Managing "Size Creep" in Software Development Projects

Robert D. Leinen Jr.
Specialist Master, Deloitte Consulting LLP

Introduction

Over the last 30 to 40 years the software industry grown from small entrepreneurial ventures to an enormous entity that in one form or another plays a critical role to most businesses today. As the software industry has grown so has the need to improve and mature capabilities for developing quality software products and solutions. Consequently, there has been a lot of investment focused on perfecting methodologies and tools for managing projects and developing quality software, and the seemingly unending search for the holy grail of estimation. And although there are many identifiable risks to software projects (e.g., inadequate processes, poor quality, unclear goals and objectives, poorly defined requirements, unstable objectives/requirements, inadequate stakeholder involvement, etc.) that the innovations in tools and methodologies seek to address, there are hidden and less recognizable sources of risk which can persist unnoticed . Sometime these hidden risks are quite complex and very difficult to identify, even in situations where they have materialized and created issues for a project.

One source of risk at the onset of a software project is the high level of "uncertainty" about the product to be developed. Until projects have captured and analyzed the client's requirements they simply don't know everything that should be included in a software product and sometimes this unknown can be quite significant. Yet, when some projects do not meet their budgets and schedules, many forget this fact and don't recognize it as a key contributing factor when conducting a root cause analysis to determine why the project estimate was inaccurate.

Preferably, the client has defined many, if not all, of their product requirements in advance of the proposal (which rarely occurs), and/or as the software vendor has previously developed something similar and can use past experience to base the current estimate on (assuming the client's requirements don't deviate significantly from what was built before). Unfortunately, the desired situation does not exist for many projects, and as a result there is a high degree of uncertainty which presents a very real risk. It is this uncertainty that exists at the onset of most software development projects, coupled with system development contracts that do not limit size, which can leave many software projects exposed to a seldom noticed budget-busting risk referred to as size creep. Size creep is a condition that occurs when the size of the product grows beyond what the project budget and schedule can realistically support. It is a risk that exists for many software development projects the moment the contract is signed. Size creep can be a silent killer of projects because few projects will have implemented the measures to detect or control it and are consequently blindsided by it on the occasions where it becomes reality and strikes. What is found by managing and controlling size creep is that projects can actually control and minimize the negative impact uncertainty can have on a project's budget or schedule.

In this paper, readers will be introduced to the concept of size creep, how it first came to be recognized, and the risks it can present to many software development projects. Additionally, given that some readers may not be familiar with the concept of "size" and its influence on software development, the paper will provide a broad definition of software size and the metrics the software industry tends to use in measuring it, and discuss how size may be underutilized in the support of software development projects. The paper will conclude by providing readers effective alternatives for reducing the risk of size creep and minimizing its impact. The general outline of this paper is as follows:

1. Defining size creep
   - Background of how the concept of size creep was first realized
   - Overview of size as it pertains to software development, how it is derived, and how it is used within the software industry
   - Detailed explanation of size creep and how some common business practices inadvertently create the perfect environment for the risk of size creep to exist

2. Alternatives for minimizing the risk and impact of size creep

Originally published as a part of 2011 PMI Global Congress Proceedings – Dallas, TX

Defining "Size Creep"

Background

Prior to joining Deloitte, I spent 22+ years working various roles in the IT Services industry (e.g., systems engineer, project manager, process improvement manager, etc.). During that time I was given many opportunities to support, analyze, and sometimes salvage projects that were challenged or had not met their schedule and budget commitments, with some exceeding their commitments by more than 100 percent. Although there seemed to be many contributing factors as to why these projects had overran their budgets and schedules, there was one extremely perplexing aspect commonly found with most, which was that there was little or no warning that their budgets and schedules were in jeopardy of being missed. These projects had been rolling along week to week, month to month, reporting everything on-track and under-control, and then in the course of a single reporting period, the projects suddenly found themselves forecasted to be substantially over budget and behind schedule. What would later be found through root-cause analysis and the lessons learned is that the risks and issues that ultimately caused the demise of these projects were often present the moment their contracts were signed, and the true causes were tied to factors not considered in their estimates or contracts, and not measured or tracked by project management. One particularly interesting risk found to exist on such projects was size creep. Size creep is so interesting because it is a risk many software development projects can unknowingly have, few projects recognize it before or even after they have experienced it, and for the projects where it had become an issue, the results were often devastating to their budgets and schedules.

The road to discovering size creep was not easy to follow as the observed evidence sometimes led the analysis in the wrong direction. When first looking at the evidence yielded from the projects that did not achieve their desired results there were three observations that quickly stood out as being significant and seemed to reveal what had occurred.

1. There was little warning these projects were in trouble. The projects had implemented processes and procedures to measure and report progress and performance against their baseline budgets and schedules. And most had been reporting they were tracking to budget and schedule up until the reporting period where they first discovered they were going to be significantly behind schedule and over budget. In the span of a single reporting period, these projects had gone from reporting a "green" status (indicating the project was performing to plan with no critical issues) to reporting deep in a "red" status (indicating the project was severely falling short from meeting its commitments and in deep trouble).

2. The amount of effort they were forecasting to complete their projects was well above what they had estimated would take and had agreed to in their contracts.

3. The contractual scope of the project had not changed from a product scope perspective. In fact, traditional scope management approaches were ineffective as the requirements and resulting attributes of the software product could be traced back to the contractual requirements.

Based on these observations, there were a couple of seemingly obvious conclusions that, although logically true, later proved to be false. First, projects typically do not go from being on schedule and on budget to being over budget and months past due within a single reporting period. Problems of this type generally build up over time and do not occur unless there are serious issues with the project management processes or the project manager's skills. For that reason, the logical conclusion was something had occurred, which the project management processes did not address, or that the project managers were too slow to respond or not skilled enough to see the issues impacting their projects.

Second, given that the forecasted amount of work required to complete these projects was significantly higher than what had been estimated, while the contractual scope of each project had not changed, another logical conclusion was that the estimation process had to be fundamentally flawed to have so grossly underestimated the effort. How else could the estimates have been so far off target without a change in scope?

Originally published as a part of 2011 PMI Global Congress Proceedings – Dallas, TX

To correct these situations, mandatory training for project managers was mandated, some project managers were released, and significant time and resources were spent seeking more effective ways to produce more precise estimates. Yet despite these efforts, the situation didn't improve. Too many projects fell short of meeting their budgets and schedules in the same manner as before, and the three observations discussed above continued to persist. Clearly, the conclusions drawn from the empirical evidence were either inconclusive or false, and there had to be other causes that hadn't been considered.

Further analysis uncovered a significant finding not previously recognized. The assumptions and parameters used to estimate effort for many of these projects that did not achieve their results proved to be false as the final software products produced were significantly different than assumed when the estimates were made and their contracts formed. The reason this was such an important finding is that it invalidated the prior conclusion that the estimation approach was flawed. Logically, the estimation approach couldn't be blamed for producing an incorrect estimate when so much of the final software product was unknown at the time the project estimate was produced. This was also the first indication these projects were at risk or in trouble the moment their contracts were signed, which meant the original project managers were likely placed in no-win situations. This finding led to a number of new questions, which meant that the analysis had to be expanded into areas not previously considered. The answers, as well as the concept of size creep, did not become apparent until later when the notion of size was first considered.

Size

No discussion on size creep makes sense without first having a very basic understanding of the concept of size. In the simplest of terms, size is a measurement of the amount of functionality provided by a software product, and sizing metrics are viewed as yard sticks for measuring it. Many terms are used within the software industry for conveying the concept of size, of which most include the word size and are synonymous (e.g., project size, application size, software size, product size, scope size, etc.). There are established and standardized metrics for measuring software product size (e.g., Source-Lines-of-Code, Function Points, Use Case Points, Feature Points, etc.) and although each is very different, they have one thing in common. They are based on one or more tangible attributes of the software product. To be a tangible attribute means it is something which physically exists, or is an entity that everyone observes to be the same thing (e.g., screens, reports, interfaces, data bases, lines of source code, etc.). Most of the standard sizing metrics provide their own specific approach for deriving software product size. However, the basic process most follow is to identify a specific set of one or more software product attributes, and then use weighting factors to convert each identified attribute to a common unit of measurement.

For years sizing metrics have been used primarily for two purposes, estimating effort for software development projects and as a base measurement to calculate many of the derived metrics that have shown to be useful for managing software development projects and process improvement. The theory behind using size-based estimation is that each unit of size for a given programming language or technology, and of similar complexity, generally will take an equivalent amount of effort to develop within a reasonable range. Hence, by projecting the size of a software product, one can estimate a range of effort it should take to develop it.

To more effectively demonstrate this concept the simple analogy of building a brick wall will be used. In this analogy, the size of the wall can be measured in a number of ways, which can then be used to estimate the effort and cost to construct it. For example, if the expected dimensions of the wall is known then it is easy to determine how many bricks it should take to build it. Each brick being a tangible attribute of the wall has an equal cost and takes approximately an equivalent amount of mortar and effort to set into the wall. Knowing or having an estimate of the total number of bricks needed to build the wall (i.e., the total size of the wall) and the amount of effort it takes to lay one brick, allows us to estimate the total effort required to complete the wall. Furthermore, once you have an estimate of the number of bricks and effort required, by applying the cost to lay a single brick and the hourly rate for a brick layer, you can calculate an estimate of the total cost to build the wall.

Although estimating software projects is not as clear-cut and simple as the brick wall example, there are many tangible attributes of a software product that can be used to size it and estimate the effort and cost to develop it. To demonstrate, the following provides examples of two size-based approaches commonly used to produce estimates for software projects. In the first example, Source Lines of Code (SLOC) is being used as an example of a software sizing metric that uses a single attribute of the software product (i.e., source lines of code) to estimate project effort. For illustration purposes, let's say it takes a particular organization 700 to 1,300 hours to produce 10,000 lines of

Originally published as a part of 2011 PMI Global Congress Proceedings – Dallas, TX

JAVA code. If the lines of JAVA code that it will likely take to produce a specific software product are known or have been estimated, then these performance limits can be used as conversion factors to easily estimate a range of effort for the total project to build it. For example, take a software product estimated to be 500,000 SLOC, then using these performance conversion factors the expectation is that the project should take 35,000 to 65,000 hours of effort to build the product.

For the second example, Function Point Analysis (FPA) is being used as an example of a software sizing metric that calculates size based on a set of common software attributes (e.g., Internal Logical Files, External Logical Files, System Inputs, System Outputs, System Queries), which are individually converted to a single unit of measurement. For the purpose of this paper, readers don't need to be familiar with the aspects of the FPA attribute types, but what is important to understand is that each instantiation of the relevant attributes is converted (based on its complexity) to an equivalent number of function points.

For example, a complex data base (an Internal Logical File) is equivalent to 15 function points, whereas a simple query screen (an External Query) is equivalent to three function points. Once all of the relevant attributes have been identified and converted to equivalent function points, the identified function points are then totaled and adjusted to derive the total count for the software product. As with the SLOC examples, each function point can be delivered within a range of effort, and this range can be used to estimate the project's total effort. Using the JAVA example again, assume it takes the organization 26 to 38 hours to deliver a single function point. If the software product is calculated to be 10,000 function points, then using these performance limits as conversion factors it is expected that the estimated effort range will be 260,000 to 380,000 hours of effort to build the software product.

The Project Management Body of Knowledge (PMBOK® Guide, 4th edition) defines scope as two distinct entities, "Project Scope" and "Product Scope". Project scope is defined as the work that needs to be accomplished to deliver a product, service, or result with the specified features and functions. Product scope is defined as the features and functions that characterize a product, service, or result. Stated another way, product scope comprises the detailed product requirements, which ultimately are used to determine the attributes of the software product. The reason for mentioning the PMBOK scope definitions is that there is a direct correlation between project and product scope, where project scope is largely dependent on product scope. This relationship can be seen through the PMBOK definitions of scope where the definition of project scope encapsulates product scope (i.e., the "features and functions that characterize a product"). This relationship is important to understand because it basically means a project should keep product scope under control or it will likely find it difficult to manage project scope. Given that size is a measurement of product scope, it therefore follows that size is a measurement that can be used to manage product scope, and to a great extent keep project scope in check.

Size is difficult and application professionals often ignore it (Gerush & West, 2009, p 6). Consequently, many software projects do not recognize size as a parameter to be managed, monitored, and controlled. As will be seen next, this issue coupled with software contracts that do not limit product size, can create an environment for size creep to occur, and can place a software development project at risk for size creep the moment the contract is finalized.

Size Creep

Earlier in this paper, size creep was defined as a condition which exists when the size of the product grows beyond what the project budget and schedule can realistically support. But, what does this really mean? Take a software development project committed to a baseline budget and a schedule. Logically, there is an inherent size limitation for the software product being developed where anything larger cannot be supported by the project's baseline budget and schedule. This size limitation exists whether or not the size of the product is known or was used to estimate the project effort. Stepping back to the definition of size for a moment, size is derived from the attributes making up the software product, and there is a direct correlation from product size to effort, and from effort to cost. A project budget can realistically support only so much effort, and the amount of effort required to build software is ultimately derived from the product size.

Certain software development projects can be at risk for size creep the moment their contract is signed, due primarily to how their contracts are formed. Unlike some industries where projects are often engaged to build well-defined products, many software projects are commissioned to build a new product that may only exist conceptually

Originally published as a part of 2011 PMI Global Congress Proceedings – Dallas, TX

in the minds of the client. Consequently, software vendors are expected to bid on projects at a time when the least amount of information is known about the product to be developed, and as a result many assumptions are considered when preparing their proposals. In order to generate accurate project estimates and proposals, software vendors should be very proficient (or very lucky) at predicting the software attributes that likely be needed to fulfill the agreed requirements in the final product. In other words, they should be very good at predicting what the client is going to ask for, or predicting the unpredictable. Taking this into account, and the fact that certain software contracts don't include terms or conditions for limiting size, and that certain software vendors may not consider size to be a factor they need to measure and manage, one begins to see how size creep can sneak up unnoticed on a project and cause major overruns. And if this wasn't a big enough issue to manage, many software contracts are fixed price which shifts the risks and liabilities of cost-overrun to the vendor.

To further elaborate on how this contracting approach can lead to size creep, a closer look of the process is needed. The process typically begins when a client issues a "Request-for-Proposal (RFP)" to one or more vendors for the purpose of developing a software product. The RFP usually provides a fairly detailed description of the problems or opportunities the client is seeking to address, their goals and objectives for the project and software product, and a detailed description of the software solution and its major components. By and large, the information provided in many RFPs amounts to the client's business requirements for the software solution they are seeking. The issue here is not what is in the RFPs, but what may be missing from it. Many RFPs do not capture the user and functional requirements that define the product scope at a level of detail deep enough to carefully predict or determine the size attributes of the software product to be developed. Consequently, certain software development contracts are estimated, negotiated, and agreed to at a time when uncertainty about the product to be developed is greatest. By basing software development contracts on the client's high-level business requirements, there may not be detailed enough requirements to know all the attributes that will be required, as software attributes are generally revealed through the progressive elaboration of requirements that typically takes place during project execution. This can leave projects exposed to the risk of size creep, as each detailed requirement shown to trace back to the project's contractual requirements is in scope for the project, as are the resulting attributes that have to be developed to address them.

At times, a client's detailed requirements can result in more system attributes than can be realistically developed under the project's budget and schedule. This coupled with a contract that does not limit the size of the software product places the software vendor in a very precarious situation where they are contractually obligated to build a product the project budget cannot support and someone has to bore the additional cost. The reality of this situation is that the software vendor has agreed to build a product without knowing its size and without placing any limitations on its size, and then left it up to the client to determine its size through the progressive elaboration of requirements. Basically, the software vendor is presuming that the client's requirements will likely not necessitate the construction of a product that is larger than what the project budget and schedule can support.

As a final thought on size creep, the "*cone of uncertainty*" is being used to illustrate how uncertainty is a major factor leading to the risk of size creep. In his book "Software Estimation, Demystifying the Black Art", Steve McConnell popularized the "*cone of uncertainty*" to illustrate how uncertainty is extremely high at the onset of a project, and how this has a negative impact on a project's ability to estimate with accuracy.

Originally published as a part of 2011 PMI Global Congress Proceedings – Dallas, TX

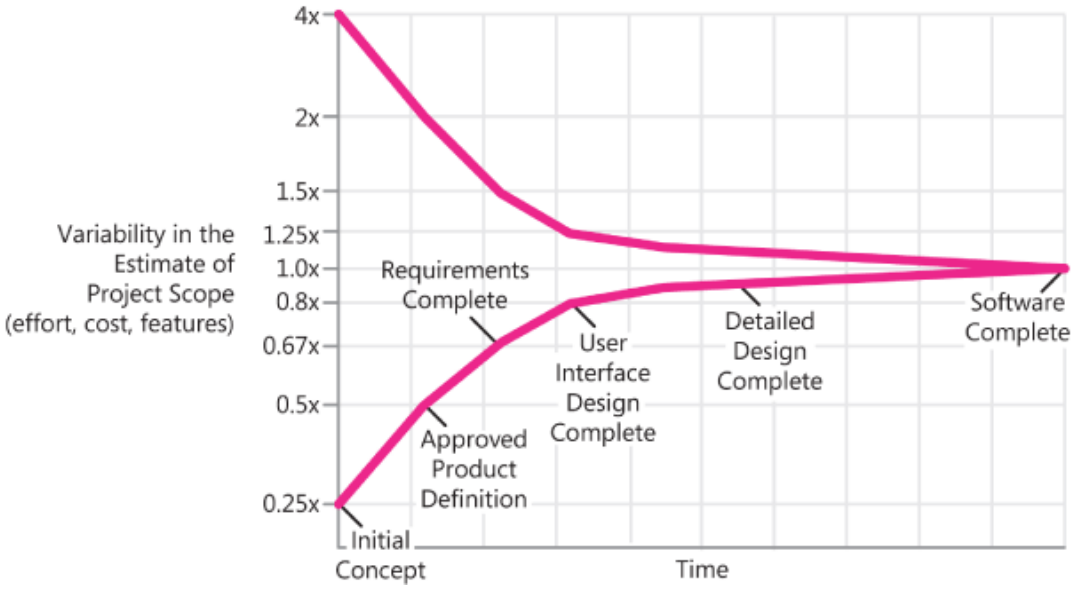

Exhibit 1: Cone of Uncertainty (McConnell, 2006, page 37)

The cone of uncertainty is fairly straightforward and easy to understand. The horizontal axis represents a conceptual timeline for a project, and is labelled with key milestones of the software delivery life cycle. The vertical axis represents the amount of uncertainty about the product to be developed at any given point in the delivery life cycle. The left edge corresponds to the beginning of the project where most contracts are estimated and established. Here, it can be seen that the cone of uncertainty is at its widest point symbolizing the point of the most significant uncertainty about the scope of the product. In other words, most software development contracts are formed at a time when the least is known about the scope of the product to be developed. At the right edge of the chart the cone of uncertainty closes, representing the end of the project and all uncertainty about the product. As figure-1 demonstrates, moving from left to right it is not until the project has completed requirements and moved into design, where the cone of uncertainty narrows to a point where the project knows enough about the product to realistically identify the attributes to be developed and determine the product's size. This is also the point where many of the projects impacted by size creep suddenly realize the development of the product is going to take significantly more effort and time than they estimated and their remaining budget and timeline will likely not be enough to the complete it.

Options for Avoiding and Mitigating the Risk of Size Creep

When searching for strategies to prevent and mitigate size creep, there is good and bad news. The bad news is that unfortunately for most projects which have fallen into the trough of size creep and did not establish the safeguards for preventing and managing it before the project began, there is likely little that can be done to remedy the situation. These projects have unknowingly agreed to build a software product, without size limitations, often at a fixed price, and then left it up to their client to determine product size later in the project, based on their requirements.

The good news is that although size creep is difficult to remediate once it has become an issue and is adversely impacting a project, there are measures that can be taken to prevent or at the very least, minimize its impact. However, to effectively prevent size creep, projects should either limit and control the size of the software product or find a way to greatly reduce the amount of uncertainty about the software product before negotiating and entering a contract to develop it. In other words, projects should find a way to reduce the risk and impact of uncertainty or find a way to accelerate the shrinking of the cone of uncertainty before committing to a contract. Surprisingly, preventing and mitigating size creep does not involve inventing anything new or learning skills the software industry has not had at its disposal for years. Instead, it can involve using existing software sizing metrics for more than estimation and in a manner certain organizations may not be accustom to.

Originally published as a part of 2011 PMI Global Congress Proceedings – Dallas, TX

Next, three approaches for addressing size creep will be discussed. The first approach is not really viable, but is being provided in response to the typical Project Management Professional (PMP) position on fixed price contracting. The final two options are approaches that have shown to be very effective at preventing size creep.

Approach #1 – Avoid Fixed Price Contacting (the PMP response)

Following this approach the software vendor avoids entering into fixed price contracts, instead opting for contractual terms that fairly compensate them for the work performed in development of the software product (e.g., cost plus contracts, time, and material contracts). Project management professionals (PMPs) are taught fixed price contracting is often not the most effective option for larger projects as it can leave the vendor disproportionately liable for the project's cost risks. Having researched a few projects that experienced size creep, it is difficult to disagree with this approach, particularly when you consider a few of these unfortunate projects ended up costing more than twice their original budget. The major advantage to this approach is that it can shift the risks and liabilities of cost overruns from the vendor to the client or to a situation where the client and vendor share the risk. Consequently, the vendor is fairly compensated for the work performed and the client pays for everything they request.

The problem with this approach is that it may not be realistic or practical for many software projects. What makes this approach problematic is that many perspective software clients demand fixed priced contracts to protect themselves from the risk of potentially large budget overruns. In addition, fixed price contracting is the most commonly used contract type (PMBOK Guide, 4th edition, p322), and therefore not likely to go away.

A second disadvantage of avoiding fixed price contracts is that it does not necessarily recognize or resolve the issue of size creep. It merely shifts the cost risk to a more favorable position for the software vendor. Although this can minimize the software vendor's losses, the client's, confidence and loyalty wane when they realize their software product is going to be extremely late and cost them significantly more than what they were promised.

In summary, software vendors should still avoid fixed price contracting for large projects where possible. Unfortunately, for many software projects this may not a viable approach for combating size creep as many organizations continue to require fixed price contracts from software vendors.

Approach #2 – Split Phase/Dual Contracting

Split phase/dual contracting generally involves conducting a project under two contracts. The first contract, which is ordinarily much smaller than the second, focuses on the activities typically performed in the requirements/discovery phase of a software development project. The second contract then covers the project phases to design, build, test, and deliver the software product. The purpose of the first contract is to capture the requirements for the software product in sufficient enough detail that the system attributes are identifiable. The work under the first contract is used to reduce much of the uncertainty about the software product to be developed.

The second contract is then estimated and negotiated based on the findings of the first contract, and the estimate is usually much more precise than if the entire effort had been estimated as one project. This is because most of the system attributes that need to developed (i.e., the product scope) are known and serve as input to the estimate for the second contract. Essentially, the size of the software product is established based on the detailed requirements identified through the work of the first contract. The major advantage of this approach is that it can accelerate the closing of the cone of uncertainty through the first contract, thus substantially reducing the risk of size creep for the second, much larger contract. The vendor can essentially enter the second contract with a very solid understanding of the product scope and the attributes to be developed and delivered.

There are a couple disadvantages to this approach that make it impractical in some situations. First, this approach is better suited in sole-sourcing situations where the work is not put out for competitive bidding. Generally when a client issues an RFP for competitive bidding, cost is a critical factor in their selection criteria. Therefore, they require vendors to submit proposals addressing the total cost to develop and deliver the software product. Vendors proposing a split phase/dual contract approach are generally excluded right away.

Originally published as a part of 2011 PMI Global Congress Proceedings – Dallas, TX

A second disadvantage is that the vendor still shoulders much of the cost risk and liabilities, albeit the risk likelihood and potential impact are greatly reduced as a result of the requirements work completed under the first contract.

In summary, this approach is an option for minimizing size creep, and reducing the risk of massive cost and schedule overruns due to size creep. But do remember it is often not feasible in a competitive bidding environment.

Approach #3 – Contractually Limit the Size of the Software Product

The third approach seeks to reduce the risk of size creep by establishing contractual size limits which the project cannot exceed without going through formal change management. Rather than trying to manage product scope solely through traditional means that use requirements as a primary basis for determining if something is in or out of scope, this approach acknowledges that the business requirements that certain projects start off with can be too vague to keep product scope in check. Therefore, this approach introduces and uses size as a measurement of product scope and continually measures and manages it against the contractual limits as the project progresses. Under this approach, the standard sizing metrics previously used to estimate effort and cost are also employed in a completely unaccustomed manner throughout the project to monitor and manage product scope and prevent size creep. This can provide a win/win situation for the client and software vendor. The client wins because they can still require fixed price contracts to keep their cost risk at a minimum for the functionality they require (as long as the software product stays under its size limits). The vendor wins because the cost risk caused by product scope uncertainty is greatly minimized by capping the size of the software product at a limit that can be reasonably supported by the project budget. As a result, the client should pay for everything they ask for in the software product, and the vendor is fairly compensated for the work they perform in delivering it.

As with the other approaches, this approach comes with issues of its own that should be overcome. To address these issues, the following items should be considered when contractually limiting the size of a software product.

1. Establish the size limit as something not to be exceeded and not as an absolute. When size is firmly established, and the size limit is not approached, the client may seek to be compensated for the difference. They may ask for additional functionality, additional work, or a refund of the difference. By capping size as a "limit not to exceed" projects can avoid this situation.

2. Use an industry recognized sizing metric as opposed to a homegrown, proprietary metric. Sometimes clients may want assurances that the software vendor is not manipulating the size measurements for their own financial gain, and therefore may hire an Independent Verification and Validation (IV&V) vendor to validate the software product size on their behalf. It makes it very difficult for a client or IV&V vendor to validate a product's size if the contract limits size based on the software vendor's proprietary sizing metric. Using a standard industry sizing metric can allow a client or IV&V vendor to more easily and competently validate the actual size of the software product.

3. Use a sizing metric that converts tangible attributes of the software product to a single unit of size as opposed to a completely attribute-based approach. Although size limitations can be established and managed based purely on a set of predefined attributes (e.g., number of screens, reports, databases, interfaces, etc.), the process of managing size is greatly simplified if these attributes can be converted to a single sizing metric. In all likelihood, the set of product attributes assumed would be needed when the contract was created will likely not be the same set ultimately delivered. By using a single unit sizing metric, projects can avoid the horse trading and disproportionate number of change requests that would be introduced when trying to manage size based on the software product attributes. For example, if the contract limited screens to 500 and data bases to 50, but through requirements the project finds it needs 57 databases but only 485 screens, then how do you balance the tradeoff? Is the new work required for the additional seven databases equal to the work given back by not having to build 15 of the screens, or is it considerably more? Furthermore, these tradeoffs only get more complicated if you try to add other factors into the equation, such as complexity. For a single unit of measurement the tradeoff is very simple. For example, if the contract sets size at 10,000 function points, the project only needs to track the actual function point total and not worry much about the attribute mix.

Originally published as a part of 2011 PMI Global Congress Proceedings – Dallas, TX

4. Use a sizing metric that can be measured and validated early in the life cycle. There are many sizing metrics used by the software industry. Some can be measured and validated early in the project life cycle, whereas others cannot. For example, function point analysis (FPA) is based on attributes that can be determined early in the project at the end of requirements. Whereas, source-lines-of-code (SLOC, another popular metric) is limited because it cannot be fully determined until the product is developed. The earlier in the project that you can determine the actual size of the software product, the more effective the project will likely be at managing size and preventing size creep. Consequently, FPA is a good metric for managing size creep, whereas SLOC is not.

5. Plan milestones in your work plan for measuring and updating the actual size of the software product and comparing it against the limitation established in the contract. Size can increase as a project progresses, and projects will want to know as-soon-as-possible when this occurs so applicable actions can be taken to prevent out-of-scope work from being performed. Therefore, the updating and monitoring of size should be planned milestone activities to make certain they are performed.

Final Words

Size creep occurs when the size of a software product to be developed grows beyond what the project budget is capable of supporting. Size creep is not the same as scope creep, as size can grow beyond the limits of the budget, while scope remains traceable to the accepted requirements. For many software development projects the risk of size creep can exist the moment the contract is signed, due primarily to the high level of uncertainty which exists about the product scope when most software development contracts are formed, and the fact that many software contracts don't consider size as a limiting factor. This can create a situation where the software vendor has agreed to build a product, without knowing the product size, with nothing in place to measure or manage product size, and then left it up to the client to dictate the product's size through the progressive elaboration of requirements. Furthermore, many software development projects do not measure, monitor, or manage size. For these projects size creep can be a significant factor, in that there is no warning until the damage has already occurred.

The good news is that the risk and impact of size creep can be minimized and controlled if the project does two things:

1. Establish contractual size limits for software development contracts

2. Measure, monitor, and mange software product size throughout the life of the project

Typically, by following these basic steps projects can effectively confine the uncertainty which leads to size creep, and minimize the risk of taking a major hit to the project's bottom line.

Originally published as a part of 2011 PMI Global Congress Proceedings – Dallas, TX

References and Appendices

Dominguez, J. (2009) *The Curious Case of the Chaos Report 2009* [Electronic Version]. Retrieved on 4/5/2011 from http://www.projectsmart.co.uk/the-curious-case-of-the-chaos-report-2009.html

Garmus, D. & Herron, D. (2001) *Function Point Analysis*, Boston, MA: Addison-Wesley

Gerush, Mary & West, David (2009) *Software Size Matters, AndYou Should Measure It* [Electronic Version]. Retrieved on 07/06/2011 from http://www.forrester.com/rb/Research/software_size_matters%2C_and_you_should_measure/q/id/54937/t/2

Jones, C. (1998) *Estimating Software Costs*, New York, NY: McGraw-Hill

McConnell, S. (2006) *Software Estimation: Demystifying the Black Art*, Redmond, WA: Microsoft Press

Project Management Institute. (2008) *A Guide to the Project Management Body of Knowledge (PMBOK® Guide)* (4th ed.). Newtown Square, PA: Project Management Institute.

Q/P Management Group, Inc. (2008) *Controlling Project Scope with Function Point Analysis* [Electronic Version]. Horvath, D. Retrieved on 10/08/2009 from http://www.qpmg.com/pdf/articles/controlling_project_scope_with_function_point_analysis.pdf,

Ross, M (1999) *Size Does Matter: Continuous Size Estimation and Tracking* [Electronic Version]. Retrieved on 10/16/2009 from http://www.qsm.com/qw99cse.pdf

Ross, M (1999) *Software Size Uncertainty: The Effects of Growth and Estimation Variability* [Electronic Version]. Retrieved on 10/16/2009 from http://www.compaid.com/caiinternet/ezine/ross-size.pdf

Ross, M (2005) *Managing Software Size* [Electronic Version]. Retrieved on 07/06/2011 from http://ai.kaist.ac.kr/~jkim/SEP583-2005/Resources-shl/R4-SWSizeManagement-Galrorath.pdf

Vogelezang, F. (2007) *Scope Management – How Uncertain is Your Certainty* [Electronic Version]. Retrieved on 3/25/2009 from www.peerevaluation.org/pdf/download/libraryID:23765