Counting Function Points for Agile / Iterative Software Development

Abstract

Function points (FPs) are proven to be effective and efficient units of measure for both agile/iterative and waterfall software deliveries. However, inconsistencies come to light when comparing FPs counted in agile/iterative development with those counted in waterfall or combination development – and those inconsistencies can create confusion for cost, productivity, and schedule evaluations that span multiple software delivery methods.

IFPUG's definitions for such terms as "project," "elementary process," "consistent state," and "enhancement" do not directly translate to agile/iterative delivery methods in which the term "project" is often interchanged with such terms as "sprint," "iteration," "release," or "story map." This paper seeks to marry IFPUG definitions with equivalent concepts in agile/iterative processes so as to create a basis for consistent comparison.

Introduction

Functional size is a pre-requisite for estimating the cost, effort and duration of software development "projects" (labeled by IFPUG as development projects for the first "release" of a software product and enhancement projects for its subsequent adaptive maintenance and enhancement). Function points are a convenient and reliable common denominator of size and are often used as the basis for comparing project productivity (FP/effort), duration delivery (FP/elapsed time), maintainability (hours/FP), product quality (defect density) and other important aspects of software delivery.

With today's IT landscape dotted with agile, iterative, spiral, waterfall and combination approaches to development, businesses are searching for the ultimate approach to delivering the right software (functionality- and quality-wise) at the lowest unit cost for the least amount of effort. But for estimation to succeed in this climate, we'll need consistent FP definitions across all methods of software delivery.

Agile Is Here to Stay

Gone are the days when agile/iterative development methods were considered "rogue" and without structure; today, agile methods are held in high esteem, even in conservative software development shops where waterfall still prevails. Indeed, the penetration of agile in the IT marketplace has had numerous positive impacts, including:

- Change is no longer seen as the enemy;
- Users are more receptive to participating on projects and better understand the impact that non-involvement can cause;
- Business stakeholders are more engaged; and
- Developers can better respond to changing business requirements.

Now, with both agile/iterative and waterfall methods at their disposal, businesses are searching for the optimal combination of talent, tools, techniques, cost, and schedule that will deliver good-enough quality software for a reasonable investment of time and money. But finding that "sweet spot" relies on measuring the same elements in the same ways across various delivery methods. And consistency in measurement depends on consistency of definitions and the application of measurement techniques, such as IFPUG Function Points.

We need to start by aligning the IFPUG definitions and then looking at how to apply them consistently to count FP on agile and waterfall deliveries alike.

IFPUG Functional Size Measurement Definitions

IFPUG FP methodology (IFPUG 4.3.1) gives us guidance on how to count FP based on projects and the delivery of unique (and complete) elementary processes that leave the business in a consistent state. Agile/iterative techniques deliver software incrementally. Defining what constitutes a "project" and the delivery of an "elementary process" in agile/iterative is the KEY element for consistent function point counting across development approaches.[1]

Since FP counting of software *development* is meant to measure the size of the functional user requirements delivered via a *project (development or enhancement)*, the methodology used to implement that functionality (be it agile/iterative, spiral waterfall, or any other development method) should have no effect on the size of the delivered software product. The application FP (also called the baseline or installed application FP size) is the same regardless of the delivery method used and can be measured consistently at the completion of any type of software delivery. Application FP counts are of secondary concern for this paper; the primary concern is defining what constitutes a "project" (either development or enhancement) in agile/iterative development.

Terminology Presents Challenges

Before we get into the issues and challenges of counting FP in an agile environment, let's add a bit of strictness and consistency to a few of our terms:

- **Release:** A release is the distribution of the final version of an application. A software release may be either public or private and generally constitutes the initial generation of a new or upgraded application. A release is preceded by the distribution of alpha and then beta versions of the software. In agile software development, ***a release is a deployable software package that is the culmination of several iterations.*** (source: http://searchsoftwarequality.techtarget.com/definition/release)
- **Project:** *A collection of work tasks with a time frame and a work product to be delivered* (IFPUG 4.3.1 glossary). According to the Project Management Institute (PMI.org), a project is a *temporary endeavor undertaken to create a unique product or service.*
- **Iteration:** In agile software development, an iteration is a *single development cycle, usually measured as one week or two weeks.* (source: http://whatis.techtarget.com/search/query?q=iteration)

  (Side note: Some proponents of agile insist that all iterations be the same length, and that the particular length of iterations (anywhere from 2 to 6 weeks) is of less importance. For our purposes in this paper, the key element is that an iteration represents a single development cycle.)
- **Sprint (software development):** In product development, a sprint is a *set period of time during which specific work has to be completed and made ready.* (source: http://whatis.techtarget.com/search/query?q=sprint)

For **waterfall development**, it is fairly easy to identify and count FP for discrete development and enhancement projects. "Release" and "project" are often used synonymously to refer to the scope of a self-contained software delivery.

For **agile development**, a "project" is not so easily identifiable. "Sprint" and "iteration" are used more often than "release," and those terms are based on *elapsed calendar time or work effort* rather than functionality.

“User stories” (or “use cases”) are used to describe functionality and are useful for identifying functional user requirements, but there is no requirement that they constitute an elementary process or that they leaves the business in a consistent state – both of which are required for FP. The notion of using “story points” (a sizing approach intended to quantify the relative size of a user story) as equivalent to FP (as suggested by a few agile advocates) is not feasible for the following reasons:

- Story points are not convertible to FP (there is no conversion factor);
- Story points are not standardized (FP are standardized through IFPUG and ISO); and
- Story points capture user story size differently (and based on different concepts) than FP.

When functionality is delivered in discrete and well-defined construction projects, as is intended with **waterfall-style deliveries**, counting FP is easy and based on a single set of functional user requirements. Even when a subset of the overall features is delivered in one release and then enhanced in a later release, the discrete “chunks” of new/enhanced functionality make counting FP a straight-forward process using IFPUG methodology.

With traditional waterfall delivery, the terms “developed” and “delivered” are almost always used interchangeably. But in **agile/iterative development**, there is a difference between “developed software” (which is not yet ready for mass deployment) and “delivered software” (ready for full deployment).

Therefore, with agile/iterative forms of software delivery, counting the “delivered” functionality is not so easy. IFPUG (and other ISO conformant) functional size measurement techniques are counted based on complete elementary processes or functions that leave the business in a consistent state, not on parts thereof. A “function point” count consists of delivered (or anticipated to be delivered) functions that are whole business processes. Partially delivered functions that are incomplete and cannot support the business without further work would not typically be counted as “function points delivered.” For example, a business process such as create hotel reservation would not be an elementary process until a reservation is made and stored. If the first part of the reservation process was delivered in one sprint (such as checking the availability of a hotel for given dates) and the latter part was delivered subsequently (enter customer information and book reservation), FPs would be counted on the elementary process (both parts.)

Why Does All This Matter?

At this point, you may be asking why we don’t just use the word “release” in place of the word “project” and be done with it.

Or, couldn’t we simply count up the delivered function points at the end of a “release” no matter how the software is developed, and then compare the productivity across releases to perform our estimates?

Actually, yes. This is absolutely the right way to go, but only if our definition of “release” can remain consistent across our various forms of delivery. For starters, we’ll have to determine the number of agile iterations or sprints that constitute a “release.” Let’s consider the following situations:

- When a single software “product” (i.e., the result is a working piece of software) is delivered via two or more distinct releases, each of which is completed and implemented into production (i.e., fully-functioning software), the software application in place at the end of the two releases is exactly the same size as it would have been if delivered all at once. (Consider the analogy of a floor plan that is built in stages versus all at once – the resultant square foot size is the same.) Each release is discrete and self-contained, and the sum of the FP of the two releases likely will exceed the installed application base (because some of the functionality completed in release 1 may be enhanced through adaptive maintenance

FPin release 2, yet may not increase the application size (also called “installed base” size or baseline). Think of how a house can be delivered in a first construction and then renovated in a second – the square foot size of the two constructions added together may exceed the overall size of the house.

- However, when the software is built iteratively over the course of a year in two week sprints, there is a lack of discrete delivery. (Think of building a house bits at a time and slowly developing the underlying floor plan.) Where is the “elementary process” for FP counting? Likely, the “complete” functions were delivered through multiple user stories or use cases spanning a number of sprints or iterations. Therefore, the challenge to counting function points in agile/iterative lies in the question of when and where a business process or function leaves the business in a consistent state.

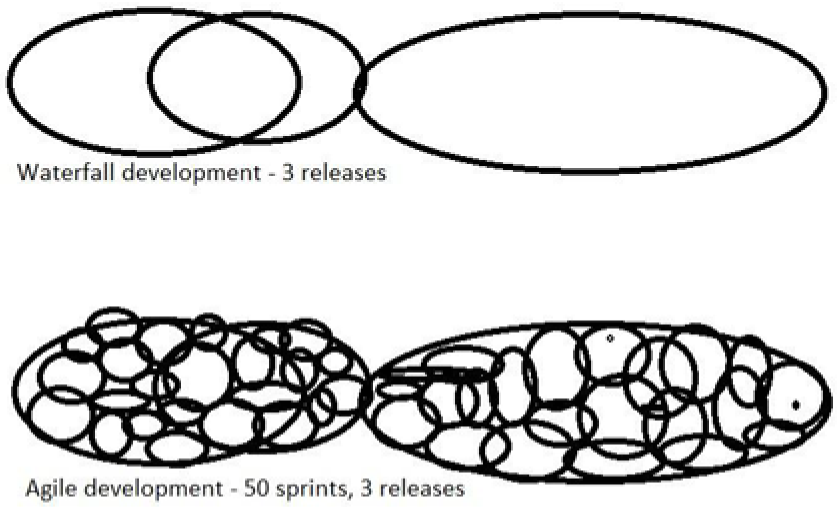

Waterfall development - 3 releases
Agile development - 50 sprints, 3 releases

Diagram 1 Waterfall releases vs agile releases

Some would suggest that each sprint or iteration should count as a discretely countable functionality in FP; however, this approach would contradict FP definitions and concepts including:

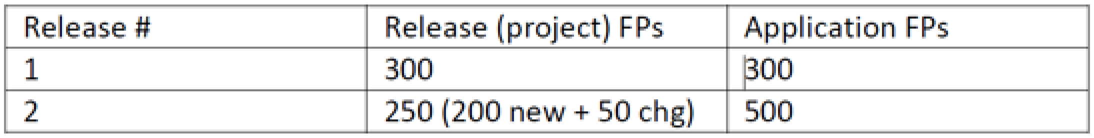
- Elementary process
- Self-contained
- Enhancement (defined as the adaptive maintenance of a delivered software product)
- Functional user requirements (which relies on elementary processes)
- Development project
- Enhancement project
- Counting scope (a set of Functional User Requirements)
- Base Functional Requirement (BFC) – elementary unit of Functional User Requirements
- Consistent state

Why would counting “sprints” violate these definitions? The answer is that a sprint, by definition, is based on elapsed time or work effort – not on functionality. Certainly user stories (and use cases) describe user functionality, but there is no requirement that they describe complete or self-contained functionality. Consider that two user stories for an airline reservation system might be written as:

a. Choose the flights on which you want a seat
b. Pay for the reservation

Clearly these are two different steps in the sequence of steps needed to complete the airline reservation. But each one is NOT its own separate elementary processes they're part of a single elementary process where both steps are needed to leave the business in a consistent state.

Further examples:



Release #	Release (project) FPs	Application FPs
1	300	300
2	250 (200 new + 50 chg)	500

Function Points in Agile/Iterative Development

When we talk about implementing user stories or use cases, the assumption may be that each user story or use case equals at least one standalone and self-contained function. This is seldom the case. While the scope grows and morphs with each iteration or sprint, the requirements are often progressively elaborated. This means that one "functional user requirement" may span multiple user stories or use cases – especially if it is a complex one. Thus, the application may not satisfy the full user requirement for a process or transaction until after a set of sprints is implemented.

A full agile software development implemented in a series of two- to six-week sprints will deliver functionality in a piece by piece fashion. It may not be easy to determine/predict when the functionality will actually be delivered, especially once the IFPUG definitions for "elementary process" and "leaving the business in a consistent state" are taken into account.

Using the home building analogy, agile development is similar to pouring the foundation and building rooms a bit at a time, as the overall floor plan eventually comes into being. When a home is constructed in this manner, it is not ready to be occupied until the rooms are finished and a roof covers the structure. In agile development, functionality is typically delivered partially – in sprints – and it isn't until several sprints are delivered that the business can begin to actually use the software. Yet there is a tendency to assume that sprints and iterations are akin to a new development project for the first sprint and enhancement projects for each sprint thereafter.

The challenge to counting FP on agile projects lies in determining *which* functional user requirements have been satisfied (and *when* they are satisfied) by the software delivery.

For instance, if a function is "delivered" in a sprint (i.e., we count FP for the initial sprint, assuming that the user story completely describes an elementary, self-contained, and complete business function), and subsequently enhanced in a second sprint, was the original function:

a. Incomplete (i.e., the elementary process was NOT fully delivered in the first sprint) – and we shouldn't have counted/taken credit for FP in the first sprint?

b. Complete at the time of the first sprint but now enhanced due to changing requirement – and we should count delivered FP for sprint #1 and count the entire transaction's FP a second time for sprint #2?

c. Flawed in the first sprint – therefore the FP counted in sprint #1 should not be recounted in sprint #2 (because sprint #2 was only corrective maintenance)

d. Some other variation?

AND waterfall development) are the same. The actual size of the installed application baseline (FP installed) is – *or should be* – the same, regardless of HOW the software is developed.

When does the delineation of FP across sprints become an issue? (Or, why is it important to count delivered FP in a consistent manner regardless of development methodology between agile and waterfall projects?)

There are two significant situations where the FP "delivery" is critical to businesses:

1. **Productivity assessment.** Businesses want to compare the cost per FP or effort per FP between agile and waterfall projects, but doing so requires a consistent baseline. If we count FP for each sprint the same way as we do for an entire project, the total project FP delivered in agile (the sum of FP across all sprints) may be 10 or more times the total project FP delivered using waterfall (the sum of FP across several releases) thereby invalidating productivity comparisons;
2. **Outsourcing.** When businesses commit to paying for software as it is "delivered", it makes no sense that the business should pay over and over for partial delivery of functionality just because it is delivered using an agile approach. From the client perspective, the overall delivered software (base) is the same size.

Therein lies the dilemma and the challenge in using function points on agile projects – it is problematic to credit agile projects that deliver same completed functionality (i.e., complete elementary processes leaving the business in a consistent state) with having produced MORE functionality than waterfall projects!

To recap, let's look at one example using the two different methods:

Waterfall software delivery: The business needs a new customer service application where the final installed software will equal 1000 FP. Through negotiation and agreement, three phases/projects are outlined, each of which delivers working software in production.

1. Phase 1: New development project = 300 FP. (Installed baseline at the end of the project = 300 FP.)
2. Phase 2: New functionality of 200 FP and enhancement of 50 that were already delivered in phase 1. Project count = 250 FP. (New installed baseline at the end of the project is now 500 FP.)
3. Phase 3: New functionality of 500 FP and enhancement of 50 that were already in place. Project count = 550 FP. (Installed baseline at the end of the project is now 1000 FP.)

Agile development: The business needs a new customer service application delivered using an agile approach. User stories are iteratively documented and a series of 2-week sprints is agreed upon to allow developers and the business to discover the requirements and define them as they go. Twenty-five different sprints are worked on over a year period, and at the end of the "project(s)" the installed baseline software is 1000 FP.

1. Sprint #1 outlines the need for users to sign in and validate their password. (It is not yet certain where the data will reside. We cannot yet count a datastore definitively but it is envisaged that it will be maintained in either an ILF or EIF in a future user story/user case.) Sprint 1 also delivers the first of several screens needed to set up a new customer. – Estimated FP count = 1 Low Complexity Query (for user validation) + 1 Average complexity datastore (for customer) + 1 Average Input process (create customer) = 16 FPs. (Baseline = 16 FPs.)
2. Sprint #2 adds a second screen of data for customer creation and identifies the need to allow changes to and deletion of customer records. Customer details (all of the data added across both screens) can be displayed. What should be counted in Sprint #2?

   a. The new functionality introduced: Change customer = 1 Average complexity Input; Display customer =

Average complexity Query; Delete customer = 1 Low complexity Input EI;

b. The datastore called Customer - it was already counted in the first sprint (and it is the same complexity.). Should it be counted again in sprint #2? It seems nonsensical to do so.

c. The add customer function – it was delivered partially in the first sprint because there wasn't enough time to deliver it fully. It was incomplete (i.e., did not leave the business in a consistent state) in the first iteration – the question is whether the FPs should be counted in sprint #1, in sprint #2, divided between sprints (1/2 and ½ perhaps) or as the entire number of FPs in both sprints (i.e., appearing as double the FPs).

Guidance On FPs counting For Agile

The following list of recommendations is provided to increase consistency across the various forms of software delivery:

1. Identify the user stories and use cases that contribute to a single elementary process, group them together, and count the FPs for the elementary process (and document what contributed to the function);
2. Count an ILF only when its maintenance is introduced and consider future DETs and RETs it will include (i.e., count a Customer ILF only when the first transactional function to maintain it is delivered, and count its complexity based on all DET and RET envisaged in that release);
3. Count functionality at a release level according to #1.
4. Count as development project FP all functionality for the first release as long as working software is implemented (i.e., users can input data) and elementary processes are complete;
5. Count as enhancement project FP all functionality for subsequent releases as long as working software is implemented and adaptive maintenance is performed on each release;
6. If data or transactions describe code data, do not count (not as ILF, EIF, or any associated maintenance or query/drop down functions for such data). This needs to be spelled out because this is easy to overlook when counting from use cases or user stories.
7. Document your assumptions used in the FP count(s).

Comparative and consistent FP counts across various development "projects" can be done through consistent terminology, and the application of FP rules. Being careful not to size "bits" of functionality partially delivered, and instead grouping use cases and user stories into elementary processes according to IFPUG 4.3.1 will go a long way to creating consistent FP counts.

---

[1] The reader is encouraged to refer to IFPUG's Counting Practices Manual (CPM) 4.3.1 for information regarding definitions for terms used throughout this piece of literature.