Project Management: Software versus Kitchen Fitting

By Dr Ant Kutschera, Principal Consultant / Technical Architect

From a project management point of view[1], software projects have often been compared to those in the construction industry. In many late night conversations with project managers, the author has often discussed how software development could actually be compared to any industry where products are developed within a project environment, for example, anything from building a motorway, to constructing a kit car.

As a result of moving into a house with an old kitchen the author of this paper was assigned a *Kitchen Fitting* project when *senior management* took the decision that a new kitchen was required. The delivery date was set based upon a plan negotiated between the developer (the author) and management. No sooner had implementation commenced, than problems were encountered. These problems had uncanny resemblances to those previously experienced by the author in software projects. This paper draws parallels between project management problems in the kitchen project and those of software projects, shows conclusions between the similarities, and suggests ways of reducing the impact of these problems for the software project manager.

The Successful Project

Many kitchens have, over time, been fitted successfully. In fact, many software projects have been completed successfully too. However, with either type of project, there are many things that can go wrong. If these problems are not caught quickly and managed correctly, there is a chance that the project may fail. Project failure can be measured in terms of the project running over budget (in terms of either time or cost or both), failing to meet quality standards, or not containing the required functionality. In fact, in some cases the project will get to the stage where senior management takes the decision to completely cancel the project, as they feel that there is no chance that it will ever bear fruit or become profitable. Some sources[2] have even estimated that up to 70% of software projects fail to some degree. With such a high chance of failure, project managers should be concerned. This paper concentrates on the types of things that can go wrong in the kitchen and software project, and looks at ways of fixing the problems proactively or reactively, depending upon the type of problem.

Poor Initial Plan

As with any project, an initial plan is created and negotiated between the project stakeholder (the persons requiring the project deliverable), senior/executive management (those who grant the budget), project management (those who will manage the development) and perhaps a member of the implementation team or a consultant who specialises in the area of development (the term development is used throughout this paper to refer to those doing the work – software developers, the implementation team, or indeed kitchen fitters). With the kitchen project, an initial plan was produced, being negotiated and modified with influence from the developer (also the project manager), consultants from the kitchen production company, and senior management (also the stake holder).

As is often the case, all parties involved in the negotiations have different schedules in mind, but it is normally senior management who get the final ruling, as they are the ones providing the budget. Similar tactics can be used by the project manager for both software and kitchen projects in order to *buy* time. A common example of this are asking

for 30% extra budget. When senior management reduce this down to the time or cost that the development team originally had in mind, senior management are happy as they have *saved* 30%, and development are happy as they have been granted what they actually wanted. Old estimation rules of thumb, for example doubling an estimate are equally useful to software and kitchen projects. Reference 3 discusses how for software projects the actual time taken for implementing code is reduced by half what is available due to meetings, telephone calls, coffee breaks, filling in timesheets, filing paperwork and general interruptions during the day. While the kitchen project may not require the developer to fill in a timesheet, or tally up day-to-day expenses, there are certainly overheads such as putting tools away, searching for tools ("damn it, where did I put the 4mm wood drill again!"), vacuuming up sawdust at the end of each day, etc. which take up time that may not have been allocated in the first place. Admittedly, the lack of a kitchen greatly reduces snack times…

For example, only two days were originally allowed for removal of the old kitchen, channelling electrics into the walls for lighting and sockets, tidying up the wall where brickwork was exposed and painting those areas of the walls. Once this was started, the channelling alone took three days! The main reason for this was the hardness of the walls, simply requiring a greater amount of chiselling. Although this could be seen as a foreseeable problem, it was not expected. A quick examination of the amount of work that needed to be undertaken could have helped foresee this problem, much like a prototype could help in unknown technologies in software projects. However the inexperience of the developer meant this was not the case.

Inexperience in Technologies

While there are not many 'unknowns' to an experienced kitchen fitter, there are often many unknowns related to the technologies used in software development, even to the experienced developer. Software technologies evolve and become trendy and then outdated much faster than technologies in most other industries. This would normally make it difficult to draw comparisons of this facet between the kitchen and software, however due to the author's inexperience in kitchen fitting, parallels can be drawn.

In the kitchen, inexperience came in all respects, from chiselling to re-plastering, as well as the electrical wiring. From a safety point of view, the wiring had to be completed accurately, first time, meaning that there was no room for inexperience. To overcome the inexperience, two potential solutions existed. Firstly, work on a prototype may have allowed the author to gain the experience required. From the chiselling and re-plastering point of view, there was no point in practising on a prototype first, especially because the plastering only required a rough finish, as tiling would eventually be put on top of it. The second solution would have been for the author to consider getting some expert advice. In fact, this was the case for the electrical work, due to the safety aspect. Either way, the impact of inexperience meant that the chiselling took much longer than originally anticipated, that the plaster had to be re-worked because the wrong type of plaster was originally used, and the project began to overrun.

In software, inexperience in technologies is normally handled in one of two ways. Either the developer realises their inexperience, and a prototype is built to test out a design and learn the technology, or the developer does not realise or will not admit their inexperience, and development takes longer and may contain more bugs. Either way, development time is detrimentally affected.

The Prototype

Building a prototype allows the development team to explore the technology being used, and come up with the best technical solution to the problem. Often the time taken to create the prototype is not too great, because it does not need to be fully tested, nor does it require a tidy finish (in terms of design or aesthetics). However, due to these facts, plan to throw the prototype away before re-commencing the development of the production system.

Calling in the Consultants

Using the local DIY store for hints and tips is a good way of getting free advice, although the development team loses time during these visits. A software parallel is when developers search on Google Newsgroups[4]. This is a free resource where people who have faced similar problems in the past, document the solutions. However, finding the exact solution can take a long time, and while it is being searched for, the developer cannot actually develop.

For really tricky problems, true professionals can be called in to advise. It is remarkable, but true that the daily rate of a plumber or electrician can be similar to that of a software consultant, except that the former often charge a callout fee – something that should be considered as adoptable by software consultants when only a part days work or short term contract is given.

Now, just because a consultant is called in, does not mean that the problem has been seen before, and that it will be resolved quickly. And in both types of project/industry, the quality of the consultants knowledge is not necessarily dictated by the flashiness of their advertising/marketing, nor by their callout fee nor daily rate. Word of mouth is certainly the way forward in consultant recruitment, however, this often has the problem that particular individuals are sought after, and as such they may not be instantly available, resulting in either selection of a different candidate, or slip in the project plan while waiting for that resource.

Availability of consultants deserves more thought. It was found that high-tech skills such as plastering and electrical installation are actually extremely sought after skills, and their availability is closely related to the current market conditions. When the market is favourable for the consultants and there is a lot of work for them, the lead time for getting the consultants is increased. The chance of getting hold of a 'cowboy' – a rogue labourer – is also increased, as people with fewer skills see the opportunity of quick and easy money in the particular industry. If the market is less favourable for the consultant, it is easier to find one with good skills quickly and at a cheap daily rate (the cowboys soon move onto the next booming industry).

The Technical Guru

Kitchen fitters often work as a team made up of one very experienced fitter, leading one or more less experienced fitters. The experience of the leader will span all aspects of the kitchen being fitted, including plumbing, electrics, tiling, carpentry, unit building, appliance fitting, etc, as well as aspects outside of the current kitchen. Whenever a less experienced fitter has difficulty, he can refer to the master fitter.

In software, projects often have a technical architect, who has all the experience in the technologies being used on the project, in addition to technologies outside of the current project. He will have an understanding of computer networking, be an expert in the

business domain for which the product is being built and have a deep knowledge of the technologies being used to implement the product. He will have excellent soft skills for leading the team from a technical perspective, and will review all designs which are made. He will grasp the *bigger picture* and understand what the customers needs are. He will act as the project manager's right hand man and attend most meetings that take place on the project. He will be responsible for all project documentation. Finally, the technical architect will have the natural ability to know when the time is right for implementation to begin. The author has seen many cases where the architecture is argued over for far too long and even cases where it is changed after implementation has started. A good technical architect knows when the architecture will be adequate for the needs of the project, at which stage it will be fixed, and implementation can commence.

Budgetary Constraints

In any project there should be a contingency budget, which is used for completing tasks that were not originally considered in the initial plan. What happens when money runs out? Tradeoffs must be made. In the case of the software projects, this often results in reduced functionality of the final product, or lower quality. In the kitchen project, similar effects occur. Either certain planned features of the kitchen are removed from the plan, such as those *must have* overhead lights for £100 a set, or lower grade materials are purchased, resulting in reduced quality of the kitchen, for example that slightly cheaper model of dishwasher, that is 5 decibels louder than the originally desired item, or even worse, a halogen light fitting which due to low grade workmanship, flickers annoyingly until it is up to working temperature. In software, the budget is usually fixed by the start of the project. However, if it is discovered that for example, more consultants are needed, paying for this additional work may result in the functionality of the project being reduced.

The Bidding Process

The bidding process relates to tendering work out to subcontractors. In terms of the kitchen project, this means choosing its suppliers. There is an irony in this process which should be considered with great care. When tendering out work, the suppliers will know that they are competing against other suppliers, and as a results in a bid to gain the work, all suppliers have the fact that they must produce the lowest possible cost, in relation to their standard of quality that is possible. By aiming to cut costs, they are already considering reducing quality and/or functionality before you have even hired them.

In software it is not even unknown for a supplier to offer work at such a low cost in order to guarantee the contract, based upon the assumption that they will get more work in the future, and be able to make profits at a later date. The author has experienced this, together with the associated reduction in quality of the deliverable.

In determining your suppliers, firstly calculate what you think it would cost you to develop the component yourself, to your standards. Then, create a table of criteria which are important to you for this component. Give a weighting to each criteria to show its relative importance to you, and then grade each prospective supplier in each category. Sum the weighted scores, and the three suppliers with the highest overall scores are who you should start talking to. Always get at least three estimates. Ensure that the cost is one of your criteria, but neither the only nor the most heavily weighted one. If the highest scoring supplier has a cost lower than everyone (including what you think it will cost to deliver), move on to the next highest scorer.

Changing the Delivery Dates

Two types of change to the delivery date can occur. The first is related to external deliveries of components which fail to be delivered on time.

In kitchen fitting, different manufacturers can create different appliances or fittings for the kitchen, due to open standards that have been agreed upon, for example the standard width of a base unit, into which a fridge or oven fits. The benefit of this is that other constraints such as financials or aesthetics can be considered when choosing the component to purchase. Different manufacturers are then used for delivering the different components, and each provides a scheduled delivery date.

Similarly in software, a product is often broken down into components, and these can be farmed off to different development teams. The only thing that needs to be known for each component before development commences is the interface that should be delivered. In fact, the development can be given to teams internal to the customers company, or they can indeed be bought or developed using external companies.

However, things start to go wrong when management get a phone call stating that a component is not ready (out of stock in the case of a kitchen), and that it will be delivered a week late. This is not too bad if it only affects the overall functionality of the project, like an oven, but can be disastrous if it is some of the housing units which will be delivered late, when they are the foundation units of the kitchen.

In software terms, this is akin to missing interfaces in components delivered by other software development teams. While this may sound strange, the author has experienced a software component being delivered on the delivery date in order to meet that deadline, only to find that the component didn't actually contain any binaries, and the source did not even compile. It had however been "delivered on time"…

Retrospectively, using many suppliers eases the problem of changes in delivery dates, as if only some of the components will be late, it allows the project to be re-planned, so that at least some work can continue. Consider what would happen if all components were to be delivered by a single supplier, who then informed you that they would all be late. Productivity would have to completely lapse until the components were delivered!

The second type of change in delivery date is the internal type. This is when senior management take the decision to change the delivery date, bringing it back in time and demanding an earlier delivery. This may occur if management take the view that the original plan has been given too much budget, or the decision many be related to other dependencies if the project being built is part of a much bigger system. If this were to occur, motivation would drop, as project members begin to wonder whether their hard work to date has been noticed and they start preparing to work even harder. Even the project management will become unmotivated, because effectively their power and ownership over the project has been totally overruled. There is also the question of whether the new deadline is actually achievable. Adding manpower or overtime is the typical reaction, but as reference 5 discusses, this will increase the cost. This extra cost as well as the extra cost due to demotivation, must be explained to senior management as soon as their request is made.

Motivation and Productivity

If kitchen developers feel that the product they are developing is of poor quality, they can become unmotivated. If they feel that the component they are building is not of value to the project and that it may not be included in the final kitchen, they can become unmotivated. In fact if a developer has been developing the same component or even product for too long, they can become unmotivated. Other factors that contribute to reduced motivation include repeatedly seeing the same types of mistakes (bugs) cropping up, not seeing appreciation for their hard work, and failure by project management to properly manage a failing project. Even project management can become unmotivated, if as previously stated, senior management removes part of their control over the project.

All of the above mentioned have been noted as motivation reducing on kitchen projects, but are equally relevant to software projects. Developing that low level controller code which bills customers for using their mobile phones[6], and not being thanked for the hard work, can be just as unsatisfying as having to slog away at channelling out the wall for those electrical cables, or having to work on that endless task of tiling above the work surface in the kitchen.

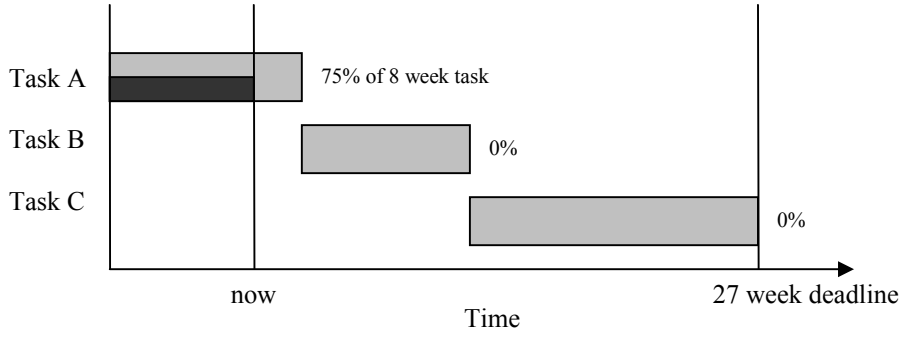
In any case where the developer becomes unmotivated their productivity starts to drop. They start to think about the future, and other projects, or perhaps their career progression. Soon enough they are even considering moving to a different company. Once this has happened, you can guarantee that they will discuss their situation with other developers on the team, and it won't be long before they too become converted to the demoralised ways of thinking.

Management has to be extremely quick to notice when this is happening, otherwise the project is bound to fail. On a kitchen, project management might think that because there is no other solution other than finishing the kitchen, that they are safe. After all, the development team and management are both stakeholders, and have to live together and use the kitchen as part of everyday life. However, there is no real reason why the development team cannot give up and move on to a different project, leaving management no option but to buy in a different development team to finish the project, which undoubtedly will result in late arrival of the deliverables, and at much greater cost!

Project Plans

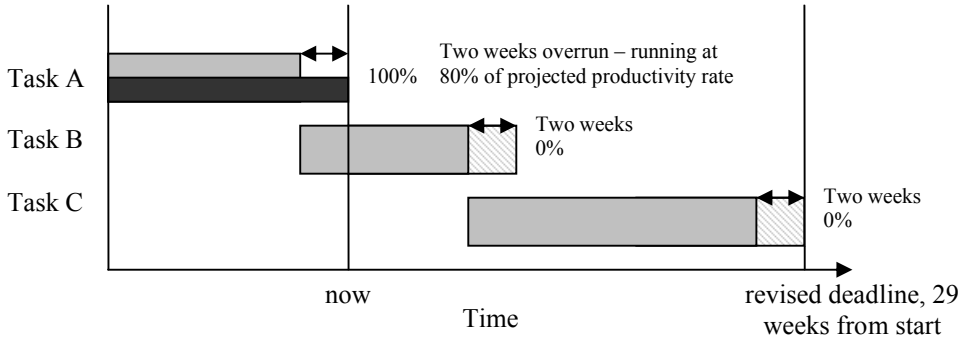
The Gantt chart provides a good way for the project manager to quickly view the tasks at hand on a project, to determine the critical path through the project, to manage resources on the project (making sure everyone always has something to do), to track project slippage, and in such an event to re-arrange/reschedule tasks in the project to reduce the impact.

In fact the Gantt chart is not specifically a software management tool, but is indeed used in many industries. Hence it might come as a surprise to some, that they are not always used. In fact sometimes, no form of tracking tool is used at all. And not surprisingly when this happens, projects begin to become disorganised quickly, but no one notices the slippage, and it becomes increasingly hard to re-plan the tasks at hand to reduce the impact of the slippage.



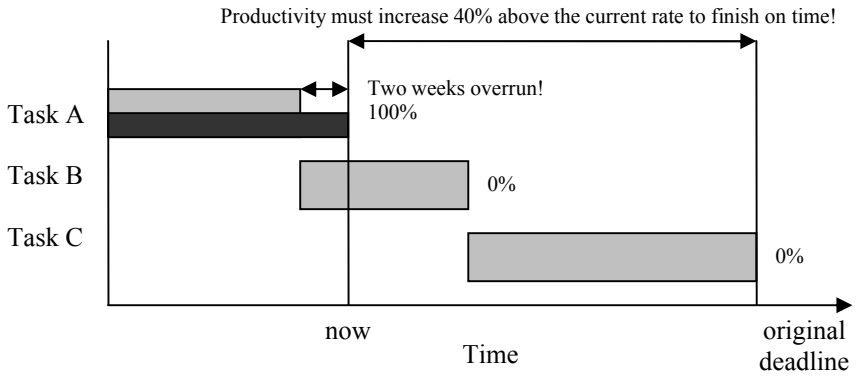
A Typical Gantt Chart

The Gantt chart shows the actual progression to date on the left hand side, and the projected progression on the right hand side. Often management notices slippage, but only records it by moving the entire right hand side of the chart more to the right, effectively shifting the end date for the project by the amount of current slippage. Doing this is artificial, because if slippage has occurred on the left (in the past) it is likely to occur on the right (in the future).



Gantt Chart With Revised Deadline

Even worse, where the deadline is absolutely not moveable, the right side is simply compressed, leaving the end date in the same place. The effect of this is no longer artificial, but effectively twice as bad as simply moving the end date by the current amount of slippage. In the given example, it has taken 10 weeks to complete 8 weeks of work, equating to a 25% decrease in projected productivity. If you continue at this rate, it will take nearly 34 weeks, instead of the originally planned 27. If you keep the original deadline, you how have only 17 weeks to complete another 24 weeks of work. This equates to requiring a 40% increase in the current work rate. Using the Gantt chart to notice slippage early greatly reduces the problem. If you do not notice the slippage until halfway through the project, productivity has to increase 67% above the work rate of the first half[7].



Gantt Chart With Fixed Deadline

Sadly the kitchen project did not use a Gantt chart, or any other form of project tracking tool. Hence it goes without saying, that the project suffered as a result, in the form of much later delivery than expected. Had a project plan been used, it would have provided all the advantages discussed above. However, a further advantage with good project tracking is that the expectation of the stakeholder can be managed – they will now early on if things are not going well, and will not have a huge disappointment on delivery day when you inform them the project is not complete. This provides the opportunity to discuss the tradeoffs that could be made if the deadline is not flexible, like reducing the functionality or quality.

Communications

The term communications refers to multiple streams of information flow across a project. Firstly, communications refers to keeping in constant contact with the customer to keep them informed of what is going on, where potential issues exist (with the delivery, or also with the requirements).

Secondly, weekly team meetings need to take place. These should cover development discussions, architectural decisions and project planning. They are a good way for everyone to communicate quickly on the problems they have encountered during the last week, and to share knowledge. If harder or more time consuming tasks are identified, then resources can be reallocated during these meetings. A developer who has encountered a particular problem before, or who has taken less time than anticipated to finish their current tasks can assist other developers having problems or struggling to complete their tasks on time. To document all the architectural decisions, create a website where all project meetings are documented, where progress reports are available, and where project specific documentation can be read. Care should be taken to limit the size of these meetings. If there will be more than 10-15 people attending, break the meeting down into several meetings, each covering a particular functional component.

Equally, the project manager should also communicate frequently with the developers to aid tracking the project. The project plan should be updated regularly, and where required, re-planning should take place.

Finally, communications also refers to talking to suppliers regularly, for the reason stated earlier in this paper. Good communication will help reduce the impact of late realisation of delays, which have just been shown to get dramatically worse, the later they are discovered.

Trade Off of Attributes

When things on the project go wrong, tradeoffs may be required. Consider the various attributes of a project, from a management point of view. Quality, Functionality and Cost are all related, and important to a project. In fact, their relation can be viewed as a balancing equation. So long as nothing goes wrong in a project, these attributes balance. But as soon as something starts to go wrong, (for example more time is required, which will increase the total cost as more man days are added), the other attributes must be altered to maintain the balance. The problem that the customer faces is that they cannot dictate all of these attributes. If for example, they want the cost to remain the same, so that no extra man days are added, then either the functionality or the quality must be reduced. In relation to software, these are discussed in reference 1.

In the kitchen, the quality relates to the finish of the entire project, for example how well the units are put together, whether they stand upright or slanting to one side. The functionality relates to what the user is capable of doing in the kitchen (cooking on a hob, cooking in an oven, cleaning using a dishwasher, having enough space to chop vegetables, etc). Cost is related not only to the cost of the materials, but importantly to the time taken by the developers to build the kitchen.

The equation dictates that when things go wrong, at least one of the attributes will be out of their control, in order to let the equation balance. If the original plan does not reflect reality, the customer will have to agree to a change in at least one of the attributes.

For example, if the customer decides that the functionality which must be contained in the kitchen, as well as the overall cost must remain the same, then they will not be able to dictate the overall quality, which will have to reduce as the developer has to cut corners (usually via writing less tests) in order to finish on time. The manager cannot add developers to help, because the cost has been fixed.

Similarly if the customer decides that the quality of the kitchen and the cost must be maintained, then they will have to agree to a reduction in the functionality of the kitchen, so that development can be completed on time – and again, no extra developers can be added because cost has been fixed. And if the customer chooses to keep the functionality and the quality the same as originally planned, they will then have to pay more as the project will take longer to complete or manpower is added in order to maintain the deadline. Interestingly this is often why the customer prefers the fixed price contract, because they can then fix all three attributes. In such a case, what really happens is that the development pays the additional cost, in terms of reduced profit (or they overspend on their budget, in the case of an internal development team).

Comparing these examples to software, we can apply units of measure to the attributes, and show the relationship of the attributes in the form of an equation.

If functionality (F) is measured in number of Use Cases, quality (Q) is measured in the Number of Bugs present in the final product, and cost (C) is measured in currency (and itself is a function of the number of man-days of work, the number of people on the project, and the charge rate of each developer[5]), the equation looks as follows.

$$\frac{F}{Q \cdot C} = \text{constant}$$

This can be interpreted to show that if for example the number of use cases that need to be in the final project needs to be increased, either the number of bugs will also increase or the cost will increase.

The Impossible Deadline

There can be serious implications to the success of the project, when senior management sets a deadline for the project *without* an accurate plan, and sets it in stone. Similar to the way that senior management might demand the rollout of a new software system on a given date, using an initial and inaccurate project plan from the project stakeholders, the kitchen project can also have a deadline set in stone, when it too has an inaccurate plan. The *Dinner Party* to launch the completion of the kitchen project and to show off its success to friends and family has to be booked weeks in advance to ensure availability of attendees. Changing the rollout date for the kitchen project is not viewed as acceptable by senior management, as it would mean a let down to friends and family. This poses no problem, until project slippage starts, due to the poor initial plan, and lack of contingencies in it. At the stage that project management notice the slippage, there are a number of *knee jerk* reactions that they take. The result is always the same. They look at the functionality-quality-cost equation and they determine that it will cost more money to fix, since they (rightly) assume that the customer will not agree to a reduction in quality or functionality. The assumption then goes that an increase in cost means they have to increase the number of man-days, which is either achieved by adding overtime to the project, or by adding more developers to the project.

In the kitchen, overtime is constrained by things like it not being sociable to start chiselling away at the walls or drilling them late at night. Software does not exactly have this problem, however demanding overtime has an impact on motivation. To start with, the impact can be positive, if the company pays its staff for overtime. However, anything more than a couple of weeks of daily overtime starts to wear down the staff. Not only will they become tired, which will reduce quality, they will become unmotivated, because all they seem to do is work, and they don't have time to spend relaxing or with the family. Furthermore, some staff cannot work overtime due to family commitments, for example young children who need looking after, or travel commitments like the long commute to work each day.

Adding man power to a project also does not necessarily help. The problem is that the relationship between man-days, number of people, and resultant delivery time is not linear, as discussed in reference 5. The time spent mentoring the new project members and the overhead involved with the extra staff will cause overall productivity rates to drop.

The solution in the author's opinion is not to consider reducing the quality of the project, and it is also not to consider moving the deadline, because the customer has already stated this is not an option. The only other attribute that can change in order to maintain the cost, is the amount of functionality that will be available. Often the customer's requirements for a system are in excess of what is actually needed to perform their daily

tasks. Requirements specifications will contain a lot of *nice to have* features which the customer has justified because they have been given the budget to procure a new system. Working closely with the customer, you will be able to work out which features can be left aside for the first release (still to be delivered on the original delivery date) and moved into the second release (which will be delivered at a later date). The overall effect is similar to simply moving the deadline, except that on the original deadline, the customer will at least have something that is useable. Obviously the additional cost of the second release must be paid for. Determining the root cause of the delay using historical project plans will help work out who is liable for this extra cost.

In the case of the kitchen, the dishwasher and washing machine were not actually required for the dinner party. Legacy systems like the sink could still be used to replace the missing functionality of the new product, until such a time, as their installation was complete.

Changes to the Requirements

In the kitchen, when the cooker hood was delivered, it was found to be much bigger than initially anticipated. The result was a small change in the positioning of the hob, resulting in a change to the layout of the base units, and custom hacking (literally!) of one unit in order to fit the hob.

No matter how much requirements gathering, or analysis and design is carried out before implementation begins, discoveries will be made which will mean changes to the overall project requirements. This is often due to technical limitations in the technologies being used, and as such cannot necessarily be foreseen during the requirements gathering stages.

Also, the customer may discover more information that they did not have at the time of signing off the requirements, which makes them want to change them.

As soon as requirements changes are received from the customer (or agreed with the customer in the case of changes related to discoveries in the development teams work), the project plan should be revised to allow additional re-design and re-analysis work to be carried out. The impact of these changes must be completely explained to the customer in terms they understand, and changes to the project plan must be agreed upon. If the deadline is set in stone, refer to the earlier section.

Flagging Morale Towards the End

As any developer nears the end of a project, whether it is the end of the project, or simply the developers roll off time, there is a natural decrease in their morale on the project. They start thinking about the future, what the next project will be, and what new fruit it will bear. Just as this happens with software, the kitchen project starts to get boring, and the developer starts thinking about how ugly the bathroom now looks (relative to the shiny new kitchen). Catalogues are ordered showing the latest in tap technologies for bathrooms, and a sense of excitement overcomes the developer, just as they might look forward to their first Java project after a string of C++ projects. Productivity drops, and those last few finishing touches take longer and longer to complete. If management are good, they will realise quickly that this could have an impact on the completion date for the project, and they crack the proverbial whip to maintain productivity. However, if management also get swept away with all the

excitement of the new project, it is quite possible that the current project will be rolled out with bugs and missing functionality.

Anger Management

Project Management, take note: visibly pacing up and down, threatening behaviour such as swearing or shouting, actual threats, showing visible signs of stress and pressure, revoking privileges such as lunch, shaking of the head and similar outward expressions of dissatisfaction, do not encourage a developer to work quicker and more accurately. In fact, putting a developer under pressure will certainly decrease the reliability of the product they produce.

The Incremental Delivery

The kitchen was built one step at a time, but at all times, some of the functionality was available. For example, once the sink was installed, it could be used at anytime. The same was true of the oven.

In software, a typical project may not have a deliverable until the very deadline. None of its functionality will be available to users until it is delivered. This is the way software used to be built, and still is when it comes to the software products industry with projects such as Microsoft Word.

However, for a large majority of bespoke systems being built for customers of all ranges of industry, this need not be the case. Iterative delivery cycles, where frequent deliveries are made (as often as every few days) through the life of the project, allow the customer to gain early access to their product.

This not only has the benefit that they can see what they are paying for, and are happy to keep paying the money, but many other benefits too. They can critique the product allowing for improvements along the way (which can make it cheaper as components that have not yet been coded can be redesigned, reducing the total cost of ownership [TCO]); they can assist in testing the product; they can start to use some of the functionality and get an early return on their investment [ROI]; and they can refine the requirements.

This last point about requirements is extremely important. Iterative delivery cycles allow the customer to start changing the requirements as they start to see the product and realise they might have wanted something else, or something better. This is fine, so long as project management get a handle on it, and allow changes to the requirements in a controlled manner. A controlled manner is one in which redesigns occur with enough thought, and are paid for by the customer. Development paradigms such as Extreme Programming[8] (XP) cater for and encourage incremental deliveries and changing requirements.

The author has experienced both single delivery (where the product has been delivered at the prearranged delivery date) and the incremental delivery (lots of deliveries, each providing the customer with more functionality than the last), and by far, in every case the incremental delivery was the more successful project, resulting in a much happier customer, often willing to pay more for more functionality, as they saw the project delivering against their requirements. Although there is extra overhead in managing the associated problems of an over eager client, the benefits outweigh these overheads.

In fact, the incremental delivery actually helps to solve many of the problems discussed in this paper, such as changes in delivery dates, the impossible deadline, and flagging towards the end (as there is no major end).

Reducing the Impact for The Software Project Manager

For each of the examples given in this paper where something goes wrong during the project, anecdotes for use by project management exist for reducing the impact that these cases have on the overall success of the project. They are as follows. While it may not be possible to use all of them, the more that are used, the more likely it is that the project will succeed.

1. The Successful Project Do not be complacent. Up to 70% of software projects fail to deliver to some degree, so there is a good chance your next project will fail too. Use the anecdotes given below to help avoid failure.
2. Poor Initial Plan Always negotiate for at least 30% more budget than you expect the project to take. Remember that many hours are used in day-to-day tasks that you may not have considered. When doing an initial estimate, factor in leave (annual, sick, training, etc) and productivity based upon metrics that you have from previous projects. Remember to record the metrics for this new project too, to add to your historical data for future estimations.
3. Inexperience in Technologies Create an environment where the developers are free to express their inexperience early on. This way, the impact can be included in the initial project plan.
4. The Prototype Plan to build prototypes, but also plan to throw them away and start the production development again using the lessons learned from the prototype.
5. Calling in the Consultants After assessing the inexperience in the technology, and if required, allow for a generous budget to get the relevant expertise. If possible use personal recommendations or interviews when selecting potential candidates.
6. The Technical Guru Always have a technical architect on the project, and when choosing them, ensure that you gel with them, that they have all the required technical prerequisites (a deep understanding of the technologies being used and an excellent grasp of IT systems in general), as well as all the interpersonal skills of a good manager. Ensure that they are the type of person who gets things done (like an engineer), and not the type that continuously tinker with the architecture (like the computer scientist).
7. Budgetary Constraints Allow for a "slush" fund – a small budget which can be spent on tasks which are not in the initial plan. If it starts to run out, prioritise the tasks in hand with the customer and start with those which are more important. Sometimes these are functional and quality related, but they can be aesthetically related for projects producing user interfaces.
8. The Bidding Process To choose the supplier for a specific component, create a weighted scoring chart (score the supplier in each criteria that is important to you, giving each criteria a weighting as to its importance) to compare all suppliers. Work out how much you think it will cost to develop a component. If a supplier is offing it to you at a cost less than this, work out why. Applying penalties in the contract for the deliverables may also help reduce lateness and poor functionality or quality, however this may increase the cost of the components.

9. Changing in Delivery Dates Communicate as much as possible (even on a daily basis) to find out as early as possible about problems with deliveries from externals. Using multiple suppliers will help to reduce the impact of a late delivery. Know the critical path through the project plan, and know which components the project depends on most heavily. Before problems even arise, consider contingency plans related to the critical path, and most heavily depended upon components. To avoid changes in delivery date from internals, project management must have an understanding with senior management from the start that they will be fully responsible for the successful delivery and financial success of the project[9]. From the start they should make it clear to senior management what the results of a change in delivery date will be, which are increased cost due to adding manpower and/or overtime, as well as reduced quality, functionality, and/or morale.
10. Motivation and Productivity It is acceptable to pass the work on to a different developer within the team. Pass on big tasks, or indeed break things down into smaller more manageable tasks. Either way, there is actually a big benefit, which is knowledge sharing, as more developers start to become aware of more parts of the system. Also remember to show your appreciation of the work that the developer delivers through general gratitude as well as social events paid for from the slush fund. And keep overall quality in the front of your mind – if a developer asks for better tools or more development time, it will result in a higher quality product. This then protects the project from attrition rates, although ironically once you increase morale, attrition rates drop anyway.
11. Project Plans In order to create and maintain a project plan which is accurate, the manager needs to make projections based not only on actual current trends from the project, but also on historical knowledge. Project management data must be collected from many projects and a database should be built up so that accurate projections can be made. The project manager should also not only look at average development rates, but should also consider single events, which cause slippage, especially those which have knock-on effects. A project tracking tool must *always* be used even on the smallest of projects, as they act as a reminder of what is due and when, and how slippage will affect the bigger picture. Use the project plan to determine early on when slippage is occurring, and then re-plan the project, based on current and historical productivity rates for the upcoming work (the testing phase will have different productivity rates than the development rates, so you cannot forecast the time required for testing based on your current productivity rate for development). If slippage is unavoidable, discuss it with the stakeholder early on, to reduce their stress, and work with them to consider tradeoffs that can be made. Remember that early recognition of slippage will make it much easier to get back on target. Consider incremental delivery is slippage occurs.
12. Communications Ensure that weekly development meetings take place, where information on the project can be exchanged, and architectural decisions can be passed on to the team. Ensure that communications are kept regular with the customer, informing them of problems in the plan, issues related to requirements, and indeed letting them know when things are going well.
13. Trade Off of Attributes According to the equation given, functionality, cost and quality equate to a constant. Changing one of them requires at least one of the others to change in order to maintain the original constant. If the initial project plan is wrong, or problems outside of the scope of the project plan occur, then

the equation becomes unbalanced – it can no longer equate to the same constant without one of the attributes changing.

14. The Impossible Deadline If things start to go wrong on the project, and the customer will not allow the deadline to be moved, the only realistic option is to cut the functionality which will be delivered on the deadline. A second phase should be planned, and arbitration over who should pay the cost of the second phase should commence. Use your project plans to determine the root cause of the delays, which may show who is responsible.
15. Changes to the Requirements Analyse the impact of the changes in terms of additional re-analysis and re-design on the project. Determine the overall impact of the changes and work closely with the customer to resolve whether the changes should be applied or not, which could be the case if the deadline is fixed. The customer must understand fully the new risks involved with you accepting changes to the requirements. Consider running the project as an incremental delivery project to reduce the impact of requirements changes, which come naturally with a project run in such a fashion.
16. Flagging Morale Towards the End Start to plan the end of project party early, and get everyone on the project involved. Do not encourage staff to think about the future, leave them to it. Remind them of the bigger picture and involve them in demonstrations of the product.
17. Anger Management Project managers should do their utmost to portray a confident and relaxed manner showing that they are in complete control at all times. Not doing so, will rub unhealthy stress off to the development team, who will already be stressed by the daily sight of a project plan, to which they will naturally work their hardest as it is what they have been trained to do. A good HR department will ensure that lazy staff who neglect responsibility are never hired in the first place.
18. The Incremental Delivery Plan the project so that you can work closely with the customer, and provide them with frequent functional deliveries. This will give them confidence in your ability to produce the product, allow them to critique the deliverables, encourage them to increase the budget as they are happy with progress and realise that they need more functionality. Finally, if things do go wrong, at least they will have something to work with – you will greatly reduce the chance of a total project cancellation. Be very careful to manage the customer's eagerness to make changes to the requirements.

Conclusions

To be fair to the kitchen project, this was the first one undertaken by the implementer in question. As a result, problems discussed above like unknown technologies, poorly set initial plans, etc. may not be as problematic in the future, considering the experience gained from this project. The same could be said for software projects, however, these problems recur time and time again for them. It takes an extremely experienced and gelled software development team (project management *and* developers) to get projects right, repeatedly. Hence the author predicts the same is true of kitchen projects. Hopefully however, it will be many years before the author tests this theory!

This paper has discussed most of the common problems that occur on software project, but from the point of view of a kitchen fitting project. The resolution of these problems is the same, regardless of the project, showing that they relate mostly to management issues, not to the software industry, which is often slated for the poor record of successfully delivered projects.

As such, software project managers should be aware of the resolution of the problems highlighted in this paper, and should take the anecdotes on board for the future, as doing so will certainly not increase the risk of a failed project.

References

1. *eXtreme Programming Explained*, Kent Beck, Addison Wesley, 1999, ISBN ISBN: 0321278658
2. http://www.it-cortex.com/Stat_Failure_Rate.htm
3. *Peopleware: Productive Projects and Teams*, by Tom DeMarco & Timothy Lister, December 1999, ISBN 0932633439
4. Google Newsgroups / Dejanews: www.google.com
5. *The Mythical Man Month*, Brooks, F.P., Addison-Wesley, 1995.
6. The author started his career developing extremely low level code used in the mobile phone industry, adhering to outdated standards and developmental processes.
7. If a project is set for 27 weeks, and you take 17 weeks to do half, you are running 25% slower than planned. Now you have to finish the project in 10 weeks, but at the current rate, it will take a total of 34 weeks (25% slower). So the required rate becomes 27 weeks left / 10 weeks in which to do the work, which is a productivity increase of 67%.
8. www.xprogramming.com and www.extremeprogramming.org
9. *Kelly-More Than My Share of it All*, Smithsonian Institution Press, 1990, ISBN 0874744911

Dr Ant Kutschera is a principal consultant and technical architect, working for a private company specialising in J2EE Enterprise Systems (particularly Enterprise Application Integration, Composite Application Networks, Business Process Management and Work Flow) conducting work mostly for Global 500 companies. He can be contacted at ant@maxant.co.uk.

The author would like to express his thanks in the preparation of this paper to the following, in no particular order: Thorsten Wenzel, Clare Kutschera, and Shaun Le Geyt.

The author would be interested in hearing your views on how the advice in this paper relates to your projects. Please contact him if you would like advice on how to apply this paper to your work and projects.