altexsoft
software r&d engineering

WHITEPAPER

Estimating Software Engineering Effort: **Project and Product Development Approach**

Estimates Everywhere

How much did it take you to get to work today? Are you sure that it'll take the exact same amount of time to get there tomorrow? Probably not. Depending on your route, the traffic, weather or dozens of other circumstances, the required time might vary. For a short distance the difference might be insignificant. But if you have to travel to another city or country, it can grow exponentially.

Yet, your commuting time is your own business. In any professional activity, where a number of other people or processes depend on your ability to accomplish your tasks in a timely manner, the accurate time/effort estimations are far more crucial.

With the global **contract value ranging from** $63.5 billion to $159.1 billion, according to the different sources, software engineering is one of the top services to be handled by outside contractors. In this case, knowing the estimated time/budget prior to the start of any cooperation is crucial for businesses. Otherwise hiring a contractor or an agency without a clear understanding of the cost-benefit would be a matter of absolute trust.

At the same time, software engineering is a complex area of knowledge. It often requires extensive research and out-of-the-box solutions. Thus, making any assumptions as for the effort or duration of one or the other engineering task is quite risky.

As a Technology & Solution Consulting company, we conduct on average 30-40 estimates per month which totals **up to 500 estimates per year**. With such vast experience in this area, we have developed our own approach to quoting software engineering efforts. In this paper we cover all aspects of the software development cost estimation process and the techniques we typically use. But first, let's define the role of the estimates in the software development life cycle.

1. Conventional Approach: Software Development Life Cycle

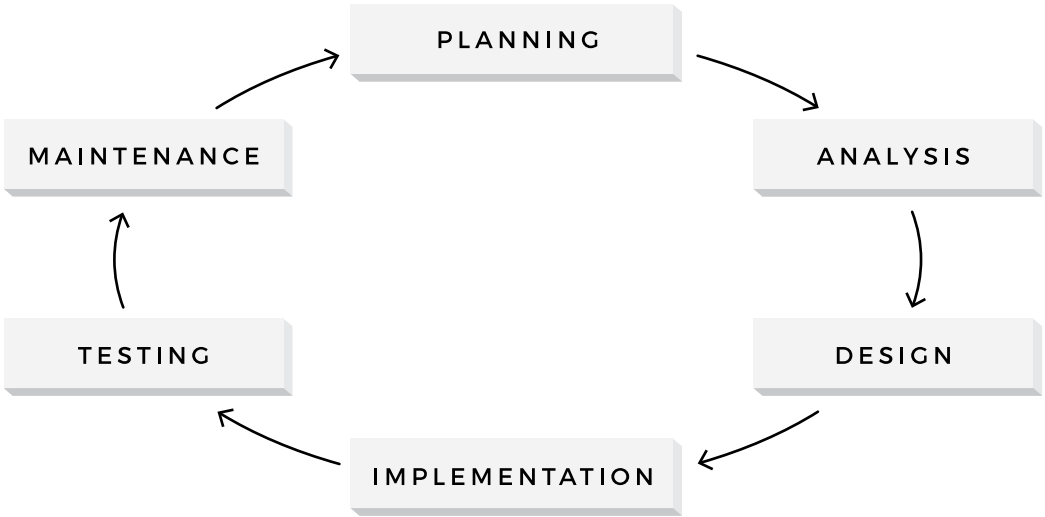
Over the last couple of decades, the software systems have been getting increasingly complex. At the same time there is barely any industry or area of knowledge that hasn't been experiencing the impact of technology. As a result, there was a need to formalize the software development process and determine a common model for its life cycle management.

Software Development Life Cycle (SDLC) was the first formal project management framework, used to define the major stages and tasks within a software development process.

In his book, *"Global Business Information Technology: an integrated systems approach"*, Geoffrey Elliott described the main purpose of the methodology as:

> *"...to pursue the development of information systems in a very deliberate, structured and methodical way, requiring each stage of the life cycle from inception of the idea to delivery of the final system, to be carried out in rigidly and sequentially."*[1]

Software Development Life Cycle



Traditionally, the Software Development Life Cycle includes **6 stages**:

Stage	Activity	Key Roles	Deliverables
Planning	Preliminary requirements' analysis, research, basic project vision and scope.	Customer, sales representatives, business analysts	Understanding the customer needs; basic project roadmap and technical recommendations.
Analysis	Identifying the project goals and functionality, finalizing the technical specifications, requirements and finding solutions to potentially challenging issues.	Customer, development team (business analysts, technical experts, project managers)	Complete functional and design specifications, Work Breakdown Structure (WBS), rough cost estimate.
Design	Creating basic system architecture and visual design (UI/UX).	Development team (architects, UX and UI experts, project managers)	Draft system architecture, final product design, and revised estimate.
Implementation	Software development process.	Development team (software engineers, architects, project managers)	Full-featured functioning software product.
Testing	Quality assurance process.	Development team (software engineers, QA engineers, project managers), Customer	Finalized software product of the required quality.
Maintenance	Deployment, support and updates.	Development team (software engineers, project managers)	Up-to-date software product.

Providing a roadmap on how the project is planned and managed from start to an end, the original SDLC formed the basis for a number of software development methodologies used today. The approach that most fully complies with the given step-by-step process is the **waterfall model**.

This model works best with well-defined projects that have clear requirements and a relatively small scope of work with low complexity. Such projects are typically used to solve a secondary business problem or automate a certain internal task. It might be a landing page or a simple tool that streamlines a certain business process. In this case a fixed price collaboration model is possible: The effort is specified and outcomes are predictable.

Being a simple and straightforward approach, the traditional SDLC still has a number of downsides. One of them is the huge uncertainty that occurs at the early stages of the software development.

2. A Flaw in the Theory: Uncertainty in Project Estimation

Eight out of 13 of the most famous failed projects, state cost overrun and delays in delivery as major problems that led to the failure. It means that more than 60% of the project success depends on meeting the cost and time estimates, provided by the engineering team.

Similarly, Fred Brooks noted in his all-time classics, The Mythical Man-Month: Essays on Software Engineering: "*More software projects have gone awry for lack of calendar time than for all other causes combined.*"[2] In his opinion, this is due to the fact that "*...all programmers are optimists. ... the first false assumption that underlies the scheduling of systems programming is that all will go well, i.e., that each task will take only as long as it "ought" to take.*"[2]

Aside from being optimistic, engineers are often under a lot of pressure: After all, they bear the responsibility of the unrealistic estimates that result in a delayed project. Like it or not, the estimates, even the "ballpark" ones, end up being considered commitments in the client's mind. Experienced engineers have an unwritten rule: when forced to provide a quote, make a guess about the amount of time and effort the work might actually take and double it. Yet, even these preventive measures prove to be useless when estimates are made too early in the process. Without proper documentation and detailed project requirements it's impossible to make any accurate guess.

To make a realistic estimate one should generally consider:

- **Detailed specifications** – The more information you have on the scope of the project and the desired outcomes the better.
- **Graphic design** – Complex UI elements usually require more engineering effort and take longer to implement.
- **Technology stack** – Depending on project specifics, the team might need to use complex tools, third-party APIs or even find custom

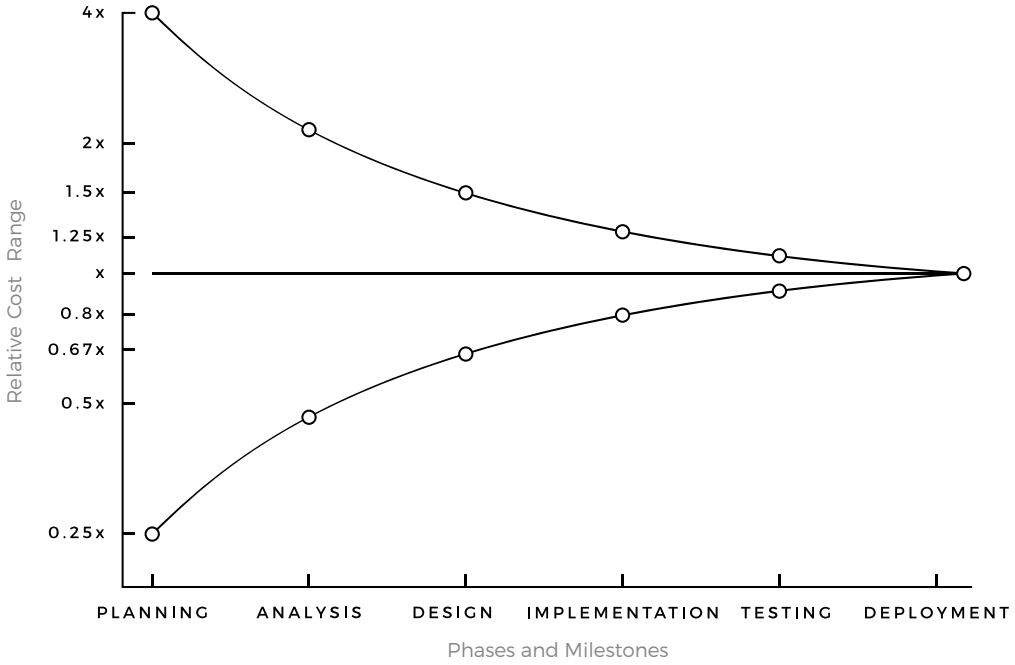
solutions to some problems. Thus, an estimate needs to cover the research or the learning curve involved.
- **The experience and personality factor** – What takes a senior software engineer an hour to implement might take a trainee several days. Therefore, estimates should be tailored to the team that will work on the project.

And this is only the tip of the iceberg. While a requirement in project documentation might seem straightforward, it is never too detailed. A scope of a simple user story "**As a User I want to login into the app**" can vary greatly:

- What type of a login should we implement (email and password login or social networks login and if yes, which ones)?
- Should the fields have any restrictions (maximum number of characters, type of characters, password strength requirements)?
- How should the input fields detect and handle errors (invalid email, password, etc...)?
- Is the "Remember Me" option needed? If yes, for how long should the information be stored?
- Which password recovery option should we use?

That's a lot of questions for a simple and straightforward feature! And those are only the functional requirements. There are even more technical details: testing, documentation writing, code review and refactoring.

As one can clearly see, it is practically impossible to define the scope of work early in the process. Usually, the clearer the project requirements become, the more accurate the quote will be. A concept that perfectly presents this phenomenon is the Cone of Uncertainty. It was introduced by Barry Boehm in his book Software Engineering Economics (1981) and then developed further by Steve McConnell in "*Software Project Survival Guide*" (1997).



The Cone of Uncertainty

According to this example of the Cone, represented by the chart above, the highest level of uncertainty is typically observed early in the process (planning stage). At this point the estimate variability might range from 4x to 0.25x. This means that if a project is estimated to take a month, it could actually end up taking from 1 week to 4 months.

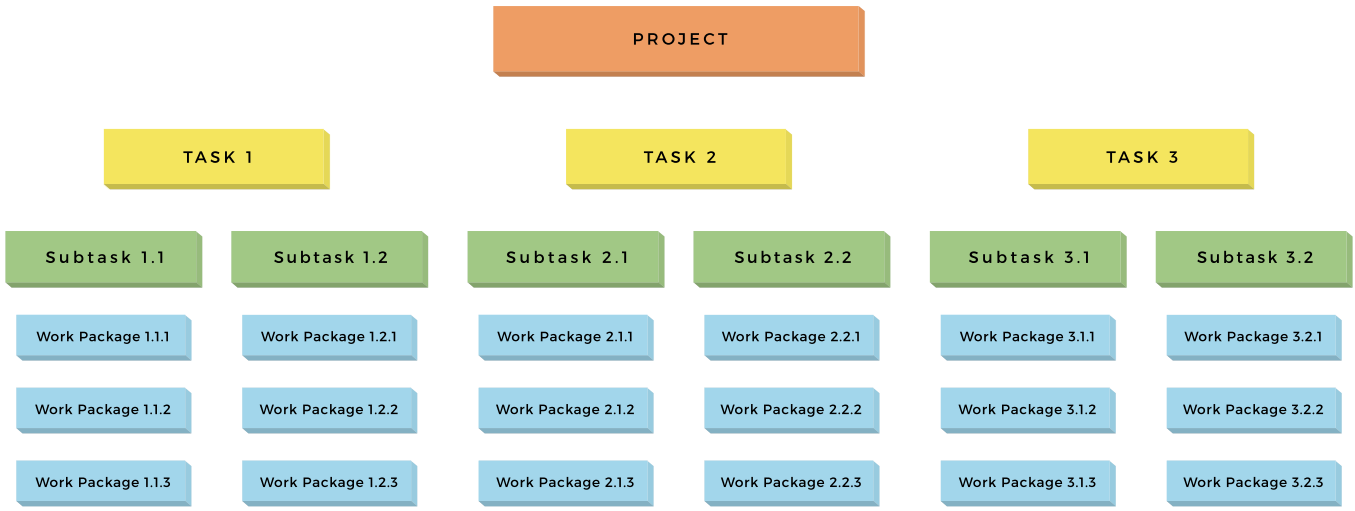
However, the degree of uncertainty decreases as the project progresses. Through the analysis and design stages, the team might reduce the variability of the estimate. In theory, having the specifications and design at hand might decrease the average deviation for a 1-month project estimate to less than one week. The exact duration of the project might remain unclear until the software is deployed.

The examples provided above explain how the Cone of Uncertainty principle should work in theory. Based on our practical experience, the range of deviation often depends on the scope of the project and may vary accordingly. Yet, one thing remains clear: the estimates made early in the process are often referred to as "guesstimates", they rarely prove to be of any value in the long run. Software engineering estimations require a far more rigid approach.

3. Seeking a Compromise: Estimation Methods and Tools

3.1 Mapping out Work Breakdown Structure

As mentioned above, the requirements and project roadmap are usually finalized at the planning/ analysis stage. They serve to minimize the uncertainty of the software development estimation. Thus, Andrew Stellman and Jennifer Greene, bestselling O'Reilly authors and Agile coaches, in their book Applied Software Project Management (2005) state: *"A sound estimate starts with a work breakdown structure (WBS)."*[3]



Work Breakdown Structure - sample

In the PMI Project Body of Knowledge®, WBS concept is defined as *"..a deliverable-oriented hierarchical decomposition of the work to be executed by the project team to accomplish the project objectives and create the required deliverables."*[4]

Often compared to a backlog in Agile methodologies, this document breaks down the project into measurable and manageable deliverables. The WBS focuses on the deliverables and their visual representation, while the backlog is centered around the so-called user stories and features. Being one of the artifacts in the Scrum methodology, backlog is often referred to as an Agile WBS. Both of them are living documents, constantly "groomed" in the process of the development.

3.2 The Units of Measure in Software Development Estimation

With a WBS at hand, engineers might be able to provide an elaborated estimation of the efforts needed to build a software product. Yet, there is another aspect of the problem: How can this effort be measured?

Some teams use relative terms, such as story points, function points or even T-shirt sizes. Others prefer more substantial units, estimating the software engineering effort in man-hours/days/weeks/months.

Yet, for business-oriented customers these numbers don't make any sense. When negotiating with the potential contractors or discussing the project with the in-house engineering team, they expect to be told when the product will be ready and how much it would cost. Thus, the effort estimation is usually translated into hours/days/weeks/months and the cost is calculated accordingly.

3.3 Software Development Estimation Techniques

Depending on the project management methodology that will be used in the process, the most common estimation techniques are divided into Traditional (usually applied to waterfall method) and Agile.
Klaus Nielsen in his article "Software Estimation using a Combination of Techniques" provides the following classification of the most common software engineering assessment techniques[5]:

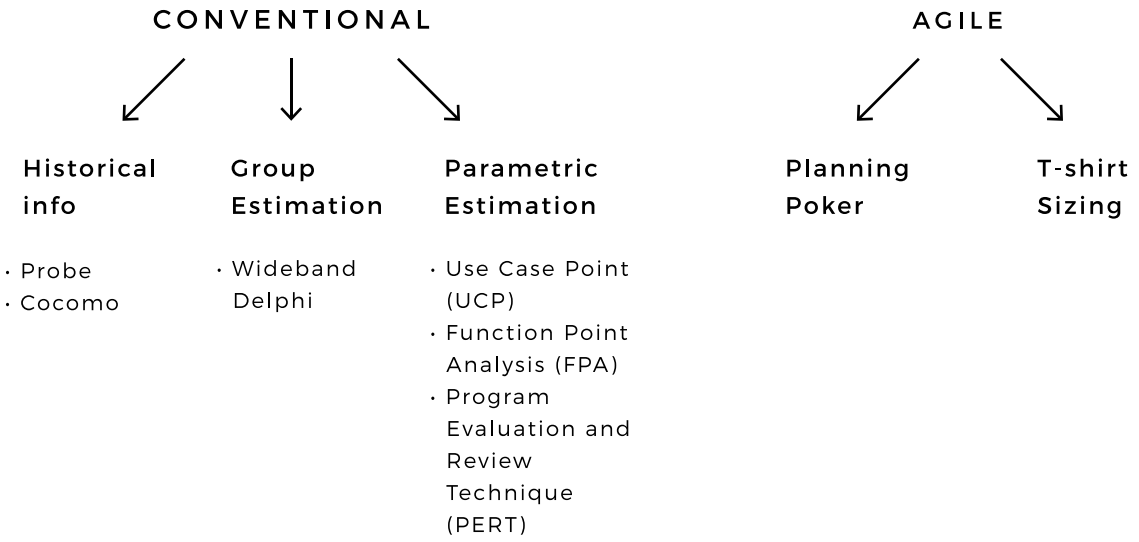
Traditional

- Analogy
- Effort method
- Programming techniques (lines of code)
- Expert Judgment
- Delphi/Wideband Delphi
- Program Evaluation and Review Technique (Beta or normal)
- CoCoMo 1/CoCoMo 2
- Albrects Function Points
- Parametric methods (e.g., PRICE)
- Various top-down or bottom-up techniques

Agile

- Relative sizing
- Wideband Delphi
- Planning poker
- Affinity estimation
- Ideal time and elapsed time
- Disaggregation
- Bottom-up/top-down estimation
- Constructive agile estimation algorithm
- Time, budget, and costs estimation

Another approach to this classification can be found in IBM developerWorks knowledge library[6]:



3.4 Handling Project Estimations at AltexSoft

We at AltexSoft have developed a unique approach. Namely, there are three basic methods we use when estimating a typical waterfall project:

- Analogy-based estimation predicts the required effort based on the information from former similar projects or features.
- Statistical method uses statistical data about the previous experience in the field in general.
- Expert judgement is used when the in-house team does not have the experience building similar features, or the project implies use of the latest or industry-specific technology solutions.

Our standard process starts with a high-level project vision finalization. This document is then further transformed into the Work Breakdown Structure. The proposed solution is broken down into smaller modules and features, to make sure that we are on the same page with our client regarding the scope and functionality of the software product. Every feature within a module is analyzed and estimated separately, using one or a combination of the following methods: Analogy-based estimation, Statistical method, Expert judgement.

As a result, we get an elaborated document with the detailed breakdown of the scope of work and estimated duration of each task. Minimum and maximum values define the possible variability range.

Project Summary					
Module Name	**Feature name**	**Sub-feature Item**	**Description**	**Development estimate (m/d)**	
				min	**max**
Task Management App			Native Android app		
	Mobile app core infrastructure				
		Android app bootstrap		1	2
		Integration with Google Maps		3	4
		General UI Layout		2	3
		Backend API integration		3	4
		Push notifications		1	1.5
		Social Networks integration	Facebook, Google+	2	4
		Google Calendar integration		2	3
		Task synchronization	Automatic and manual sync between devices of one account	1	1.5
		Calendar synchronization	Sync of tasks into/from Google Calendar	3	4

This is a draft estimate for a hypothetical project. The numbers on the right denote the expected effort, calculated in man-days.

Having estimated each separate feature, we can now provide a quote for the whole project by summing up the duration of all tasks.

Total Features Implementation	min	max
	77.5	112

Yet, this estimate includes only the time for feature implementation. If we add other related activities, such as documentation writing, UX/UI design development and implementation, QA and communication, we will have a more realistic vision of the project scope and duration.

Project Summary		
Code Review & Refactoring	4	6
Bug-fixing	8	12
Feedback Processing	8	12
Deployment	1	2
Total Additional Development Tasks	**min**	**max**
	21	**32**

Software Engineering and Infrastructure Tasks		
Software Architecture Design	2	3
Database Implementation	1	2
Project Infrastructure/Environment Setup	1	2
Contingency	16	23
Total Software Engineering and Infrastructure Tasks	**min**	**max**
	20	**30**

Additional Project Activities		
UX Design	14	18
Graphics Design	16	24
Project Stabilization	10	20
Quality Assurance	29	42
Total Additional Project Activities	**min**	**max**
	69	**104**

Project Management and Analysis		
Requirements analysis	10	15
Team communication	10.5	14
Project management	30	44
Total Project Management and Analysis	**min**	**max**
	50.5	**73**

Adding these results to the development estimate we will get the final quote for the whole project.

Planned resources				
	Min man/days	**Max man/days**	**Quantity**	**Position Title**
Total Development	133	200	3	Software Engineers
Total QA	30.5	44	1	QA Engineers
Total Data Science	0	0		Data Science Engineers
Total UX Design	15.5	20	1	UX Designers
Total Graphics Design	17.5	26	1	Graphics Designers
Total UI Implementation	0	0		UI Engineers
Total BA	11.5	17	1	Business Analysts
Total PM	30	44	1	Project Manager
Calendar Duration Full Week	**14**	**17**		

While several activities, such as QA and project management are conducted in parallel to the main development activities, they do not normally increase the calendar duration of the project. In addition, the total estimated scope of work is provided in man-days and does not equal the actual calendar duration.

Yet, as we can see, using detailed work breakdown and dedicated methodologies is still no panacea: There is still some variability between the minimum and maximum estimated scope. That is why we have developed another approach to handling complex projects.

4. Product Engineering Approach

As Barry Boehm wrote in his book "Software Engineering Economics":

> *"Whatever the strengths of a software cost estimation technique, there is really no way we can expect the technique to compensate for our lack of definition or understanding of the software job to be done."*[7]

This is why we have come to using a more solid approach to building software solutions - The product development model.

Building complex software products from scratch requires more flexibility and a long-term dedicated team effort. The requirements as well as the whole business model might largely evolve during the development process, so extensive investment in research and detailed planning at this stage is simply unfeasible. That is why this approach finds its best application when a customer request goes far beyond a trivial engineering task, when there is a need to build a product that plays a major role in the client's business. For instance, this model was successfully used by our team when building a booking portal for an online travel agency or a clinic management platform for a practicing physician.

While traditional SDLC presupposes a project-based collaboration, based on the waterfall approach, product engineering mostly complies with agile methodologies. Therefore, instead of investing 30-40% of all project time into planning and analysis, we focus on high-level requirements and build up a dedicated team, hand-picked specifically to best fit the client's needs, that will handle the development. After that, a typical agile process takes place: We deliver functioning product builds iteration by iteration. We limit the sprint duration to 2-4 weeks to make the progress measurable and to deliver predictable results.

Being agile at its core, this approach cannot be limited by deadlines or commitments from the very beginning, that's why a Time and Material model is the best option. Instead of making total product estimates, we provide quotes for a limited set of tasks, one iteration at a time, using agile software estimation techniques:

- Planning Poker
- Relative Sizing
- Ideal Time & Elapsed Time

Planning Poker is a gamified agile estimation and planning technique. Instead of speaking estimates aloud, the team members use a deck of cards. Estimating every item out of the list of features proposed for an iteration, each of the team members puts a card that denotes how long the task will take in his/her opinion. After that, the team discusses each proposed estimate and agrees on one option. This method eliminates the cognitive bias, allowing every team member to make an impartial judgement, based on his/her own opinion.

In **Relative sizing estimation** the team agrees on a feature or task that would serve as a measurement standard. All the further items will be estimated based on how they compare to this standard.

Ideal Time & Elapsed Time technique is based on the assumption that instead of working an "ideal" 8-hour day, most of the engineers will spend about 6 hours a day fully focused on the task. Taking into account this focus factor, the team will estimate a 24-hours task as the one that will be finished in 4 days instead of 3 days.

According to the Standish Group 2015 Chaos Report, only 3% of large projects based on waterfall approach turn to be successful, whereas the success rate for agile project of the same size is 18%.[8] For reference, the success rate for small waterfall projects is 44%. This perfectly explains why we encourage our clients to choose the product development approach.

#NoEstimates

Based on the concept of reducing waste and Just-in-time development, the key principles behind some of the most popular Agile methodologies, Vasco Duarte in his book #NoEstimates says: *"People who find value on estimates are just addicted to a practice (estimation) to get something they find valuable: plans, comfort, uncertainty reduction, financial projections, sales proposals... But, again, these are not customer value. [...] So they immediately fall under the label of waste."*[9]

Still, being able to predict and plan your expenses in advance might be vitally important for your business. The best way to go about this dilemma is to think of any estimate as of an assumption, not as something that is set in stone. After all, deadlines and budgets may change. What doesn't change is the value and advantage that you gain with high-quality software.

References

1. http://www.amazon.com/Global-Business-Information-Technology-integrated/dp/0321270126
2. http://www.amazon.com/Mythical-Man-Month-Software-Engineering-Anniversary/dp/0201835959
3. http://shop.oreilly.com/product/9780596009489.do
4. http://www.workbreakdownstructure.com/index.php
5. https://books.google.com.ua/books/about/Software_Estimation_Using_a_Combination.html?id=JU1grgEACAAJ&redir_esc=y
6. http://www.ibm.com/developerworks/library/d-estimation-agile-or-conventional-trs/index.html
7. http://www.amazon.com/Software-Engineering-Economics-Barry-Boehm/dp/0138221227
8. https://www.infoq.com/articles/standish-chaos-2015
9. http://noestimatesbook.com/

About AltexSoft

AltexSoft is a Technology & Solution Consulting company co-building technology products to help companies accelerate growth. The AltexSoft team achieves this by leveraging their technical, process and domain expertise and access to the best price-for-value Eastern European engineers. Over 100 US-based and 200 worldwide businesses have chosen the company as their Technology Consulting Partner.

US Sales HQ

701 Palomar Airport Road,

Suit 300, Carlsbad, CA 92011

+1 (877) 777-9097

Global HQ

32 Pushkinskaya Str.,

Kharkiv, Ukraine 61057

+38 (057) 714-1537