

A Challenge to the GUI-Centric Paradigm of Information Education for the General Public

Chao-Kuei Hung and Yen-Liang Shen
Information Management Department,
Chaoyang University of Technology, Wufeng, Taichung, Taiwan

Abstract—We contrast the current paradigm of information education for the general public with that of mathematics and of foreign languages. This contrast exposes the questionable mentality, in IT education, of emphasizing ease of use at the cost of life-span of knowledge and the combinatorial nature of knowledge. We propose for educators to carefully choose only software tools that conform to the *longevity* and *combine-ability* criteria. This includes insisting on teaching technologies that are cross-platform and support open interfaces. The prevalent view of information technology as a field of commerce is identified as a possible source of confusion and the free/libre/open source software movement as a possible solution. GUI and ease of use are more clearly seen as convenience and help, instead of the main body of knowledge when we constantly remind ourselves of the analogy. Hopefully investment in future IT education for the general public will have longer term effectiveness and produce the combinatorial explosion effect as we see in other fields of education.

Index Terms — longevity, combine-ability, combinatorial, command line, regular expressions, open interface, open file format, cross-platform, free software, open source

I. INTRODUCTION

Today the Information Technology (IT) education for the general public usually takes the form of learning various popular application programs (AP) with convenient and attractive graphical user interface (GUI) such as Microsoft Word and Microsoft PowerPoint.¹ The conventional wisdom is, or at least the existing curricula both in the education and commercial sectors imply, that the computer should be made easy to use for the general public, exposing the less technical details the better. [1] Command line interface (CLI) and regular expressions (regex), for example, are considered too technical, thus nearly nowhere to be found in high school computer curricula. Yet are they really more difficult than trigonometry, or than the English language's tenses and moods for an oriental student whose mother tongue does not express tenses by changing verb forms at all?

In this paper we propose a thought experiment to challenge this attitude and strongly held, even though never-proven preconception. Learning computer is contrasted to learning mathematics and to learning foreign languages. We hope that such contrast will reveal hints on how to drastically change the direction of present day IT education in order to produce more effective and long lasting results.

¹See for example the web site of the European Computer Driving Licence: <http://www.ecdl.com/main/>

II. CONFUSING MATHEMATICS EDUCATION WITH MATHEMATICAL SOFTWARE OPERATION

Is it possible that we have confused IT education with software operation? More specifically, how much emphasis should we place on software operation while designing an IT curriculum for the general public, and what other ingredients, if any, should be placed in the curriculum to complement software operation? To answer this question, we will make a thought experiment by applying the prevalent IT education paradigm to mathematics education and see the inadequacy of the result. Specifically we will deliberately confuse mathematics education with mathematical software operation.

Imagine we have a powerful model of calculator CK05 (of the famous brand name Chao-Kuei, not Calvin-Klein). It does more than just the arithmetic and trigonometry. Every formula a high school student or even a non-math major college student should know, is included in its user-friendly menu system. Euclid's algorithm for finding the greatest common divisor of two integers, for example, is no longer a difficult algorithm for the students to learn and understand. Instead it is just a matter of clicking through the menus and filling in the integers in a friendly and obvious dialog box. In fact the system is not structured by formulas – that would be user-unfriendly. “Gaussian elimination for finding the answer to a system of linear equations” is too intimidating a name for the general public even to read. These should be implementation details which only the programmers and the mathematicians care. In the menus there will be many entries listed in the form of application problems such as “finding the number of chickens and rabbits given the number of total head counts and foot counts” or “finding the amount of investment in each category given the total profit and the interest rate of each category”, etc. Learning mathematics for a non-math major should never be more difficult than learning how to navigate the menu system of CK05. Why should a business major ever be bothered with the details of simplex method, or even its name at all? These again should belong exclusively to the realm of the mathematics majors. With the comprehensive, user friendly, and attractive menu system, CK05 is the way to go for the mathematics education of our future generations. Mathematics teachers still teaching in the old way or considering using some other less capable products should heed the potential troubles of their unpopular choices to say the least.

The authors apologize if all this sounds too commercial and

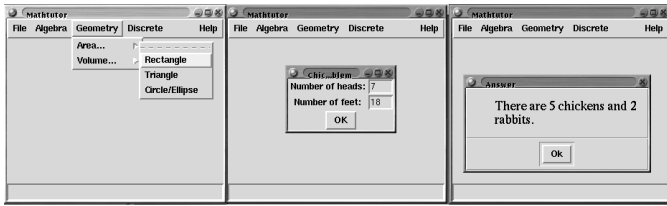


Fig. 1. CK05 calculator provides easy access to many math problems. No more math hassles! Tedious and difficult math formulae/proof/derivations are no longer needed in our math education for the general public!

ridiculous. Yet it seems inevitable, be it adequate or not, that *arguments and educators' attention be misled to friendliness, ease of use, or even feature comparison among commercial products if mathematics knowledge becomes deeply entrenched in a commercial product.*

III. WHAT'S IMPORTANT IN MATHEMATICS EDUCATION?

What else then should be the focus of education? Even without any formal training in the theory of education, one sees a few important features of education as one begins to reason about the inadequacy of the above hypothetical situation. We are certainly capable of realizing such “mathematics education” with today’s information technology, and yet we never let it happen, for good reasons.

- The main body of the knowledge should have a lasting value.
- It is important that students learn to analyze, to decompose a problem into smaller, familiar sub-problems.
- It is important that students learn to synthesize, to combine the solutions for smaller, familiar sub-problems into the solution for the original, larger problem.

Let’s call the first item “the longevity criterion”, and the last two collectively “the combine-ability criterion”. These observations apply to the education of foreign languages as well. We could conceivably write software to help students learn compound sentences, complex sentences, finite and non-finite clauses, etc. but the software itself would never be the main body of the education itself. For one thing, the operational knowledge of the software itself may not have a long life span. Besides, we expect the students to be able to analyze and synthesize the sentences without the help of the software some day even though we might actually employ such software during some stage of the learning process.

The long life span of our knowledge in mathematics and/or natural languages makes it possible for us to learn these subjects in our youth while still being able to apply them many years later. The combinatorial nature of our knowledge makes it possible for us to memorize a relatively small amount of building blocks (say a 3000 word vocabulary) while being able to compose a huge amount of possible solutions (say a million sentences) and able to apply the solutions in vastly different, seemingly unrelated contexts. (as in the example of the system of linear equations) Not unlike Lego games, the effect of combinatorial explosion is what makes such knowledge valuable and desirable. It is also similar in spirit

to the orthogonality of the instruction set of a computer architecture.²

Of course, learning a subject in this “combinatorial way” also demands a higher price than learning it the “CK05 menu calculator way”.

- One’s familiarity with the (linguistic or mathematics) building blocks needs to go far beyond passively recognizing the terms in the menu.
- One needs to learn to find the right blocks to use in an unfamiliar context.
- It takes efforts to compose a syntactically correct and semantically meaningful combination of the blocks.

These are certainly challenging to non-math or non-linguistic majors, and yet we do not abandon the combinatorial way in favor of the menu way. It is unlikely that we will develop enough technology for the computer to read the user’s mind or user’s ambiguous natural language in the foreseeable future, much as we don’t expect machine translation to replace the general public’s need of learning foreign languages any time soon.³ There is no healthy reason why we should choose the more convenient but less effective alternative when dealing with IT education. The convenient alternative prevailing in today’s IT education is in direct contradiction to our general consensus that IT education for the non-technical, general public is of commensurate importance with math and foreign language education.

IV. WHAT'S IMPORTANT IN IT EDUCATION?

If we choose to pursue the combinatorial way, what should we look for as the long-lasting, combine-able building blocks for IT? We propose two possible lines of thought.

The first line of thought recognizes the similarity between the command line interface and a natural language. [2] In fact one easily recognizes the command line as an imperative, the command as the verb, the options as adverbs, and the command line arguments as the objects. For example, `ls -l /usr/share/doc` directly translates to “Please list, in detail, the contents of /usr/share/doc.” In this view, the mechanism that combines simple blocks into larger constructs are such shell features as the pipe and the back quotes. The Linux Professional Institute Certification (LPI)⁴ seems to agree with this line of thought, but their emphasis is more on the building blocks (the commands) than on how to combine them (the pipes and the back quotes).

Regular expressions is another mini-language that comes to mind. It has been repeatedly and independently mentioned in [3], [4], and many other sources discussing IT education as an important yet largely neglected subject. With only a dozen most frequently used symbols and at most 3 dozen symbols in total, it is even simpler than the CLI. It is of course both long-living and combine-able.

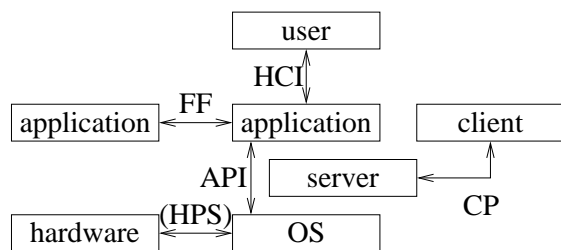
²The latter becomes less important nowadays only because humans no longer interact with machines at the assembly language level.

³One ponders, for example, how it is ever possible to replace regular expressions with a “less intimidating” user interface even a hundred years from now.

⁴See <http://www.lpi.org/>

Not only do the CLI and regexp have combine-able blocks within their respective domain, but also they combine well with other larger block of IT. The same CLI is available across all flavors of Unix systems, OS/2, and MS Windows [5]. Regexp takes combine-ability one step further. Being available in almost all popular programming languages Perl, C/C++, etc. and in many end-user AP's such as vim and less, it is not only platform-independent, but also language- and AP-independent.

And this brings us to the second line of thought – at the level of software components. Be it a GUI application or a CLI command, an AP almost inevitably needs to communicate with other AP's through some kind of interface. It is therefore essential that IT educators insist on teaching only software that supports open interfaces if we agree that the combine-ability criterion is important. In fact, open interface is not only the key to combine-ability, but also the key to longevity as we shall discuss immediately.



Choose open interfaces. Keep your options open

Fig. 2. To meet the longevity and combine-ability criteria, IT educators should insist on teaching software that can be decomposed into components and can exchange components with other software, i.e., software that supports open interfaces.

V. A PLEA TO TEACH ONLY TECHNOLOGIES SUPPORTING OPEN INTERFACES

The reason for requiring that software be decomposable into relatively independent and replaceable components is more than of aesthetic or theoretical interest. Consumers' and educators' failure to insist on this requirement results in damaging social effects, as is currently the case in the IT world. We will elaborate on this point using the most typical interface – file format (FF) – as an example. Specifically, we investigate the dynamics of software vendor's monopoly on the word processing technology through the propagation of the distorted conception of “popularity” and “compatibility”.

It is unfortunate that the term “popular” is usually understood, in the IT field, without consideration of the depth of time. It is frequently taken without second thought that an FF is popular if it is in heavy use by the population *at the time of writing or speaking*. This psychological propensity of the general public gives an AP vendor great incentives to release gratis copies of software that creates proprietary FF's which only AP's from the same vendor can import/export. Later versions of the AP could then be sold at a high price since the users' files have been locked in this proprietary FF. They could not use AP's from other vendors or AP's

in the creative commons to further process their files, and they grow exclusively dependent on this vendor. Yet through exchanging files with associates and colleagues, the general public collaboratively further the “popularity” of this FF, that is, until a newer version of the FF appears from the vendor, possibly without downward compatibility. At this point, the public forget that it is the old format that is popular, not the new format. Their mentality unconsciously switches from requiring popularity to requiring catching up with the new trend – even if it means giving up the presently popular FF and the presently popular AP. The old “popular” FF becomes history and the cycle begins again.

Such misconceptions of popularity and compatibility result in a trend to “standardise” on a certain *software*, when in reality it would be much more sensible to standardize on an *interface* instead. [6] Therefore we “standardize” on Microsoft Word in order that we may exchange files, and “standardize” on Microsoft Windows in order that we may run the same application on different computers. Monopoly receives consumers' and IT educators' collaboration and is inevitable. In reality *requiring sameness is exactly the indication of lack of compatibility*.⁵ We therefore urge all IT educators to stop the questionable practice of distributing *and* requiring submission of files in proprietary FF such Microsoft .doc to/from the students. Such practice is partly responsible for the illegal copying of software, indirectly promotes software monopoly, *and unnecessarily widens digital divide at the same time. It should not be treated merely as a technical issue. Its adverse social dynamics need to be clearly explained to the students and the practice itself be discouraged as an ethical issue.*

But we digress. The reason for insisting on open interfaces from the educational viewpoint is the two key criteria – longevity and combine-ability. We have explained it in terms of FF. Let's turn to the OS level for another example. No one knows which operating system will be most popular 10 years from now. Learning AP's that are locked into one single OS/hardware platform is making an investment without regard to the longevity criteria, and greatly limiting the possibility of combining it with other software components at the system level. We therefore urge IT educators to choose cross-platform AP's in preference to single-platform alternatives. Again we make such a statement with reference to the two criteria we learned from mathematics and language education. We will leave the debates about the feature comparisons and ease of use among various alternative AP's and alternative OS's to people who are only interested in today's technology snapshot.

Other interfaces for AP's to communicate with each other include network communication protocols (CP), application programming interfaces (API), etc. The same argument applies to these interfaces. In the long run, it will be most effective for IT education resources to be devoted to AP's that have good support for open, publicly available interfaces at the FF level, CP level, and/or API level. There has long been a call for the public awareness of such issues. [8] [9] However,

⁵Obviously we don't require cell phones to come from the same vendor, or even to be of the same model, in order for them to communicate.

a precise definition of “open interfaces” does not seem to exist until recently, and mostly centered around open file formats alone. [10] In short, there is really a very strong need for the consumer, industrial, open source, legal, government, and academic communities to reach a consensus on an exact definition of open interface. It will have to encompass not only such interfaces as mentioned above, but also those emerging from technologies/scenarios such as client-server architecture, application service provider, etc. For lack of a precise definition of “open interface” today, we propose an easy approximation. A relatively safer choice of AP’s to avoid vendor lock-in and to achieve the longevity and combine-ability criteria is to mix components from competitive vendors and from FLOSS equivalents. If two components do not talk to each other nicely, it is likely that at least one of them does not actively support open interfaces.

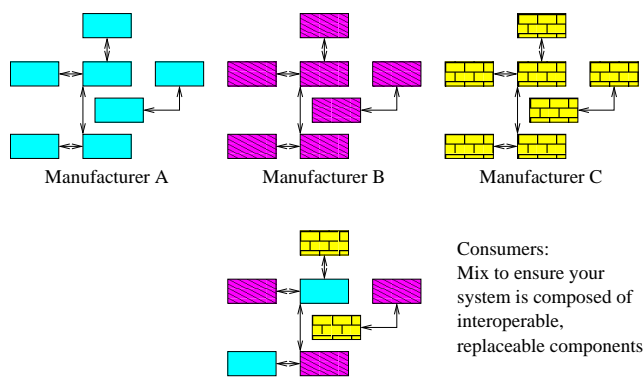


Fig. 3. Mix components from competitive vendors and from FLOSS equivalents to ensure interoperability. For lack of a precise definition of “open interface” today, this provides an approximation.

VI. IT AS A FIELD OF COMMERCE OR IT AS A FIELD OF KNOWLEDGE?

It is possible that most IT educators did not have a chance to make a choice between the menu alternative and the combinatorial alternative simply because they didn’t know that the latter existed. Very unlike mathematics or languages, IT has been more a field of commerce than a field of knowledge. As is the case with many fields of commerce, immediate or short term results are much easier to sell than long term results. In conversations and decision makings, ease of use and user friendliness naturally replace longevity of knowledge and combine-ability when sales figures are prime motives at work. Even if a high school teacher knows of and seriously considers the combinatorial alternative, she doesn’t have 10 years to verify her theory. Of those few who seriously think that the combinatorial way is likely more beneficial to the students from a long term point of view, even fewer would decide against the surrounding atmosphere of looking for immediate results.

On the other hand and from a proprietary software vendor’s point of view, longevity of software is nearly exactly contradictory to profit making. The combine-ability criterion appeals only to small vendors. For a vendor that already captures the

largest share of the market, it is only natural that combine-ability with AP’s from competitors be avoided in order to raise the barrier of entry of competition. Compounding this observation with the psychological tendency of the public to follow the choice of the majority, monopoly is inevitable. Combine-ability thus becomes a difficult goal which can only be achieved through reverse-engineering, and this can sometimes even face legal threats such as software patents.

Might it be better for the world in general and might it be more resource-effective for IT education in particular if IT is viewed more as a field of knowledge than as a field of commerce? The answer may lie in the free/libre open source software (FLOSS) movement. The atmosphere may become friendlier to the longevity and combine-ability criteria of knowledge when the prime motive at work is reputation, like mathematics and any other sciences in general. [12]

We note that open interface and open source are not equivalent. Interfaces provided by open source software are always open, since the code itself is open for everyone to study. Whether such interfaces are in popular use is another issue. However, presently unpopular open interfaces may turn out to be popular *in the long run*. The *sxw* format of OpenOffice.org is likely to become such an example. Or such open interfaces may after all remain unpopular, in which case later they can always be adapted to other more popular open interfaces through filtering programs, etc. If we draw a spectrum of software according to the degree of freedom software users enjoy, open source software lie at the most free (as in free speech) end. To make an analogy, open source software is like a telephone set with transparent shell, revealing all of its inner circuits. It is worth emphasizing that there are proprietary software that support open interfaces. They lie in the middle of the spectrum. They are like ordinary telephones whose circuits are proprietary but which talk to other makes and models of telephones nicely. At the other extreme of the freedom spectrum lie software that promotes closed interface to achieve vendor lock-in, which we term “opiumware” for lack of a more descriptive name.

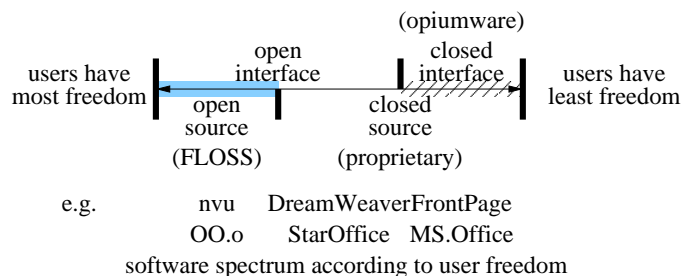


Fig. 4. Three levels in the spectrum of software freedom. Open interface is a must. Open source has many extra benefits.

Openness of source code is not the focus in this article, but we cannot avoid noticing that many FLOSS tools seem to meet both criteria better than most proprietary tools. Also, with its source code openly available, a FLOSS program more readily becomes cross-platform with relatively less extra modification

efforts once the compiler and the development libraries are ported to a new operating system. Cygwin on the Microsoft Windows systems is a case in point. [5] It may be worth while for IT educators to check out education tools such as the freeduc-cd [13], which provides both tools meeting the longevity and combine-ability criteria, *and tools that are easy to use* in one single CD that does not require installation. Besides, it is encouraged to duplicate, modify/adapt, and distribute such CD's much as it is encouraged to duplicate, modify/adapt, and distribute knowledge. This is certainly important for an IT educator facing the moral dilemma of forcing students to buy software or indirectly encouraging illegal copying as is the case with proprietary software.

VII. EASE OF USE IS NOT BAD

A short clarification is necessary lest the readers misunderstand the authors point. All this discussion is not to deny the usefulness of GUI or other user-friendly software in education. They may:

- help students understand the abstract ideas [14];
- reduce the amount of mechanical and repetitive work [15] [16];
- hide low level details that are irrelevant to the subject of study [17]

They are useful tools but they are not substitute for the body of the knowledge itself. The CK05 calculator might be a really useful tool after all. If a newer version of the CK series calculator obsoletes the CK05 calculator, however, we throw away the calculator, but not the mathematics knowledge. We use tools; we simply don't identify the tools with the knowledge. Much less do we emphasize the importance of GUI and ease of use to the exclusion of the more difficult, but long-lasting and combine-able building blocks.

VIII. CONCLUSIONS

In summary, we propose to scholars and educators in the IT education field that the following elements be given considerably more emphasis in favor of the operation of GUI software:

- regular expressions
- introductory CLI
- composing long commands using pipes and back quotes
- editing an html file with plain text editor plus tidy
- the social and hence ethical concerns raised by distributing files in proprietary formats
- the importance, from a consumer point of view, of having the option of breaking a piece of software into components exchangeable with components from other vendors or from FLOSS

The LPI tests need to put more emphasis on the combinatorial aspects and less emphasis on the memorizing aspects of the CLI. We also point out the urgent need to give a precise definition of open interface that the consumer right organizations and other interested parties can agree upon.

Of course analogies are not perfect and should not be taken to the extreme. Yet with today's IT education highly inclined in the GUI and easy-of-use extreme, there is ample room for us to move in the other direction. By constantly reminding ourselves how IT education has been heading in a rather different direction from that of mathematics and foreign languages, hopefully we will pay more attention to knowledge longevity and combine-ability in future IT curriculum designs, and have more long-lasting and explosive (as in combinatorial explosion) results with the society's investment in IT education resources.

REFERENCES

- [1] B. Brown and D. Kester, "College Students and Computers" The Education Resources Information Center., Lanham, Research Rep. ED366291, Apr. 1993.
- [2] A. Robbins, "Opening the Software Toolbox" *Linux Journal*, vol. 1, no. 2, article no. 9, April-May 1994.
- [3] M. Vermeer. (1998, December). Unix as an element of literacy (2nd ed.) [Online]. Available: <http://linxutoday.net/stories/1846.html>
- [4] C. Hung. (August 2001). How to learn computers effectively (23-Aug-2004 ed.) [Online]. Available: <http://www.cyut.edu.tw/~ckhung/a/c013.shtml> (traditional Chinese)
- [5] S. Chamberlain, C. Faylor, et. al. Cygwin Information and Installation [Online]. Available: <http://www.cygwin.com/>
- [6] E. Fair, (1997, October). Software standards versus protocol standards [Online]. Available: <http://www.clock.org/~fair/opinion/open-standards.html>
- [7] D. Raggett. Clean up your Web pages with HTML TIDY [Online]. Available: <http://www.w3.org/People/Raggett/tidy/>
- [8] J. Dennis, (2000, April). Protocols, APIs and File Format Libraries [Online]. Available: <http://old.lwn.net/2000/0504/backpage.phtml>
- [9] I. Smith, (2003, August). The Rise of Proprietary Formats [Online]. Available: http://opensource.mimos.my/fosscn2003cd/paper/slides/09_imran_william_smith.pdf
- [10] S. Baker, T. Hancock, et. al., (2003, March). Open file format definition [Online]. Available: <http://www.anansinspaceworks.com/Documentation/BuildImage/Legal/tosi.openformatdef.2003.03.19.html>
- [11] C. Burstein. Viewable with Any Browser: Campaign [Online]. Available: <http://www.anybrowser.org/campaign/>
- [12] C. Kelty. (1998, December). First Monday Peer-Review Journal on The Internet, vol.6, no.12, 2001. Also available [Online]: http://firstmonday.org/issues/issue6_12/kelty/index.html
- [13] H. Fernandes and G. Khaznadar. Freeduc-cd, a live-cd for education [Online]. Available: <http://www.offset.org/freeduc-cd>
- [14] H. Fernandes. Dr. Geo, interactive geometry [Online]. Available: <http://www.offset.org/drgeo/>
- [15] I. Searle. Rlab Web Site [Online]. Available: <http://rlab.sourceforge.net/>
- [16] J. Eaton. Octave Home Page [Online]. Available: <http://www.octave.org/>
- [17] C. Laurel. Celestia [Online]. Available: <http://www.shatters.net/celestia/>