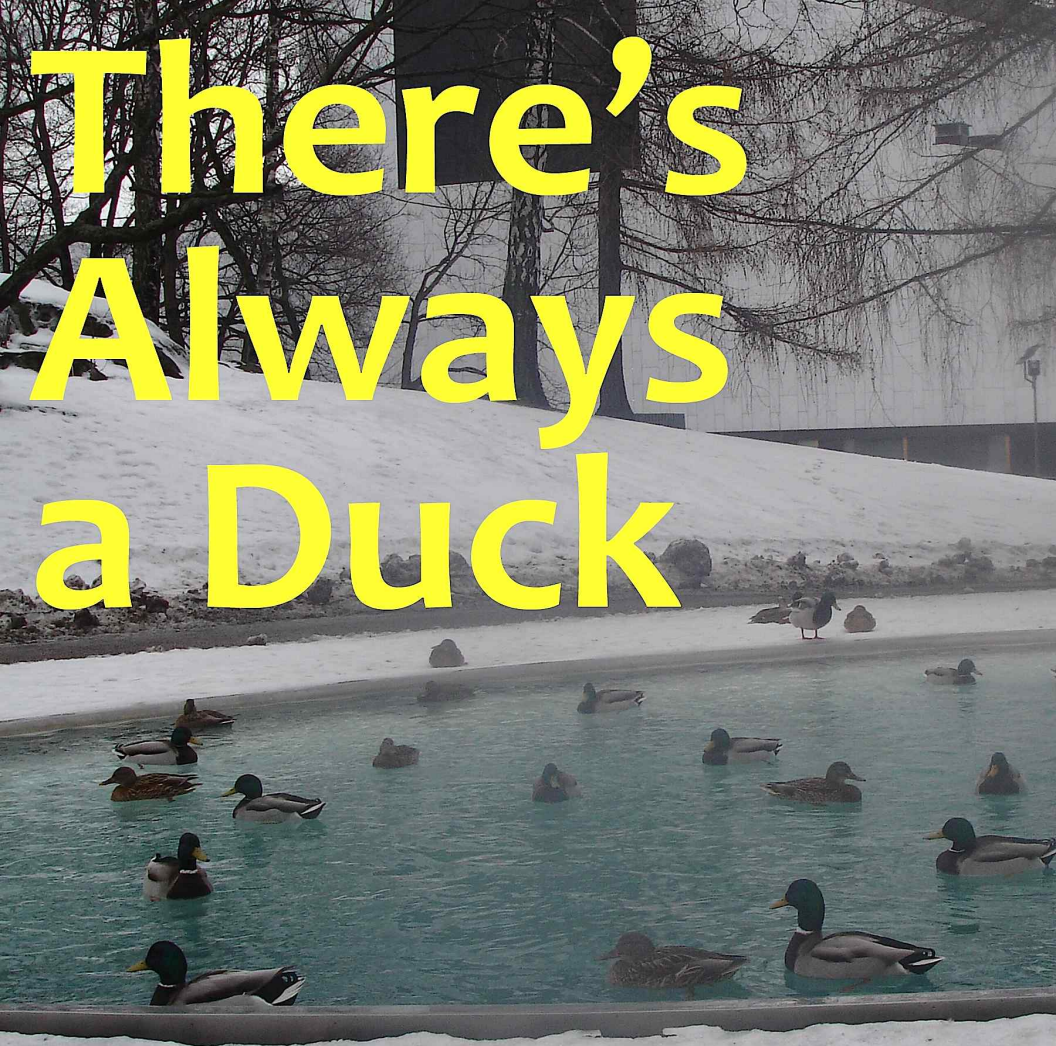
There's Always a Duck



A collection of essays about people, culture, and teams

Elisabeth Hendrickson

There's Always a Duck

A collection of essays about people, culture, and teams

This is a Leanpub book which is for sale at http://leanpub.com.

Leanpub helps you connect with readers and sell your ebook, while you're writing it and after it's done.

Contents

Preface

This book is 15 years in the making.

I started my first website back in 1997 with the intention of writing a new post every day. When I told people that, they thought I was crazy. "You'll never keep up with that pace!" they said.

Because I started my site two years before the term "blog" was coined, the idea of someone with a full time job and a family making time to post a new column online was a foreign concept. Folks were skeptical about my even attempting such a thing.

Of course, it turns out that the skeptics were right; I didn't keep up the pace. I publish sporadically, averaging about a blog post a month, but sometimes going months without publishing anything at all. But they were wrong too: any number of people do post to their blog daily.

That first website was painstakingly handcrafted, written in raw HTML and using Microsoft Paint to create images. Because I had spent years writing technical

documentation in troff, writing in raw HTML wasn't as difficult for me as it otherwise might have been. However, whenever I decided to move things around on my site I had to update all the links by hand. As you can probably imagine, it was an excruciatingly laborious process.

Fortunately blogging has evolved. I've evolved too. Over the years, I imported my content from one publishing mechanism to another.

In 1998, I went to a WYSIWYG editor (NetObjects Fusion) that then generated and uploaded my pages, automatically updating internal links whenever I moved things around. In 2003, I moved on to CityDesk, a content management system from Fog Creek Software. It wasn't WYSIWYG, but what it lacked in usability, it made up for in power.

Somewhere along the way, I decided to move my blog posts from my corporate website at qualitytree.com to my personal site, testobsessed.com. Eventually I moved all my content into WordPress where it currently resides.

Each time I imported my old content into a new technology, I pruned it. I removed the posts that I felt were weak or out of date. And finally in the last big move I decided to cull all the old content no one was reading. "Why bother maintaining content no

one cared about?" I thought.

When I purged the old stuff, I made a backup of it. Or so I thought, right up until I started assembling this book. I have a favorite essay titled "Trucks in the Sandbox" that I wanted to include. When I went looking for it, I couldn't find it on any of my hard drives.

The Internet Archive Wayback Machine[1] saved me. They have snapshots of qualitytree.com going all the way back to 1998. Initially, I just went foraging through the Wayback Machine looking for the small handful of essays that I remembered as my favorites. I ended up copying down all my old content, the good and the bad. It was a nice trip down memory lane.

I also went out to find the content that I had published elsewhere but that I still owned the rights to. I published a number of columns on stickyminds.com and elsewhere over the years.

As I dug into my older writing I realized that I had enough blog posts to fill multiple books. For this volume, I decided on a theme of people and cultures. I selected my very favorite older pieces plus a number of my newer pieces that fit the theme.

Within these pages you will find stories about organi-

[1] http://archive.org

zations, change management, software development, and quality. But ultimately every essay is about people: people and their interactions; teams and their decisions; culture and context.

I should note that I edited all the pieces, especially the older ones: polishing up a rough edge here and clarifying some wording there. But I tried not to polish away the original tone or intent.

So this volume contains essays spanning the full 15 years that I've been writing, assembled from a variety of sources. In addition, I wrote some new bits to stitch the collection together. So even if you are a regular reader of my blog, there's certainly something in here you haven't read before.

I hope you will find the pieces I selected both entertaining and thought provoking.

And since all the essays in this book are favorites of mine, I hope one or two might become favorites of yours.

Elisabeth Hendrickson
Pleasanton, California
January 2012

Something Familiar

You may have wondered where the title of this book came from and why I have a fascination with ducks. It's a family story involving the wisdom of a small child, as you will discover below.

The child is not so small any more. As of this writing, my daughter Anna is now twelve. She is, however, still wise and is developing a wicked, dry sense of humor.

Anna's wisdom about ducks taught me to look for the familiar in strange places. The essays in this section reflect that lesson.

Always a Duck

This story appeared on my blog at testobsessed.com on February 14, 2007.

Some years ago while on a family car trip, my youngest daughter piped up from the back seat: "There's always a duck!"

"Huh?" we replied. "Where did that come from?"

"I don't know," she said. "But it's true. There's always a duck."

She was four at the time, and prone to odd-ball exclamations. I've forgotten most of the bizarre things she said at that age. But that one stuck.

Perhaps that's because I've noticed that it's true. Wherever I go, I can usually find a duck. My young daughter's non sequitur about the ubiquity of water fowl has become a philosophical statement about the way the world works.

It's comforting. Wherever I go, no matter how strange a place it is, I know that I will find something familiar.

More Similar Than Different

I posted a version of this essay on my blog as a post titled "On the Road..." on September 25, 2007. This version is substantially edited.

Until 2001, I had not been off the North American continent.

I had traveled to both Canada and Mexico when I was younger. I'd even been to Guatemala to see the Mayan Ruins. I thought of myself as reasonably adventurous and a seasoned traveller.

And yet on that first trip to Europe as I headed to Stockholm for the EuroSTAR conference, I discovered much to my surprise that I was extremely nervous. My anxiety level rose as the time for departure neared. I was nearly hyperventilating in the International Terminal at SFO.

This wasn't a fear of flying and had nothing to do with the events of 9/11 just a couple months before.

No, I was uncharacteristically terrified that I would not be able to find my way around or make myself understood. Perhaps it's because it was my very first trip abroad all on my own; I had traveling companions on my trips to Canada, Mexico, and Guatemala.

So with time to spare, I decided that I needed a

distraction. I found a warm beverage and a quiet spot and began journaling. As I was writing, the following poem came to me:

> Take comfort in this,
> for this thing I know.
> People are people
> wherever you go.

(As Seussian as those lines sound, I believe they are original. I searched.)

I found it tremendously comforting to remind myself that people are people everywhere. And once I arrived, I found it to be true.

To my great joy, I discovered that even when I couldn't understand the language at all, I could guess at the conversations from the body language and tone.

As I sat in a restaurant for lunch in Stockholm, I could tell that the rapid-pace, whispered, giggly conversation between two middle-aged ladies was probably some salacious story about a romance, a mutual acquaintance, or perhaps a mutual acquaintance having a romance.

Later that evening in the hotel restaurant, I surmised that the serious conversation punctuated by nods and formal gestures between two professionals in business

suits was probably an Important Business Dinner. I imagined that it might result in some kind of understanding or deal or agreement.

And the aggravated sigh of the mother with two slightly mischievous children in a department store? All too familiar.

I recognized an exchange between a mother and adolescent daughter in the elevator in my hotel. The daughter spied the mirror in the elevator and made her mother stand next to her back-to-back, then waved her hand over their heads as though to say "See! I'm almost as tall as you!" I couldn't understand her words, but there was no mistaking her meaning. I struggled to hide my smile at the private moment in a public place. My 12 year old does the exact same thing to me.

In the last ten years, I have been all around the world: Sweden, Finland, Norway, Germany, France, Portugal, Poland, China, India, Russia, the Ukraine, Japan, Australia, New Zealand, England, the list goes on. I feel extremely fortunate to have had the opportunity to see the world and meet amazing people.

In all those places I have had similar experiences. I watch people converse, and even when I cannot understand a single word that they are uttering, I feel a sense of familiarity in their interactions.

When I travel, people I meet often ask: "How do you find the people here in this country different from people in other places?"

My answer is: "There are differences, but the similarities are more striking."

No matter where we live or what languages we speak, we have far more similarities, large and small, significant and insignificant, than we have differences. People love, and fight, and gossip, and struggle, and celebrate everywhere much the same.

Business is increasingly global. If you find yourself working with people who live in different time zones and speak different languages from you, rather than focusing on those differences, I hope you take time to notice the similarities.

People really are people wherever you go.

We're All Human

This column originally appeared on stickyminds.com on August 12, 2002.

I wrote this at a time when I self-identified as a tester and not a programmer, so I refer to programmers as "they." This is incorrect on two counts:

1. *as Alistair Cockburn says, there is no They, only Us; and*
2. *I actually am a programmer, even though at the time I lacked the confidence to call myself one.*

Although this column was inspired by programmer bashing, you could replace the word "programmer" with any other role. Programmers, testers, project managers, product managers, designers, technical writers, system administrators: we're all human, and we're all doing the best we can with what we have.

Over the last several days, I've encountered numerous instances of programmer bashing. I've heard nontechnical managers complaining about the "idiot programmers" in their shop, had test managers claim the programmers were "user hostile," and most recently read a comment on a Web site calling for legal

action against individual programmers who introduce security holes.

In each case, it seemed that the commenter intended the comment to be humorous. At least for me, the humor is wearing thin. An attitude of blame lay just behind the humor: "The programmers make the bugs, blame them for poor quality!"

Every time I hear a comment painting large groups of programmers with the same brush, I think of programmers I've known who don't deserve that treatment.

Sam, one of my former managers, has been around the industry for a while; he learned to program on punch cards. He had a patient and calm demeanor no matter how hairy the project. He took the time to do things right, even when that meant standing up to a manager who wanted him to cut corners. Sam's approach consistently paid off. By the time he delivered his code to test, there were very few bugs. Those that did appear were minor oversights or unpredictable integration issues.

Karen, a former coworker, specialized in internal tools. If you could envision it, she could build it, and build it quickly. She was so speedy, she often had it built within a few hours after you said, "wouldn't it be cool if…?" Generous with her knowledge, she was

a patient if sometimes demanding mentor.

Brian Marick tells the story of a young programmer who prided himself on producing 100 percent reliable, robust code. He had a standing offer: If you found a bug he didn't know about in his code, he'd buy you lunch. He didn't have to buy many lunches.

I knew a similar programmer. Mikey was the youngest member of the team, recruited while he was still in high school. But he was meticulous. I rarely found bugs in Mikey's code.

In fact, the more I think of all the programmers I've ever worked with, I can remember only a bare handful that were incompetent or even remotely deserving of ridicule.

The vast majority of programmers I have met are diligent, capable folk. They truly care about the quality of their work and want the software they produce to be useful. They are more interested in producing a good product than playing CYA games. They work hard to make sure they are implementing the right features and writing solid code.

I have, however, seen a few situations where programmers seemed to lose their focus on the customer.

Betty, an otherwise quality-conscious programmer, surprised me by insisting we should ship software that

was causing intermittent blue screens on Windows machines. She pointed out that the blue screens were rare and couldn't be debugged because they couldn't be reproduced. She asserted that they couldn't possibly be caused directly by her code.

I was shocked. I insisted, in a nastier tone of voice than I intended, that we weren't going to ship anything that caused blue screens. Betty and I ended up in a screaming fight, one of the very few cases in my career where I completely lost it. It took a VP to force both of us to back down long enough to talk about the real problems.

If I could have stayed calm just a little longer when talking with Betty initially, I would have realized how much pressure she was under.

This was a high profile, quick turnaround project. It had been handed to her because of her previous track record for releasing well-received products within tight timeframes. Betty had a reputation to protect and difficult expectations to meet. Her bonus and quite possibly her career with the company were on the line.

Unfortunately for Betty, she was building on a none-too-stable base. Her code worked on top of an existing system that was rife with stability problems. The intermittent blue screen crasher showed up in the

feature she was building, but the underlying fault that caused the problem had been there all along in supporting code that Betty had no control over.

Sick of still having a six-week project on her plate after twelve weeks, Betty was anxious to declare victory. From her perspective, the project had been done some time ago. She figured it was someone else's responsibility to fix the other code on which her product depended.

Of course from my perspective her attitude was irresponsible. I'm sure I called her an "idiot programmer" and worse during that project. But Betty didn't deserve it. She was simply reacting to a difficult situation.

Ultimately, Betty agreed that we shouldn't ship until we could resolve the blue screen issue. She realized that no matter who was at fault, she was responsible for fixing the intermittent crash. And the VP who helped us settle our differences assigned someone who was familiar with the rest of the code base to help fix the issues.

The next time you're tempted to think of your programmers as idiots, incompetents, or quality hostile, remember that no matter what else they may be, they're people first.

Even if it seems like they're hostile or incapable, it

is far more likely that they are having a very human reaction to a particularly bad situation. Perhaps, like Betty, they're feeling trapped in a no-win situation. So before you condemn them, ask what's going on from their point of view.

And before you blame someone else for a mistake, remember the last time you made one. I've made some real whopper mistakes in my time. We all have, whether or not we choose to admit them or even remember them.

It may be that some programmers don't care about users, but it's more likely that bugs are honest mistakes made under difficult circumstances.

I don't expect you to put a bumper sticker proclaiming "Have You Hugged a Programmer Today?" outside your work area. But the next time you're tempted to vent your anger at a programmer, see if you can imagine the factors contributing to his or her behavior.

After all, we're all human, with all the brilliance and fallibility that implies.

A Duck by Any Other Name

I posted this follow up to "Always a Duck" on my blog on March 7, 2008.

While in Portugal on business, a colleague and I decided to talk a walking tour in the mild weather. I'd just told him about my daughter's proclamation the day before. As we were walking, I spotted a duck.

"See, there's always a duck," I smiled and pointed.

"That's not a duck," he replied.

"Yes it is," I said.

"No, it's not," he replied.

I'm from the US; my colleague is from Finland. It occurred to me that maybe ducks in Finland look different. "OK," I said. "If that's not a duck, what does a duck look like?"

"White," he said. "And bigger. That's not a duck, it's a *sorsa*," he gave me the Finnish name.

Certainly, the duck we were looking at was not white. It had the colorful markings of a male mallard. "Right," I said. "There are white ducks and colored ducks. But they're both ducks. A *sorsa* is a kind of duck, right?"

"No," he insisted. "Ducks are white and have bigger beaks than this."

I began to sense that we might be talking about two entirely different things. "Big white water bird," I mused. "with a bigger beak. Do you mean a goose?"

As we sorted out the difference between ducks and geese and *sorsa*, it dawned on me that this English-Finnish discussion offered a good example of the difficulty in sorting out a common language on software projects.

I started remembering all the conversations in which someone used an overloaded term like "server." Or "test." Or "done."

For the record, it turns out that we were, indeed, looking at a *sorsa*[2]. And it was, indeed, what I would call a Mallard duck in English.

But just contemplating how much effort it took to establish this simple basis of understanding about something we could see and hear, I am astonished that business stakeholders and technologists on software projects—something truly intangible—are ever able to achieve any kind of shared understanding about what we're building and how it should function.

[2]http://fi.wikipedia.org/wiki/Sorsat

Translations and Failures in Communication

When I first visited England I heard the phrase, "We are two countries divided by a common language." As the story in the previous section about ducks and sorsa *shows, even when we are speaking the same language and about the same thing, it's hard enough to communicate. Add in the global nature of software development and various domain-specific languages and communicating can become downright impossible.*

The essays in this section consider challenges in communication and the importance of communicating well. As Matthew tells us in the last essay in this section, you can measure an organization's intelligence by the number of connections between people within the organization.

Lost in Translation

I posted this essay to my blog on March 30, 2009. And I had altogether too much fun playing with Google's translation feature while writing it.

A colleague recently described the requirements process in his organization to me. In their process, the business people talk to the business analysts who talk to the systems analysts who give requirements to the programmers.

As he was explaining all this, I couldn't help but reflect on all the possible points of failure.

I've seen conversations around requirements go horribly astray with just two people: a business person specifying what they want and a developer who is supposed to implement it. How much more likely must misunderstandings be when requirements are coming from people who are multiple levels of indirection removed from the originators of the requirements?

The first thing that came to my mind is the Telephone Game. In case you've never played it, it's a game where one person thinks up a story, then whispers it to the next person. That person whispers it down the line to the next person, and so on. Eventually when the story reaches the end of the line, the last person

tells the story as they heard it to the whole group. There's usually much laughter as the group hears how the story evolved through many successive tellings.

Then it occurred to me that this situation is less like the Telephone Game and more like a succession of translations.

The business person tells a story in BizSpeak to the Business Analyst who interprets it and retells it in BA-Speak to the Systems Analyst in terms of "Must," "Should," and "May." The Systems Analyst interprets the story and retells it in SysSpeak to the Programmer through UML Use Cases and/or Sequence Diagrams. The Programmer translates it into TechSpeak, creating State Models, Class Diagrams, Entity-Relationship Diagrams, and ultimately code.

All those languages may have some commonalities, but they have a lot of differences too. That made me wonder what would happen if I played the game with the Google translator[3].

So I picked an innocuous sentence out of one of my blog posts, and translated it from English to German, then to French, then to Italian, then to Spanish, and then back to English. Here's the result.

English: Agile means delivering busi-

[3]http://translate.google.com

ness value frequently and consistently while adapting to changing business needs.

to German: Agile Business Wert bedeutet, liefern häufig und konsequent, während die Anpassung an sich wandelnde Bedürfnisse.

to French: Agile Business valeur signifie, et fournissent souvent de cohérence, tandis que l'adaptation à l'évolution des besoins.

to Italian: Agile Business valore significa, e spesso prevedono la coerenza, mentre l'adeguamento alle mutevoli esigenze.

to Spanish: Agilidad de negocios de valor medio y, a menudo, la coherencia, mientras que la adaptación a las necesidades cambiantes.

back to English: Agility and business value, often, the coherence, while adapting to changing needs.

Hmmm. Something got lost in translation.

Now, I wondered, what would happen if I chose more radically different languages?

> ***English***: Agile means delivering business value frequently and consistently while adapting to changing business needs.
>
> ***to Indonesian***: Tangkas berarti memberikan nilai bisnis yang sering dan konsisten sementara beradaptasi dengan perubahan kebutuhan bisnis.
>
> ***to Hungarian***: Fleet átlagos üzleti és következetes, alkalmazkodva a változó üzleti igényekhez.
>
> ***to Dutch***: Gemiddelde vloot en samenhangende activiteiten, aanpassing aan de veranderende zakelijke behoeften.
>
> ***to Finnish***: Keskimääräiset laivaston ja siihen liittyvää toimintaa, mukauttaa muuttuviin liiketoiminnan tarpeisiin.
>
> ***back to English***: Average fleet and related activities, to adapt to changing business needs.

As I read the final result I wondered wether and "average fleet" was an example of "related activities," or whether the words were supposed to be grouped the other way resulting in averaging "fleet and related activities."

Clearly something wasn't just lost in translation, but became twisted around as well.

Perhaps the lesson here is that the farther apart the languages are, the more the message is garbled. And perhaps that points firmly toward the need for a ubiquitous language, as Eric Evans and the Domain-Driven Design community describe it[4], across the business stakeholders and the implementation team.

Or perhaps it's just further evidence that it's amazing that any software ever works. (Truly, given how many ways things can go wrong when developing software, I'm often astounded that technology works as well as it does.)

However, going back to the original story that prompted this post: I think this simple example in translations across natural languages illustrates the risk of having multiple levels of indirection between the business stakeholders and the technical team.

If we rely on multiple levels of indirection, or worse,

[4]http://martinfowler.com/bliki/UbiquitousLanguage.html

use multiple levels of documentation to convey messages, the real meaning and intent of the requirements will be lost by the time the message gets to the people who have to implement the requirement.

What's lost in translation might not just be a few details, but the entire business value.

What They Forgot to Say

I posted this essay to my blog on April 14, 1997. This version is substantially edited to increase clarity and add in some important details that I left out of the original.

Project teams have to make tradeoffs every day. Sometimes the end result is wildly successful, sometimes not.

Teams make the the best decisions they possibly can about those tradeoffs based on the information they have. When the team is operating from only partial understanding, they're more likely to misjudge the situation. Often the team has only a narrow view on the situation while the leaders in the organization have more of a big picture view. That's why communication between the team and management is so critical.

Let's take the example of a product that was released without international support because the project team was told "Get it to market FAST!"

If you've never had to worry about internationalization, here's how it works. Instead of hard coding messages, you put placeholders in the code and put the actual strings in a file. To translate the product into another language, you (or more likely, a translation

service) translate the file. No source code changes needed and therefore no need for a separate code branch for each supported language.

For this product, management made it clear that speed, not quality, and not forward looking features, was absolutely paramount. The team had been told that the initial release of the product was intended for just a few key domestic customers. As a result, externalizing the strings was the last thing on the developers' minds.

It turned out that management did actually care about localization though. When he found out that the strings were all hard-coded, the Vice President of Development was shocked. He had expected the team to externalized the strings as a matter of course. He was upset that the team made a decision that limited the size of the potential market for the product without talking to him. He was furious that the localized version of the product would have to wait until the next release. And he blasted the programmers for being, "irresponsible and sloppy."

In the VP's defense, externalizing strings is something that takes very little time to do, and it is required in most commercial products since software is a global market.

In the team's defense, they were told explicitly that

their mission was to get the thing done as fast as possible. They did what they were asked to do.

This is a case where a little improvement in communication could have made a vast difference. What could have changed the outcome?

If upper management had been more precise in their communication, the development team would have more information with which to make technical decisions.

If the project team had communicated to management the tradeoffs they were making as a result of the schedule pressure, upper management would have had the opportunity to clarify the requirements.

Unfortunately, this is not an isolated story. This kind of thing happens all the time. The only way to fix it is for both the executive sponsors and the project team to take the time to make sure they are communicating precisely and being heard correctly.

What's in a Name? Everything.

This was originally published as a column on stickyminds.com on May 28, 2001. It turns out that communication problems can start as early as the moment we accept a title in an organization.

Several years ago, I was having one of those difficult conversations with a manager. You know the kind of conversation I mean. A shift-uncomfortably-in-your-chair and hope-the-pain-is-over-quickly conversation, something like a visit to the dentist.

He was concerned about bugs found after the software was released. So was I. He was in charge of all of engineering. I was in charge of something called "quality assurance," a term we'd never really defined in the context of this company and my role. Not defining the term when I was hired was my first mistake.

As the intensity of the conversation escalated and my manager became more visibly agitated by my responses about how we released the software in its current state, I finally realized what was going on.

"You don't really think I OWN quality, do you?" I asked.

"If you don't, who does?" he countered.

Oh dear, I thought. I just stared at him.

I wanted to say, "If I own quality, then let me determine which bugs get fixed. Let me decide under what circumstances we'll slip the schedule to produce a better product. Let me also own the requirements process because we end up doing one heck of a lot of rework around here."

But I knew better. He wanted me to own quality but not the release schedules, requirements process, or engineering decisions: an impossible responsibility.

Unfortunately, he seemed to think I'd signed on as the owner of quality because I'd accepted the job title.

I said as gently as I could, "Just because I have 'quality' in my title doesn't mean I personally own quality. We're all responsible for the level of quality of the software."

He settled down a little, became reflective. He seemed to understand. At least that's the last time he pointed a finger at one particular group for a bad release. The conversation was a turning point for us.

Others aren't so lucky.

I was talking to a test manager the other day facing a similar situation. His organization expects that he'll

test everything important. Only no one tells him what's "important." He's supposed to know.

Nothing specifies which versions of the operating system will be supported, so he has to guess. (Woe to him if he guesses wrong.) If a new version of a Windows OS is released, his organization expects him to know about it and begin testing their software on it immediately. Yet he's seldom given more than two weeks to test "everything."

The other folks in his organization have interpreted his title, Test Manager, to mean that he manages everything associated with test (except the schedule). They don't understand why he can't "test everything." From their perspective, the Test Manager owns anything with the word "test" in it. So they believe testing everything is his job.

This isn't just about titles; it's about responsibility and accountability. When I ask people what could be done to improve the quality of the software their company produces, many people say, "raise the level of accountability." What they really mean is, "I'm doing my part. Hold the other guy's feet to the fire."

Perhaps the real problem is that we don't know what our parts are. We don't have agreement about who's doing what. But we think we do because everyone has standard titles. These titles have become a kind of

shorthand. Everyone knows what the Quality Assurance group does, right? We don't explicitly define the scope of responsibility of the groups because we think we already know. Unfortunately, we also often have very different expectations.

A "Senior Developer" doesn't have "Test" in her title. So should she be responsible for testing to make sure the changes she just made in a shared piece of code will work with the other pieces of code? If you ask the testers, they'll probably say, "Yes! The developer is responsible for making sure that her code plays nicely with others' code." If you ask the developer, she's more likely to say, "I'm not a tester. I made sure my stuff works; it's up to someone else to make sure everything works together."

Some organizations solve the problem of no one taking responsibility by creating positions that bridge the gap. A new job title. Someone who can truly own that piece of the puzzle. When it works, it's an excellent solution. Unfortunately, it tends to becomes a crutch when other members of the organization shift more and more of their work onto the new bridge organization.

Maintenance engineers become responsible for almost all bug fixes so the other programmers can focus exclusively on new code development. Integration

engineers become responsible for unit testing. External beta testers become the sole source of use case testing. Will forming a new group work in your situation? Maybe. Maybe not.

Perhaps the most powerful question anyone can ask is, "If our development process worked really well, what would it look like?" In answering that question, and in combining answers from many different areas of the company, you can get a clearer picture of where reality isn't meeting expectations. Then you can work to address the difference between reality and the idealized process. If their expectations are unreasonable, you can negotiate.

So what's in a name? There may be a bushel of expectations and a barrel of responsibilities, not all of which you might think you signed on for when you took the title.

Make the expectations and the responsibilities explicit. After all, your title might not mean what you think it means.

Becky and Matthew

This is one of the earliest pieces I wrote for my website. I posted it on August 18, 1997. It remains one of my favorite pieces, and I've left it almost entirely as I wrote it originally. Matthew's insight that the number of connections within an organization determines its intelligence has remained with me, and it's one of the things that I look for when visiting clients.

I invited my friends Matthew and Becky to lunch with the idea of interviewing them for my website. We've known each other for years ever since Becky and I worked for Matthew at a large software company. The interview provided a good excuse to get together for lunch, catch up on each other's lives, and reminisce about the old company.

It turned into a very interesting conversation about software development practices and organizational intelligence.

Matthew, Becky and I are in a trendy Italian restaurant near Berkeley. Most of the other diners wear cellular phones, pagers, or both. The other diners are wearing suits. We're wearing jeans. Our waiter, a young woman with very short hair and multiple body piercings, brings us menus.

Knowing that the company where we all met began

an intensive ISO 9000 certification effort after I left, I ask, "Is ISO worth it?"

Becky immediately says, "No."

Matthew replies simultaneously, "Yes."

Matthew looks at Becky, surprised, "No?"

I'm also surprised. I remembered Becky being a strong proponent of ISO 9000. In fact, she was the ISO 9000 coordinator at the software company.

"I admit it, I was wrong." Becky confesses.

Matthew clearly disagrees with Becky's change of heart. "Why do you think it wasn't worth it?" he asks.

"Because it was a lot of effort and it didn't have much effect. We put too much detail into it, and the paperwork was overwhelming."

Matthew asks, "Was that degree of detail necessary?"

Becky replies, "Not really."

"ISO did help. It helped communication," explains Matthew. "It used to be that if you needed something from another department, you couldn't get it unless you knew exactly who to ask, and you already had a working relationship with them. At least with ISO, the other person had an idea why you were calling

and would help you. ISO is common sense written down."

Becky looks thoughtful. "When you have a complicated process, just the act of writing it down makes you look at it, possibly for the first time," she says, thinking about some of the other consequences of the ISO 9000 certification effort.

"There's an idea that the intelligence of an organization is a function of the communication within the organization." Matthew looks at the two of us to see if we understand that he's referring to the benefits of ISO improving communication.

Matthew continues, "So it's in your best interest to be helpful and friendly in an organization. Being nasty is counter-productive. That's something that changed at that company. When you and I started there," Matthew gestures to me, "people were generally helpful. That attitude changed to one of information hiding, hostility, and scapegoating."

Matthew's idea explains how a company that started out smart ended up stupid: communication within the organization deteriorated as politics played a larger and larger role in corporate culture. Once a technology leader, the company had fallen behind and was now trying to catch up.

"Why do you think the software industry has begun

paying more attention to process?" I ask.

Matthew responds, "I think it's because the industry recognizes the fact that process is important. Software is more complicated these days, and it all has to work together."

I toss out another question, this time about something I'd been curious about for years. "We all worked together on that particularly disastrous project. If you could change anything about that project what would it be?"

Matthew's rapid-fire response gives me the sense that he's considered this question often, perhaps sometimes in the wee hours of the morning when he can't sleep.

"There were a lot of problems with that project: the lack of a specification for two and a half years; no test plans; no unit tests; and no integration tests. To make matters worse, we only did huge integrations, not frequent, incremental integrations. The product was a constantly moving target. Especially with 60 people on the team, that doesn't work," he says, not pausing for breath.

"Massive egos," Becky chimes in. "The people working on the project had massive egos."

Lunch is over, and we all head back to our respective

offices. I give Becky a ride, and on the way ask, "What was it that made you leave the company anyway?"

Becky had left the company a year or two after I did. She relays an amazing story of political maneuvering that resulted in multiple reorganizations. She says that sort of thing had become commonplace. She couldn't take the politics anymore so she left.

I drop Becky off and spend the rest of the day thinking about what Matthew and Becky said about ISO, failed projects, and communication, wondering how the culture at the company changed from cooperation to hostility. Politics managed to fell a technology giant.

It's sad, really. The best technology, the best programmers, and the best projects can all be hamstrung by lack of communication, massive egos, political intrigue, and power grabbing.

There's More to Read...

I hope you enjoyed this sample. Thank you for downloading and reading it!

The full book, There's Always a Duck[5] *contains 179 pages of essays like these. If you enjoyed this sample I hope that you'll buy the book.*

Thank you!

Elisabeth Hendrickson
esh@qualitytree.com

[5]http://leanpub.com/alwaysaduck