Senses of Craft in Computer Science

Antranig Basman
amb26@ponder.org.uk

Abstract
Computer Science, yet to establish its role in the world, inhabits an invidious position between mathematics, engineering and craft — losing out by the comparison to each. We draw a distinction between different levels at which the term 'craft' could be applied and list four key virtues desirable for crafted items at a higher level. The current products of computer scientists or software engineers are deficient in these four virtues, but not necessarily so — we see no inherent limitations that require future incarnations of our field to be forever without them. We propose a shift in the value system underlying our work — away from correctness, towards truthfulness, rewarding the workmanship of risk, rather than certainty, and considering time horizons of generations rather than at most a couple of years.

1. Introduction
Computer Science, yet to establish its role in the world, inhabits an invidious position between mathematics, engineering and craft - losing out by the comparison to each. A recent line of discourse (Lindell, 2014; Blackwell & Aaron, 2015) considers that software development constitutes a form of craft.

We argue that the craft status of software development remains limited, in the crucial senses which mark the phenomenon and results of craft as highly desirable to society. In a looser sense, software development shares some of the aspects of a craft — in the same way that a DIY enthusiast, journeying to their local hardware store in search of cheap, highly standardised components with poor manufacturing values can be said to be indulging in a craft as they seek to make some more or less successful home improvements. But these are not the values we hope to appeal to as we seek to ennoble our "craft" by comparison with its more established brethren.

2. Virtues of Craft
As William Morris remarked as he began to establish the English Arts and Crafts movement in 1880, "Have nothing in your houses that you do not know to be useful, or believe to be beautiful" (Morris, 1880). A true craft produces artefacts that are

Beautiful - giving pleasure in use and contemplation
Durable - long-lived, stable, and if degrading, degrading gracefully
Ownable - can be the possession of their owner, living amongst them in their household
Handleable - fulfil a function which is immediately obvious to sight and touch

In contrast, the outputs of computer science are widely acknowledged to be ugly, brittle, masterful and obtuse.

Let us consider our virtues in turn.

2.1. Beauty
There is an aesthetics in computer science, but it is not for the user or rarely even for the developer. Any model for beauty with any currency is imported wholesale from mathematics — an algorithm can be elegant, a proof beautiful. In contrast, any actual piece of software, especially one which performs a useful function, is without fail fantastically ugly, reflecting the haphazard and throwaway values of real software culture. As Simon Peyton-Jones remarked at the PPIG keynote of 2015, of open source - "It's where people go to contribute, but it's not beautiful". (DeMillo, Lipton, & Perlis, 1979) have recognised that "Formal verification of programs, no matter how obtained, will not play the same role in the development of computer science and software engineering as proofs do in mathematics."

2.2. Durability

The lack of beauty in actual software is intimately tied up in the lack of possibility for any kind of durable expression — and still less, of any durable function. It is well-known that software will spontaneously "rot" as a result of a progressive violation of its environmental assumptions. Sadly, it does not rot gracefully in the manner of a forest log, but disgracefully in the form of simply failing to install or run correctly or indeed at all. The only hope one has for running an old piece of software is to create a complete replica of its original home (a "virtual machine") correct in every detail, and keep it captive there. If software were a lifeform, it would be an impossibly brittle and ill-adapted one that would mark in every case an evolutionary dead-end.

Unfortunately, our field does not so far have the courage to face up to this problem. Indeed, one of its leading lights, Alan Perlis, in (Abelson & Sussman, 1985) speculated whether software might not inherently lack durability, comparing it to a "soap bubble". We can't infer from Perlis' remark whether he believed software should lack durability or simply that we lack the understanding to make it so. We suspect the latter — and if most software didn't possess the beauty of a junkpile or shantytown rather than a soap bubble, we would be more sympathetic to its lack of durability.

2.3. Ownability

Because it is not durable, but also through further causes, software can never effectively be owned by its user. Software is less ownable now than it has ever been — with the prevalence of "leasing" models for software, and increasingly aggressively pushed "updates" which will sometimes interrupt vital human activities in order to deprive them temporarily or permanently of the use of the artefact that they just believed was theirs (Steve Hoffer, 2016). This is not the behaviour of a tool or a product, but of a tin-pot aristocrat transplanted into the user's home.

2.4. Handleability

Software is legendarily opaque in form and function. It is well-attested that the vast majority of users are unaware of or unable to exploit the majority of configuration or function which is in theory available to them (Jared Spool, 2011). The output of a craftsman, whether it takes the form of a spoon, bowl, basket or chair, even if it is of an unfamiliar form, typically appeals to an immediately obvious use value. Part of the problem could be ascribed to the novelty of software function, but users have little incentive to invest effort in becoming familiar with their software, because of our previous point — they have no control over its form or any ability to resist change. At any moment the familiar may once again be swept away by the unfamiliar.

3. Our Lack of Materials

Some of the discourse around software "craft" centres on the property of its "material". We argue that the core of the deficiencies of software can be ascribed to the lack of anything which properly qualifies as its material. A true material is the result of a natural process, accumulated and shaped for years if not millenia. When one applies craft to a natural material, one is engaging with the natural accident of this process when meeting difficulties of resistance. By contrast, when meeting resistance in crafting software, one is primarily tangling with the obtuseness of one's colleagues and their insistence in retaining authorial power for themselves.

Before software could become a craft, we need to construct its material. The durability of a true material, as well as enjoying the sheer ability of continued survival, also has the aspect of the ability to participate in creative networks. A product made of a material can be passed from hand to hand, at each point being seen in a different role or being the starting point of a fresh creative cycle.

4. Workmanship leading to Craftsmanship

(Pye, 1968) makes a crucial distinction between grades of workmanship — the "workmanship of risk", where the outcome is at risk at every stage, and under the direct control of the worker, and the "workmanship of certainty" where variability has been eliminated through a mechanical process. Pye is clear that only the former kind of workmanship can lead to true craft, whereas computer science exclusively

values the latter — in its use of proofs, abstractions, implementation insulation, and other techniques designed to take power away from the user. Pye is also clear to value the contributions of engineers, who alone are responsible for defining the nature of materials which lead to real results, as opposed to theoreticians and scientists who accept such materials as a *fait accompli* without assigning or recognising credit. (Gabriel, 2012) is also clear to point out that it is most often the case that principled engineering work long precedes formalisation.

(Lakatos, 1976) on certainty:

> "Certainty" is far from being a sign of success, it is only a symptom of lack of imagination, of conceptual poverty. It produces smug satisfaction and prevents the growth of knowledge.

5. Lutyens' Latch

As an example of the kind of craft we are appealing to, we exhibit the example of this door latch (Figure 1) designed by Edwin Lutyens, a prime proponent of the Arts and Crafts movement. Latches like these can be found throughout his restoration of Lindisfarne Castle which he undertook starting in 1902, as well as some of his other designs. It self-evidently enjoys all the virtues we have listed — it is highly beautiful (even achieving a kind of wittiness), and its method of construction, based on readily available and widely understood materials, is a direct guide to its function. In the unlikely event that it requires maintenance (having endured intact for over a century), this can be achieved through widespread means.

A lack of durable and natural materials, as well as a suitable craft culture, represent the primary deficits in computer science which prevent it from constructing crafted items such as these.

Figure 1 – Door latch designed by Edwin Lutyens

6. Spectrum of Forgiveness

Whilst the software that we have is constituted of a material far more brittle and less malleable than the least forgiving of the human crafts (for example, ceramics), being a product of the mind, it has the potential to be far superior to the most forgiving (for example, knitted textiles). The following table illustrates the affordances that we can expect from different materials.

Ceramics

- Once the product's form is fixed, it cannot be further altered in a graceful way.
- It is brittle and once damaged cannot be gracefully repaired (a glued repair is always obvious, and a damaged surface can never be reformed like the original)

Wood

- With some effort, an item can be reformed or repaired, for example by sanding, varnishing, repolishing etc.

- For some items, a broken element can be removed (unglued, unscrewed) and replaced with a new part which functions and appears as well as the original

Textiles

- Most items can be repaired, sometimes indefinitely, by patching or darning
- In some cases the original raw materials can be fully recovered by unravelling the garment and a completely fresh one constructed

In contrast, we have

Computer Science of the Present:

- A product may spontaneously disintegrate without warning, suddenly becoming wholly unusable
- No modifications may be made by the owner after delivery of the software, even through amateurish affordances such as superglue
- The raw materials for software are endlessly distant from the owner, separated by a series of barriers operated by successive priesthoods (compilers, linkers, integrators, hosting services)

Whereas nothing inherent in Computer Science prevents it from being

Computer Science of the Future:

- Any product, either in form, function, or both, may be preserved indefinitely in a working condition
- Any owner of software may modify and maintain it using only the affordances to hand — or else share or receive such modified versions from peers or communities of interest
- The raw materials for software are constantly immanent — any user of software, simply through an act of perception, may choose to see it either as raw materials, finished product, or anything in between

From its current position as the world's least forgiving craft, computer science could surpass all those currently known.

7. Resolution

Our field is making only tiny and halting steps towards craftsmanship. It is increasingly popular to show relevance to craftspeople by an appeal to our ability to work in their spaces or with their materials (see (Victor, 2014) or other "Maker" movements), but the only result will be to export our own aesthetics of frustration and power-worship into already functional communities. The experience of (Blackwell, Aaron, & Drury, 2014) exhibits the behaviour of craftspeople when provided with pieces of "techno-junk" such as a USB keyboard which doesn't work with a USB socket and other similar obstructions:

> *experiences were … "frustrating" — a word that was used consistently and repeatedly in the artists' reports*

with the result that

> *busy artists are equally likely to respond to obstacles by shifting their effort to other projects in which they are making rapid progress*

Computer scientists show little recognition of the frustration that their products cause in practice, and instead imagine that the world would be a better place if more people thought and worked like them — witness the rise of "Computational Thinking" (Wing, 2008) as the vehicle for this imperialism. Rather than the virtues we list above, the highest virtues of computer science are the tedious accountancy of efficiency and correctness. These will create a civilisation of bureaucrats rather than craftspeople.

In some quarters, there are signs of a change of heart. (Blackwell, Church, & Green, 2008) turn against the tide of computational thinking towards a respect for the individual human, and (Blackwell & Collins, 2005) admit the possibility that "an end-user programmer may well prefer to accept the results of an imperfect execution" — an appeal to the "workmanship of risk".

However, there are few concrete signs of the development of the durable yet forgiving materials that must underlie true crafts. There are some simple, house-clearing tasks that we should turn our resources

to, before we are prepared to achieve the higher virtues:

- To guarantee to every user that, once given an artefact, they can always choose to use it in its current form, regardless of "improvements", updates, or environmental changes
- For every piece of software to be worked on by means of itself, rather than through the use of specialised tools only available to an elite at a different site, at a different point in its history
- Given a user's preferred way of working and seeing, for any system that they start to work with, it adapt to whatever extent it can to meet those preferences

The latter three of our virtues are something that can be worked on in our lifetimes — and if we succeed in constructing better, more forgiving materials, our descendants might possibly use them to construct something which is beautiful.

8. References

Abelson, H., & Sussman, G. J. (1985). *Structure and Interpretation of Computer Programs*. MIT Press.

Blackwell, A., & Aaron, S. (2015, July). Craft practices of live coding language design. In *Proceedings of the First International Conference on Live Coding* (pp. 41–52). ICSRiM, University of Leeds.

Blackwell, A., Aaron, S., & Drury, R. (2014). Exploring creative learning for the internet of things era. In *Proceedings of the Psychology of Programming Interest Group* (pp. 147–158).

Blackwell, A., Church, L., & Green, T. (2008). The abstract is 'an enemy': Alternative perspectives to computational thinking. In *Proceedings of the Psychology of Programming Interest Group* (p. 34-43).

Blackwell, A., & Collins, N. (2005). The programming language as a musical instrument. In *Proceedings of the Psychology of Programming Interest Group* (p. 120-130).

DeMillo, R., Lipton, R., & Perlis, A. (1979). Social Processes and Proofs of Theorems and Programs. In (Vol. 22, p. 271-280).

Gabriel, R. P. (2012). The structure of a programming language revolution. In *Proceedings of the ACM Onward 2012* (pp. 195–214).

Jared Spool. (2011). *Do users change their settings?* Retrieved from `https://www.uie.com/brainsparks/2011/09/14/do-users-change-their-settings/`

Lakatos, I. (1976). *Proofs and refutations: The logic of mathematical discovery*. Cambridge University Press.

Lindell, R. (2014, March). Crafting interaction: The epistemology of modern programming. *Personal Ubiquitous Computing*, *18*(3), 613–624.

Morris, W. (1880). *The Beauty of Life*. A lecture before the Birmingham Society of Arts and School of Design (19 February 1880).

Pye, D. (1968). *The Nature and Art of Workmanship*. Cambridge University Press.

Steve Hoffer. (2016). *Windows 10 Update Interrupts Weather Report.* Retrieved from `http://www.huffingtonpost.com/entry/microsoft-update-weather-report_us_572241f2e4b0f309baf0082a`

Victor, B. (2014). *Seeing spaces.* Retrieved from `http://worrydream.com/SeeingSpaces/`

Wing, J. M. (2008). *Computational thinking and thinking about computing.* Retrieved from `https://www.cs.cmu.edu/afs/cs/usr/wing/www/talks/ct-and-tc-long.pdf`