Automated Software Engineering 7(4), 377-380

Desert Island Column: A Trip to Carthea

John Mylopoulos
University of Toronto

We travel from the port of Piraeus to the island of Kea by private boat. It takes ninety minutes to reach cape Sounio, and then another ninety minutes in the open seas of the Aegean to reach Kea. We follow the southern edge of the island moving east. The landscape is barren: small bushes, hills and mountains in the background, the odd farm, beautiful beaches. Every bay we pass seems inviting. But we are heading for Carthea. We find it, yet another bay among many. But it's special too. This was the site of a 30,000 inhabitant city sometime 25 centuries ago.

We set anchor in the bay. There are steep mountains all around, making the place unreachable by land. At the centre of the bay there is a hill with the remnants of a temple and a fortress, typical accessories for this kind of landscape anywhere in the world. Left and right from the hill there are valleys with fig trees and bushes. At the rightmost edge of the bay there is a sharp rocky cliff, with signs curved out (ancient?…modern?..who knows). There are stones everywhere, large, small, flat, round... Could it be that the ancient city was built out of these humble stones?

We reach the coast, climb up to the temple and admire the setting. We eat wild figs, and take some with us for the boat. As we walk downhill the sun is beating down on us hard. Then we reach the water, and it's instant bliss. We swim where ancient seafarers launched their boats. We swim some more, back and forth between the beach and the boat. The water looks like patches of blue and green, feels cool and clean. There are no other boats or people in sight. Our bay is deserted. OUR bay!!!

We will spend a day in Carthea. At night we sleep on the deck and watch the Milky Way in all its splendor till our eyes logout. Sleeping in the fanciest hotel never felt as comfortable as this. The air is crisp, there are no bugs, the temperature is perfect, and the universe is performing, just for us.

This is my desert island. The rescue is on-site and the departure time is fixed. But, there is time to kill in the heat of the afternoon and the early hours of the evening. So, I check what reading material I brought along. Not much, actually, ("…wise move John!" I think to myself) except for some required reading from my graduate course on "Requirements Engineering". I choose to re-read the two papers by Doug Ross [Ross77, Ross77a], published in the *IEEE Transactions on Software Engineering*, January 1977 issue. I've read these papers many times over 20 years. Could it be that the landscape of Carthea will bring forward new inspirations? Let's try. My perspective won't be technical. I don't believe that anyone can think technically on a desert island, certainly not me, and most definitely not in Carthea. (Hope this doesn't disqualify me instantly from publishing

in this column.) Instead, I'll try to give a feel of how visionary Ross' ideas were, then and now.

I actually met Doug Ross during a panel discussion on Requirements Engineering at the IFIP World Congress in Paris in 1983. He was an established and well-respected researcher in the Software Engineering community. I was a junior intruder to Software Engineering, presenting work we had done with Sol Greenspan and Alex Borgida on formalizing Structured Analysis (hereafter SA), Ross' brainchild. Ross was polite, thought that our project was worthwhile (but complained off-line that we missed the intended semantics of SA. Oh well…)

The first of his TSE papers offers Structured Analysis (SA) as "a language for communicating ideas". Communication is accomplished through SA diagrams which serve as blueprints to describe the subject matter in an organized fashion. Right off the bat, Ross claims that his language is "universal and unrestricted" in scope, exploiting decomposition to structure and organize the subject matter. Decomposition, he argues, is the fundamental principle of good storytelling. Decomposition works, he continues, when it involves six or fewer non-overlapping pieces. To make his point, he proceeds to present SA diagrams which describe the activity of making SA diagrams. A language for communicating ideas, indeed!

The second paper, co-authored by Ken Schoman, focuses on requirements definition as an application area for SA:

> "...Requirements definition is a careful assessment of the needs that a system is to fulfill...must say **why** a system is needed, based on current and foreseen conditions, which may be internal operations or an external market...must say **what** system features will serve and satisfy this context...must also say **how** the system is to be constructed..."

The paper argues that requirements definition is a critical first step during software development. Moreover, it is the step that is consistently done poorly due to the lack of appropriate notations and techniques. Predictably, the paper offers the Structured Analysis and Design Technique (SADT), founded on SA and developed by Ross and colleagues at SofTech, as a suitable technique for the task. Apart from describing in detail the technique itself and the roles that need to be played during a requirements definition project, the paper sketches some of the experiences SofTech has had in using SADT for a number of industrial projects.

These are remarkable papers for several reasons, starting with the very idea of a visual modelling language. Of course, people played with visual languages since the times our ancestors lived in caves. So did computer scientists like Peter Chen and others working on semantic data models at roughly the same time as Ross' work. And so did folks working on semantic network representations in AI. But none of these alternatives are as ambitious and unrestricted in scope as SA. Moreover, these proposals on semantic network representations and semantic data models were focusing on a suitable *internal* representation of knowledge, to be used by a computer system, rather than a language to be used by people for communication. For a different type of comparison, look at UML

as another visual modelling language gaining a foothold in software engineering practice today, thanks to the efforts of the three amigos and the backing of some rather big players in the software industry. Its principles are founded on research carried out over several decades. Contrast this with the solo efforts of Ross to set down principles for visual modelling and make them practice too, and all that 25 years ago! If there is a "pioneer" rank in our discipline, Ross deserves the honour hands down.

Another fundamental contribution of these papers is their emphasis on structuring, both as a foundation for building SA models, and as a means for understanding them. The structure of the message is more important than the content, Ross argues, and laments the fact that systems analysts never paid attention to Marshal McLuhan (for whom the medium IS the message!). Of course, structured programming, the dominant programming paradigm of the day, offered decomposition as a methodological aid [Wirth71]. Ross takes this idea further than his structured programming colleagues by offering a visual representation for decomposition (the box-within-a-box notation) and making it the centre piece of his SA notation.

These papers also happen to have served as foundations in launching Requirements Engineering as a research area. Ross was correct in his claims that software requirements involve much more than system description, hence the need for a rich notation such as SA, in terms of which one can describe (almost) anything. He was also correct in claiming that the cause of the poor track record of requirements engineering is the lack of suitable notations and methodologies. Predictably, Ross also got it right when he offered a notation that is understandable by non-technical stakeholders. They are the ones who need to communicate their respective perspectives of a problem, and their vision of a solution that defines the functional requirements of the system-to-be. His stance on all three of these issues is taken for granted today in the Requirements Engineering community. To my mind, we should thank Ross more than anyone else for this.

If these papers were to be “updated”, what updates would I recommend? Firstly, I would hope for a visual modelling language that offers a richer set of primitive concepts for communicating ideas. After all, the world consists of more than things and happenings. Human intentions, such as goals, beliefs and desires can only be represented in the crudest sense as either things or happenings. So are social concepts and social structures consisting of agents, social dependencies, responsibilities and the like. Moreover, each type of concept is amenable to different kinds of analysis. It’s interesting that UML does offer a richer set of modelling constructs, but still doesn’t cover the intentional or the social world, as it should. In [Ross77], Ross cites the slogan “Tell me why it works, not that it works!” which he quotes from one of his earlier writings. For such a slogan to become reality in Software Engineering, one needs to represent explicitly goals, somehow. Given such goal representations, one can trace implementation decisions to design elements, design decisions to requirements, and requirements to stakeholder goals and objectives. See, for instance [Dardenne93] for a principled framework for fitting Requirements Engineering with an early goal analysis phase.

My other recommendation for an updated SA would be to support several structuring mechanisms, not just decomposition. After all, decomposition will do little for you if you are dealing with a large number of relatively simple and similar concepts, such as different classes of students, employees, or products. By now the merits of specialization as an orthogonal structuring mechanism to decomposition are well known and generally accepted. And specialization is appropriate precisely for application domains involving large numbers of simple concepts. Multiple levels of instantiation (which means that there are objects, classes, metaclasses, etc.) comes in handy if metamodelling is useful during a communication exercise. Some means for partitioning a large model into overlapping subsets in terms of packages, contexts or workspaces, is also important to day as we build ever-larger and more complex domain models.

In summary, these TSE papers encompass all a journal editor could ever dream of in a technical contribution. They propose a visual (and visionary) modelling language, demonstrate through concrete examples its usefulness in communicating ideas, argue that such models are important in software engineering, thereby launching the field we now call Requirements Engineering, AND report on encouraging experiences involving uses of this language in large scale industrial projects. Wow!!! These papers are classics, on the (modest) Computer Science scale of history. Speaking of modesty, Ross doesn't come through these writings as a modest fellow. But then again, why should he be? He doesn't come through as a careful researcher either, justifying his every decision and validating his every claim. I suppose there are times when a blitzkrieg approach pays off in research, as much as in anything else.

We sail away from Carthea. It's afternoon and the sun has mellowed. One last glimpse of the temple and the bay, then we are back in the open sea heading for Sounio. The wind is southerly and brisk. It's always like this in the sea beyond Sounio, the sailors tell us. There may come a time when travel will simply mean clicking on a URL. That time hasn't come yet, and I'm happy for this.

[Dardenne93] Dardenne, A., Fickas, S., and van Lamsweerde, A., "Goal–Directed Requirements Acquisition," in *Science of Computer Programming, 20,* 1993, pp. 3-50.

[Ross77] Ross, D. T,. "Structured Analysis: A Language for Communicating Ideas," *IEEE Transactions on Software Engineering 3(1),* Special Issue on Requirements Analysis, January 1977, 16-34.

[Ross77a] Ross, D., and Schoman, K., "Structured Analysis for Requirements Definition," *IEEE Transactions on Software Engineering 3(1),* Special Issue on Requirements Analysis, January 1977, 86-95.

[Wirth71] Wirth, N., "Program Development by Stepwise Refinement," *Communications of the ACM* 14(4), 221-227, 1971.