My Friend the Customer

Your customer must speak with one voice ...
if that is not the case you will suffer
Chet Hendrickson [2].

Abstract

The Customer is the only non-developer role in eXtreme Programming (XP). The Customer's explicit responsibilities are to drive the project, providing project requirements (user stories) and quality control (acceptance testing). Unfortunately the customer must also shoulder a number of implicit responsibilities including liaison with external project stakeholders, especially project funders, clients, and end users, while maintaining the trust of both the development team and the wider business.

This paper provides a small collection of patterns that describe useful models for the customer, showing how other roles involved in the process can be adapted to also serve as the XP customer.

Introduction

The initial description of Extreme Programming (XP) assumed an in-house project with an in-house customer, so that the goals and attitudes of customers and developers are closely aligned. Many projects are much more remote: often, developers never meet or speak to their actual customers, who may work for a different organisation in a different continent. Even the flagship C3 project failed because they were targeting the wrong customer in the end [2].

The Customer is the only non-developer role in eXtreme Programming. The Customer's explicit responsibilities are to drive the project, providing project requirements (user stories) and quality control (acceptance testing): unfortunately the customer must also shoulder a number of implicit responsibilities including liaison with external project stakeholders, especially project funders, clients, end users, and defending the project against office politics, while at the same time maintaining the trust of both the development team and the wider business environment.

The customer role is critical in making decisions about "what to build" and, in the minimalist philosophy of XP, the following are recommended for the customer role [6]:

- The customer is an integral part of the team and should be on-site with the team
- The customer writes user stories and then discusses each requirement directly with the programmers
- The customer is responsible for all business decisions including prioritising user story development
- The small 2-3 week iterations allow the customer to evolve their requirements based on concrete working software

- The customer regularly tests the software to confirm it works as expected.

XP explicitly assumes that the customer knows the domain well and is able to make decisions and as such does not provide ‘how-to’ advice on gathering, expressing and prioritising requirements.

This paper presents several patterns that all address this general context, that identify several different ways of solving the same problem: how to find customers, especially for out-of-house development projects, whether these projects are short-term contract developments, long term outsourcing, or some intermediate situation.

These patterns do not form a pattern language; rather, they may one day form part of a larger language collection describing outsourced Extreme Programming.

To save your time and our effort, there’s no repetition of the common context in each pattern. Rather, each begins by describing the cast of characters and forces involved in the problem, and then focuses on the description of each pattern on the solutions each provides and how it resolves the forces present in common problem.

Each pattern contains a detailed example of an XP development project, generally constructed from several situations with which the authors are familiar. All these patterns are based on observations of real projects. The examples reflect the key details, although names and details have been changed to protect the innocent.

The Patterns

The patterns in this paper are as shown below. You can guess the thrust of each from its name. These patterns were identified in our observations of real projects, where we found the customer role being played by someone with another role in the overall process, or played by someone taking on characteristics inspired by that other role.

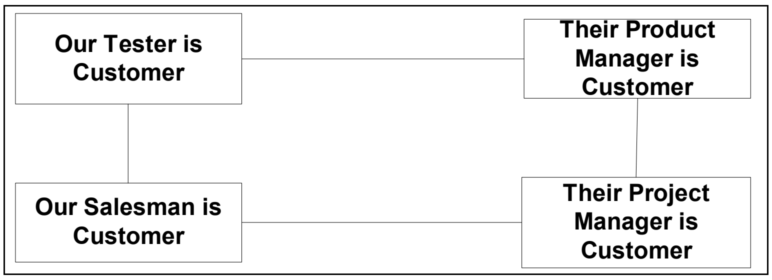
Our Tester is Customer
Their Product Manager is Customer
Our Salesman is Customer
Their Project Manager is Customer

Figure 1: The Patterns

Cast of Characters

The main categories of roles in our process are framed as follows:

- “We”, a software development shop using Extreme Programming or other Agile practices.
- “Them”, or “The Client”, the poor schmucks who not only need some software but are paying us to build it for them.
- “Their Users” the even poorer schmucks who will end up having to use the software that we have written for them.

This is the language we hear listening to people working on projects: us and them, and users out there somewhere. The Customer is the crux of these patterns: the interface between “Us” and “Them”. Here we can typically identify several roles. Figure 2 shows the main roles, with lines showing typical communication between them.

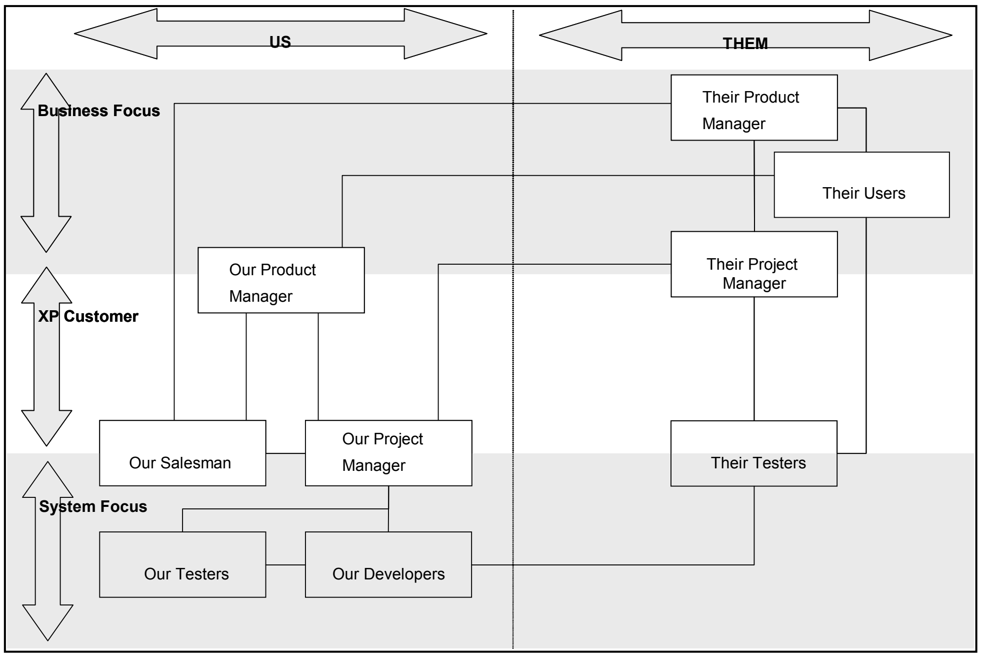

Figure 2: Key Project Roles

The Problem: the Customer Role

The customer’s role in XP is [1]:

- To define the project goals (what constitutes success),
- To write and explain User Stories,
- To specify Functional Tests,
- To set priorities, and
- To attend planning sessions[1].

We can add some extra demands implied by the role:

- To ensure consensus of what will be considered success amongst the main interested parties (those providing finance, those responsible for managing the product). [2]
- To answer day-to-day and moment-by-moment questions by programmers on required functionality.
- To be a champion for the project, to ensure it keeps going regardless [3], and
- To coordinate releases with the requirements of the outside world.

This is a complex role. A candidate with the ability to do all of these things best will be trained as an administrator, as a tester, as a salesman, as an office politician, and as

[1] Strictly this reads “to attend CRC sessions”, but the authors have rarely encountered these.

a software designer. It's unlikely we can identify a perfect candidate for this; even if we could, the workload will be overwhelming [5].

So in the real world we'll have to make do with a less than perfect solution, with someone who is less than the 'ideal customer'. The patterns in this paper explore the consequences of a variety of possible non-ideal candidates.

Forces

Let's examine first the key forces involved – the main issues that constrain our choice of customer:

- *Risk of failure*: how likely the pattern is to cause the project as a whole to fail. Patterns that *resolve* this force make it more likely that the project will be completed to the satisfaction of everyone involved; patterns that expose this force make it more likely the project will suddenly, capriciously fail or be cancelled. Limiting this kind of risk is the motivation for much of XP: anything that interferes with the key XP practices is likely to increase this risk.
- *Development time*: how long a project takes. Patterns that resolve this force mean the project will finish sooner; those that expose this force make it more likely the project will take longer. Note that this is related to (but different to) forces listed below including like turnaround and feedback. All these forces are about time (or inversely, about speed): development time is about the length of a whole project, while turnaround and feedback are about the length of individual iterations.
- *Development cost:* how expensive it will be for a project to complete. Resolving this force makes things cheaper; exposing it makes it more expensive. This force is often correlated with development time. Costs can also be divided into those that they pay directly; those that we pay but that we charge them for, or those that we pay and cannot get back from them.
- *Development staff:* a major driver of development costs, patterns that expose this force require more staff to be assigned to the development, while resolving this force reduced the number of staff required. As with other forms of costs, this can include their staff, our staff that we change them for, or their staff that end up working for us or for the project without charging us.
- *Predictability:* Even more important for many customers than the actual cost is the predictability of costs: how easily we can guess the development time and development cost before we start work. Patterns that resolve this work make it easier to complete fixed-price contracts successfully; unfortunately they often decrease the flexibility, and overall cost and time of the development.
- *Flexibility:* how much they can change the scope or requirements of the project after we have started work on it. Patterns that resolve this force make it easier for them to make changes; patterns that expose this make it more difficult!
- *Turnaround:* how quickly we can turn their requirements into something they can complain about. Patterns that resolve this forces help us to get things to them faster (not necessarily finish the whole project faster – that's development

time); Turnaround is about how fast we can go: in contrast to feedback, which is about how fast they can go.

- *Feedback:* how quickly they can tell us that the software we've built from them is crap. Resolving this force makes it easier for them to shout at us (considered a bad thing in traditional methodologies); exposing this force makes it harder for us to get shouted at (considered a bad thing in agile methodologies). This paper separates turnaround, and feedback because they often related to different aspects of a project. A good team may have high turnaround, producing things quickly, but low feedback, because the customer is unable to evaluate them quickly enough. A project's overall velocity will be a balance of feedback and turnaround.
- *Trust:* may seem like two forces: our trust – how much we trust them, and their trust, how much they trust us. Patterns that address one of these forces may (or may not) also address the other! Success really requires both. Ideally, increasing turnaround and feedback will increase trust; but in toxic projects, the reverse could be the case!
- *Motivation:* how much their contact people care about the project. In a small scale, patterns that resolve this force make it more likely that they will provide us with feedback.
- *Engagement:* how much their organisation is engaged with the project on a large scale. In particular, patterns that resolve this force make it more likely that they will give us more money. The C3 project ended because different factions in the Client organisation disagreed on goals [2]; increasing the *engagement* of such factions would reduce the likelihood of this happening.
- *Sustainability:* whether the workload will burn out part of the project team. Again, this may apply to us (bad), to them (not so bad!) or to both (worst).

The importance of each force will depend significantly on the nature of the project. For example, in a two-week project *trust* and *motivation* and *sustainability* will be less significant than in a five-year one.

Pattern: Our Tester is Customer

Problem: *How can you fill the XP customer role in a project that has a well-understood specification?*

Example:

As HaggisCorp, the famous Glaswegian software development house, we have accepted a development contact with WurstMobil, the equally famous mid-European digital telephony multinational. This project, codenamed Offal, is to produce the SausageStack next-generation telecommunications framework for supporting MXT (multimedia messages, pronounced “mixed”) that will allow full-motion 3D videos to be sent as quickly and easily as today’s SMS. The MXT protocol has a full specification from international standards bodies (the 2 3/4 G specification association) and a reference implementation is available under a public licence. Both we and WurstMobil are keen to use Agile development techniques, however we are based in the fragrant, dank, dark early-19th century back streets of Glasgow, whilst they are based in a fragrant, dank, dark, early-21st century skyscraper in Frankfurt.

Forces:

- We want to minimise *risk of failure*
- We want to provide good *turnaround* and get good *feedback* to our programming team
- We want to keep them *motivated* to provide us with important information about the project
- We want to ensure *sustainability* so our team can take on further development work.

Therefore: *Employ one of our testers to act as customer for the programming team.*

Our testers will have to understand exactly what the product does in order to write effective test specifications. They have a very clear ideal already, from the specifications they have. Moreover, they already need to liaise with them in considerable detail. So we make them the XP customer: testing is already their forte; priorities and specifications they can do.

Forces Resolved:

Making our tester the customer reduces the *risk of failure:* testers are good people at avoiding the fatal errors, keeping everyone on the straight and narrow. It improves *turnaround* and *feedback* — testers are good at running tests and telling developers that their code is poor – after all, it’s their job. Testers are good at tracking the status of a project, improving *predictability.*

Motivation is improved: because they produce better software, understand bug requests and can ensure fixes are made, testers are good at ensuring the client organisation’s technical people think well of the project.

Microsoft has two testers per developer. Given a suitable number of testers, perhaps just one for a small XP project, this pattern can be quite *sustainable*.

However: making our tester the customer can increase *development time* and *development cost* – testers usually prefer correct software to somewhat buggy but cheaper software. We have to provide extra *development staff* as a direct cost to us. But many clients will understand the need to pay for testers, so we can often charge this cost back to them.

There's not much *trust* – who trusts testers?

Most importantly, our testers are unlikely to be good at *engaging* client organisation support for the project. Because they have no status within that organisation, this pattern greatly reduces *flexibility* because they (and thus the whole team) will be working mainly from a static specification. In that situation, this pattern can be quite successful. On the other hand, this means that while *risk of failure* from detail issues will be avoided, it means that *risk of failure* from wider business issues may be increased.

Example Resolved:

Dougal McScroggin, a senior tester with HaggisCorp, was detailed to act as the customer for the Offal project. After some initial training (his personal instincts encouraged him never to accept any software at all to avoid the risk of being blamed for a defective release) this arrangement worked well.

For each iteration, Dougal was able to develop acceptance tests based against the standard and the publicly released code, and eventually conducted interoperability tests with a competitor's project released by TelecommeSaussisonne. Due to the diligence of the 2 3/4 G standards association, it was always clear to Dougal what the specification means and what the result should be.

During the project, Dougal and other staff from HaggisCorp visit Frankfurt twice to demonstrate intermediate results, and once more to deliver the final software, and enjoy a trip to Munchen during Oktober.

Pattern: Our Salesman is Customer

Problem: *How can you fill the XP customer role in a project with a very dynamic and changing specification?*

Example:

We, SpivSoftware, are a team of three developers based in lockup garage in Sarf London, and we have Terrence “Tel-Boy” Spiv as our founder, principal, director, CEO, and salesman. Because Tel-Boy plays many extreme sports – or at least has go-faster stripes, aftermarket mag wheels, and third-party led-enhanced rear spoiler on his purple BWM 318i – Tel-Boy was very interested in the idea of eXtreme Programming. Furthermore, his latest project – a WAP based “magazine” subscription service to manage downloadable ring-tones and phone wallpaper to accompany the latest “UK Pop Stars” reality TV show – seems to fit really well with this paradigm.

However the TV project is commissioned by the “New Tory Brutalist” advertising agency Hirst & Emin, famous primarily for their slogan “take our money and do it!” Hirst & Emin are philosophically opposed to playing any part in this project other than paying for it (oh, and perhaps using the results). Tel-Boy is concerned that it will be difficult to convince anyone from them to play the role as an XP customer for this project.

Forces:

- We want to control and *development costs* (and thus *development staff*).
- We want to keep the client organisation *engaged* (and thus paying us); yet the customer doesn’t have a lot of time or effort to devote to the project.
- We want to keep working at a *sustainable* pace and avoid burning people out.

Therefore: *Employ one of our salesmen as customer for the development team.*

Given the dynamic nature of the requirements and the lack of a formal specification, we need a customer who’s really up to date with the exact current thinking within the client. It can’t be someone within their organisation; they don’t have the time or inclination. So the candidate XP Customer must be our person who interacts most with the real customer: our salesman.

Consequences:

Turnaround and *feedback* from the development team’s perspective are good (so long as the salesman is generally available).

Salespeople are supposed to be professional at building up *trust* with their clients. They are an anecdotal reputation of being less effective at building *trust* with developers (see http://dilbert.com); though in practice the authors have found many exceptions.

*S*alespeople are supposedly better at *engaging* with other organisations to help them spend their money.

However: There will be a higher *risk of failure*: Salespeople typically focus on getting the client to sign, and are less concerned with delivery. We need more *development staff*, because we have to allocate a salesperson's time, post-sale, to this project. That gives a higher *development cost*.

Salespeople are typically *motivated* by commission; without normal commission structures a salesman is unlikely to be well-*motivated* to this project. If they still have to do other sales simultaneously, then you're probably asking for trouble. But where salespeople have built up a good relationship with their clients, and understand development's costs and timeframes, this pattern can work very well.

Example Resolved:

After a couple of heavy Tuesday evenings down at the Old Bank of England on Fleet Street, Tel-Boy hit upon the idea of doing the job of XP customer himself. After all, he washes every day and drives a BWM, while the rest of SpivSoftware staff simply do not own enough Armani to talk to anyone from Hirst & Emin. This way, he could talk to them, tell the developers what to do, and answer the developers' questions.

And indeed, this seemed to work well. Hirst & Emin could just about bear to meet Del-Boy occasionally at exclusive London bars to talk about their "vision things". The design went well, especially the QuickTime VR images of Pop stars pickled in formaldehyde. Upon delivery, Hirst & Emin promptly paid an inflated fee in a briefcase of five-pound notes – according to Emin, "to get that retro-sixties-Get-Carter feel that was soooo 1999s Guy Ritchie". Del-Boy assumed that they were happy with the delivered product, or, as Hirst put it: "That's what we wanted all along. Now shove(*) off".

Discussion

In most projects the client organisation will interact most with two of our roles: our Salesman and our Project Manager. With less grotesque clients than Hirst & Emin, much of the day-to-day communication with them will fall to the Project Manager. So with this pattern, it becomes the responsibility of the Project Manager to ensure that the Salesman is always kept 'in the loop'.

Pattern: Our Product Manager is Customer

Problem: *How can you fill the XP customer role in a project with multiple clients, and a rigorous but unwritten specification?*

Example:

We are GameCatz, an American company making electronic music production software. We use and produce open source software, having been founded by hackers who created electronic music for free computer games. Our early hit was for the game "Nethack", where the music was based on electronic interference sounds made by cell phones.

We still make many products available free on the Internet. But in 1998 we were bought by the large distillery Dan Jackals, and we now focus on commercial clients in the growing convergence between computer games companies and the arms industry. We are starting a new project to create a completely new version of our flagship product: CatGutz. We've decided to use XP because we're trying to be seen as more professional, and XP appealed to our institutional culture favouring rapid response – and to our devotion to anything called "extreme".

We've rented an experienced XP "Coach", and our developers have done a small internal pilot project to add electronic marching music generated automatically by web browsing, where the coach also played the customer role. We are now set for the CatGutz project, but are unsure who should be the customer.

Forces:

- We want *feedback* with good *turnaround* because we don't know what the users will accept.
- We want someone *motivated* to really make sure we deliver the right thing.
- We want someone with enough authority to allow us *flexibility*.
- We need to *trust* what they tell us, and trust their discretion about the implementation details.

Therefore: *Our product manager should act as the XP Customer for this project.*

If we don't already have a Product Manager, we should appoint one, and that person should also be the on-site customer.

A Product Manager is distinct from a Project Manager. In a traditional management approach, a *project* manager manages the development effort and has a focus on *how*. But the traditional management approach also features a *product* manager with a focus on *what*: the finished product and its role. This is an especially important role when the product is being developed for many external clients, and even more potential clients: the Product Manager has the responsibility to make sure committed clients are satisfied, and that potential clients are likely to be satisfied too.

The Product Manager has a significant business responsibility, and the appropriate power, to make sure the product succeeds in the market. The Product Manager must therefore stay closely in touch with the Users, and becomes critically aware of their needs. The job is to make sure the product is a success, so the Product Manager is motivated to tell the developers what the Users really need, and provide feedback with fast turnaround because they are in frequent contact with the Users. Product Managers often also control the development budget; so can also allow flexibility, while also taking development time into consideration.

Consequences:

Motivation is improved because the customer has an institutionally supported personal stake in success. F*eedback* and *turnaround* are improved because the customer has all the right contacts to make this happen.

F*lexibility* is assisted because the customer has budgetary and institutional support to *put things right*. D*evelopment time* will not blow out because budget overflows will adversely impact the customer.

However: *trust* may be problematic because a traditional source for Product Managers is the sales department, where they care little about implementation details.

Example Resolved:

GameCatz was still a bit new to the corporate world and so we did not already have a formal Product Manager for CatGutz. We decided to appoint one, and selected Clark Bruce. Bruce has been working for us as a mild-mannered marketing rep, and spent much of his time visiting computer games companies. Bruce originally worked at a games company himself as an engineer, and so he is familiar with many of the issues involved.

Bruce made sure when he accepted the position that it included oversight of the development budget, and management of the sales and marketing efforts for the product. He also asked for a larger salary, and a bonus package, including a new Hummer and the necessary two parking places. We agreed to this, but only after making it clear that responsibility for the success of CatGutz was his; we reminded him that their industry treats failures with eXtreme Prejudice (XP).

With his pride and prestige on the line, Bruce worked hard. He continued to visit the Users in the games industry frequently, but also spent time with the developers every day. When the developers had questions about how realistic the musical screaming should be, his contacts in the games industry steered him right. When it became clear that it was taking too long to develop thunder claps keyed to the speed of player movement, he took responsibility for changing the budget. He made sure that the new Synth-of-Doom features were ready early, so he could give sneak previews to his most important Users, to stop them from buying a competitive product (Microsoft Kill-Tunes) when it was released.

With all this, CatGutz was a success: our developers got what they needed, and Bruce got his Hummer. We also found that this approach also helped them in working with our parent company Dan Jackals, whose Product Management had followed this path for many years.

Pattern: Their Project Manager is Customer

Problem: *How do you manage projects with companies that outsource professionally?*

Example

MegaCo are a large Dutch telecoms company. One of their many subsidiaries, our client MegaRude, is a company that produces a product Bother* (yes – even the names used in this paper have been changed to avoid offence). Bother* lives on a mobile phone, and speaks rude or amusing comments at random intervals to the user based on location, time of day, current world situation. It's proved very popular with a certain section of the population!

Unfortunately, Bother* has become a victim of its own success: MegaRude now has a successful global franchise operation, and is busy signing up network operators in every market except Japan and the Deep South (apparently Bother* is considered inappropriate there, for *cultural reasons*). This means MegaRude has a porting problem: Bother has to be ported to each new phone and network.

Luckily, Bother* consists of a core component, the AnnoyEngine, developed by MegaRude, which can generate comments as PCM data given location and support for Internet requests. The AnnoyEngine is written in C and is completely portable. But to support different hardware platforms, MegaRude have taken the decision to outsource all their software development to subcontractors who are specialists in each platform.

Since they run many outsourced projects MegaRude have developed a structure geared to outsourcing. They have project managers for each project, and a large testing group with well-devised testing procedures and test scripts. After experience with undisciplined projects, they instruct their suppliers firmly to channel all communication and decisions through the project manager assigned to the project.

We are Allsorts Software, a small company contracted to port Bother* to the new BuzzyPhone platform. We don't have enough staff to answer the phones reliably, let alone provide an XP customer; MegaRude's Product Manager, on the other hand, oversees many different global ports of Bother*. So who should be the Customer for this project?

Forces:

- They want *predictability*, with minimum *development cost* and *development time*.
- *Flexibility* is not a major issue for either them or us.
- We need to *trust* what they tell us, and trust their discretion about the implementation details.

Therefore: *Their project manager should act as the XP Customer for this project.*

As described in the **Our Product Manager is Customer** pattern, many development organisations maintain separate Product and Project Managers. The *Project* Manager drives the development effort with a focus on process, while the *Product* Manager concentrates on the finished product.

Because their Project Manager will be primarily responsible for a particular project (or set of small projects) they can make excellent XP Customers. Typically Project Managers can pass larger issues onto Product Managers for resolution. Especially important for XP, Project Managers often have their own testing teams on projects, so the Customer acceptance-testing role can be covered very effectively.

Consequences:

Their Project Manager is wholly responsible for changes that may affect costs, and will filter out expensive changes required by their Product Management, giving them their desired *predictability*, and reducing their *development cost* and *development time.* It's an approach understood well by organisations that outsource a lot of development, so it's highly *sustainable*. The Project Manager champions the project within their organisation, improving *engagement*. Over time, a relationship can develop between their Project Manager and ours, increasing *trust.*

However: *Turnaround* and *feedback* are long; typically a week or more, especially when the Project Manager must to consult their Product Manager for a final ruling. *Flexibility* may also be considerably reduced, as Project Manager generally do not have much scope for make changes without deferring to Product Managers.

Example Resolved

MegaRude assigned one of its highly experienced Project Managers, Drop van Zoutwinkel, to supervise the BuzzyPhone port. Luckily, van Zoutwinkel had managed several similar ports before, and so he was able to answer many of our project team's questions immediately.

MegaRude trusted Zoutwinkel enough that he was allocated a contingency fund within the project time and cost budget. For example, late in the project, our internal testing discovered that the random number generator in the BuzzyPhone always returned the same constant result. Due to his position as project manager at MegaRude, van Zoutwinkel was instantly able to authorise an extra iteration to correct for this problem, especially when, in the middle of the crisis meeting every BuzzyPhone in the room announced simultaneously: *"Who's this then — my friend, the customer?"*

Discussion: Using the Patterns

The patterns in this paper stem from our observations of XP projects. The authors have been studying the customer role, and have found it to be a challenging one. The customer role is critical to the success of an XP project, but we find that it requires an impressive individual, excellent communication links, and a lot of work.

The authors have identified that one of the strategies being used in XP projects is for the customer role is to be associated with an already established business role. This might mean that the customer role is actually performed by the person in this business role, or it might mean that a person playing the customer role takes the business role as inspiration. These patterns can be used in either situation. There are probably also other role patterns for customers, but the four discussed here are the ones observed so far.

Getting on with it

How would you, the reader, actually make a particular pattern happen for you? In practice it is the choice of your development team – in practice, probably, of the coach – and it will be severely constrained by the practical situation. Occasionally a project may try out more than one potential customer role before finding what works best.

Deciding which Pattern may work best

The many forces identified show the complexity of the customer role. The different patterns all manage forces differently, each according to the business issues of most concern to that particular 'customer'.

For example, one of the key forces is simply the *risk of failure*. The patterns each handle this force differently:

- "Our Tester is Customer" means that preventing failure from the detail-level risks associated with technical problems is a key concern.
- With "Our Salesman is Customer", the customer will be more distant from the detail-level risks, but have a greater focus on the immediate appeal of the project to potential clients.
- Similar to these two is "Our Product Manager is Customer": like a salesman, a product manager is not concerned about the details; yet like a tester, a product manager does have to be concerned about quality, though with more emphasis on longer-term issues.
- "Their Project Manager is Customer" shows the other side of the picture: the world view of a project manager is also about quality and the longer term, but with intent more custodial than entrepreneurial.

Risk of failure is an important force, but there are others. The analysis works in the same way. Each of the role patterns brings a particular view and focus for the customer, a different coherent set of emphases that can help a customer do their job.

This approach shows how a customer role pattern can be selected by an organisation. Every project is subject to many forces, and many different risks. But the main risks are often known. For example, it would often possible to identify the greatest *risk of failure*, and choose a role pattern for the customer to best address that risk.

For situations that can be identified early, it may be possible to align the customer role with the pattern role, and so actually make, for example, the product manager also play the customer role. Where the situation is more dynamic, these patterns can also assist someone in the customer role, by suggesting the kind of role that can best address a particular set of circumstances: *who do you need to be today?*

Consider the Alternative

All the patterns described here have proved effective in real projects. Perhaps the least effective solution is a pattern deliberately omitted from this paper: no customer at all.

So if you're planning or in the middle of an XP project, please do opt for one of these patterns and go for it!

Acknowledgements

Our thanks go to our EuroPLoP shepherd, Arno Schmidmeier, and to Kirk Jackson at MCS who put together the writing wiki where this paper originated.

References

1. Extreme Programming Roadmap, 'The Customer' http://c2.com/cgi/wiki?TheCustomer
2. Extreme Programming Roadmap, 'C3 Project Terminated' http://c2.com/cgi/wiki?CthreeProjectTerminated
3. In Search of Excellence, Peters and Waterman, Harper&Row 1982.
4. Peopleware, DeMarco & Lister, Dorset House 1987.
5. Martin, A., Noble, J., & Biddle, R., Proceedings of the Fourth International Conference on eXtreme Programming and Agile Processes in Software Engineering, Succi, G. (Ed.), chapter Being Jane Malkovich: a Look into the World of an XP Customer. Lecture Notes in Computer Science, Springer-Verlag. 2003.
6. Schalliol, G. Challenges for Analysts on a Large XP Project. in Marchesi, M., Succi, G, Wells, D & Williams, L ed. EXtreme Programming Perspectives, Addison-Wesley, 2002, 375 - 386.
7. James O. Coplien, A Development Process Generative Pattern Language, in Pattern Languages of Program Design. edited by James O. Coplien and Douglas C. Schmidt Addison Wesley, 1995.
8. Alastair Cockburn, Agile Software Development, Addison-Wesley, 2001.