The baseline SQUARE process is available on the CERT website. These are SQUARE's nine steps:

1. Agree on definitions.
2. Identify assets and security goals.
3. Develop artefacts to support security requirements definition.
4. Assess risks.
5. Select elicitation technique(s).
6. Elicit security requirements.
7. Categorize requirements.
8. Prioritize requirements.
9. Inspect requirements.

Because many operational systems problems are traceable to requirements problems, the SQUARE project team hopes to enable the development of systems that are more secure and survivable by successfully using requirements engineering methods. In addition, the SQUARE project team hopes that this focus on security requirements will result in more predictable development activities and processes, as well as systems whose costs and schedules are more predictable.

For more information on SQUARE, visit http://www.cert.org/sse/square.html or contact Nancy Mead at nrm [at] sei.cmu.edu.

Simon Hutton, Editor

RE-writings

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

Olly Gotel

During the summer of 2009, I decided to build myself a skin on frame kayak[1]. Being a complete kayaking fanatic, I wanted to learn more about the history of the kayak and to understand how design decisions impact a kayak's behaviour on the water. What better way to do this than to go through the process of building a traditional Inuit kayak for myself? Having absolutely no practical skills, it was clear that I was going to have to learn to do this in a workshop setting under the watchful eye of a master kayak builder.

With my master kayak builder located, along with a very appealing workshop location (a waterfront boathouse on the island of Vinalhaven in Maine), I received a list of required tools. Given that I was clueless about woodworking tools, this was obviously going to be an interesting shopping experience for me. What is a Japanese pull saw exactly? What is a spokeshave used for? Does it matter if I get a plane that is not low angle? I was probably the first person to request photos of all tools. After having borrowed everything possible from friends, and having raided almost every hardware store in New York City, I finally packed up my bag of tools and headed off to coastal Maine.

In the weeks leading up to the workshop, I had been thinking long and hard about the kind of kayak I wanted to build, only to realise that I did not really know. I knew that I valued fit, comfort and aesthetics, so much so that I took these things for granted. I also simply assumed that my kayak would float and be water tight, so I gave that no thought at all. However, I wanted my kayak to be fast and to track well (so it had to be long and narrow), to be an excellent roller (so it had to be low volume and sit low in the water), and to be able to handle rough water, waves and wind (so it had to be short enough to turn easily and have some rocker on its waterline for manoeuvring). These three things were, of course, all in conflict. I wanted to build myself the multi-purpose kayak that excelled at everything … and, as I jumped on an airplane to go north for two weeks, I thought I could actually do it!

Day one of the workshop was my wake-up call. Along with four fellow kayak builders, we discussed what we each wanted to build with our master kayak builder. Did I really want to build a compromise kayak, a kayak that would never be good at any one thing? No, but I could not decide what to prioritise. I was asked to think about what I actually wanted to do with my kayak above all else and most of the time. It was at this point that I realised that I was about to embark on a requirements engineering journey. I therefore decided that I should see if I could learn a few lessons for the day job as I sawed and drilled my way through the next couple of weeks.

To cut a long story short, I decided that I would build myself a rolling kayak. Rather than take you step-by-step through my internal requirements negotiation, and kayak design and building process, I have compiled a list of my top twenty observations from my efforts, and I draw some simple lessons for requirements and software engineering from them.

[1] 'Qajaq' (pronounced 'kayak') is the traditional Inuit term used to refer to a Greenlandic skin on frame kayak. For ease of reading, I shall use 'kayak' throughout this article.

1. *Rip a Story Stick to serve as a baseline*

On day one of the workshop, we each ripped ourselves a 'Story Stick'. A Story Stick is a long piece of off-cut wood that acts as a reference point throughout the entire kayak building process (as shown in Figure 1). It is used to record all the key measurements of an individual in one physical place, such as their wingspan (i.e., the tip of one middle finger to the other with outstretched arms) and their cubit (i.e., the tip of the elbow to the tip of the middle finger), measures that translate to important dimensions of the kayak so that it is custom built to fit. You start with the Story Stick measures that you design to and then, so long as you comply with these key dimensions, you can make ongoing design decisions every step of the way. In fact, it is somewhat possible to shape the characteristics and behaviour of the kayak as far as 90% into the woodworking portion of the build, the critical point of no return being the placement of the chines[2].

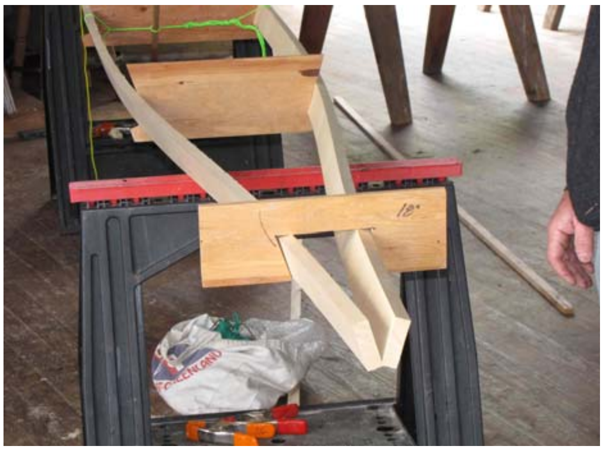

Figure 1. *The Story Stick for a kayak is always close to hand (it is the stick on the floor close to the hand!)*

We have been struggling with the process of requirements discovery ever since the dawn of software engineering. Our approaches range from attempting to specify all the requirements upfront (i.e., Waterfall) through to incrementally adding requirements stories as they arise (i.e., Agile). In the kayak-building world, we take a middle way. We work towards an overarching goal (e.g., a functional rolling kayak) and identify key measurements to then work within as we decompose and realise this goal. We neither make all the decisions upfront nor add features arbitrarily as and when they arise in our minds. We proceed along a truly incremental path that is governed by a few key constraints. In requirements and software engineering, we still need to find our middle way.

[2] The reader is directed to a number of online glossaries at the end of Part II of this article. These define all the terms that are used in this article, more precisely. The chines are two long pieces of wood that define the edge of the kayak, from the side of the kayak (the gunwale) to the bottom (the keel), thereby providing its hull shape.

2. *Agree Terminology*

We used traditional Inuit and maritime terms to refer to all aspects of the kayak, so we knew exactly which part of a kayak we were referring to at any one time. When there are multiple deck beams and ribs in a wooden frame, it is important to have a scheme to refer to each beam and rib uniquely and precisely. For instance, the first deck beam in front of the cockpit is called the Masik and the first deck beam behind the cockpit is called the Isserfik. The traditional Inuit kayak terms are shown in Figure 2. The same rigour was applied to the tools and techniques we used. For example, if we were instructed to chamfer the wood, we knew we were to put a 45-degree angle on to the edge to take out a square corner. We were hence able to understand what parts of a kayak were being discussed and what techniques were required in any group instruction or communication. We were therefore also able to work on each other's kayaks interchangeably and as needed.

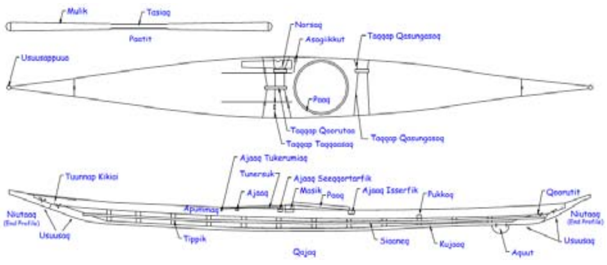

Figure 2. *The traditional language of the Inuit kayak*

(Please visit the Image Map of Shawn Baker and Craig Bumgarner to hear the terms spoken and to view their English translations: http://www.qajaqusa.org/Movies/audio_glossary_map.html)

In requirements and software engineering circles, we still debate the meaning of fundamental terms. Does requirements management refer simply to the management of requirements once specified or to all the activities of requirements engineering (i.e., elicitation, analysis, negotiation, specification, management, etc.)? When we ask someone to validate a requirement, do they verify instead? Are we able to use agreed terms to refer to the internal contents of a requirements specification or to the component parts of a software system? Use of agreed terminology facilitates the conduct of any complex task that involves more than one person. In requirements and software engineering, we need to go back to basics and agree on our terms.

3. *Make marks consistently*

The manner in which a pencil is held affects the angle and width at which a line is drawn. This can have quite dramatic consequences if a sawing task is soon to follow. I learned that there is a 'proper' way to sharpen a pencil to get a crisp line. I also learned that there is a 'right' way to hold a pencil against wood to draw this line in a repeatable way. There is a need to be consistent in how marks are made in woodworking for reliable results. The marks remaining from newly cut wood are shown in Figure 3, along with the conventions for deck beam and rib positioning on the gunwales. Where multiple lines or thick lines are visible on wood, and where these cannot be erased completely, standard conventions are then used for indicating where exactly to cut.

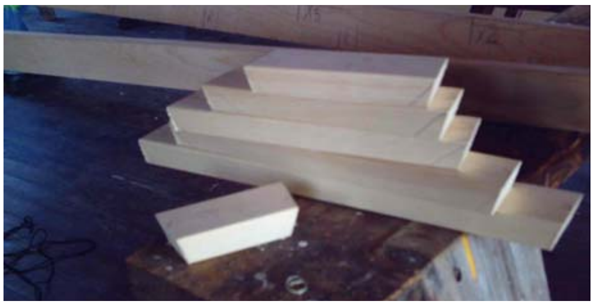

***Figure 3**. Clear markings for fixing and cutting*

How do we indicate to others where we want task attention to be focused when we engineer requirements and software? It is somewhat tricky to mark-up the artifact of interest itself, in the hope that the required action will be noticed and undertaken as anticipated. Instead, we maintain associated project plans and create tickets for work, and we attempt to describe what needs to be done and where within these. The addition of ever more fragmented written artifacts, encodings of our intentions, leads to growing coordination and communication issues across teams and over time. This then sets up a challenge for subsequent traceability. In requirements and software engineering, we need a better way to juxtapose process information with our product for communication and recording purposes.

4. *Proceed from an architectural backbone*

With a kayak, all the woodwork required to build the frame is completed prior to the skinning[3]. The first woodworking task was to join the gunwales, the two long pieces of wood that form the port and starboard sides of the kayak, to mark the midpoint of both, then to pull them apart to define the desired shape of the overall kayak. This shape was retained temporarily using a number of windlasses, as shown in Figure 4. There are three fundamental shapes for the kayak: neutral, where the maximum breadth of the kayak is at the midpoint, making the kayak symmetrical; fish-form, where the widest point is in front of the midpoint; and Swede-form, where the widest point is behind the midpoint. This decision determines the distribution of the kayak's volume. Although the eventual completed frame may resemble the bones of a fish, the backbone of the kayak is not the keel at the bottom of the kayak, but it is these very important gunwales. These provide the kayak with strength, and a solid basis to build upon and attach further wood to. Once these are in place, work can proceed.

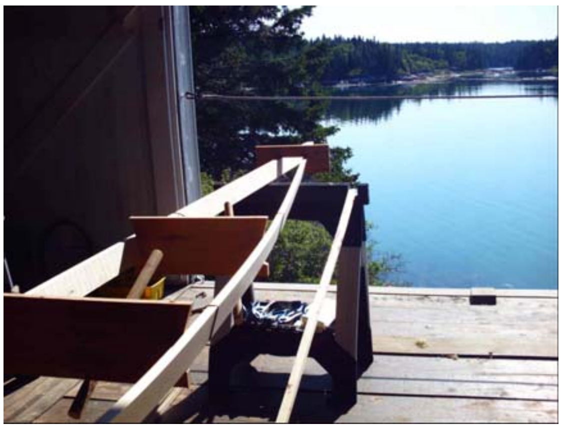

***Figure 4.** Shaping the gunwales using windlasses to hold their position*

In software engineering, requirements could be said to be the backbone of the system, since the end result is likely to be unsatisfactory if these are found wanting. However, we rarely elucidate and articulate the core requirements upon which all the others depend, let alone experiment with options for the shape they could take before proceeding with all those that follow. As per the gunwales, the core structure may be framed by non-functional properties, such as the overall shape and strength required of the system. It is all too easy to build on shifting sand, in our attempts to consider either every possible requirement upfront, or to accommodate any requirement that arises later. In requirements and software engineering, we need to differentiate those requirements upon which others depend. We need to do this far more concertedly, early on in the process, and create an architecture that allows other requirements to emerge later.

5. *Envision prior to any commitment*

Progress appeared to be extremely rapid at the onset of the workshop, as pieces of wood were assembled into something that resembled a kayak in one day. There was then a visible sign of progress with every subsequent day, although less pronounced in the second week. This ability to see a kayak materialising

[3] Note that fabric is mostly used nowadays to skin a kayak, not seal skin! This is usually a heavy-grade nylon.

kept people motivated and positive. The use of clamps made it possible to attach pieces of wood to each other to experiment with positioning and shaping, prior to any permanent change or attachment. This enabled a three-dimensional view of the envisioned kayak at every step of the way, visibility that was indispensible for fixing deck beams and ribs to the gunwales, positioning the keel and chines, and for finalising the profile of the hull, bow and stern before commitment. The ability to experiment with the keel position is shown in Figure 5.

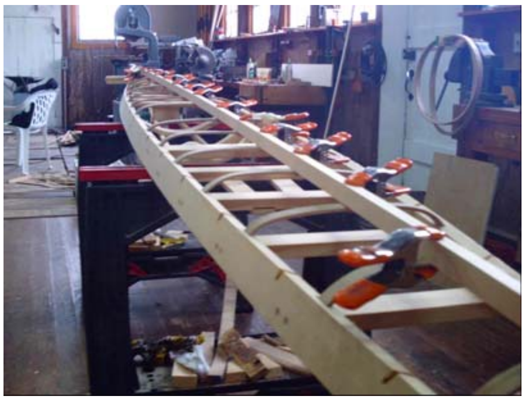

Figure 5. *Working out the positioning of the keel before drilling*

It is useful to be able to see the impact of decisions before they take hold. We attempt to do this in software and requirements engineering with rapid prototyping and user interface design mock-ups, but we rarely see beyond a two-dimensional surface or gain a view of the bigger picture. In particular areas, like change impact analysis, our ability to pre-visualise the impact of change prior to commitment, both locally and systemically, would be invaluable from a cost perspective. However, we perceive many of our actions to be reversible, so we sometimes forget the cost that rework incurs. In requirements and software engineering, we need ways to visualise the intentions and likely outcomes of our abstract intangible work.

6. *Measure twice, cut once… revisited*

The Carpenter's Maxim is well known and quoted in software engineering circles. As suggested by Figure 6, wood once cut stays cut. When it comes to kayak building, not only is this maxim imperative, it actually goes much further than this. We did measure more than once before cutting any wood. More important, we used different strategies and techniques to achieve any one measure. The trickiest task for me was placing the chines on the kayak because, when I took two different approaches to measurement, I kept coming up with two different results for their placement. It was the only way to catch that I was doing something wrong.

Figure 6. *Preparing to make that cut*

It is not an exaggeration to say that requirements and software engineers rarely think about measurement. When they do, they cling to one of a few well-known metrics and use those as a basis for decisions. In requirements and software engineering, we really ought to take a multi-pronged approach to how we obtain our measurements, particularly if we are to gain confidence in our measures and advance as a true engineering discipline.

7. *Stabilise to localise change and care for what remains*

Whether sawing, drilling or planing, we always ensured that the kayak was stable so that we could work on a specific area without causing a negative impact on the rest of the kayak. Throughout the entire building process, it was important to ensure that the wooden frame rested on two saw horses (or more) as we worked on specific areas, as shown in Figure 7. This was to maintain the frame's overall integrity and to prevent sagging as pressure was applied to areas. When sawing, we learned to hold the piece of wood that was to remain, rather than the piece of wood that was to be disposed of. When fixing, we would clamp the join for a number of hours to hold the wood in position until the glue had set.

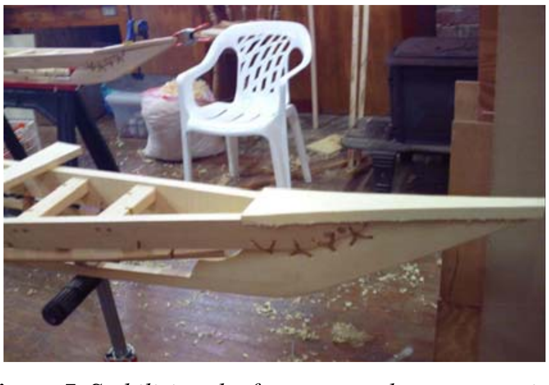

Figure 7. *Stabilising the frame to work on a stem piece*

We do a reasonable job in requirements and software engineering when it comes to change management, particularly when working on versions of documents or code with a team of people. However, given that it can be quite tricky to understand all the hidden or emergent interrelations in an abstract system of this nature, assuring stabilisation of the whole as changes are made and take effect is not a trivial matter. The perceived ability to rollback any changes to a previous state, as mentioned earlier, is a belief that sometimes leads to a lack of initial care and attention with initial modularisation. In requirements and software engineering, we need to focus on this proactively when structuring requirements and the designs that flow from them.

8. *Work within acceptable tolerances*

Small mistakes propagate in kayak building and the impact accumulates. For example, if the ribs are positioned slightly off of the original marks when drilled and dowelled, you may find that you can no longer lash the ribs to the gunwales as securely as desired. To minimise the impact of errors, you need to decide what will be acceptable tolerances early on. Whenever we measured anything (see Figure 8), everything was acceptable to 1/16th of an inch and we learned to work within this margin of error. The consequence of every small deviation becomes even more critical, however, as the kayak nears completion. When it was time to attach the final stem pieces to the bow and the stern of my kayak, I was too terrified to do the drilling, as a slight mishap in the drill angle would have meant redoing a lot of work. When undertaking critical tasks nearing completion of the build, the master kayak builder played a critical handholding role for the under-confident.

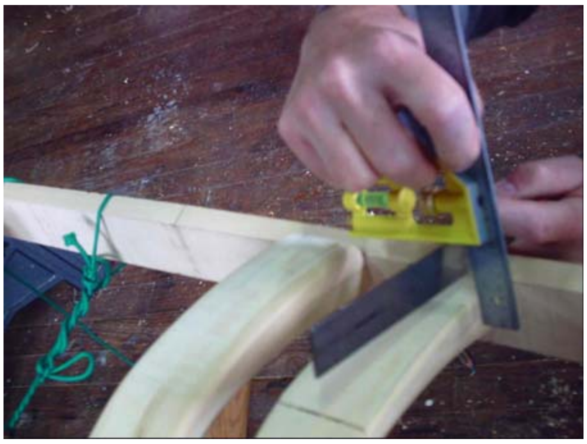

Figure 8. *Checking those tolerances again*

Do we make sufficient allowance for inevitable human error when we engineer software? We have approaches to help us detect, address and tolerate errors. We also have strategies to prevent errors in the first place, if we do a good job at requirements engineering. However, do we pay sufficient attention to all the small innocuous deviations that can eventually add up, or do we focus mostly on the larger and more obvious failure modes? Do we know what our acceptable tolerances are for everything we do and work within them? In requirements and software engineering, we should begin to understand the acceptable tolerances for different tasks and try to work within them.

9. *Avoid single points of failure and unnecessary rework*

The reliability of a kayak relies upon engineering multiple back-ups in the final product. For example, the integrity of the wooden frame is secured in three ways: using wooden dowels and glue for a primary attachment of wood to wood; lashing wood to wood for a secondary attachment; and tensioning the skin around the final frame for a tertiary. The first two levels of this system are shown in Figure 9. Kayak building also requires pre-planning if you are to avoid having to do work over. For instance, you need to plan for the attachment of float bags and put the deck lines in place before closing up the skin, else you have a nasty cutting and additional sewing job to do later.

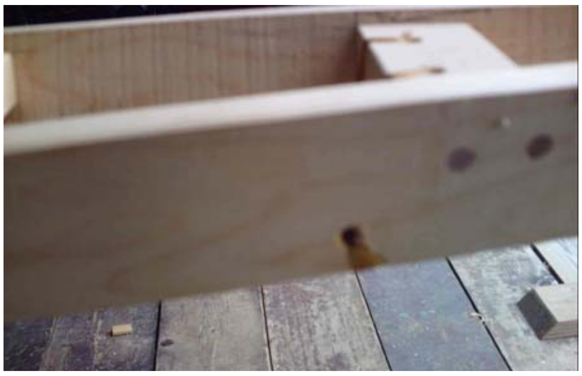

Figure 9. *Dowels and lashings to secure each deck beam*

When engineering software, it is common to have to undertake much rework; refactoring is an acceptable term these days. With a physical structure such as a kayak, rework is incredibly difficult, frustrating and costly in time. As such, it demands anticipation and thinking ahead from the kayak builder, continuously reflecting on what is required in the eventual end product and understanding what needs to be put in place in the present to enable this to happen with ease. It does not mean doing all the requirements upfront, but it does mean understanding the overall concept early on and not flying blind into the build. Pre-emption is a precursor to gaining resiliency in most forms of engineering. In requirements and software engineering, we need to start to share more information on rework efforts and failures across projects, especially if we are to avoid learning the same painful lessons over and over.

10. *Employ models and create informal sketches where useful*

At the start of the workshop, we made use of a reusable wooden rig to simulate sitting in a kayak, in order to get a feel for the required dimensions of the cockpit (e.g., length and width) and to assist with the positioning of critical deck beams. This would ensure that we would be able to slide in and out of our kayaks once completed without removing our kneecaps. When it came to forming the cockpit itself, we selected a prior cockpit coaming template to refine to our own dimensions. These would be used to bend wood to form our own individual cockpits, as shown in Figure 10. The role of a model or template is clear in kayak design and, once used, it is set aside. Throughout the entire kayak building process, impromptu sketches on scrap pieces of wood were also used for the communication of concepts and techniques, and to support decision-making tasks. For example, a two-dimensional matrix was sketched out on wood (resembling a 'Go' board) to help seat the ribs evenly, while the performance characteristics of differing hull shapes were sketched as needed to help with difficult chine positioning decisions.

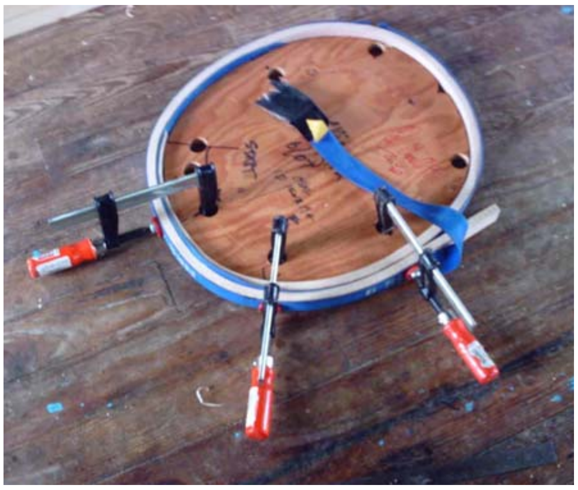

Figure 10. *Bending wood around a cockpit template to form a coaming*

Requirements and software engineering is all about modelling, and we reuse prior patterns in our design and coding activities with some regularity. However, we tend to call anything a model, so the line between what is the model and what is the final product can get quite blurred. As a consequence, the role of the model may mutate beyond its intended role and its planned useful life may be extended unwisely. Our use of informal sketching in requirements engineering is also still quite limited, even though sketching makes for a natural way to express tentative ideas and concepts early on amongst non-specialist stakeholders. In requirements and software engineering, we need to clean up our modelling practices and also look at how we can exploit much simpler means of informal communication.

11. *Understand the role and value of the expert*

The entire kayak building process would have been a complete disaster without the oversight of an expert eye. In our case, these were the four expert eyes of Figure 11. Our master kayak builder was able to see things that the rest of us overlooked or simply missed, moving seamlessly from the most intricate detail to the overall line of the kayak, across all five kayaks in parallel. The expert was also able to troubleshoot and improvise fixes to all our disasters, from poorly drilled holes, dowels popping out of wood, severed lashings, to split ribs. Continuous quality control and endless resourcefulness was achieved thanks to years of accumulated knowledge. The critical role of the expert even came into play long before the workshop began, in selecting the raw materials. Shamefully, I was unable to discern oak from cedar at the start of the workshop, let alone differentiate viable rib stock for bending from that with bad grain. I am proud to say that I am far savvier with wood now! With an expert on hand, we learned that there are very few points of irrevocable disaster. With an expert on hand, we had the confidence to try things out.

Figure 11. *Our experts, Turner Wilson and Cheri Perry*

(Photo from http://www.kayakways.net/, used with permission)

One of the best ways to learn any engineering discipline is to apprentice with a master, and this is indeed what we did with our kayaks. Having the opportunity to learn about failure modes, and to witness the tactics derived to mitigate these first hand, provides for a deeper level of understanding than simply repeating activities and getting everything right the first time. In requirements and software engineering, we rarely have the opportunity to work so closely with such an individual. This is problematic to do, not only because the majority of the work that we do is solitary, abstract and intangible in nature, but also because there is not always a single individual with all the requisite skills and necessary patience. Instead, we tend to be thrown into the deep end. Our own mistakes are not always so immediately visible either. Many software-related errors are left for others to find and fix in the deployed future. Furthermore, when the inexperienced are left to procure core platforms and technologies to support a new software development venture, the shortcoming may never be overcome. In requirements and software engineering, we need to revisit the way in which we teach and train, and we need a way to identify those inspiring master builders.

12. Know the most precious resource and the most ubiquitous

The bandsaw was the most indispensible tool for all the kayak builders in the workshop; its reliability lay on the critical path for each of us as it was used to pre-cut the wood to approximate size. However, the bandsaw malfunctioned early on in the workshop and was subsequently unreliable. We further exhausted all the replacement parts for it. Thus, work on the critical path stopped for everyone as acquiring new parts became a lengthy off-island experience for one person. The most used item in all of our kayaks was dowels and we exhausted our supply twice. Running out of dowels therefore led to another hold up in operations. Clamps were both precious and needed in large quantities, and these were the one item that we had in abundance, since emphasised in the kayak building literature. When building a kayak, you can never have too many dowels, clamps or back-up bandsaw blades (see Figure 12 to appreciate why).

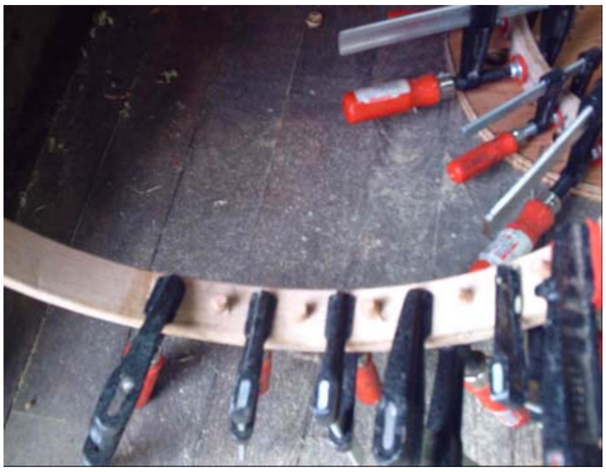
Figure 12. *You can never have too many clamps*

In requirements and software engineering, we tend to think of our key resources as people and money, and indeed they probably are. But, skilled people and money are prerequisites to do most things in life. A reliable inventory of traceable requirements, moreover, can be considered precious in software development and is something that we therefore need to maintain. Requirements drive the critical path of what we design and build. Traceability helps these requirements change as we learn more about them and facilitates confirmation of their eventual satisfaction. As to the most used resource in requirements and software engineering, it is possibly the myriad of channels that we rely on for communication, but generally take for granted. We often do not pay as much attention to planning for softer skills and team working needs, as perhaps we should. In requirements and software engineering, there are many critical resources that we neglect and need to pay far more attention to.

To be continued…

Did I avoid that irrevocable disaster? Did I return from Maine with a handcrafted skin on frame kayak? If I did, did it actually fit and float? Or, did this kayak-building project go the way of a typical software project? In Part II (RQ54), find out about the tooling, the testing and the teaming of kayak building. Find out what happens when you work in an unpredictable environment. Share my pains of balancing perfection with letting go.

Contact Details:

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