Requirement Surveys

I am aware of two surveys being conducted at present, and would encourage you to participate - those conducting the surveys have kindly agreed to share their results with RQ.

Firstly, Yuri Chernak invites you to participate in his Requirements Reuse 2010 survey at:

www.surveygizmo.com/s/246670/requirements-reuse-2010

The goal of this survey is to gain visibility into the state of the practice with software requirements reuse in the IT industry in three areas:

- Practical experience with requirements reuse
- General project information
- Demographics of the survey participants

There are 22 questions in this survey, it should take you about 15 min to complete.

Secondly, Lihi Raichelson of Haifa University is conducting a survey as part of a research project based on the premise that organizational reality is a tight correlation between Information System (IS) requirements and business process design. In contrast, Requirements Engineering (RE) and Business Process Management (BPM) are separated and almost unrelated research disciplines.

The questionnaire aims is to explore the perception of RE and BPM by those who are involved in such activities. The response time is approximately 10 minutes and the questionnaire is at:

http://questionpro.com/t/CHS0xZDiIGu

Simon Hutton, Editor

RE-writings

Building Myself a Kayak: Some Lessons for Requirements and Software Engineering - Part 2

Olly Gotel

In Part I of this article (RQ53), I explained my determination to build a traditional Inuit skin on frame kayak last summer. With no prior knowledge of either kayak building or woodworking, I became an apprentice to an expert in a small kayak-building workshop on an island in Maine. As I went through the process, I recognised so many lessons for requirements and software engineering that I felt compelled to share my top twenty observations with others. In Part I, I discussed the role of communication, measurement, modelling, visualisation and architecture in kayak building. I also highlighted the issues of resourcing and the pivotal role of a keen eye. In Part II, I discuss eight more observations … and you get to find out the outcome.

Continued…

1. ***Use the right tool for the job***

While it was possible to use a small number of tools when building our kayaks, life was made far easier when the right tool for the particular job was used. The difference in size and angle of the plane that was used could make a task last either ten minutes or take two hours. A metal rasp could equally halve the time of a sandpapering task. A hot knife would seal the cut edges of the fabric at the same time as it was cut, thereby completing two tasks in one. Curved sewing needles would make a directional pull of thread around difficult wooden corners both feasible and easy, as evident in Figure 13. The smart kayak builder not only has a well-equipped boathouse and toolbox, but knows exactly what to use and when, and they can interchange between the use of these tools seamlessly[1].

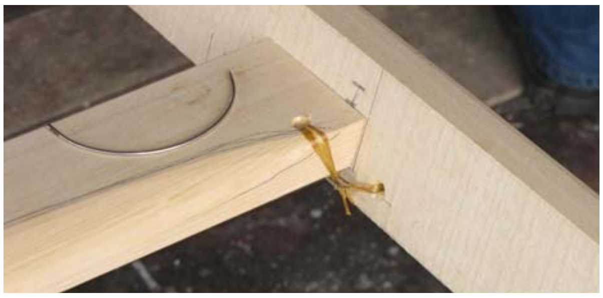

Figure 13. *Lashing made easy using a curved needle*

The large majority of tools in requirements and software engineering are general purpose and require considerable knowledge and tailoring to exploit fully. While there are also tools dedicated to niche techniques and tasks, they can often be difficult to use in concert, or it is prohibitively expensive to do so. More critically, there may be little that is transferrable from one tool to another by way of technique, so the value in taking the time to learn the tool could be

[1] On reading a draft of this article, one of the kayak builders (Eben) remarked on how he has always been fascinated by the power of tools to increase the equivalent skill level of a person: *"With a workshop full of tools we novices were able to produce a complex hunting craft equivalent to what an expert using just a hook knife and bow drill could make."* Being a professional software engineer, he also drew the parallel: *"With the right IDEs (Integrated Development Environments) and function libraries an average developer can create and debug complex systems like database and web apps."*

debatable. Given that the learning curve for tool use can be high, and the longevity of the acquired skill uncertain, we naturally resign ourselves to the use of the few tools that we own and know. Our relationship with tools is also quite different from that of a kayak builder's – do we keep our tools organised, neat, sharp and ready-to-hand? In requirements and software engineering, a collection of smaller and simpler tools may be the way to go, but only if we can get back to the essential underlying techniques that we need.

2. ***Master the underlying techniques to capitalise upon use of the tool***

It is not sufficient simply to have a collection of different sized chisels or planes in your toolbox. You need to learn the technique of chiselling or planing if use of any one of these tools is to be effective. As suggested in Figure 14, this is only achieved through practice on the job. Even the way you swing a simple hammer, either with or without gravity assistance, is going to impact how well and how easily it works in practice, irrespective of its size or your strength. Some activities we repeated over and over during the workshop, and we began to master the techniques with practice, such as sawing, dowelling and lashing. Other techniques would obviously take a few more kayak-building experiences, like mastering the transfer of angles with a sliding bevel and intricate spokeshave manoeuvres.

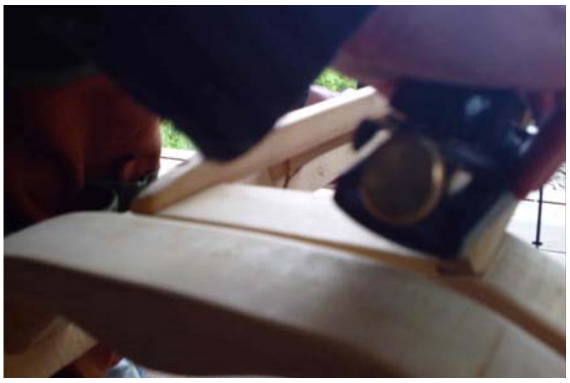

Figure 14. *Mastering that planing technique*

In software engineering, we sometimes do not distinguish the technique from the tool, assuming that we will acquire any necessary technique through use of the tool. This is a notable issue when it comes to requirements management tools. Do we really know what the fundamental techniques are that we need to learn and master to capitalise upon use of our tools? We can name the underlying techniques in the kayak-building world, but in the software world they tend to be somewhat vague and large in their scope: elicit requirements, specify requirements, code, test, etc. Also, where and how do we pick up these skills? This is on the job and under the watchful eye of a master kayak builder when learning to use the tool in the kayak world but, in the software world, we are often expected to make the transfer to a practical setting from the classroom on our own. Many of our tools are so overwhelming and so feature rich that we never quite get to become fully competent in the core underlying techniques that matter, the techniques that will serve us best in the longer term. In requirements and software engineering, we need to highlight the foundational and transferrable techniques of the discipline if we are to focus on them more overtly.

3. ***Embrace diversity and pull together***

Although each of us in the workshop was learning all the skills that we needed to build our own kayaks, some of us were naturally more talented at some tasks than others. I was considered a good lasher and seamstress, so I was able to trade the application of these skills for intricate carving or drilling tasks on my own kayak by others, two things that I certainly did not master. Eben carved the shark bow piece for my kayak that is shown in Figure 15[2]. While much can be done individually, some tasks do work better in pairs, such as having an extra eye to help maintain an angle whilst drilling, while some tasks require a whole team, such as four people to steam wood, then bend and affix the resulting ribs quickly. Given the nature of the workshop, as a joint learning experience, we all needed to stay somewhat in step in our kayak building schedules to make it manageable. So, when one person fell behind, we would all pull together at the end of the day to ensure alignment. We all wanted everyone to succeed in his or her own work.

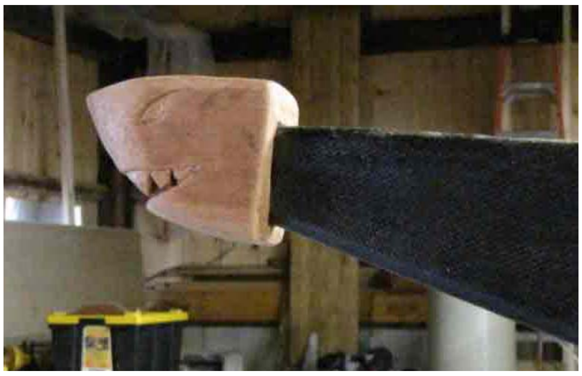

Figure 15. *We may not all be skilled at carving, but we can share skills*

It was once accepted in requirements and software engineering that individuals would specialise in particular areas and work together as an integrated product team to leverage complementary skills. More recently with the agile movement, the idea has become that individuals should be able to undertake any task in the software development process, any one member of the team being replaceable if the knowledge and skills are circulated continuously within the team. Competence and skill in all areas takes time to master though and we cannot always all be acknowledged experts at everything. What if all the team members are at the beginning of their skill learning curves? In

[2] For someone who spends so much time on and in the ocean, I still have a great fear of sharks (I blame it all on reading and watching 'Jaws' when I was way too young!) This bow piece is my talisman.

requirements and software engineering, we need to take care in resourcing our projects to blend the competencies and personalities of individuals, while nurturing their individual career aspirations, rather than simply expecting people to be autonomous cogs in jobs.

4. ***Provide for a healthy and safe working environment***

In a woodworking environment, there can be a lot of dust and noise when many people are sawing, drilling and hammering all day long. Protecting eyes and ears is critical, as is ensuring adequate ventilation and light. Equally important is ensuring that people are alert and aware around any machinery, and so they need to be well nourished and work at a sustainable pace. We were fortunate to have a team of fantastic cooks on hand who made sure that we had lots of refreshment breaks, often outside on the dock. We also listened to music to help some of us focus, while ongoing chatter about kayaks and paddling kept others fully immersed in the process. Where there was idle time, as some people had completed tasks before others, such time was used to sweep and clean the boathouse, collect and tidy tools, and to ensure the retention of a workable environment. A tidy moment is captured in Figure 16. We were not just workers doing a job; we were enthusiastic apprentices keen to learn and share, passionate about the product we were building and we wanted to enjoy the process in every way possible.

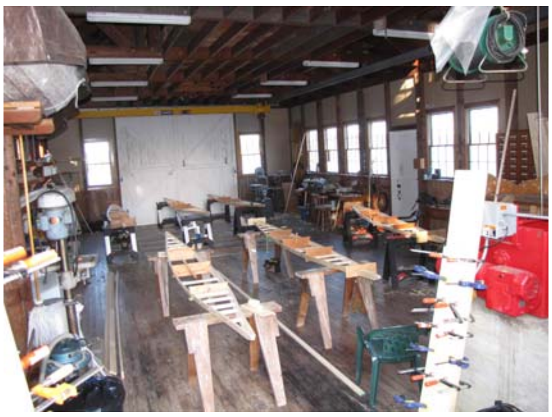

Figure 16. *Maintaining a tidy work environment*

Health and safety issues are not so easy to identify when it comes to the work of requirements and software engineers. Nevertheless, endless days of computer usage, poor posture and flood lighting can all take their toll. The idea of refreshments often only extends to pizza and caffeine in our world. It is consequently common for individuals and teams to reach burnout as deadlines arise. While some enlightened workplaces focus on ergonomics and comfort for their employees, imagine what a team of requirements and software engineers could accomplish with an in-house chef cooking up delights and forcing imposed downtime in natural surroundings! Also, imagine the commitment if the team had some future stake in the ongoing use and success of what they are each creating. In requirements and software engineering, we need to prepare and tend to our working environments much more carefully if we are to demand excellence from our engineers.

5. ***Recognise that the best is the enemy of the good***

For the first five days of the workshop, I wanted everything to be perfect. I even smoothed out rough edges on wood that would remain hidden inside my kayak and created a thumb nook to help me get into my kayak (which, to date, I have never used). With some non-critical measures, rough approximations using fingers were wholly sufficient as a heuristic, but I would insist on lining everything up and using my tape measure to be absolutely precise. As time passed, I realised that I was sweating the small stuff too much and needed to compromise on perfection if I was to finish on time. It was the 7:00am to 2:00am work day on day six[3], when steaming, bending and fixing the ribs for my kayak (as a team, at top speed and still falling behind), which led to this epiphany (Figure 17).

Figure 17. *There is no time to fuss about when working the steam box*

When we undertake requirements and software engineering activities, we rarely have any benchmarks to judge how efficiently and effectively we are working. Most of our estimates are arbitrary and individual productively, for one, can be a controversial thing to measure. Also, many projects do not start with a clear understanding of what it takes to be considered 'done', so we keep on with many activities long past the point of return on investment. It is essential to have a prioritised set of requirements and accompanying acceptance tests to ensure that we focus on what matters most and, in the interests of a wider schedule, know when it is time to move on. In requirements and software engineering, a perfectionist can be as hazardous as a negligent, so we need to learn how to recognise what is 'good enough'.

[3] Yes, I really did work for about seventeen hours that day, once all the enforced meal breaks had been factored in!

6. ***Watch that urge for closure***

Come the final days of the woodworking, I just wanted my kayak frame to be finished. All the care and attention that I had put into the previous days could easily have been wiped out with a careless slip of the drill. The trigger for the change in my behaviour was the realisation that I was not going to be completed in the two weeks I had expected, so I would not be taking my kayak home to New York City with me. None of us would be finished and we would all have to return to Maine later in the summer to skin our kayaks. The reasons are explained later and mostly come down to the environment (the Maine weather). At the time, I thought that I could speed things up and skin my kayak before the waterproofing was dry, just so the process would all be over in the one visit. Although waiting was painful, as the inactivity of Figure 18 shows, the alternative would have been detrimental. On my return visit to Maine, I was concerned that something else would go wrong and require a third visit, so I sewed up the skin at top speed on my arrival. I then desperately wanted to get the skin painted before the caulk sealant had dried, so that the paint would have a chance of drying before my scheduled departure. That would have been detrimental too, so I waited. The problem was that I could see my kayak emerging in front of my eyes. All I wanted to do was to take it home, get it on the water and paddle it before the summer was over. I did not want to chance nature again. As luck would have it, we had a glorious sunny painting and drying day, and we all took our finished kayaks home after the second trip north. Patience is so much harder to muster when the end is in sight, but yet so far from attainment. It is consequently the easiest time to introduce that irreversible disaster. This is the one time when the advice of the expert is the most invaluable, even though it can be very hard to digest at the time.

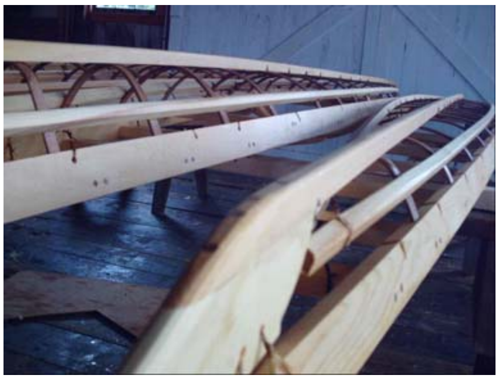

Figure 18. *Waiting for the waterproofing to dry (Note that the horizontal wood is the keel, chine and gunwale as you look down the upside-down kayak)*

In software engineering, the time when activity seems to peak on a project is when the reality of a deadline draws near. It is the time when a bug is fixed hastily, thereby introducing far more new bugs, and when other corners are cut to ship a product. This is exacerbated with an abstract intangible product like software – who can see that omission or shortcut? Irrespective of how or why this happens, it is the time in the process when more quality checks and balances than usual need to be imposed. In requirements and software engineering, if any part of a software project is to be scrutinised more than the initial requirements, it is the activities of those final hours.

7. ***Conduct early testing and a final fairing***

At many points during the workshop, we could actually sit in our wooden frames to check the dimensions and fit of our kayaks, as illustrated in Figure 19. One person even dropped his frame on the concrete floor of the boathouse to check that it would not fall apart. It didn't. We all undertook our own forms of idiosyncratic testing along the way, to convince ourselves that the kayaks we were building would actually fit, because they all looked so very small. However, we never once tested that they would float, so we took the primary requirement for any kayak completely on trust. We also never tested out their likely behaviour ahead of time to see whether they would perform as expected on the water. We relied on sketches and trusted the opinions of our experts. Prior to the skinning, we all conducted what is called a 'final fairing'. This is a careful look over the shape of the kayak, and a smoothing of all the rough wooden edges, to assure the eventual profile and silhouette once skinned. Any lashings needed to be seated into bevelled wood to permit for a non-lumpy skin to avoid drag in the water. We focused entirely on testing the form and fit; we relied entirely on expert judgement for all the rest.

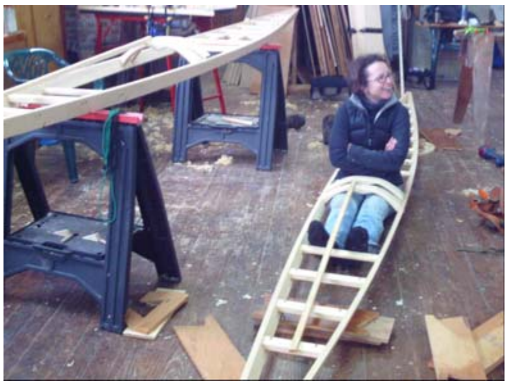

Figure 19. *Testing it fits before taking the next step*

In software engineering, we unit test fragments of code with some regularity, though we often tend to leave the overall functional testing of the assembly towards the end of the process. Early user testing is possible when we work with customers and end-users during the requirements activities, but we sometimes forget to continue to do this as the software evolves and as time becomes pressing. It is obviously easier to do all forms of testing when the product you are building is for yourself and when it is in your hands, literally. However, the idea of a final fairing applied to requirements and software engineering is an appealing one, ensuring a final check over all aspects prior to points of commitment. We attempt to do this with inspections, but we do this neither routinely nor rigorously. If only we could judge how a set of requirements would behave by standing back and simply looking at them from different angles. In requirements and software engineering, we are beginning to bring testing to the forefront, but we still have plenty of scope to make this practice integral to the forward engineering process every step of the way.

8. ***Determine the risks and have a back-up plan***
There were certain time windows that we needed to adhere to if we were to finish building our kayaks in two weeks. After the first few days, we were on schedule, and then things slipped. It became evident that our most valuable resource, the bandsaw, was compromised (as discussed in Part I). More critically, the damp Maine weather meant that everything took twice as long to dry than expected, quite the problem when the first level of integrity of the kayak frame is provided through glued wooden dowels. The Maine weather that we had counted upon is shown in Figure 20, but we only got that kind of weather maybe three days out of the two weeks. As a result, we experienced delays and only finished the wooden frames on the original visit, despite working twelve to seventeen hours every day (these long working days were only possible for the reasons discussed in 16). We returned to Maine to skin the frames six weeks later over a long weekend. On that trip, we had glorious sun.

Figure 20. *One of those rare good weather days in Maine*

Not finishing a project on time, with the functionality as initially anticipated, is accepted as the norm in software engineering. Risk identification and mitigation strategies can help us build in better contingency to project plans but, to some degree, we cannot anticipate everything and control the natural environment. If we did not accept some risk, we would never get to start on engineering anything, so we learn to proceed with acceptable risk. Time criticality is perhaps what really sets software engineering apart from kayak building. Deliver late and you may blow a contract or miss a market, and consequently do not get paid or lose a business. With my kayak building, there would always be another day for my indulgence. In requirements and software engineering, irrespective of the best project planning and risk assessment, perhaps we too can blame the weather sometimes?

In Closing…

My rolling kayak was worth waiting for and well worth building right. Yes, it floats, and the evidence of this is given in Figure 21! It also fits me like a pair of trousers and I can roll it like a whirling dervish when in the water. Not only did I plan, design and build something that required hard physical labour (hard for me that is), as well as acquire many new skills, I gained a wider appreciation for why it is necessary to apprentice with an expert when learning any engineering-related craft. The value of such an experience is further magnified when you can observe and learn from other apprentices who are going through the same process with you.

The technical world of requirements and software engineering is far removed from that of traditional Inuit kayak building. I would not even begin to claim that they are comparable in their complexity, as every now and again we see software systems that are engineered that stretch the bounds of the imagination. However, the kayak building process is engineering in the most classical sense, and it is a tradition that has survived and been passed on through the generations in Greenland. In going through the learning process myself, I decided that the requirements for a personal kayak are quite easy to determine once you get to the heart of what you actually want and why. Getting to pare down to those essentials however, in a world of endless possibilities and dreams, is the really hard part to do. I got what I really wanted ONLY by letting go of what I did not really need. That was a critical insight for me.

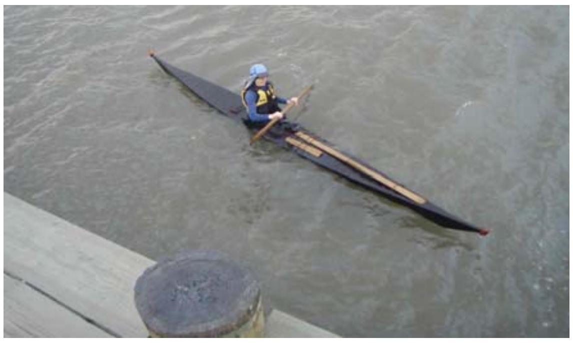

Figure 21. *My kayak and me, together at last!*

However, I probably could not have undertaken this exercise had I not been a pretty advanced kayaker. The trade-offs that I made and the precision that I required only made sense because I felt them, personally. We are rarely experts in the domains that we create our requirements and software for, but apprenticing to learn about the product domain is as essential to our success as apprenticing to master the engineering process. But even as domain experts, we certainly could not have built a single kayak to suit all five of us at the workshop. The problem of achieving an agreed set of requirements amongst a set of stakeholders was blatantly clear in this context. This issue remains one of the largest and underappreciated challenges of software requirements engineering; the only way in which this is sometimes possible is to build a software system that has compromised the requirements away.

Most importantly, we could not have evolved our requirements, and then satisfied them in our kayak designs and builds, without our kayak master builder's hard earned, tried-and-tested experience being passed on to us every step of the way. I therefore urge all requirements and software engineers to experience being an engineering apprentice in this same spirit. Build something that you have always dreamed of, but lacked the know-how and skills to do. You will gain a wider appreciation for why we, in requirements and software engineering, have such a long way left to go in understanding the nuances of our own engineering discipline and in passing on our craft to others. You never know, your reflections may help us all as we proceed on this important journey, and you might even have some fun! ☺

Acknowledgements

My special thanks go to Turner Wilson and Cheri Perry of Kayak Ways LLC (*http://www.kayakways.net/*) for making this kayak-building workshop such a memorable engineering experience for me! It was a real joy to see such a skilled and passionate pair of kayakers selflessly enabling others to pursue their own kayak building dreams. As a pair, Turner is the kayak master builder of this article, while Cheri is one of the best Greenland-style kayak rollers in the world.

The workshop would not have been such great fun if it had not been for my four fellow kayak builders: Dave, Eben, Bob and Geoff. I learned so much from each of them and their contributions are all now part of my aptly named 'Shark Attack' kayak. Thanks to Dave and Eben for taking the photos that were used in this article and, given they are both IT professionals, for suggesting connections between kayak building and software development that I had overlooked. I also thank Eben for making his wonderful waterfront boathouse in Vinalhaven our home for the workshop. I particularly thank Ali, Elise and Patsy for making sure that the workshop felt more like a luxurious gourmet eating vacation than the extremely hard work that it actually was!!!

Finally, my mum read and commented on early versions of this article, under some duress! While she is a keen kayaker, she is a reluctant user of technology and she has never quite understood what I do for a living. Not only does she now have a vague idea about requirements and software engineering, she also ensured that I did not veer off into a technical reverie and stuck to using Plain English! I also thank Ian Alexander and Simon Hutton of the RESG (Requirements Engineering Specialist Group of the British Computer Society) for their insightful comments and their enthusiasm to see my musings shared in RQ (Requirements Quarterly).

Glossary

Kayak and maritime terms used in this article:

Bow, Chine, Cockpit, Coaming, Deck Beam, Deck Lines, Final Fairing, Float Bags, Gunwale, Hull, Inuit, Isserfik, Keel, Lashing, Masik, Port, Qajaq, Rib, Rocker, Roll, Skin, Skin-on-frame, Starboard, Stem Piece, Stern, Track.

Please see:

http://www.qajaqusa.org/Movies/audio_glossary.html (for qajaq terms).

Please see:

http://andrews.com/kysc/terms.html (for maritime terms).

Woodworking terms used in this article:

Bandsaw, Chamfer, Chisel, Clamp, Dowel, Drill, Hammer, Hot Knife, Japanese Pull Saw, Plane, Rasp, Sandpaper, Saw, Saw Horse, Sliding Bevel, Spokeshave, Story Stick, Windlass.

Please see:

http://www.woodworkinghistory.com/glossary_access.htm.

For Further Reading

- Cunningham, Christopher. *Building the Greenland Kayak: A Manual for Its Construction and Use.* Camden, Maine: Ragged Mountain Press, 2003.
- Dyson, George B. The Aleutian Kayak. *Scientific American Magazine*, Volume 282, Issue 4, April 2000.
- Golden, Harvey. *Kayaks of Greenland: The History and Development of the Greenlandic Hunting Kayak, 1600–2000.* Portland, Oregon: White House Grocery Press, 2006.
- Morris, Robert. *Building Skin-on-Frame Boats.* Hartley and Marks, 2001.
- Peterson, H. C. *Instruction in Kayak Building.* Nuuk: The Greenland National Museum and Archives and Atuakkiorfik/Greenland Publishers, 2001.
- Qajaq USA. *The American Chapter of the Greenland Kayak Association.* (Peruse their extensive website of resources, *http://www.qajaqusa.org/.*)
- Zimmerly, David W. An Illustrated Glossary of Kayak Terminology. *Canadian Museums Association Gazette*, Volume 9, Issue 2, 1976. (Can be downloaded from his extensive Annotated Bibliography of Arctic Kayaks, *http://www.arctickayaks.com/biblio.htm.*)

Contact Details:

Olly Gotel, PhD (olly@gotel.net), New York City, December 2009.