# TickIT*plus* Implementation Note

| | | | |
|---|---|---|---|
| **Title** | Understanding Base Practices – Requirement Sizing | | |
| **Date** | April 2015 | **Reference** | TIN015-1504 |
| **Originator** | Dave Wynn | **Version** | v1r0 |
| **Key Terms** | Base Practices, Implementation, Requirements, Sizing, Estimating, Process and Practice Order | | |

There is one base practice hidden away in the Requirements Analysis process that is probably just briefly considered, simply justified and rapidly mapped in the PRM, but often without much thought as to what it is trying to achieve and why it is in the Requirements Analysis process. This base practice is BP.2 Estimate System Requirements Size. Take a look and then read on for a discussion around this base practice.

Three immediate questions might come up when thinking about this base practice:

1. What does it mean by size?
2. Why, if it sounds like something to do with estimating, is it in a development process and not in the project management process where estimating is done?
3. And, if it should be in a development process, why not in the Stakeholders Requirement Definition process?

To answer the last question first; in writing the Base Process Library (BPL) there was always going to be a risk that it would be interpreted as suggesting an order or time sequence to practices and more so to the processes, which was certainly not intended.

The BPL provides a set of processes made up of a collection of base practices that have to be implemented in the most appropriate, and beneficial, way by organisations using the TickIT*plus* scheme. This might involve some processes being done sequentially in the order within the BPL, or in another order, or in parallel or even iteratively, such as a case of going back and forth between defining stakeholder requirements and requirements analysis. Furthermore some processes, or practices within the processes, could be event driven rather than time or sequence based, e.g. handling customer complaints in the Customer Focus process, ORG.5.

The BPL does not infer order or sequence and was to some degree designed to emphasise this. For example, the Integration Management process (TEC.3) and the Transition and Release Management (TEC.6) process both come before the actual design and development processes. Note that although there is no specified order there is clearly some logical order to the processes, i.e. you cannot transition and release something before it has actually been created and you can't really create something until you have a rough idea of what it is that needs to be created.

Additionally, at the practice level within a process it may look like there is a specified order to the practices, but again this was not intended, although clearly there will sometimes be a logical order to the practices. In designing the BPL, practices where added at points in processes relative to the last logical point that makes sense to undertake them. Let's look at this a little further with some real base practice examples.

In addition to the practice mentioned above on requirement sizing, TEC.13.BP.1 Establish Development Approach, TEC.14.BP.1 Establish the Development Environment and TEC.14.BP.2 Identify Component Sources are all examples of practices which appear to be 'out-of-sequence' or 'later-than-expected' in the normal sequence of events.

TEC.14.BP.2 Identify Component Sources base practice in the Development Implementation process is all about looking at the classic 'make-buy-reuse' question, which is often something usually done as part of the early stages of a development, maybe even as early as during the bid phase, to understand some of the parameters involved in quoting, i.e. should we make it ourselves from scratch, adapt some existing system or buy the system or components from a supplier, either as a specific development or as a configurable COTS product. This decision will usually have a major impact on cost, timescales and almost certainly potential risks. If we make it ourselves there will be more effort and hence cost but possibly lower risk, whereas if we buy it, the effort could be less, although not necessarily the costs, but the risks could be higher. So given this, why isn't this base practice in a logically earlier process than the development implementation process, for example the establish stakeholder requirements process or even in the more general project management process. Again, this is not trying to suggest a defined sequence, but just that establishing requirements, to some degree of other, must come before we start developing.

However, this base practice has been placed at a point when it would not really be possible to go any further without considering this 'make-buy-reuse' decision, i.e. in the development implementation process where the decision must be made before any development would start, if indeed it was actually going to be built in-house. Placing it here allows the base practice, and hence the decision, to actually be undertaken at an earlier time and at the more logically considered places mentioned above, but without excluding situations where it is made later.

TEC.13.BP.1 Establish Development Approach in the Architecture Design process and TEC.14.BP.1 Establish Development Environment are similar base practices. Again, it would be normal to consider the development approach and hence the associated environment as part of the early planning activities, if not again as part of the bidding or proposal phase as there could be implications on cost. However, this is something that isn't necessarily the case in all situations and therefore the BPL accommodates the various options by placing the base practices in the last relative position that it would be sensible to consider them; just before development starts as it wouldn't make sense to consider the development approach or the environment after the development was supposed to have started.

So back to the original base practice in discussion, TEC.11.BP.2 requirement sizing, part of the Requirements Analysis process. This is similar to the second base practice above, although much less obvious as to why until its purpose is understood, which touches on the first 2 questions originally raised. Firstly though, the positioning of this base practice is again representing the last possible logical stage that the base practice should be done relative to other logical processes and practices. But, the real questions are what is it doing, what's it for and what's its purpose?

Sizing requirements is all about supporting and improving the engineering estimating process and hence why its positioning is in the requirements analysis process and not in the more logical process of establishing stakeholder requirements or even in the project management process; more about this in a moment. Again, however, there is no reason why it cannot be done in a logically earlier process; it's just that it doesn't really make sense to do it in any logically later process.

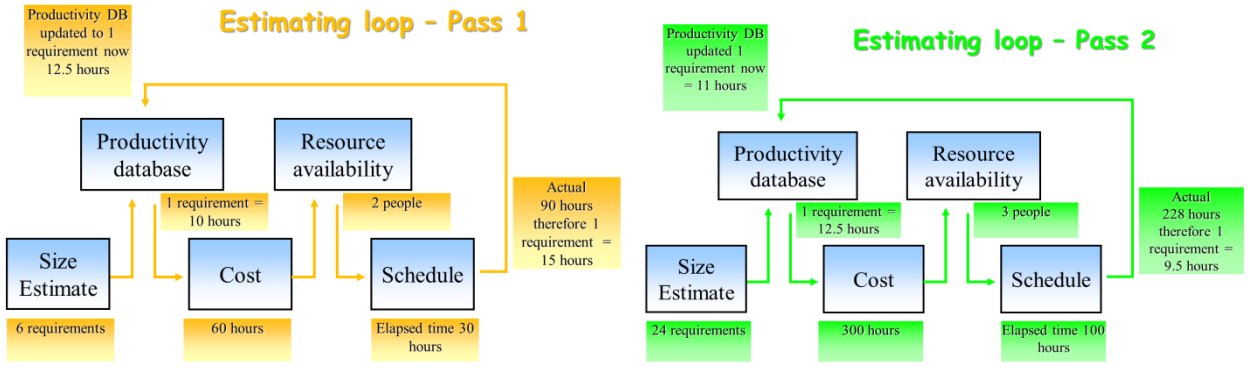
Back in the 80's, software development estimation techniques were starting to be refined and improved. The approach being established tended to involve the classic lines of code (LOC) or thousand lines of code (KLOC) as an indication of development size. This progressed over the years to function point counting (still in use today), widget counting and other, more specific, scientific and sometimes more obscure, software size measuring parameters.

The fundamental concept was to establish, or estimate, the number of lines of code and then through historical productivity data which aligned a line of code (or KLOC) to development effort in hours, to produce an estimate of the total effort needed to undertake the development. The task of establishing the lines of code estimate was well and truly the responsibility of the engineering staff. Ask a project manager how long something will take and there will be an over-riding desire to reply, "when do you need it", but ask how many lines of code it will include and will often be a blank face.

The historical productivity data would typically have been collected from previous projects by looking at the actual lines of code and the actual effort and approximating a productivity measure between the two. By merging in subsequent projects, the productively estimate gains greater accuracy related to the particular organisation, and type of work, gathering the historical data.

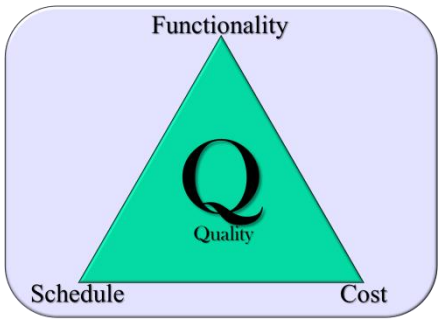
Techniques and tools such as COCOMO for example, came about that took this raw approach and extended it to provide rough estimates for effort and duration for the classical waterfall lifecycle phases. As the development of a line of code can be influenced by many factors, COCOMO introduced what it called 'cost-drivers' which were simple adjustment factors based on typical development parameters. These factors include such things as experience and newness of the development team, familiarity of the development tools, clarity of requirements and other aspects that could influence the development. The cost drivers were relatively simple weighting factors that refined or weighted the lines of code estimates before applying the historical productivity effort function.

The general approach was to boot strap the system by using 'industry' norms for productivity in the type of development being undertaken and refining these norms over time with organisational specific data as discussed above. The initial accuracy was fairly limited, but tended to improve as subsequent real data was collected.




By collecting the effort per phase the approach could provide rough estimates, based on the initial lines of code figure, for each phase in a new development.

One important factor here is that the lines of code estimate is done by the developers, not the project managers, and this to some degree tries to ensure that the final proposed costs and schedules are owned by the development groups. If the business, through the project managers, wants to reduce either or both of the costs or schedules involved, then there will be a much more visible method on which to discuss which part of the functionality should be dropped, delayed or phased into the delivery. Arbitrarily cutting costs or schedules without due consideration of the development activities will, and usually, does result in overrun, budget extensions etc. which in practice often end up looking like the original estimates proposed before the cuts. You often hear of these situations with developers saying that the project ended up being delivered around the time and for the cost they originally expected it to be. The only real effect is usually a lot of confusion, fire-fighting, negotiations, compromises and grief. This is one reason why this base practice was added to a development process rather than the project management process.




Now, many people have said that it's not possible to do this because every development is different and therefore the estimate would be inaccurate. This is probably true, but the main thing is that it is only just an estimate; it's not supposed to be an exact science, it's not supposed to be any better or worse that a good engineer providing an estimate. Ideally it should be available as another estimating approach by which to corroborate other estimates that might be being used. It's also true that every development is often quite different, although not always, but usually the development approach isn't significantly different; developing two completely different systems both through the same lifecycle have one thing in common; the lifecycle. The main aim of the Silver level in TickIT*plus* is to identified a standard process set albeit including allowable, defined and understood tailoring[1]

The art of sizing requirements is to consider something that is common to the development approach, not the actual development. It doesn't have to be a perfect measure; it just needs to be a repeatable measure. It is, after all, just going to provide an estimate, not an exact number, of the effort and duration of the development and not of some facet of the system, e.g. storage capacity, processing speed etc.

Over the years the measures used, the approach and the perceived value have all chopped and changed, and these days it's not very common to see any real development sizing being undertaken, and if it is, rarely for estimating purposes. More commonly, sizing is done at the end of projects to provide some form of benchmark indicator of the performance of the development organisation, usually needed as part of contract requirements on productively.

So, let's look at how we might make this work and what benefits can be obtained for relatively little effort. To start, there are two key principles to consider. Firstly, the size estimate is not in hours; hours are a measure of effort and this comes about by breaking down what needs to be done into measurable estimable chunks and then understanding how long it will take to do these chunks based on previous experience of doing them or similar ones. This is almost certainly what 'George' will do when you ask him for an estimate for developing some code or

[1] Please see my webpage on TickIT*plus* Capability Levels at http://www.omniprove.co.uk/capability-levels.html

system. Interestingly, all organisations have a 'George', maybe with another name, who is the person who provides estimates because "they've been here the longest and have seen it all". The idea is to get some of that valuable 'George' knowledge into a system that is more visibly available to the organisation and can therefore let 'George' retire in peace when he wants to without feeling guilty or without the organisation stressing that it will never be able to estimate accurately again.

One analogy could be a window cleaner, possibly even called 'George'. Ask him how much he would charge to clean the windows on your house and he won't simply say $20, he'll probably reply immediately with "well how big's your house or how many windows and floors do you have?". Then, based on previous experience on cleaning the average downstairs and upstairs window, he'll be able to give you a rough estimate for the time and hence cost. In fact, he probably doesn't need to even ask about the number of floors, because over time, he'd have worked out a very good approximation that provides a good enough estimate for what people would want irrespective of which floor the window is on. Of course he could always supplement this calculated estimate with some good old-fashioned questions in order to construct a bottom up estimate.

This leads nicely to the second key principle, that the result of this sizing exercise is only an estimate and is rarely perfectly accurate; it's not supposed to be, it's an estimate. This shouldn't however inhibit using the approach, which unfortunately it often does. It should be used with other estimating techniques to provide additional confidence on the overall development estimates for effort and duration.

So, what should be used as the measure? What should the measurable chunks be? Well, as mentioned earlier it can be pretty well anything that can be repeatedly measured across different projects. This could still be the LOC or KLOC measures or the function points mentioned earlier or anything else that is typically an indicator of the size of the development. If you ask 'George' nicely he'll probably tell you, although he's probably never been asked before and might wonder what you are after. Some options could be:

- Screen counting: how many screens are we likely to have to develop or change?
- Interface counting: how many interfaces are we likely to develop or change?
- Requirement counting: how many requirements do we have?
- Task counting: how many tasks have we identified?
- User stories: how many user stories are involved?

And there are probably many more specific counts that could exist. In many cases these simple counts, although valid, may not actually give us a reasonably good ability to gain an estimate. Sometimes therefore they are qualified with other indicators, such as complexity. So for example, requirement counting might involve considering the complexity of each requirement using a simple (or if you prefer not so simple) complexity factor of high, medium or low. Here, 1 point is allocated to a low complexity requirement, 2 to a medium complexity requirement and 5 to a high complexity requirement. A similar approach could be applied to the other counts as well.

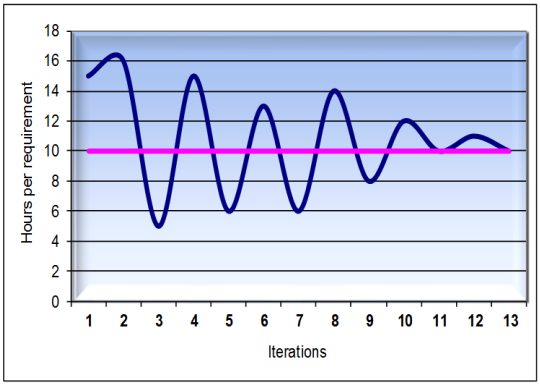
Remember that what is counted is really unimportant so long as it can be counted successively on future developments. So, the actual items being counted could just be a feeling by the development team members, such as what is typically done as part of the sprint planning mechanism in the Agile development methodology. Here, user stories are reviewed by the sprint team members and 'scored' to provide a 'size' (story point estimate) for the requirement (user story), and often this is done just using playing cards to represent an idea of complexity in development, which typically equates, although unscientifically, to development effort. In looking at this method there is really no analytical, empirical or methodical definition to the scoring other than the collective view of the team based on previous experience. In fact, you could just simple say that there is a whole bunch of 'Georges', all throwing their estimates into the pot and the team voting on the best one. This then allows the team

to consider which user stories will be pulled into the sprint that is being planned based on the velocity of the team. The velocity of a sprint team is based on past performance of the team in achieving the number of story points within a sprint. The approach is abstract in nature with the scoring and velocity being personal to the particular sprint team.

Again highlighting that it is an estimate, the resulting size typically includes every activity involved in producing the code or system, e.g. documentation, reviews, configuration management, reporting, management and even degrees of typical or expected rework.

So, we've got a good, and hopefully simple, sizing measure, some previous historical data that relates the size to effort and we can generate a rough estimate of the overall development and even possible a break down to phases, ergo as per the COCOMO method. If we've done another form of estimate, such as a more conventional bottom up estimate from 'George' we can compare the estimates. If they are relatively close all is fine, but if they're widely different, then maybe we need to get everyone around the table and try to understand why. We're not saying either one is right or wrong, just that there might be something that we need to investigate further. Ultimately, we go with the estimate that we deem to be the most reliable, either George's or our size based estimate or even some combination of them both.

One benefit of this initial sizing estimate is that if we routinely monitor it against the actual development being done we might get early indications of 'scope-creep' which might give us an early indication of potential impacts on cost and schedules.




Finally, at the end of the development we can establish the final 'size' and the final effort to gain a view of the final productively measure. Now, this new figure shouldn't simply replace the existing productively measure or we will end up with figures just bouncing up and down. Also, it's only based on estimates and not absolute figures, so it should be used to refine the existing measure rather than replace it. For example, just doing a simply average of the old and new productivity measures will provide a reasonable good final figure to be used in future.

That's pretty well about it, but to summarise, the base practice BP.2 Estimate System Requirement Size in the Requirement Analysis process:

- Is aiming to establish an estimate of the size of the development by which development effort can be estimated based on past performance of the organisation undertaking the development and using a similar development approach as previously. This doesn't inhibit the use of multiple sizing measures related to different approaches or types of development.
- Is in an engineering process because it should be undertaken by the development or engineering teams and not specifically by the project management discipline.
- Has been positioned in a process which is considered to be the latest logical point at which to perform this development estimate, i.e. even if it's not done for project costing purposes, it should be done by the engineering teams to understand any potential risk involved with the overall project estimates that may have been 'factored' for one reason or another.

Finally, just one last thing – it's just an estimate, so try it and see if it does provide some added benefits, and remember 'George' has probably being doing it for years.