The ~~Black~~ Art of Software Estimation Pattern Language

Dmitry Nikelshpur

Developers struggle with estimating time required to complete software development tasks. Software estimates tend to be considerably inaccurate, having significant implications for both development teams and business.

Managers use software estimates to calculate risk, negotiate timelines and budgets with stakeholders, prioritize projects, allocate resources, and gauge developers' performance. Estimators (i.e. developers) are faced with a dilemma: overestimate and risk appearing incompetent, not knowledgeable, or lazy. Even worse, underestimate and risk not delivering on commitments, being behind schedule and over budget, and ultimately risk having the project cancelled. Majority of software estimates are overly optimistic[1].

There is a wealth of information (books, papers, empirical and practical evidence, and tools) available to aid in estimating development activity; however, for many organizations, large and small, Software Estimation is still more a "black art" [7] than a science. We feel and hope that this pattern language will help novice, as well as experienced estimators deliver quick, accurate, confident, and useful estimates. This pattern language can be applied to virtually any development methodology, but it is especially well suited for agile methodologies, the primary objective of which is to reduce risk through embracing and managing (inevitable) change [2, 3, 5, 7, 8, 10].

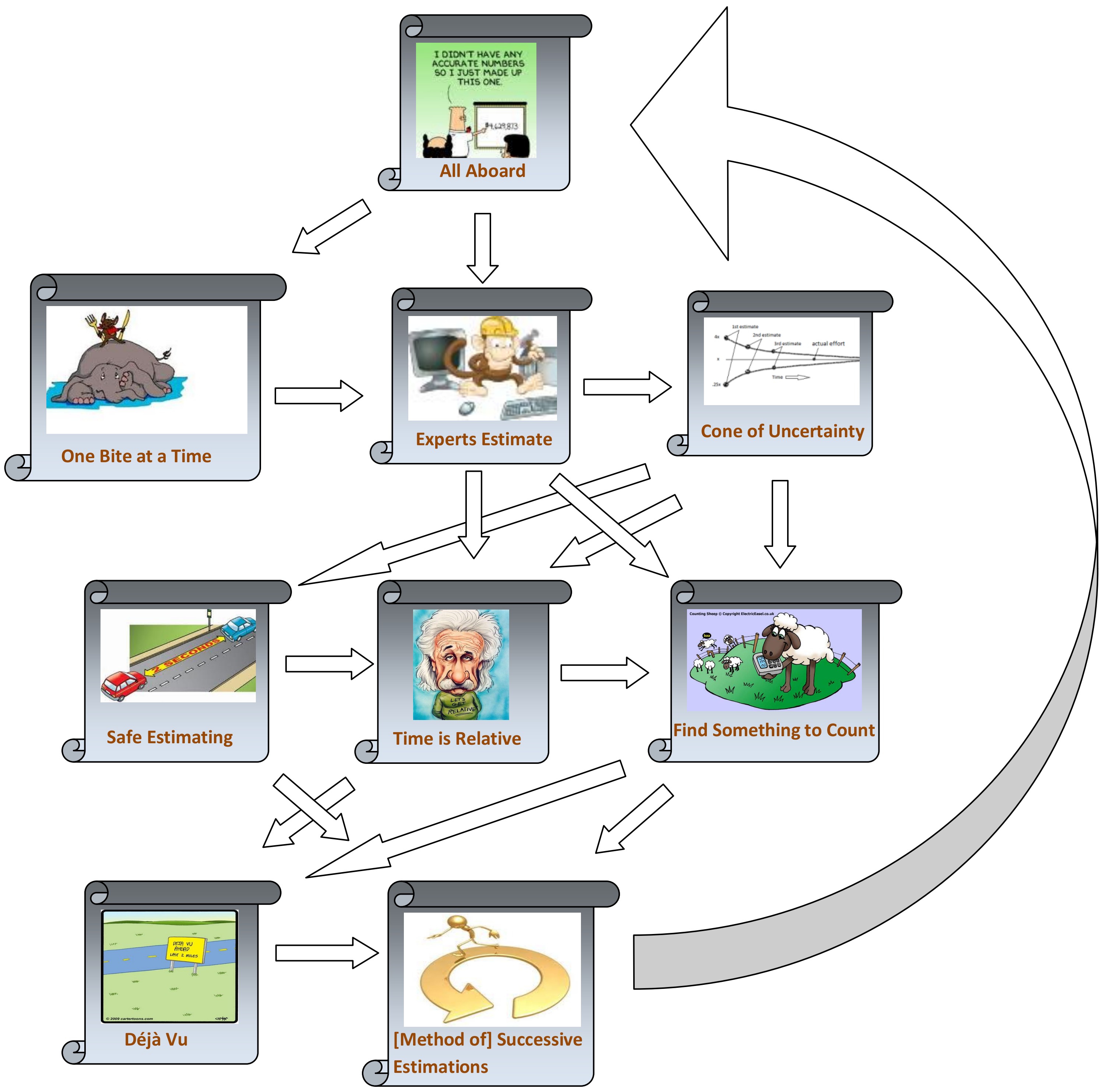
A major predicament occurs when development teams are asked for estimates early in the development lifecycle, usually before development actually begins, when there are still many unknowns and accurate estimates are unlikely . Often stakeholders require estimates that are *just good enough* to be used in high-level planning and budgeting. At other times, the preliminary estimates are misinterpreted as commitments, and development teams are held accountable for meeting these commitments. This pattern language comprises best practices that can aid estimators in providing accurate estimates quickly in both situations. To learn more about the science behind the art of software estimation, we encourage the reader to browse the bibliography section at the end of this paper.

[1] Research shows that overly optimistic estimates are more disruptive, and carry significantly harsher implications then overly pessimistic estimates [7] S. McConnell, *Software estimation demystifying the black art.* Redmond, Wash.: Microsoft Press, 2006.

Overview of the "Art of Software Estimation" patterns.

Pattern	Problem	Solution
Cone of Uncertainty	How do you do a quick, yet reasonably accurate estimate for a project when there is a lot of uncertainty?	Convey the level of uncertainty in your estimates by using a low-high range effort estimate.
One Bite at a Time	How do you achieve an accurate effort estimate for a very large, complex project?	Reduce the error inherent in estimation by breaking up the project into smaller tasks -- estimate the smaller tasks.
Time is Relative	How do you provide an absolute time estimate for a project?	Estimate the relative size of the project. Use a historical log to map the relative size estimate to a time estimate.
[Method of] Successive Estimations	How do you improve accuracy of earlier effort estimates?	Re-estimate remaining effort when you know more about the project.
Experts Estimate	Who should be assigned with providing an effort estimate for a project?	Team members possessing skills required to implement the project should agree on an estimate. Managers and stakeholders should not influence these estimates.
Safe Estimating	How to you capture uncertainty using a single-value estimate?	Use the mid-point value obtained from the "Cone of Uncertainty" pattern. Calculate and append a reasonable buffer to this estimate.
Find Something to Count	How do you provide a highly accurate estimate without using complex tools or methods?	Calculate or count something (function points, number of interfaces, number of components, etc.) from your project's data.
Déjà Vu	How do you improve accuracy of successive estimates?	Keep a historical log of your project's data, including how long it took to complete tasks of various sizes. Use this log for subsequent estimates.
All Aboard	How do you, a manager, deal with developers not being able to provide accurate estimates early in development?	Use effort estimates provided at different stages of the development lifecycle for different purposes.

The Roadmap to implementing the "Art of Software Estimation" patterns[2].



2 Usually (but not always) the more of the pattern language's patterns that are used, the higher the estimate's accuracy.

Cone of Uncertainty

Context:

Stakeholders need to know how long a particular project or modification will take before deciding when, or if, the project should be implemented. Once development commences, these estimates will be used to gauge whether work is progressing on schedule, and to decide whether corrective action is required.

Problem:

You are asked for a quick effort estimate, but you are not certain how long the project will take to implement (e.g. requirements are not yet fleshed out, or you do not know how to implement the project). *How do you do a quick, yet reasonably accurate estimate for a project when there is a lot of uncertainty?*

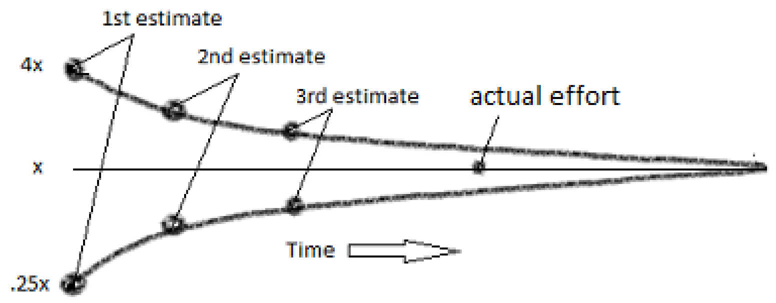
Forces:

- There is often pressure on developers to under-estimate effort required to complete a development task, in order to appease stakeholders and management.
- Software estimation is complex and uncertain by nature, with many unknowns; therefore, developers feel pressure to over-estimate effort required to complete tasks, in order to have a sufficient buffer for any unknowns[3].
- Stakeholders may be looking for estimates that are [just] *good enough* to help them decide whether the project's goals are realistic, fine-tune the project's targets, allocate resources for the project, and control and guide the project to reach its targets.
- Stakeholders or managers may misinterpret estimates as commitments (even estimates that are made early in development, when requirements are not yet fleshed out).
- Stakeholders are often unsure of what they want for the final product (especially early on), and requirements are constantly evolving and changing.

[3] Parkinson's Law kicks in if developers overestimate time required to complete a project: work will expand to fill allocated time (e.g. If a project is estimated to be completed in four days, but it can be completed in two, the developer will find *some* work to fill the remaining two days.

Solution:

Estimate effort using a low-high range. This range will likely be wider early in development to account for greater uncertainty. Re-estimate effort during development milestones (see the ” [Method of] Successive Estimations” pattern).



Discussion:

A very wide estimation range might not be useful or acceptable to management and stakeholders. The “Find Something to Count” and “Experts Estimate" patterns will help you create a grounded estimate that is confined to a reasonable range. Stakeholders and managers commonly request "man-days" estimates. This requirement is often a result of trying to conform to a popular estimation model, or to fit a project planning application that calls for a single-value estimate. If stakeholders refuse to accept a range for the estimate, use the quantity halfway between the low and high values of your range, and add a *reasonable* buffer to your estimate to account for uncertainty (see the “Safe Estimating” pattern).

Resulting Context:

You are now able to create relatively accurate estimates quickly. The size of the project and stage of development will affect your estimate's accuracy. With a large project where development is just starting, you will likely produce estimates that are further from truth than with a smaller project that is close to completion. Accuracy will improve as you re-estimate effort (see the "[Method of] Successive Estimations" pattern), especially if you follow the “One Bite at a Time” “Find Something to Count,” and the “Experts Estimate” patterns.

One Bite at a Time (a.k.a. How to Eat an Elephant?)

Context:

You are asked to estimate effort required to implement a large project. Requirements are incomplete, yet it is already obvious that this project is complex and will require a sizeable effort and time to implement.

Problem:

You are required to provide estimates for a large, complex project, but the many unknowns and long-term assumptions make it difficult to come up with confident estimates. *How do you achieve an accurate effort estimate for a very large, complex project?*

Forces:

- Large, complex projects entail many unknowns, and so are more difficult to estimate accurately than smaller projects.
- Larger tasks take longer to estimate than smaller tasks, using most estimation methods.
- Estimating many small tasks may require more effort than estimating a large project.
- Stakeholders may not be interested in estimates of decomposed smaller tasks, but rather the complete project.

Solution:

Break the large, complex project into smaller, more manageable tasks. Estimate the smaller tasks, and then combine these estimates into a single estimate of the large project.

Discussion:

This pattern takes advantage of the Law of Large Numbers: We are expected to have the same error rate regardless whether we estimate large or small projects. However, if we over- or underestimate a large project, we will have a *large* error *either* on the high or low end of our estimate. If we over- or underestimate many small tasks, some of our estimates will be a little high, and some will be a little low – these errors will tend to cancel out, yielding a much more accurate overall estimate. Ideally, each task should be small enough to be completed in up to several days. At the same time, try to decompose projects into atomic tasks that are useful on their own – doing this will make it possible for stakeholders to recompose tasks into similar, but cheaper projects. When breaking up large projects, you may find that there are temporal dependencies among some of the resulting tasks. Make sure to communicate clearly any such dependencies to stakeholders.

Resulting Context:

Through applying this pattern, you were able to better distribute and reduce estimation error. Nonetheless, larger projects increase the likelihood that requirements will change during implementation, and tomorrow your team may find itself implementing something considerably different from what you are estimating today (use the "[Method of] Successive Estimations" pattern to tackle this concern). You can further improve accuracy of your estimates by implementing the "Experts Estimate,” and the "Find Something to Count" patterns.

Time is Relative

Context:

You are about to embark on a new a development project, and you are asked to estimate how long it will take to implement a project. If this project is large, you have already implemented the "One Bite at a Time" pattern. This project is somehow different from anything your team has implemented before, or perhaps your team composition changed recently.

Problem:

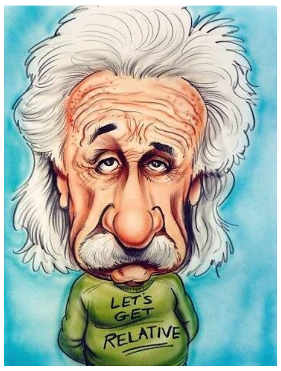
You are asked to estimate time required to complete a project. You need to provide the estimate quickly, but you are unable to rely on historical data because the project is somehow unique. You do not have access to, or otherwise cannot use complex estimation methods or tools -- perhaps you do not have the necessary knowledge (inputs or expertise) to use these methods. *How do you provide an absolute time estimate for a project?*

Forces:

- Developers are poor at estimating absolute time required to complete tasks; people are much better at estimating relative complexity, or rating a task's difficulty relative to other tasks.
- Developers have varying levels of knowledge and abilities, and work at a different pace.
- Stakeholders are usually interested in how long it would take to complete a task.
- You do not have sufficient time, knowledge, or are otherwise unable to use complex estimation techniques or tools.
- Requirements are incomplete.
- Larger tasks are more difficult to estimate accurately (see "One Bite at a Time" pattern).

Solution:

Rate, on a number scale, the perceived effort required to complete the task (including testing and deployment activities). Use the "Déjà Vu" pattern to map your effort estimates to time estimates.



Discussion:

Keep in mind that ratings of perceived effort may differ significantly among developers – people perceive effort differently, and tend to work at different pace. Therefore, effort estimates (and associated time logs) may be considerably different among teams (see the "Déjà Vu" pattern). A good rating scale might include, for example, numbers from one to ten, and a very high number to indicate projects that are too large to estimate accurately given current knowledge (e.g.: 1,2,…,10,80). See the "One Bite at a Time (a.k.a. How to Eat an Elephant)" pattern to tackle projects that are too large to estimate accurately.

Resulting Context:

Through consulting your effort-to-time logs (see the "Déjà vu" pattern), you are now able to provide quick, relatively accurate time estimates. The accuracy of your estimates will improve with time. You can make further improvements by implementing the “Find Something to Count” pattern.

[Method of] Successive Estimations

Context:

You have started development and work is progressing. You no longer have faith in your earlier estimates; based on actual time spent on tasks, it is obvious that your initial estimates were not accurate.

Problem:

You perceive a drift between your earlier estimates and the time already spent working on the project. *How do you improve accuracy of earlier effort estimates?*

Forces:

- There are many unknowns and uncertainties early in development, making it difficult to make accurate estimates; as the project progresses, requirements are fleshed out and technical uncertainties are resolved.
- As the project progresses, your team's abilities and productivity improve.
- Requirements are constantly changing, and you may find that you have estimated something considerably different from what you are implementing today.
- There is value (for business or development management) in re-estimating the projects.
- Stakeholders or managers may not be willing to accept, or may not have a use for re-estimation.
- Re-estimating multiple times may cost more (i.e. add up to more time, effort, and resources) than estimating once.

Solution:

Re-estimate remaining effort at regular, moderately spaced intervals.

Discussion:

With each interval you will have learned more about the project you are implementing and the amount of work your team can complete in a given period. Consequently, each successive estimate will be more accurate than the previous. Shorter intervals will cause you to re-estimate more frequently, while longer intervals will yield less accurate estimates. If you are implementing the "Déjà Vu" pattern, after several estimation sessions you will have a log of how long it takes your team to complete projects of a given relative complexity. This pattern works especially well with agile methodology implementations such as XP, Scrum, and Crystal.

Resulting Context:

An obvious outcome of implementing this pattern is that you will periodically re-estimate effort (as opposed to making just one estimate at the onset of a project). As you re-estimate, at intervals that make the most sense given your context, you will gain a progressively more accurate view of effort and time required to complete the project.

Experts Estimate

Context:

You are asked to provide an estimate for a project. The team is comprised of people with various responsibilities: managers, analysts, architects, senior developers, junior developers testers, trainers, technical writers, etc.

Problem:

Your team is required to come up with an estimate for a project, but you are not sure which team member is the most capable to be tasked with that responsibility. *Who should be assigned with providing effort estimates for the project?*

Forces:

- Developers feel pressure to provide an overly optimistic estimate to appease managers and appear knowledgeable and capable.
- Developers feel pressure to pad estimates in order to make sure they meet commitments.
- Managers may be compelled to pressure estimators to either pad or reduce estimates in order to mitigate perceived risks.
- Senior technical team members (or members who are more familiar with the project and technologies used) can usually implement requirements in less time compared to more junior team members.
- Senior technical team members may not be the ones implementing the projects being estimated.
- Junior team members may be less confident and less skilled at software estimation than senior team members.

Solution:

Technical team members who will, or could, implement the project should agree on the estimate.

Discussion:

When estimators disagree, they should explain and discuss the reasons for their estimates. They will usually come to a consensus. However, to limit the amount of time it may take to reach a consensus, estimates of developers who will be doing the work have greater weight. If an agreement is not reached within a reasonable timeframe, the higher estimate is chosen. Anyone outside of this group of estimators (i.e. managers, analysts, and stakeholders) should not interfere with the process nor modify the resulting estimate. The estimators should only change the estimate (using the same semi-democratic procedure) because of changing requirements or new knowledge about the project.

Resulting Context:

The estimate is grounded in developers' prior experiences, education, and knowledge. Therefore, it will be a good deal closer to the truth than relying on intuition alone, but it might still be inaccurate. The estimate's accuracy may be further improved by implementing the "Find Something to Count" pattern.

Safe Estimating

Context:

You are required to provide estimates for a project. You had already implemented the "Cone of Uncertainty" pattern, but the stakeholders (or project management software) insist on a single-value estimate. Therefore, you are using the value at the mid-point between the low and high ends of the range obtained using the "Cone of Uncertainty."

Problem:

You are asked to provide a single-value estimate for a project, and you want to capture uncertainty in your estimate without grossly overestimating. *How do you capture uncertainty using a single-value estimate?*

Forces:

- In using the mid-point value obtained with the "Cone of Uncertainty" pattern, you are not accurately capturing uncertainty inherent in your estimate.
- The higher the buffer that is added to an estimate, the greater the chance of completing the project on schedule.
- A very lofty buffer will make your estimate unreasonably high. It may appear that you are "padding" your estimate -- stakeholders may get the impression that your team is not competent and may lose confidence in your ability to deliver.
- A buffer that is too low may yield an overly optimistic estimate – quality will suffer, schedule will slip, and the project may be cancelled.

Solution:

Calculate a buffer based on perceived probability of risk, impact (time), and your risk aversion. Append this buffer to the mid-point between the low and high range of your Cone of Uncertainty estimate.

Discussion:

Developers sometimes include buffers in their estimates, so make sure that you are not adding a buffer to an estimate that already includes a buffer – doing so will result in a grossly pessimistic estimate. Likewise, if you add a buffer to the high end of the estimation range, your resulting estimate will also end up overly pessimistic. Adding a buffer to the low end of the range will not capture all uncertainty.

A potential way to implement this pattern is to use the standard deviation of your estimation range to calculate the buffer. For example, assuming normal distribution of your Cone of Uncertainty estimates, and that your high-end estimate is at around 95% certainty:

```
SD = (Worst – Avg) /2
Buffer = Avg + SD * Aversion
Single-Value Estimate = Avg + Buffer
```

Where *Worst* is the value at the high end of your estimate, *Avg* is the mean value (midway point between Low and High estimates on Cone of Uncertainty), and *Aversion* is a constant reflecting your risk aversion[4].
Note that a buffer may not be needed if the project is very small, or if accurate estimates are not required.

Resulting Context:

You now have a single-value estimate that captures your estimation uncertainty and provides a buffer that is not overly pessimistic. Nevertheless, the estimate, especially if it is made early in the development lifecycle, is unlikely to be very accurate. See the "[Method of] Successive Estimations" pattern.

[4] Setting the *Aversion* constant to 2 will capture around 95% confidence in your single-value estimate, while setting this constant to 1 will translate to approximately 70% confidence.

Find Something to Count

Context:

You are about to start working on a new project, or development has already commenced, and you need to provide a particularly accurate effort estimate to management or stakeholders. You are not able to use automated tools or complex estimation methods. You have already implemented the "One Bite at a Time" and the "Experts Estimate" patterns.

Problem:

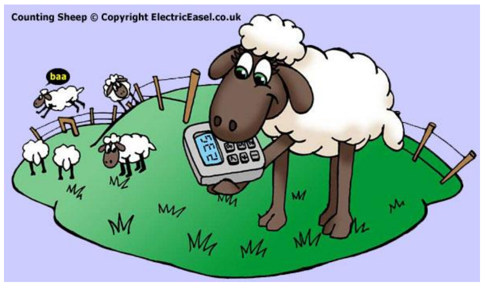
You are required to provide a highly accurate effort estimate for a project. *How do you provide a highly accurate estimate without using complex tools or methods?*

Forces:

- There are more unknowns at the onset of a project, and visibility generally improves as the project progresses.
- Similar work, on average, tends to require similar effort to complete, given that the same people work on the project.
- A requirement for a very accurate estimate may frighten estimators, causing them to pad and excessively overestimate effort.
- Different teams may require different amount of effort to complete a similar project.
- A historical log with effort required for the team to complete similar projects in the past may not be available.
- An industry's average effort required to complete similar projects may not resemble well your team's unique situation.

Solution:

Find something to calculate or count (if at all possible from the current project's data) in order to arrive at the effort estimate. Things that can be counted, and that are correlated well with effort include the number of function points, the number of GUI elements, the number of components, and the number of lines of code. If data about the current project are not yet available, use data about similar projects (preferably one of your team's projects, if available), but switch over to using the current project's data as soon as feasible.



Discussion:

As Winston Churchill once put it, "facts are better than dreams." An off-the-cuff estimate that is not grounded, even if it relies on intuition of senior, experienced developers, is not likely to be accurate. High degree of accuracy and confidence may be achieved by creating several estimates, which rely on counting or computing different measurements, and looking for convergence.

Resulting Context:

Following this pattern, you are able to create estimates that are much closer to truth than those made by intuition of even the more seasoned professionals. However, the fact remains that there is less knowledge available early in the project lifecycle, and so it is likely that your estimates will be less accurate earlier in development (see the "[Method of] Successive Estimations" pattern). Stakeholders need to be aware that you will be able to provide successively more accurate estimates as time progresses (see the "All Aboard" pattern).

Déjà vu

Context:

You are progressing on a project, and you need to provide successively more accurate estimates of work yet to be completed. You are already implementing "Time is Relative" or "Find Something to Count" patterns, and are planning to apply the "[Method of] Successive Estimations" pattern.

Problem:

You are required to provide successively more accurate estimates as the project progresses. *How do you improve accuracy of successive estimates?*

Forces:

- Similar tasks performed by the same team will likely require similar effort.
- The team might not have recorded effort required to complete tasks of varying complexity.
- It takes work and time to document effort expanded to complete a task.
- It might not be directly apparent when a task was started, when it was completed, nor the number of man-hours expanded to complete the task.
- Many variables that influence effort required to complete a task (development methodology used by the team, team composition, developers' abilities, etc.) are dynamic.

Solution:

Keep a historical log of how long it takes your team to complete tasks with various effort estimates. Also, record significant measurements such as the number of function points, components, GUI items, and lines of code. Use this log to guide future estimates.

Discussion:

"History repeats itself" -- an estimate that takes past performance into account is likely to be more accurate than an estimate that does not. However, different teams work at a different pace, and different people tend to perceive effort required to complete a task differently. Therefore, use others' historical performance only in early estimates, and only if your team does not yet have a historical log of its own. Make an effort to start using your own performance logs in your estimates as soon as possible. Make sure to keep your logs current. By keeping the logs current, you will be able to keep track of your team's velocity, and to keep the accuracy of your estimates high. For best results, the estimated features should be right-sized -- not too large, not too small (see the "One Bite at a Time" pattern). This pattern works best when applied along with "Time is Relative" and "[Method of] Successive Estimations" patterns.

Resulting Context:

Once you start using your own team's historical logs, you will be able to map perceived effort estimates to actual time required to complete a feature with good accuracy. It will take some effort for your team to keep the logs current. The accuracy of your records, and diligence in keeping them current, will subsequently affect the accuracy of your estimates.

All Aboard

Context:

You are a manager requiring an effort estimate for a project from the development team. You need effort estimates to get a sense of how long the project will take to implement, estimate costs, prioritize work, and to negotiate budget and timelines with stakeholders. Development has not yet commenced, or has just started, and requirements are still being fleshed out.

Problem:

It is still early in development (or perhaps development has not yet started) and the team does not yet have sufficient information to provide accurate time estimates. *How do you, a manager, deal with developers not being able to provide accurate estimates early in development?*

Forces:

- The stakeholders need estimates that are accurate enough to be used in projections, budgeting and controlling the project.
- You, the manager, are expecting an absolute time estimate (you may not know how to use unconventional estimates).
- You feel uncomfortable providing your managers or stakeholders with estimates that are not absolute, or that are appreciably different from their expectations.
- There are still many unknowns and ambiguities. Developers do not have sufficient information to provide an accurate absolute estimate at this time.

Solution:

Management should use estimates provided at different stages of development for different purposes (see the "Cone of Uncertainty," and "[Method of] Successive Estimations" patterns).

I DIDN'T HAVE ANY ACCURATE NUMBERS SO I JUST MADE UP THIS ONE.
$4,629,873

Discussion:

The early estimates (produced while requirements are still vague and there are many unknowns) may be used to assess the scope of the project, work out high-level budgets, or decide whether the project is too large to fit within the allotted timeframe. As the project progresses, you can use the increasingly accurate estimates to validate whether earlier estimates were realistic, control and fine-tune the project plan, and commit to a delivery timeframe that you, the stakeholders, and the developers are comfortable with.

Resulting Context:

If you, a manager, need to provide effort estimates to your superiors, success of the "All Aboard" pattern depends on whether they, in turn, are willing to implement this patter. Successful implementation of this pattern will allow you to take full advantage of the "Cone of Uncertainty", "[Method of] Successive Estimations", and the "One Byte at a Time" patterns.

Acknowledgements

I would like to express sincere gratitude to Dr. Joe Bergin, without whose guidance and insightful comments, this pattern language, and this paper, would not have been possible. Dr. Bergin was the professor for the patterns class for which this paper was a deliverable. I would also like to thank Ademar Aguiar for taking time out of his busy schedule to shepherd this paper for the PLoP 2011 conference. Ademar's keen observations and welcomed suggestions helped improve this paper significantly.

Bibliography

[1] C. Alexander, *The timeless way of building.* New York: Oxford University Press, 1979.

[2] K. Beck, "Manifesto for Agile Software Development", http://agilemanifesto.org/, 2001.

[3] K. Beck and C. Andres, *Extreme programming explained : embrace change.* 2nd ed., Boston, MA: Addison-Wesley, 2005.

[4] F. P. Brooks, *The mythical man-month : essays on software engineering.* Anniversary ed., Reading, Mass.: Addison-Wesley Pub. Co., 1995.

[5] A. Cockburn, *Agile software development : the cooperative game.* The Agile software development series, 2nd ed., Upper Saddle River, NJ: Addison-Wesley, 2007.

[6] M. L. Manns and L. Rising, *Fearless change : patterns for introducing new ideas.* Boston: Addison-Wesley, 2005.

[7] S. McConnell, *Software estimation demystifying the black art.* Redmond, Wash.: Microsoft Press, 2006.

[8] M. A. Parthasarathy, *Practical software estimation function point methods for insourced and outsourced projects.* Upper Saddle River, N.J.: Addison-Wesley, 2007.

[9] K. Schwaber and M. Beedle, *Agile software development with Scrum.* Series in agile software development, Upper Saddle River, NJ: Prentice Hall, 2002.

[10] A. Shalloway, G. Beaver and J. Trott, *Lean-agile software development : achieving enterprise agility.* The net objectives lean-agile series, Upper Saddle River, NJ: Addison-Wesley, 2010.

Contents

The Black Art of Software Estimation 1

Overview 2

The Roadmap 3

Cone of Uncertainty 4

One Bite at a Time (a.k.a. How to Eat an Elephant?) 6

Time is Relative 8

[Method of] Successive Estimations 10

Experts Estimate 12

Safe Estimating 14

Find Something to Count 16

Déjà vu 18

All Aboard 20

Acknowledgements 22

Bibliography 23