Lost in the stars

CASD software sits at the centre of an ever-expanding universe of features promising greater user-friendliness. However, according to Herbert Koelman, lack of understanding about users' needs means that some cause them to get lost in space rather than find their way to the optimal design

Reports on Computer-Aided Ship Design (CASD) software in this journal tend to lead to a waterfall of the latest features and gadgets of a particular software package, I have also been guilty of this in the past, and I will be in the future. However, in this article I take a bit of a different stance, by focussing on user-friendliness of software in practice. The first issue to address is software features which are assumedly added to enhance user-friendliness, but are without added value in the daily use of the software.

An example of such a feature was the introduction of animated characters named Clippy, Bob and Rover, in the Operating System and application software of a well-known software house. They were introduced to function as an assistant for the novice; however, their performance in that respect was poor, while their appearance was an insult to the professional. After a few years Rover walked away, never to be heard of again. Remarkably, each time such a feature was introduced and discarded, it was lauded as improvement.

This story illustrates that visual appearance offers no added value of its own accord, and neither does flip-flopping with features. Although no CASD software yet exists with cartoon character Flipper or Seal, it is still important to distinguish between appearance and user-friendliness, for the first does not necessarily invoke the latter.

A second pitfall in the quest for user-friendliness is the extension of software with too many specific functions. Although each function might fulfil a particular need, their plurality makes the software as a whole overwhelming, while the distinction between essentials and auxiliaries is not clear. So, the task here is to make functions as generic as possible.

I remember a case, more than 20 years back, where in the same week two client requests came on our (then) new ship hull modelling software. One was on the automatic generation of deck camber, with a constant ratio to the local deck's breadth at side, and the other concerned a feature to generate a shear strake in the hull, at a constant distance from the deck at the side. We could have extended our software with two such generation functions; however, how many more similar, but different in detail, generation function requests would appear going forwards? Without careful consideration, the software would end up with dozens of homomorphic functions. Looking from a distance, the two requests are actually the same, because they both express the desire to let the shape of a curve of the ship hull be dependent from another curve. A dependency editor was implemented which allowed both features to be addressed with the same function – and many, many more shape dependencies with a similar nature.

A third observation is that poor software design cannot be repaired by fancy menus or forms. We once had a software function that was configured with a plain old text file. Granted, a bit of an 80s solution, but not harmful, because it was only intended for internal use in the company. The structure of the configuration data wasn't very coherent either, but presented no problem for the same reason as above. When an external party showed interest in this function, interactive menus were created to enter the configuration data. Fortunately, just in time, we realised that although the visual appearance and the operation had changed, the underlying poor design had not. It was a typical example of mission creep, where software is taken from one environment to the other without reconsidering its design. Our customer was therefore told to wait for a better design to be developed.

SARC founder Herbert Koelman

Root cause analysis

The question to be asked is which mechanisms have led to instances of ill-designed software. We could blame the system developers; however, in general, they are expected to create what the market requests. And the market is the common denominator of the users – ship designers. So, we should look at market focus.

The first issue is what I would call the syndrome of 'electronic availability', that is, the idea that because data are present inside a computer, they can seamlessly be utilised by other software. This idea is maintained by colourful leaflets of CAD software vendors that place the system in the centre, orbited by specialised software systems which communicate flawlessly with the core by means of mysterious acronyms such as STEP or IGES. In general, this is fiction, which users appear to believe without question.

A second phenomenon is that users have become used to the modelling methods or *modi operandi* of existing software. Some methods are so ubiquitous that people come to believe that these are the only methods to use. This mechanism can also be witnessed in the case of youngsters who have grown up with Windows' 'desktop' metaphor and its implementation in File Explorer, which

makes them really believe that digital computers should work in this fashion, and that no alternative exists.

This is a belief that hampers innovation. In effect, users are willing to accept a system as it is, working around its impracticalities. A shining example is from some decades back, when our company prepared stability software for a pre-designed multihull. The outer hulls of the ship were composed of surfaces which were either fully flat, or circular-conical, which struck us a bit odd, for we had expected some kind of foil shape instead for a better hydrodynamic performance. Years later we came to find out that the ship was designed with software which was only fit for monohulls, although additional side hulls could be modelled by means of 'appendages', which were limited to flat or conical surfaces. As such, the shape of a real ship was adapted towards the limitations of the applied software.

Was this a result of long-since abandoned past behaviours? No, because today we also see many hull shapes designed with the popular NURBS-surface method, which is adequate to model regions of the hull but not the hull in its entirety. With commonly used contemporary CASD programs, making intersections between these regions is fairly easy, so that is what designers tend to do, leading to ridges and chines at the intersection of surfaces as a side product. It is astonishing that in 2019 our community is aiming at large-scale reductions of energy consumption, while we accept bad hydrodynamics caused by improper modelling tools.

It is my impression that these examples hint at the root cause, which is a merry-go-round of, on the one hand, users who have learned to utilise what they have, so don't ask for fundamental improvements, and on the other hand, system developers who let themselves be guided by user's demands. Nobody is to blame for this situation; everybody plays his or her expected and accepted role, but the result is suboptimal. Perhaps this vicious circle can be broken if software developers stop listening to their customers?

Pursuit of happiness

To be more precise, the circle may be squared if developers stop taking the customer literally, instead proactively envisioning what the user *really* needs, or will need in the future. Some examples of software developed in this fashion are taken from PIAS:

- Our hull form design method is sculpted to the way a human reasons about the hull, which is with 3D curves on the hull, fixed in an orthogonal plane if required (e.g. waterline, ordinate). 3D surfaces are interactively created between these user-defined curves. Obviously, a computer program based on this method requires many tools and features, but regardless of its implementation and

visual appearance, such a program will be fundamentally user-friendly.

- The way ship designers reason about compartments shows a duality. It can either be viewed from the compartment as such, with its boundaries (or their coordinates) as primary parameters, or from the bulkheads and decks which divide a ship hull into spaces. Our software supports both views, as well as a mixture.
- SOLAS rules for probabilistic damage stability are based on a schematic subdivision model (by so-called 'zones'), which has shown to lead to confusion and inconsistencies, because reality differs from this approximation. Fortunately, the theory of probabilities also allows for a realistic subdivision model, as has been adopted in PIAS, avoiding these inconsistencies. Obviously, to satisfy the occasional classification society that insists on conventionality, a zone-based method is also present.
- Two types of data exchange standards are commonly applied: either canonical, scientifically-based cathedrals of Product Data Technology, such as STEP, or standards that just support the transport of shape, such as DXF, 3D PDF, X3D and JT. The first require a steep and expensive development path, and the second don't contain the constituting components and their functional parameters. Fortunately, there is an alternative where higher-level product elements are exchanged, see [2]. This concept provides a feasible and practical tool for interfacing between heterogeneous software products.

To generalise, user-friendliness can be improved by looking beyond User Interfaces, naval architectural conventions and coincidentally available mathematical methods. It requires a fundamental understanding of the underlying tasks and goals, as well as the preparedness to deviate from convention – but not too much.

Disclaimer

I realise that some of my statements are a bit outspoken. An earlier version of this article was full of relaxations and exceptions; however, in that way it became illegible. So, I saved them to this end: This article draws conclusions, based on general impressions and experiences. The examples are real, however the reference to the different classes of persons – ship designers, software users, software developers – are generalised, with many positive exceptions of persons, programs and companies.

About the author

Herbert Koelman founded SARC in 1980, and is still engaged at SARC as director & principal developer. Since April 2018 he has been part time professor of Maritime Innovative Technologies at MIWB, a bachelor school of maritime operation, engineering and design in the Netherlands. SARC is the supplier of PIAS ship design and LOCOPIAS onboard loading and stability software, www.sarc.nl. *NA*

References

1. H.J. KOELMAN & B.N. VEELO. A technical note on the geometric representation of a ship hull form. Computer Aided Design, 45-11, Nov.2013. www.sciencedirect.com/science/article/pii/S0010448513001218?via%3Dihub.
2. H.J. KOELMAN. Computer Aided Ship Design 2030 – I Can See Clearly Now. Proc. HIPER'17, Zevenwacht, South-Africa, www.sarc.nl/wp-content/uploads/2017/10/Koelman-Hiper-2017.pdf.