Creating awareness / exposing problems

Background

At the last three ACCU conferences I have given semi-interactive sessions. Anyone who has attended these sessions will know the form: I present some slides and talk about some ideas for the first half. As I talk people respond and contribute their thoughts and ideas. I note these ideas down on a flip chart and after about 30-40 minutes nobody really cares about my slides because we are having a really good discussion. The audience learns something and more importantly so do I.

What might be less well know is that I take these flip charts home - or at least good digital pictures of them. Then sometime after the conference I write up notes. In 2005 this was just a web-page with 31 bullets points to capture the ideas (Kelly 2005). The 2006 presentation, "Changing your organization" led to a six-page write up on thoughts and ideas (Kelly 2006).

This year I've repeated the process, only this time I decided to expand the write up with some notes on the presentation and publish it here in Overload (Unfortunately this rather grander ambition also accounts for the lateness). I hope those of you who attended the presentation will find this reminder useful and I hope those who were not able to attend will find something of interest here. Although the audience suggested most of the ideas presented here I have added my own notes and thoughts to expand on the ideas.

Creating awareness and exposing problems

In many ways this session was a continuation from the previous two years. The unifying theme is that in order to develop software better we need to learn and our organizations need to change. It is not enough to learn, we need to act on that learning. For learning to be meaningful it must lead to action and create change.

The first step in this process needs to be exposing the problems we face and the opportunities available, and creating awareness about these issues. Hence this session.

Of course we all want to live a better life: write new code, have less bugs, get paid more, have a bigger house but things get in the way, and over coming these obstacles

is difficult. This is hard enough when it is just us but when it is our team or our entire company it is more and more difficult. It is far easier to shut up, stay quiet and accept things the way they are; but if you were such a person you probably wouldn't have joined the ACCU, probably wouldn't read Overload and certainly wouldn't attend the conference!

Overcoming these obstacles and making life better requires effort. All too often the effort required stops us from changing things. So how can we overcome these obstacles?

As it happens some of these blocks are in our own heads. These blocks are relative easy of overcome because we are in complete control of them. All we have to do is recognise the block and choose to change ourselves. Nothing is stopping us except ourselves.

Other obstacles are more difficult. In order to introduce a change into our team or organization then we need other people to agree to the change and help out. There are basically three approaches to this.

Option 1: Tell them what to do

This is the Hollywood model. Arnold Schwarzenegger parachutes into the development department to save the project. Armed only with an Uzi sub-machine gun, several hand-grenades and a hybrid-Hummer H2 he orders

> "You, you and you, code up the user interface - no bugs or the blonde gets it
>
> You there, take 3 programmers, secure the bug tracking system and eliminate all bugs.
>
> I'll deal with the customers... If I'm not back in 30 minutes call in an air-strike"

Well there are a few problems here:

- *Do you know enough to tell them?* Before you can tell someone what to do you need to know what to tell them. For big, complex, problems this isn't a trivial matter.
- *Will they do what you say?* Maybe people you work with recognise your good ideas, understand them perfectly and act on them exactly as you describe. But

since I've never ever encountered such an environment I'm guessing you don't work in such a place either.

- *Will they understand your commands?* Most developers know how ambiguous requirements can be, the same is true with instructions to change. When people don't do what you expect it may they simply don't understand your request or don't see the world the way you do. Try not to jump to the assumption that they are deliberately trying to be difficult and obstructing your efforts.
- *Do you need to check on them?* Unless you are confident that people understand what you expect, and you trust them to do what you want then you will spend a lot of time checking on them. And if you find you are spending a lot of time checking on them then ask yourself: *Am I really trusting them?*
  If you need to check on people then it is going to take a lot of your time so you will be less productive. Plus to this the people you are checking up on are unlikely to enjoy the checks and will feel less trusted. Should you leave, or stop checking, things might just go back the way they were.
- *What about motivation?* People who just do what they are told are usually a lot less motivated than those who are involved with the decision making process. Motivated people are more productive so we need to find a way to keep people's motivation while we change the way they work.

This option is predicated on the assumption that you have authority to tell people what to do; and that the people you are telling will accept your authority. On balance this probably isn't a good option.

Option 2: Scare them into changing

Faced with serious problems like "the company is going bust" or threats like "our new zero tolerance programme means we will shoot the next developer who writes a bug" it is quite possible you can persuade people to change. After all who wouldn't?

However it is also quite possible that your scare tactics will have the opposite effect. Faced with fear and threats people are quite likely to stick their head in the sand, put problems out of their mind and carry on as before. You may even make things worse, individuals may well adopt behaviours which shield them from any threat at the expense of addressing it. For example, threatened with a company failure people may

decide to find a new job; or faced with penalties for writing bugs they may simply stop writing any code.

You might just scare people into doing things differently and it might just work - great! However, what happens next time you need a change? Will the scare tactics work a second time? A third?

To complicate things further success can breed complacency, and it can make things hard to change in future. When people have success doing things one way they are likely to want to repeat that success. Persuading them to try something different risks loosing the past success. It seems counter intuitive to be told that future success depends on dropping existing practices that have brought success.

So option two isn't reliable either.

Option 3: Help others to change

Suppose most people probably feel the way you do: they would like the world to be better too. These people also have ideas on how to improve things. However for them, unlike you, the effort is too much. The solution therefore is to help them, reduce the effort needed and overcome the blocks.

The first thing to do is recognise the blocks, understand were effort is needed. When you recognise blockages then share the understanding it becomes easier to remove the blocks - many hands make for light work.

This option starts to sound a lot better. And this is what the rest of the presentation and this report looks at.

Recognising obstacles is half the problem, but it is not enough for you to see the obstacles. Other people must see the same problems, opportunities and obstacles that you do. If they can't then maybe you see the wrong ones. In order to agree on the issues you need to share the understanding.

Ideas from the group

That was the introduction. Next I asked the audience for ideas. What follows are the best suggestions from the group, with some elaboration from me.

Retrospectives

It was hardly surprising that one of the first ideas suggested from the audience was "hold project retrospectives." Now retrospectives are a great idea, they can work wonderfully. Project retrospectives are not confined to the software development domain; under the name "After action reviews" they are used by the US Army, Marines Corp and the British National Health service. (I'm not saying every NHS operation or military battle is subject to a review but they do hold some.)

Many books on process improvement suggest project retrospectives and there are two excellent books on the subject. Norm Kerth's *Project Retrospectives* (Kerth 2001) is orientated towards reviews at the end of long projects. Diane Larson and Esther Derby published *Agile Retrospectives* (Derby and Larsen 2006) last year which discusses the use of retrospectives in Agile teams and over a shorter period (several weeks as opposed to several years in Kerth's.)

The conference audience came up with a few more ideas, some of which are covered in these books and some not:

- Writing things down can help record them; projects could have a log book that records the events in a project as it unfolds.
- Retrospectives should be held regularly and acted upon within the project.
- Choice of retrospective facilitator is important. The facilitator needs to be able to facilitate the retrospective, i.e. encourage people to speak and explore the issues raised.

  They also need to be apart from the retrospective, if the facilitators have a lot to say themselves then they aren't going to be able as effective in getting others to speak. Secondly, if the facilitator is in the management team of a project their presence and control of the retrospective may inhibit the free discussion.

- Retrospectives can become dominated by one or more "big mouths" - someone who uses the forum as an opportunity to talk and talk and talk. A good facilitator will know how to manage these people and give others a chance to speak, in the extreme you may want to exclude these people from the retrospective.
- Retrospectives need safety: people can't speak openly and discuss problems unless they feel they are in a safe environment.

The group also identified and discussed several "failure modes" of retrospectives. By far the most common failure of retrospectives is that they simply don't happen. Everyone seems to agree that "retrospectives are good" but many people are reluctant to schedule the time for them to happen.

A second failure mode occurs when a retrospective happens but it is too late for it to make any difference. In many companies project teams are broken up after a project finishes, this makes it harder to hold a retrospective and harder to apply the lessons of the retrospective. Similarly, when project teams are staffed with a lot of contractors and consultants who leave at the end of the project it is both difficult to learn and apply the lessons. One solution to these problems is to put retrospectives inside the project rather than at the end - following the Derby and Larson model rather than the Kerth one.

When retrospectives do happen the biggest problem seems to be obtaining buy-in from everyone concerned and acting on the conclusions. Lack of buy-in manifests itself in two ways, firstly getting the right people (i.e. everyone involved with the project) to actually attend and contribute to the session. Secondly getting buy-in from those in authority to act on the conclusions.

However acting on the conclusions is not a management problem alone. Developers, testers, analysts and others need to act too when the retrospective suggests changes. It is not enough to blame management for a lack of change.

Personally I would add one more item to the problems identified by the group. This is: time. Almost without exception whenever I have scheduled a retrospective people have been taken aback by the amount of time I have allowed. For a project of six months I might schedule a whole afternoon, for a six-week project I might want two hours. If a project has not held a retrospective before I would allow more time.

This looks like a long time to people with busy schedules but if you want to understand what happened, understand the causes and devise meaningful solutions it is barely enough time. (And that is not counting write up time after the session is finished.) Looked at in terms of the overall project these time periods are not long: a three developer project lasting six weeks represented over 90 days of effort, probably more when you add in project management and software testing. Is two hours really too much time to spend on learning the lessons of such a project?

Pub conversations / Safe and trusting environment

One idea that came up in the discussion was the "pub" or "water cooler conversation." These are the conversations that occur out-of-band from the office work. The fact that these conversations occur isn't surprising however they are symptomatic of an environment where people cannot talk about these things in the office or to their managers.

In most organizations the people who do the work know what is wrong, they know what needs fixing and they know who are the effective people and who is free-loading. However all too often this knowledge is not available to managers. People talk about these things but they talk outside the office, in the pub, the coffee shop or at the water cooler in the corner of the office.

This information is not available to managers for a variety of reasons. Firstly managers need to make time to join these conversations and listen. Sometimes this requires creating a forum (like a retrospective) where the discussion can happen. The manager could simply join the guys in the pub but it's better if such discussions can occur in a more sober environment.

A second block to making these conversations more useful is trust. People will not discuss some matters unless they trust the person they are talking with. Without trust people will not feel safe enough to hold open conversations. Managers need to create an environment were people feel safe and can trust one another. Without trust and safety people will doubt others' motives and guard what they say.

This is not to say that every pub conversation needs bringing inside the organization and acting on. Many such conversations are riddled with personal opinions, biases and large quantities of alcohol. There may also be legal limits in both what people say and what they choose to hear. For example, if a company is likely to be taken over in the next few days people may not be able to speak freely about the future. Neither are managers at liberty to discuss individuals who may have resigned or be under disciplinary action.

It was suggested that one way of promoting safety is to keep people informed. Again there may be legal limits here, particularly if a company is publicly listed. Organizations do need to communicate with themselves, without communication different groups within organizations may become detached. This is bad enough

when it is between business units, e.g. sales and development, it is worse when it occurs between the company's leaders and the workers.

Publish problem and solution

Identifying and solving a problem is not necessarily the end of the story. In an organization that is trying to learn and improve widely - especially if the organization is large - it is reasonable to try and share your new knowledge. Even if your organization is small and ad hoc there is still benefit from writing up what you have achieved:

- You might learn something new as a result of writing up the solution
- You have a record for yourself
- You could share your solution with others outside the organization, say in the pages of Overload.

Give someone else your idea

Sometimes when you see a problem or an opportunity you are not in a position to act on it yourself. Or perhaps you can act on it but you need others to help you. In these circumstances don't be scared to give your idea away. Mention it to others, spread the idea and make sure those who can do something about it know.

However you cannot be possessive. When you hear someone else talking about your idea don't rush to say "I told you that" or "That was my idea." Smile to yourself but let others own your idea too. After all it is more important that the idea is acted upon than it is for the credit to be apportioned.

Over time people will notice you are the source of good ideas, and they will remember it was you who first pointed out what is now obvious. You have to play the long game.

One more point from my personal observation. If you have suggested or warned of something and it comes to pass then avoid the temptation to say 'I told you so' or 'That is what I have been saying.' Saying this once in a while is fine, but it can be tiring when someone constantly tells you they foresaw events. So before you say 'I told you this two months ago' ask yourself if you really need to point out how right you were.

Metrics

Metrics can be a useful way to expose problems. However there are a number of problems with metrics that the group was quick to identify. Finding a good metric is difficult: ideally they need to be easy to capture (and/or calculate) and they need to be easy to understand. If it isn't easy to get a metric they will fall into disuse, and if they are difficult to understand only a few people will be able to use them.

Do people work to a metric? Numerous studies show that people will change their performance when a metric is introduced. As a result the metric may improve but some other aspect of the system may change.

The British tabloid press have labelled this condition "targetitis" when it occurs in relation to hospital targets. For example, a hospital may be set a target of discharging patients within a certain time period, say two days. In order to meet the target the hospital may discharge all patients after two days and immediately re-admit them in. This would meet the target number but not the target intention. More troublesome would be a patient discharged prematurely who then develops complications and needs to be re-admitted. In such a case the well intentioned target becomes dangerous.

Another variation on this is known as "gaming the system." This occurs when an individual stands to gain from some outcome. The individual knows the rules of the game and attempts to use the rules to achieve the outcome even at the expense of the overall outcome. For example, say a developer offered a bonus for delivering on schedule. With this incentive they may ignore requests for changes, refuse to acknowledge bugs or cut functionality.

The problem is not so much to do with metrics but targets. Using a metric to monitor and understand a system is one thing but targeting a numeric value for a metric can change the behaviour of the system. This is known as Goodhart's law (Wikipedia 2007) after the British economist who first identified this effect.

So if you can find a good metric do not make it into a target! *Measurements are not targets.*

One suggestion from the group was to use good metrics to help understand and visualise the system. Burn-down charts are good example of this. These show how the development system is performing without creating targets.

Another suggestion was to use the metrics to promote competition between teams. This might create indirect targets but if the competition is kept good humoured and reasonable it could promote learning between the teams as they compete to do better.

Show responsibility

Sometimes individuals can play a very direct role in uncovering problems simply by taking responsibility and accepting their own mistakes. It is natural that when one makes a mistake that creates problems for others we don't want to talk about it. We might even try to avoid it or hide it. This can make life more difficult for others and sets a bad example. If we are serious about improving our organizations, exposing problems and creating awareness we need to set an example and own up to our mistakes.

The flip side of this is to be fair to people who admit to mistakes. We shouldn't single them out for criticism or behave detrimentally to them. If someone has admitted a mistake then they show be respected and rewarded. Again there is a need for a safe and trusting environment.

Showing responsibility also means you seek to understand other people and their thinking. Before you rush to brand someone's actions 'a mistake' and expect them to admit it consider if it was a mistake by their norms. What we see as a mistake, or wrong, might simply be a different way of working. This is particularly true when co-workers come from a different background, perhaps different technology, type or organisation or even country. Things are not always so black and white.

Where is the Promised Land?

When we are aiming to improve our environment and solve problems it helps if we know where we are heading. We, as individuals who read Overload, might know exactly where we are heading: we are heading to a bug free land where code is delivered on schedule, testing is fully automated and we all work just 40 hours a week. However do others see this land? Or do they see your efforts as pointless? Just something else to make their lives hard?

Explaining where we are heading is only the first stage. You also need to explain why we are heading there and get everyone to agree on where you are going. I believe Steve Jobs once said:

> "It doesn't matter how we get to San Francisco as long as we all agree we are going to San Francisco. The problem is when someone secretly wants to go to San Diego."

Once everyone agrees to go the same place it becomes much easier to do the right thing.

Finally

Hopefully by the time you read this I'll have my original presentation posted on my website at http://www.allankelly.net.

I would like to thank my audience for their many good suggestions. I would have liked to spend more time on some of the ideas in order to get down to the detail of how they can be implemented but there is never enough time to do everything.

In the end each of us has to find what works for us, in our own environment. Knowing what other people do can inspire us but it can never give us all the answers we need. Some things we have to discover for ourselves.

References

Derby, E. and D. Larsen. 2006. Agile Retrospectives: Pragmatic Programmers.

Kelly, A. 2005. "Viewing Software Development as Learning." In ACCU 2005. Oxford, UK.

Kelly, A. 2006. "Changing Your Organization." In ACCU 2006. Oxford, UK.

Kerth, N.L. 2001. Project Retrospectives. New York: Dorset House.

Wikipedia. 2007. "Goodhart's law."