Software size measures and their use in software project estimation

Abstract

Using the right size measure is extremely important for software cost engineering purposes. However, the number of software size measures is growing rapidly and there is a lot of confusion in the industry with regard to the methods that should be used. The advantages and disadvantages of the major software size measures are discussed and recommendations are given on what size measures should be used to estimate the different types of software systems.

Keywords: software size measurement, software cost engineering, historical data, estimation models.

Introduction

By far most cost estimation models for software development, enhancement or maintenance projects use the software size as the main input parameter. Software size is widely recognized as an important cost driver for the effort and cost needed for software projects. Unlike many physical objects however, there are no universal measurement standards to measure software size. While a painter who needs to estimate the cost of painting a wall usually would measure the size of this wall in square feet or square meters as an input for his estimate, the measurement of software size is hard because of the fact that it has no real physical shape. However, there are a number of methods available in the industry that can be used to measure software size. For cost estimators that are no experts on software sizing however, it may be difficult to understand the advantages and disadvantages of the methods available and therefore they might find it difficult to select the best method for their purpose. In this paper, an overview is given of the most widely used methods for software sizing available to the industry. Also the advantages and disadvantages of these methods as well as recommendations of which methods to use for which types of software project estimates are given. In the following table the software size measurement methods are displayed that are covered in this paper.

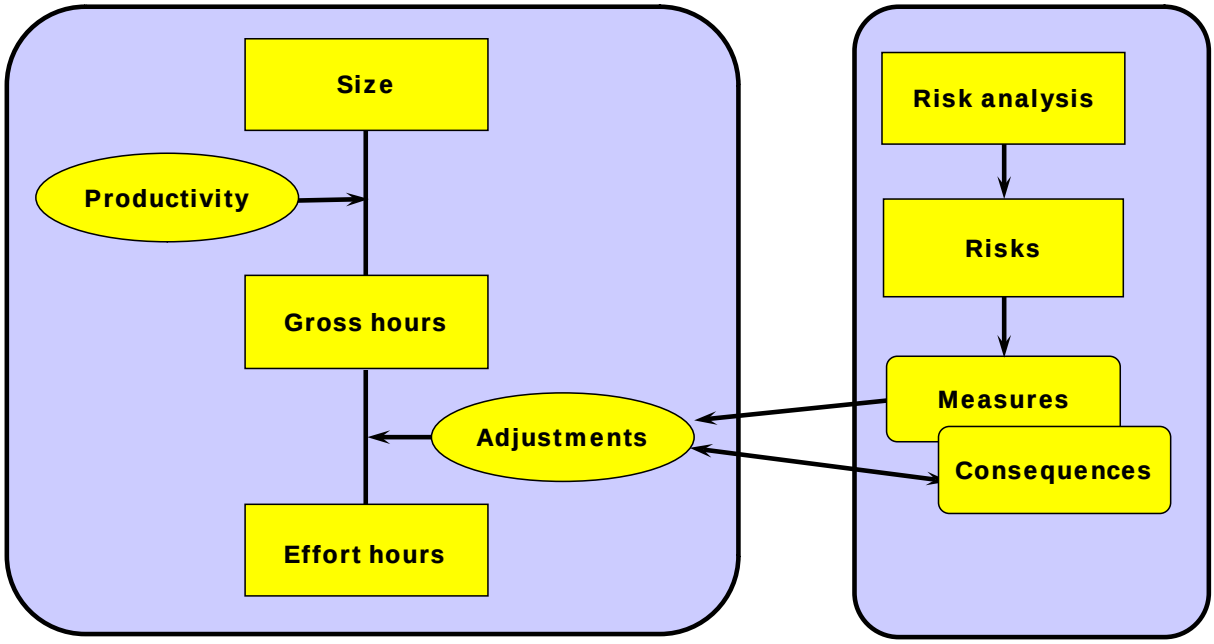
Software size measure	Measurement scope	Size
Source lines of code (slocs) [1]	'Physical' software object	Technical size in slocs
IFPUG Function Points [2]	Functional user requirements	Functional size in FP
NESMA Function Points [3]	Functional user requirements	Functional size in FP
COSMIC Function Points[4]	Functional user requirements	Functional size in CFP
SNAP Points [5]	Part of the non-functional user requirements	Non-functional size in SNAP points
Usecase Points[6]	Usecases, technical project characteristics, environmental project characteristics	'Effort/Size combination' in Usecase points
Story Points [7]	Backlog items, functional and non-functional	Story points

Software project cost estimation

In software project cost estimation, the focus is usually on estimating the effort hours of the people in the team. The reason for this is that the effort hours usually drive the total costs of the project. Of

course there are other costs involved, e.g. licenses, workplaces, infrastructure, et cetera, but these costs are not driven by the software size and can also be calculated in an easier way.

A (simplified) estimation model is displayed in the following figure.



The Size of the software to be developed is the main input parameter for the project estimate. It is crucial for the accuracy of the estimate that the size is measured accurately. Selecting a realistic productivity rate is also a very important step. This should be done based on historical data or with the help of professional parametric tools that are based on relevant industry data. Using company data is usually preferred, as this shows the actual capabilities of the organization which may be (much) better or (much) worse than industry average. Project specific characteristics can also be important to consider, e.g. schedule pressure, team size, quality constraints, etcetera. Therefore, parametric tools are usually very useful in this step of a project estimate.

Multiplying the size by productivity, the gross effort hours are calculated. Usually some adjustments are necessary based on a risk analysis. If one needs to be 99% sure that there won't be an overrun of the costs for instance, the effort hours will probably be adjusted to create a contingency.

Software size measurement methods classification

There is an important distinction between functional size measurement methods and other (non-functional size measurement methods or hybrid methods). Functional size measurement methods measure the functionality ('what does the software do') that the software offers its users and quantifies this in some way. The more functionality the user can use, this greater the size of the software. One of the main advantages of these type of methods are that the size is independent of technology used (e.g. Java, .Net, Oracle) and implementation method used (COTS, waterfall development, agile development, etc.).

Non-functional size measurement methods measure the technical artifacts of the software, usually the software code that is constructed. There are also hybrid methods available that measure functional aspects, technical aspects and sometimes also environmental aspects of the software

project in order to come up with a size, e.g. Usecase points [6]. However, most methods either measure the functional size or the technical size of software.

Functional size measurement methods – a brief historical overview

One of the many challenges for software developers products has been to find methods to properly estimate effort and time required for the development of a software product early in the project lifecycle. Allan Albrecht of IBM developed in the 1970s a measurement approach called Function Point Analysis [8] that quantifies the functions contained within software in terms that are meaningful to the software users. This method was at first intended to measure and analyze the productivity of the IBM teams. In the 1980s, it was discovered that it was possible to use FPA early in the project lifecycle as well, sizing the functional documents that describe the software system. This enabled people to use FPA to estimate new projects using the functional size of the software to be developed and the productivity rates of finished software projects.

With time, as it became more popular, FPA evolved and some variations were developed. To address this issue, ISO/IEC JTC 1/SC7 [9] decided in 1993 to initiate work in functional size measurement (FSM). A working group (WG12) was put together with representation from 12 countries and also from the principal FSM methods developing user groups. These were the International Function Point Users Group [2] the Common Software Measurement International Consortium [4],UK Software Metrics Association [10] and Netherlands Software Metrics Association[3].

The outcome of this work is that all four major FSM methods now conform to a minimal set of minimal requirements set by ISO/IEC standards, and that they are all now also ISO/IEC standards. This is very important for the industry. As well as the whole world benefits from standard measures like meters and kilos, the software industry can use standards for software size in many areas, like project estimation, project control, performance measurement and benchmarking.

Functional size measurement methods – advantages and disadvantages

In general, the main advantages of all ISO/IEC standards for functional size measurement are:

- The size measurement is carried out in an objective way. Two certified measurers should come up with the same size when measuring the same functional user requirements;
- The size measurement is repeatable and verifiable. If any assumptions were made by the measurement specialist, these should be documented as part of the size measurement;
- The size measurement is defendable. This is very important as functional size is often used in unit-priced contracts between suppliers and customers. Price per function point contracts are only possible when the functional size can be determined without a dispute;
- Some of the methods offer derived methods that enable cost estimators to fairly accurately measure the functional size quite early in the project lifecycle. For instance the Nesma indicative method [11] can be used after the requirements analysis phase and offers a quick way to get an indication of the functional size (+50% / -30% accuracy).
- Functional size can be recognized as ‘value’ to users and customers of the software system. More functionality means more value and therefore higher costs. This for instance when compared to technical size like source lines of code (slocs) where it is unclear if more slocs mean more value to the user and/or the business.

- Because of the standardization, the most important advantage is that it becomes possible to store the data of completed projects in order to use in estimates for new projects. Just like the painter that has data of his productivity in hours per square feet for specific types of walls, organizations are able to get insight in their productivity in terms of hours per function point. This enables benchmarking, for instance against the open industry database of the International Software Benchmarking Standards Group (ISBSG) [12].

Some of the disadvantages of using functional size measurement methods for software sizing and estimation are:

- Functional size measurement requires people with the expertise to carry out this activity. As it is such an important activity, it is really recommended to only have certified measurers do the sizing and even then have a proper review process in place. There are some initiatives to automatically measure the functional size of software but until now these are not very accurate;
- Functional size measurement takes some time and costs effort. Usually this is less than 1% of the project budget and by far most projects can be measured within 1 week of duration. Usually the benefits of doing a functional size measurement outweighs the cost many times, as it enables the stakeholders to draw up more realistic plans and avoid overruns and possible humiliation;
- Functional size measurement methods measure the functional user requirements that are specified in functional documentation. There are some requirements to this documentation in order for the measurers to be able to measure it. However, if these requirements are not met, the project is probably doomed anyway, as the programmers won't be able to code and the testers won't be able to test using these documents. A functional size measurement is in fact a powerful static test on the requirements, in many cases serious flaws and omissions are detected, enabling an early adjustment of the specifications and therefore preventing the costs of later detection and correcting of the defects.

The functional size of software projects can be used in the most widely used parametric estimation tools and models, e.g. QSM SLIM [13], Galorath SEER [14], Price Systems Trueplanning [15] and COCOMO II [16].

In the next sections, the typical ISO/IEC variants for functional size measurement are discussed into more detail and their use for software project cost estimation is discussed.

IFPUG

The IFPUG method is the most-widely used method for functional size measurement. Of all the requirements, It measures only the functional user requirements and it does so in the way that internal logical files (ILF), external interface files (EIF), external input functions (EI), external output functions (EO) and external inquiry (EQ) functions are identified and classified in complexity. The complexity of the software is classified in the following way: Low, Medium or High, depending on items like the number of data attributes involved, the number of files referenced or the number of record types that are part of the logical file. For an internal logical file (ILF) for instance, the number of function points connected to a low complexity ILF is 7 function points (FP), an ILF of medium complexity gets 10 FP and an ILF of high complexity gets 15 FP. There is minimum number of points

per function and a maximum number of points per function. Although this simplifies the analysis process somewhat, it is perceived as a drawback that there may not be enough nuance between the size of different logical files and functions. For instance, the size of a complex External Input function is 6 FP, but he size of a very complex EI is 6 points as well and the size of an extremely complex EI is also 6 FP.

The IFPUG method is most useful as a sizing method to estimate projects that comply with the following characteristics (not limited):

- The software is data oriented, many CRUD (Create, Read, Update, Delete) functions are specified;
- The software is administrative oriented;
- A detailed functional design is available in which the data model is complete and clear and for all the functions it is clear which data attributes are used;

When the size of the functional requirements is measured, there are numerous ways to convert it into an estimate. Using historical data of the organization that is going to realize the software is usually the best way to go, preferably supported by professional estimation tools. If historical data is not available, the open database of the International Software Benchmarking Standards Group can prove very helpful. It contains over 6700 projects (Release 13) and many of these were measured in IFPUG.

NESMA

Nesma FPA is very similar to IFPUG as it measures the same base functional components (BFC's [24]) (ILF, EIF, EI, EO, EQ) and there are very few differences now between the counting guidelines of the two methods. Many experts use the size measured in IFPUG function points as equal to the size in Nesma FP. The main advantages of Nesma over IFPUG for estimation purposes is the availability of a number of derived methods that are considered to be part of the ISO/IEC standard:

- Indicative FPA – This method can be used very early in the project lifecycle when only the data model is specified. The accuracy is not very high (-30% / +50%), but usually enough to be useful in an early stage of the project lifecycle when only a ROM estimate is needed.
- Estimated FPA – This method is considered to be the main method nowadays, since most functional documentation lack the detail to be able to use the standard Nesma (or IFPUG) FPA. The estimated FPA uses a fixed complexity assessment per base functional component: low complexity for logical files and average complexity for functions. This method can be used when the data model and the functions are known, but the details of which attributes are used in which functions are lacking or incomplete (which is often the case). The accuracy of this method is around -8% till +15% [27]

In addition, Nesma distinguishes between **project size** (the number of function points added to an existing system plus the number of function points changed in the system plus the number of function points deleted plus function points of software needed to realize to complete the project, for instance conversion software) and **product size** (the functional size of the application that is delivered to the users).

Nesma is a very active community of volunteers that are constantly working on new ways to use the method on new types or special types of software projects. Although these are not officially part of the ISO/IEC standard, these methods can be very useful for software estimation purposes. Widely used guidelines are for instance:

- FPA for software enhancement [19]: the use of this method is actually mandatory in some countries for contractors to be able to bid on government projects.
- FPA in SOA environments [20]
- FPA in data warehouse projects [21]
- FPA in the early phases of the project lifecycle [22]
- FPA applied to Usecase / UML documentation [23]

The Basis of Estimate (BoE) for software services [17] as published by Nesma and the AACE [18] is a very useful way to baseline any software project estimate, while also proving that the estimator has taken into account all the aspects that are important in his estimate.

COSMIC

The Common Software Measurement International Consortium (COSMIC) is a voluntary, world-wide grouping of software metrics experts. Started in 1998, COSMIC has developed a method of measuring the functional size of software that was designed to overcome some of the perceived flaws of Nesma and IFPUG FPA. While Nesma and IFPUG are mainly used to measure the size of administrative software, COSMIC is also applicable to measure the size of real-time and infrastructure software. The method is entirely 'open'; all method documentation is available in the public domain for free download.

One of the main advantages of the COSMIC method is the fact that it allows the measurement specialist to divide the software into software layers and peer components, which enables the possibility to measure the functional size of the software residing in separate architecture layers or in different components that communicate with each other. This overcomes one of the perceived drawbacks of the Nesma and IFPUG methods, which don't allow the software to be divided in such a way.

Another important advantage is the fact that the method follows a ratio scale. Therefore the functional size per functional process can be any size between 2 COSMIC function points (the minimum value per functional process) and (theoretically) infinity.

Usually the COSMIC method is very well applicable to use in projects developed with an agile methodology. The type of documentation that these type of projects often deliver (e.g. user stories, usecases, activity diagrams, sequence diagrams, class diagrams) are usually easy to measure with COSMIC in a very accurate way.

Although the use of COSMIC is increasing and also the number of certified practitioners are increasing, still not a lot of open benchmarking data is available. The ISBSG New Developments and Enhancements repository release 13 (released February 2014) contains about 500 projects measured in COSMIC, while the number of projects measured in IFPUG function points is well above 5000 projects.

The COSMIC method is usually particularly well applicable when using the functional size to estimate the following types of projects:

- Real-time software projects;
- Mobile app software projects;
- Infrastructure software projects;
- Software projects that deliver software in a layered architecture;
- Administrative software projects.

The functional documentation should at least show the functional processes that are implemented by the software and the data movements between the users and the software.

There are two more ISO/IEC standards for functional size measurement, FiSMA [25] and Mark II [26], but as these methods are only used in specific local communities, they are not discussed in this paper.

All methods for functional size measurement are very well suitable for software project estimation. The software size however is only part of the estimate. The size should be converted to an estimate by using relevant historical productivity data. When the functional size in function points is known, it should be multiplied by the most likely number of hours per function point that is likely to be needed in the project. Although there are several professional parametric tools available, and also databases with historical data can be procured easily [12], many organizations feel that it is complicated and time consuming to use functional size measurement and they find it difficult to determine an accurate productivity rate for the projects they need to estimate. They prefer less time consuming methods or methods that are more easy to apply. Unfortunately, this also means that these methods may not be as accurate.

Non-functional size measurement methods

While the functional size measurement methods only measure the size of the functional user requirements (what the software should offer to the user), the non-functional size measurement methods measure (often part of the) non-functional user requirements, which can be technical user requirements and/or quality user requirements (how the software should work). Also the measurement of the code that is delivered is considered to be a non-functional size measurement.

The most widely used non-functional size measure is Source lines of code (slocs).

Source lines of code

Source Lines of Code appeal to many people and organizations because once the software system is ready, it can be counted automatically by source code counters. Often, the development tools already measure the lines of code during the project.

Many organizations use the slocs measure as input for their software project estimation processes. This is remarkable, as the number of slocs can only be measured after completion. This means that to use slocs in software estimation, the slocs have to be estimated, not measured. Usually a team of experts together estimate the number of source lines of code for the new project and/or use analogy

methods with regard to previously completed projects in order to come up with an estimate of the software size to be delivered.

Although this seems like a good idea, this method brings large risks to the effort estimate. There is no ISO standard (or any other standard) available for source lines of code. Different code counting tools return a (sometimes completely) different result after counting the same code. Sometimes physical lines are measured, but also often source statements are counted instead. Since one statement can easily be written on multiple lines, this already highlights a big problem.

In addition, the number of source lines of code needed depends on factors like technical environment, complexity and programmer capabilities. Source lines of code also don't really represent value to the users. Is it better to get more lines of code or less lines of code? More lines may mean more functionality, but if one would pay a supplier based on a price per 1000 source lines of code one thing is for sure… the customer would get a lot of code! Code counters should not measure the code that is generated by the development tools, but in reality it is impossible for these tools to exclude the generated code from the measurement.

Therefore it is usually very difficult to use the slocs of a completed software project as an input for the next software project. Using experts to estimate the total number of source lines of code for a new project and then use historical data based on source lines of code is extremely risky. By estimating in this way, there is already a large uncertainty percentage in the main input parameter of the estimate.

So, why do many organizations estimate their projects this way? Some publications, for instance [28], state that estimating projects this way is a form of professional malpractice. It may work sometimes, but the risk involved is huge and failing projects with huge overruns probably cannot be avoided. In reality, this is exactly the thing that is often overlooked. There are usually many things that go wrong in large projects, and in the end it is almost always possible to 'blame' some operational issues that always appear during a software project… some technical problem, or a product owner that was not involved enough, or the OTAP environment was not implemented fast enough, or the customer changed their requirements a lot during the project… and so on. In most of the cases of software failures however, when analyzing the real cause, it proves that the project started with too optimistic expectations… the team is too small, the duration too short, the costs too low. Expert estimates, and estimates that are not based on industry standards and experience data usually start with optimistic expectations [35]. Organizations that understand the relationship between a realistic estimate and the result of the project, will focus on implementing instruments that enable them to estimate the project as accurately as possible, also not rewarding overly optimistic estimates delivered by suppliers for instance. However, organizations have to reach a certain level of maturity to understand this and this level of maturity is still far away for many organizations in the industry… even for those that consider themselves quite mature!

Theoretically, source lines of code are not useful for software estimation at all. Some people report successful projects estimated this way, but from a theoretical point of view this could be the result of chance or luck instead.

SNAP points

SNAP is the acronym for "Software Non-functional Assessment Process," a measurement method of non-functional software size. SNAP point sizing is meant to be a complement to a function point sizing, which measures functional software size. SNAP is a product of the International Function Point Users Group (IFPUG), and is sized using the Software Non-functional Assessment Practices Manual, now in version 2.2 [29].

SNAP is loosely connected to the ISO 9126 and ISO 25010 standards for software quality. It tries to size the non-functional requirements that are implemented in a software project. Although this seems like a good idea, the SNAP method seems to miss its target. It does not offer an integral measurement instrument for all non-functional requirements and furthermore a number of highly relevant non-functional requirements are not measured at all. A number of additional observations:

- Most documentation that is used in the industry does not explicitly state the necessary information about the non-functional requirements that SNAP tries to measure. In addition, it is not clear what to do when this information is missing. Count something anyway…or not count anything?
- Not all ISO 9126 or ISO 25010 categories of non-functional requirements are measured. Even when the SNAP points can be measured using the method as published, the non-functional requirements that people believe to be important cost drivers (e.g. performance or security, and so on), are ignored;
- It is not clear how the relationship between the different SNAP categories are determined and why this would be valid. Why would the UI complexity (SNAP points = Nr of properties added or configured …2, 3 or 4 times the number of unique UI elements) be of equal..or more…or less non-functional size than the non-functional size measured for batch processes (4 times, 6 times or 10 times the number of data attributes). There seems to be no relevant connection between the categories and the way the SNAP points are measured seems arbitrary;
- Professor Doctor Abran [30] pointed out that the statistical proof of the SNAP method does not pass the usual methodical validity tests, as it appears that outliers in the dataset used to demonstrate the correlation between SNAP points and effort seems to have not been removed;
- Some of the non-functional requirements that are measured are in fact functional and are also measured in NESMA and/or IFPUG methods. This is strange for a method that claims only to measure non-functional requirements.

All in all, although IFPUG and a lot of practitioners are advocating the SNAP method as a good and valid method to use in estimating, from a practical and from a theoretical point of view there are still many issues to address before this method can become useful when it comes to project estimation. At the moment there is very limited historical data of project measured in SNAP available. In 2013, the International Software Benchmarking Standards Group published a SNAP data collection form [36], but until now no project submissions with SNAP points have been received.

For now, at max it can be used to try to understand the differences in performance or productivity between completed projects, but it is not suitable for project estimating yet.

Hybrid size measurement methods

In addition to pure functional size measurement methods and non-functional size measurement methods, also a number of hybrid methods were developed. Here the following methods are discussed: Usecase Points (UCP), Fast Function Points (FFP), IBRA points and story points (SP).

During the last few decade, new size measurement methods were developed. The main reason for this seems to be the fact that many organizations don't wish to implement an 'Estimating & Performance measurement' process in their organization based on an ISO standard. Lack of expertise and high perceived costs are probably the main reason for this.

Usecase Points (UCP)

Some organizations that started to design their software requirements in so-called usecases tried to develop a new method to estimate the software projects based on these usecases. Karner [31] developed the Usecase points method, which is in fact a method that gives points to the complexity of the usecase (based on the number of transactions for instance). These points are converted into the total usecase points by multiplying the usecase size by a technical complexity factor (the impact of the technical requirements on the effort) and an environmental factor (the impact of environmental factors, like the team capabilities).

One of the main drawbacks of the method is the fact that it takes usecase documentation as the input for the estimate. Unfortunately, there is no standardized way of describing a usecase. There are multiple levels on which Usecase can be described (for instance, see [37]). Depending on the level of the Usecase, other usecase points may be measured.

Although this method may be useful to estimate projects within a single organization, the method is very subjective and can result in completely different estimates when applied by different people. Although the first part can be done fairly objectively (assess the complexity per usecase), determining the technical and environmental factor is a very subjective activity. If the organization documents usecases in a standard way and on the right level, and there are well defined rules and policies in place on how to determine the Technical Complexity Factor and the Environmental Factor, then the method can be used for estimating and should be able to give decent accuracy rates.

Fast Function Points

Fast function points is a hybrid method proposed by Gartner [32]. Although it is pushed on a management level as being faster and more accurate than IFPUG or Nesma function points, specialists in the industry know that this is not true. The method is frowned upon by measurement experts [39] and regarded as a commercial vehicle that is used by Gartner to sell their services on a management level. Claims are made with regard to aspects like training time, counting speed, accuracy and granularity which are not supported and in sometimes just wrong. There is no reason to think that the method should be either faster or more accurate than for instance the estimated (global) Nesma method. Only Gartner has access to productivity rates in Fast function points, so implementing this method will result in a vendor lock-in for the organization that allows this. The method may give good results in estimating, but the vendor- lock-in, the false claims with regard to

the comparison with Nesma/IFPUG, the fact that it is no ISO/IEC standard and the lack of available open benchmarking data would sure mean that it is very risky to implement this method.

Story Points

The last couple of years more and more organizations started to focus on agile development methods, like for instance Scrum. In this method, the story point metric is used by teams that estimate the effort of the backlog items they need to realize in a specific sprint. Although story points are sometimes referred to as a size measurement method, in fact it is an effort estimation method based on points. The team members assign points to backlog items and the number of points are related to the effort they think is necessary to finish the item. As there is no standard way of assigning points to backlog item, the number of story points can and will differ significantly between teams and organizations. Therefore, historical data based on story points can never be used for estimation purposes, except when the estimate is done in controlled circumstances within one team. Story points cannot be used across teams and definitely not across companies. Also the try to normalize story points in the Scaled Agile Framework (Safe) [33] does not really propose a solution to this, as the normalization is not done on size but on effort.

Story points are very useful for sprint planning. Once the backlog items are specified, a team should be able to rather consistently assign story points to the backlog items and therefore it should be possible to relatively accurately estimate the number of backlog items that can be realized in a sprint based on the velocity of the previous sprints. However, as Santillo [34] shows, it is more accurate to use COSMIC function points instead, in order to enable the use of parametrics tools and available industry data. Story points are not useful for project estimation. Organizations for instance wish to answer questions like: “If this team works for 6 months in 2-week sprints on this backlog, how much product will I have?”, “What is the effect of an extra programmer?” or “How many sprints are necessary to complete all the backlog items?”. These type of questions cannot be answered by using story points.

Conclusions and final remarks

The software size to be realized in a software project is considered to be the most important input parameter. However, there are multiple methods available that can be used to measure the software size and usually there is conversion possibility between methods. Some of the methods discussed are suitable for specific types of software projects, others are limited in their use, for instance they require a specific type of documentation to measure.

In general, when implementing an Estimating & Performance Measurement process it is advisable to follow the guidelines documented in the ‘Basis of Measurement’[38]. Unless there are very specific reasons to deviate from this, the industry best practice is to use one of the ISO/IEC standards for functional sizing. The objectivity, repeatability, verifiability and therefore defendability of a measurement carried out in one of these standards ensures that there is as little risk as possible in the size measurement and therefore in the main input parameter for the estimate. The availability of professional estimation tools that support these standards and the availability of industry data measured in these standards will result in objective and as accurate as possible project estimations at that specific point in time.

References

[1] Source lines of code - http://en.wikipedia.org/wiki/Source_lines_of_code
[2] International Function Point User Group (ISO/IEC 20926), http://www.ifpug.org
[3] NESMA (ISO/IEC 24570), http://www.nesma.org
[4] Common Software International Measurement Consortium (COSMIC (ISO/IEC 19761)), http://www.cosmicon.com
[5] Software Non-functional Assessment Process (SNAP), http://www.ifpug.org/about-ifpug/about-snap/
[6] Usecase Points, http://en.wikipedia.org/wiki/Use_Case_Points
[7] Story points, https://agilefaq.wordpress.com/2007/11/13/what-is-a-story-point/
[8] A. J. Albrecht, "Measuring Application Development Productivity," Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium, Monterey, California, October 14–17, IBM Corporation (1979), pp. 83–92.
[9] ISO/IEC JTC 1/SC7, http://en.wikipedia.org/wiki/ISO/IEC_JTC_1/SC_7
[10] United Kingom Software Measurement Association (UKSMA (ISO/IEC 20968)), http://www.uksma.co.uk
[11] NESMA indicative method, http://nesma.org/themes/sizing/function-point-analysis/early-function-point-counting/
[12] International Software Benchmarking Standards Group, http://www.isbsg.org
[13] Quantitative Software Management, Software Lifecycle Management (QSM SLIM), www.qsm.com
[14] Galorath SEER, www.galorath.com
[15] Price Systems Trueplanning, http://www.pricesystems.com
[16] COCOMO II, University of Southern California, http://csse.usc.edu/csse/research/COCOMOII/cocomo_main.html
[17] AACE® International Recommended Practice No. 74R-13, Basis of Estimate – As Applied for the Software Services Industries (TCM Framework: 7.3 – Cost Estimating and Budgeting). Published in 2014 by Nesma and AACE. Download for free: http://nesma.org/downloads/74r-13-basis-estimate/
[18] American Association of Cost Engineering International, http://www.aacei.org
[19] FPA for software enhancement http://nesma.org/publications/downloads/guides/
[20] Functional Sizing in een SOA-gebaseerde omgeving (Dutch), http://nesma.org/publications/downloads/guides/
[21] FPA applied to Data Warehousing, http://nesma.org/publications/downloads/guides/
[22] FPA in the early phases of the project lifecycle, http://nesma.org/publications/downloads/guides/
[23] FPA applied to Usecase / UML documentation, http://nesma.org/publications/downloads/guides/
[24] ISO/IEC standard , http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_tc_browse.htm?commid=45086
[25] FiSMA, http://www.fisma.fi/in-english/
[26] Mark II, http://en.wikipedia.org/wiki/MK_II_FPA
[27] Van Heeringen, van Gorp & Prins, Functional size measurement, Accuracy vs. costs, Software Measurement International Forum 2009, Rome, Italy.
[28] Capers Jones – 'Software Defect Origins and Removal Methods (2013), http://www.scribd.com/doc/225906827/Software-Defect-Origins-and-Removal-Methods2013#scribd
[29] Software Non-functional Assessment Process (SNAP), http://www.ifpug.org/about-ifpug/about-snap/
[30] Abran, A. prof. dr. "Software Estimation: Transforming dust into gold?", Keynote speech at the IWSM-Mensura, Oct. 6-8 2014, Rotterdam (the Netherlands), http://www.slideshare.net/NESMA-NL/iwsm2014-transforming-dust-into-pots-of-gold-alain-abran-39954154, slide 46 and 47.
[31] Usecase Points, http://en.wikipedia.org/wiki/Use_Case_Points
[32] Gartner FFPA, www.gartner.com
[33] Scaled Agile Framework, http://www.scaledagileframework.com

[34] Santillo, L. and Berardi, E. “COSMIC based project management in agile software development”, http://tinyurl.com/p8pvmof
[35] McConnell , S. “Software Estimation – demystifying the black art”. http://tinyurl.com/ogwouuq
[36] ISBSG Data Collection Questionnaire IFPUG/NESMA with SNAP - http://tinyurl.com/op9s5l6
[37] Accelerated Delivery Platform wiki for agile Development, http://tinyurl.com/qekxjsb
[38] Basis of Measurement (BoM) – www.nesma.org/publications , published 2014.
[39] Nesma website, “Challenges in Productivity Measurement”, http://nesma.org/themes/productivity/challenges-productivity-meaurement/