

- it pervades *C++* and other languages (notably, most scripting ones);
- it now goes to *C#* and even to *Visual Basic* (anonymous functions (lambda-expressions), list-processing, type inference);
- most than a half of the known to be .NET languages are FP, or have a strong FP subset

Commercial Software More Reliable than FOSS?

Empirical data ([1], [2], [3], [4]) and UNCTAD documents ([5], [6], [7]) say that free and open-source software (FOSS) tends to be more reliable than commercial software.

Trends

- scripting – many changes expected
  (see e. g. [1], [2], [3], [4])
- textual data (not images, not sound) continues to be dominating
- semi-interpreted programming languages, dynamicism
- metaprogramming
- functional programming per se and pervading basically imperative languages
- dynamically/statically typed languages both continue to grow

Trends

- multiparadigmatic programming (*Leda*, *Beta*, *Scala*, *Mozart/Oz*)
- server-based software
- distributed data, as well as distributed *programs*
- new attempts at component-based software construction