Applying Simple Function Point Analysis to an 804 Rapid Acquisition Program Cost

1

Authors

Bob Hunt, Senior Cost Analyst, Software Subject Matter Expert, NSI
Rainey Southworth, Cost Analyst, NSI
Chad Lucas, Senior Cost Analyst, Galorath
Esteban Sanchez, Senior Cost Estimation Consultant, Galorath
Heather Meylemans, Head, Acquisition Cost Estimating Division, NAWCWD
Dustin Harman, Cost Team Lead and Analyst, NAWCWD
Dave Volpe, APEO-Cost, NAWC-AD

Abstract

For large Federal Agile Programs, the Cost Analyst faces three major issues – determining if the program is fully or partially Agile, accurately defining the requirement, and determining the appropriate sizing methodology. Many Federal Programs are Hybrid-Agile (a mixture of waterfall and agile processes). When software is developed under the Middle Tier of Acquisition (Section 804), Adaptive Acquisition Framework (AAF), these programs tend to be actual full agile software development programs. These programs focus on delivering capability in a period of 2-5 years with rapid prototypes and rapid fielding with proven technology. The approach is part of the Adaptive Acquisition Framework. These programs usually do not have a Cost Analysis Requirements Document (CARD). Therefore, the analyst must search for the best statement of the program definition/requirement. Finally, determining the best sizing mechanism is the next challenge. This paper proposes unique and transferable solution to these problems.

Outline

1.0 The Cost Team

1.1 The cost team on this project was a combination of IPT personnel, NAVAIR Cost personal, and contractor support from NSI and Galorath. An integrated team is critical to the successful estimation on a large agile program. Successful cost analyses are a result of collaboration between the technical and cost personnel.

1.2 The cost team, represented in Figure 1 - The Cost Team, presented below.

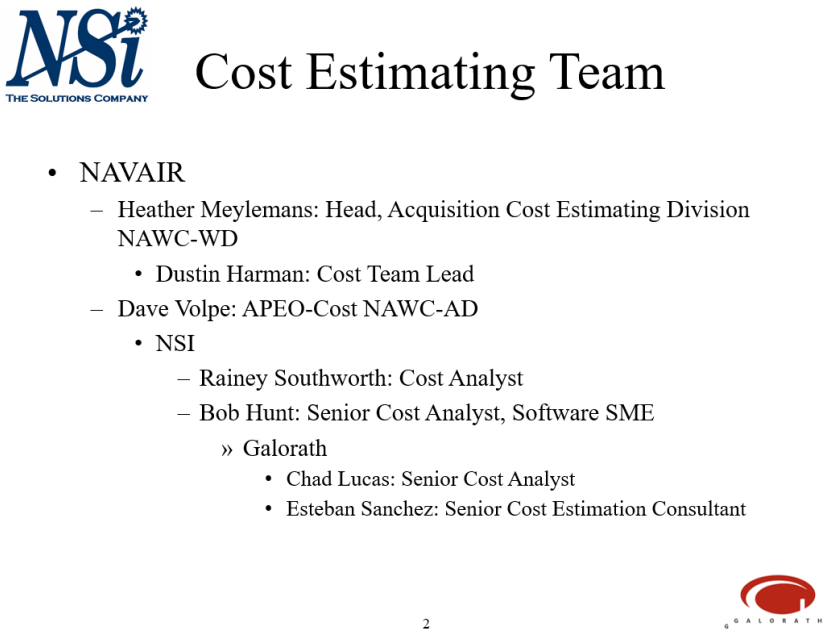

Figure 1 – The Cost Team

2.0 Problem Statement

2.1 The cost team faced several initial problems, a questionable initial estimate, no requirements document, and concerns about sizing metrics for agile software development. The problems are outlined below in Figure 3 – Challenges and Objectives. The solutions will be presented in the section below.

2.2 Comparison to an existing estimate: The program had an initial program estimate done by engineers using an analogy estimating process using T-Shirt Sizing to estimate the labor hours. This resulted in a high-level engineering judgement estimate with no hard data basis. The cost team's challenge was to develop a reliable, repeatable cost estimate for direct comparison to the initial program estimate. Initially, the only viable metric available were Agile centric, such as story points, velocity, burn up/burn down charts, etc. that were pulled from the team's data repository. Although the metrics served their purpose for measuring work accomplished, it set a limited time period to estimate to, thus creating a need to pivot to a metric that is more consistent and less dependent on team variability.

2.3 Defining the requirement: In a typical large Program of Record, there would be a Cost Analysis Requirements Document (CARD) or Estimating Technical Assurance Board (ETAB). In a truly agile development process, the requirements evolve as work is completed. The Cost team's challenge was to find a requirements statement that both matched the requirement used to develop the initial estimate and provided a sufficient basis for developing a new estimate including the new capabilities added from each subsequent update.

2.4 Developing a reliable, agreeable sizing process: As with most agile development efforts, any measure related to source line of code (SLOC) was unacceptable. The cost team's challenge was to review all sizing techniques and select the best one for this agile development program.

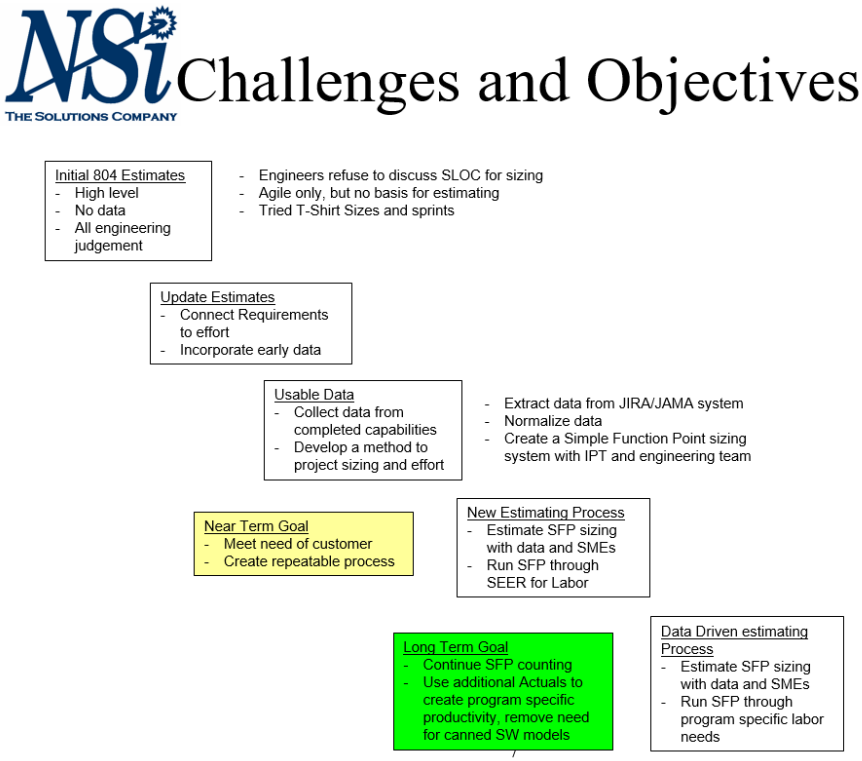

Figure 3 – Challenges and Objectives

2.5 Full Agile versus Hybrid Agile: A related problem was to determine if the program was a full agile or hybrid agile. Most large Government software development programs are Hybrid-Agile (sometimes called Scrum-Waterfall). In this development scenario, the requirement and design are done using traditional waterfall processes and the coding is done using agile processes. In a full agile process, all work is done in the agile environment. The differences are outlined in Figure 4 – Agile Development Processes. It is important to correctly define which process is most relevant to the program, as there is greater risk assigned to full agile programs. By working closely with the development team, it was clear that this program is full Agile.

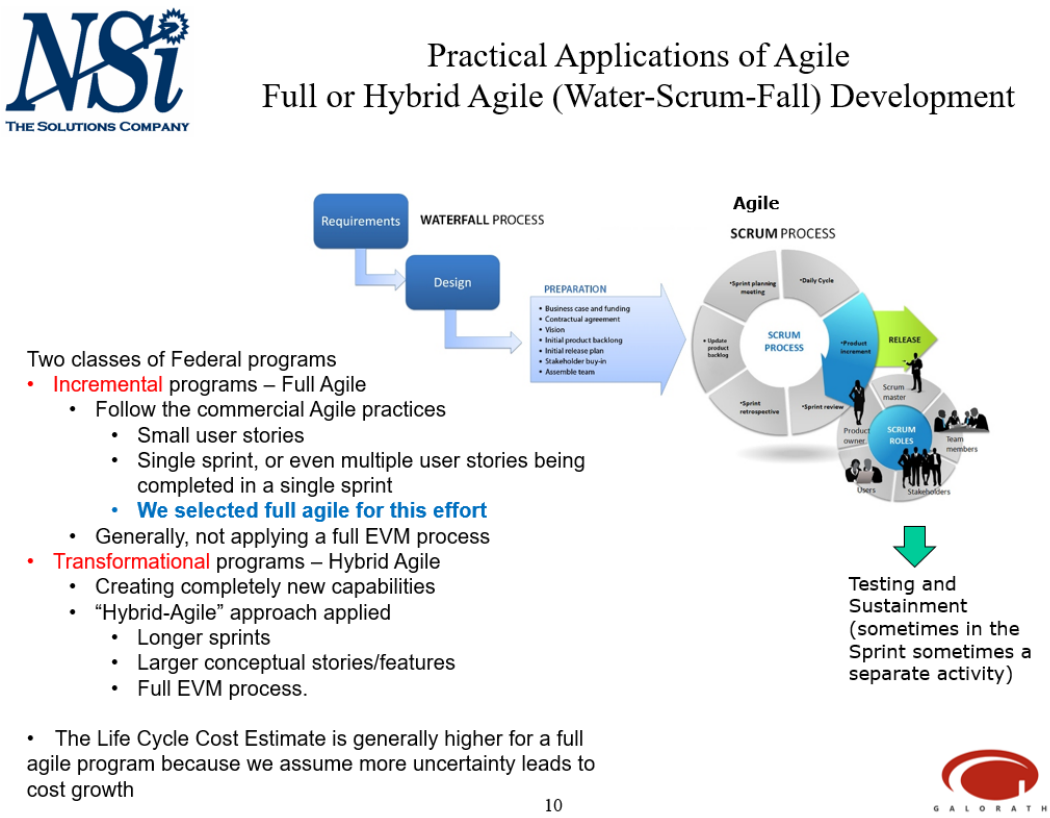

Figure 4 – Agile Development Processes

3.0 Section 804/Rapid Acquisition

3.1 Middle Tier Acquisition (MTA) Authority was granted by Congress in the FY16 National Defense Authorization Act (NDAA) Section 804.

3.2 The Middle Tier of Acquisition (MTA) pathway is used to rapidly develop fieldable prototypes within an acquisition program to demonstrate new capabilities or rapidly field production quantities of systems with proven technologies that require minimal development. The MTA pathway is intended to fill a gap in the defense acquisition system for those capabilities with a level of maturity that enables rapid prototyping within an acquisition program or fielded within five years of the MTA program start. The MTA pathway can be used to accelerate capability maturation before transitioning to another acquisition pathway or to minimally develop a capability before rapidly fielding.

3.3 The programmatic decision to move from 804 to the software acquisition pathway was made to continue rapid deployment during the Program of Record years. Due to the nature of the software, it was imperative to be able to run updates and configuration changes to the warfighter as quickly as possible, and to as many platforms as necessary. The process is outlined in the figure below, Figure 5 - Transition to SW Acquisition Pathway.

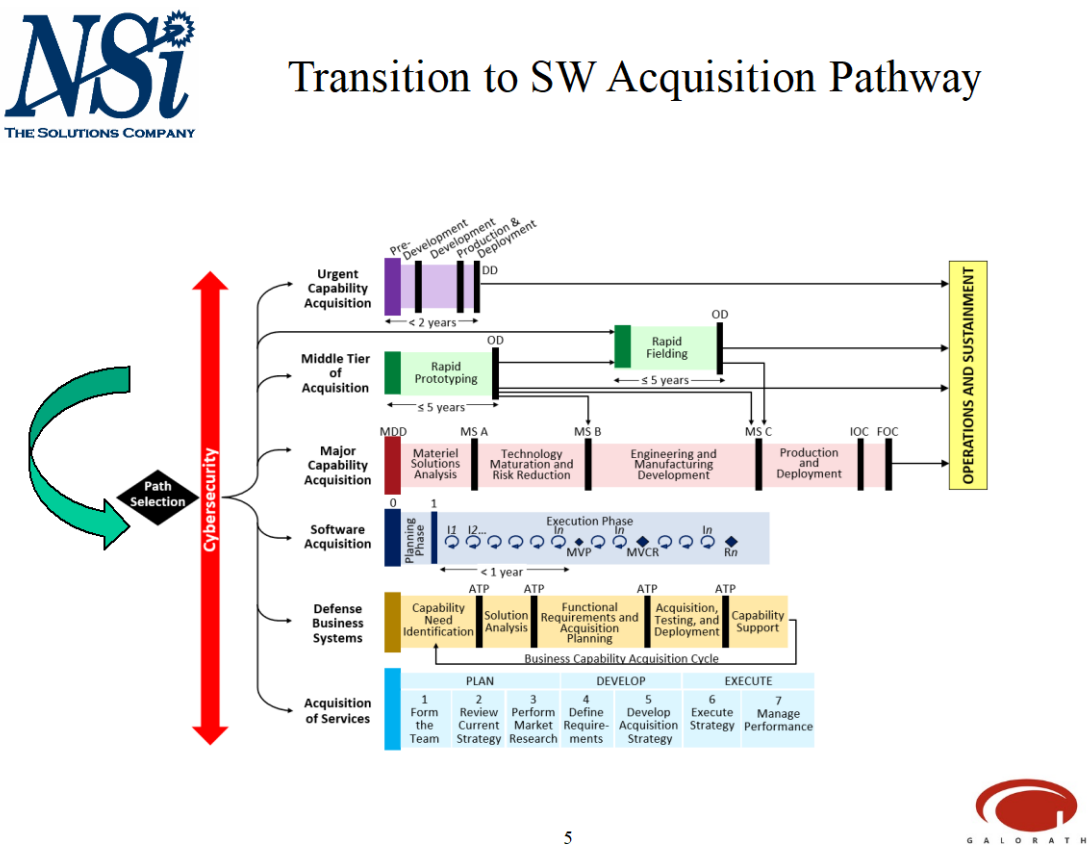

Figure 5 - Transition to SW Acquisition Pathway

4.0 Agile Metrics

4.1 Agile is an increasingly popular software building methodology. At least 71% of U.S. companies are now using Agile. Agile projects have a 64% success rate, whereas projects under the competing methodology known as waterfall only have a 49% success rate. With that in mind, Agile projects are nearly 1.5X more successful than waterfall projects. Scrum is the most popular Agile framework, with 61% of respondents from 76 countries reporting that they use it.

4.2 Figure 6 - Agile Success Rate, shows agile is more successful than waterfall but we have a long way to go before we declare success.

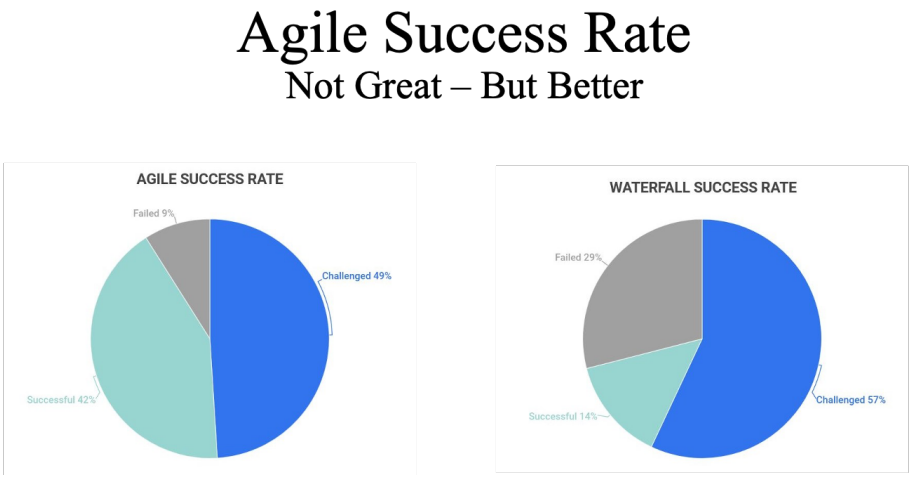

Figure 6 - Agile Success Rate

5.0 Agile Cost Process

5.1 For this cost analysis, we applied the general process outlined in the GAO Cost Manual. There are five general approaches to estimating the specific software cost.

5.1.1 Methodology 1. Many Agile programs are based on a fixed price contract. In this case the cost is labor rates times quantity.

5.1.2 Methodology 2. A simple build-up approach based on averages can be utilized. This defined as: Sprint Team Size (SS) x Sprint length x Number of Sprints.

5.1.3 Methodology 3. Structured approach based on established "velocity" (the rate at which work is completed). This methodology is most often used internally by the developer since detailed/sensitive data are available to them. The cost analyst needs several reliable iterations to apply this methodology.

5.1.4 Methodology 4. Automated Models (NEMO, SEER, COCOMO, TruePlanning, SLM, …) can be utilized. These models accept various sizing methodologies - SLOC, Function Points, User Stories, …. There models assume a fixed relationship between size and effort, e.g. Effort

=*ai*(Size Metric)*(bi)*(EAF). The results are then modified by current trends and analyses. Total effort can then be distributed over time using a mathematical probability distribution, e.g. Weibull or Rayleigh.

5.1.5 Methodology 5. An analogy/factor/complexity approach based on data generated in early iterations, e.g. T-Shirt estimation, Planning Poker, or …

5.2 For this estimate, the cost team utilized Simple Function Point analysis to estimate the size and SEER to calculate the effort. Simple Function Points provided a focus on what the software is actually doing, detached from focusing on who is doing the coding. Creating this methodology juxtaposition refines what is being delivered for the money spent, and promotes a cost-constrained roadmap.

5.3 A typical large NAVAIR program would have a Cost Analysis Requirements Document (CARD) or Estimating Technical Assurance Board (ETAB). Since this program was an 804 Rapid acquisition program, neither existed. The program is managed by an Integrated Product Team (IPT) with dynamic and evolving requirements. The cost team worked with the IPT Team to identify the most representative program description. The cost team identified a "Roadmap" that was accepted by all parties and a reasonable definition of the requirement. The development team utilized this same roadmap when their T-Shirt size methodology was applied, containing over 1300 elements.

5.4 The basic simple function point analysis was conducted at this level, and hours were estimated for each roadmap element. The function point count was converted to hours and the hours converted to dollars. Longevity wise, the program should be able to roll these elements up to describe any desired scenario (e.g., by program, platform, milestones, etc.).

6.0 Sizing Issues and Simple Function Points

6.1 Determining the appropriate size metric is still the key parameter in software cost estimation. For years, SLOC was the accepted standard, and it is still appropriate in many cases. However, with the rise of agile methodologies (and there are many), SLOC has fallen out of favor. Sizing methodologies can be separated into:

6.1.1 Physical Size: Source Lines of Code (SLOC) is an objective measure that is highly dependent on language and

programmer skill. This methodology is generally rejected by Agile developers since it was primarily utilized in Waterfall software development programs.

6.1.2 Relative Effort Size: Story Points and T-Shirt Size are examples of relative effort. Individual Agile Teams determine these measures. These metrics can be very effective/accurate for seasoned programs and agile teams. However, they are very subjective and difficult to replicate. T-Shirt Sizing was initially used by the program's IPT.

6.1.3 Functional Size: Is a standardized objective size measure that is, and can be, independently estimated and replicated. Particularly in a DoD setting, this consistency is highly sought after for long term programmatic estimation needs.

6.2 The Cost Team selected a Simple Function Points (SFP) Estimate to improve and validate the current T-Shirt/relative sizing. Simple Function Point (SFP) analysis utilization and accuracy has been validated by International Function Point User Group (IFPUG), the Department of Homeland Security (DHS), and other large DoD programs. By utilizing simple function points, the cost team was able to attach a metric to the entire project, and attach value to each effort consistently.

6.3 Figure 7- Software Size Metrics, lists many of the major sizing metrics utilized today. As noted previously, the program's derivative problem was that continual updates to the requirements created an unstable program size. Without an adaptive metric that would give fidelity to incoming projects, estimation technique relied heavily on engineering judgement, which created inconsistencies in previous data sets. Application of Simple Function Points allowed incoming capabilities to be sized quickly and consistently, bridging the gap between engineering judgement and the estimation process.

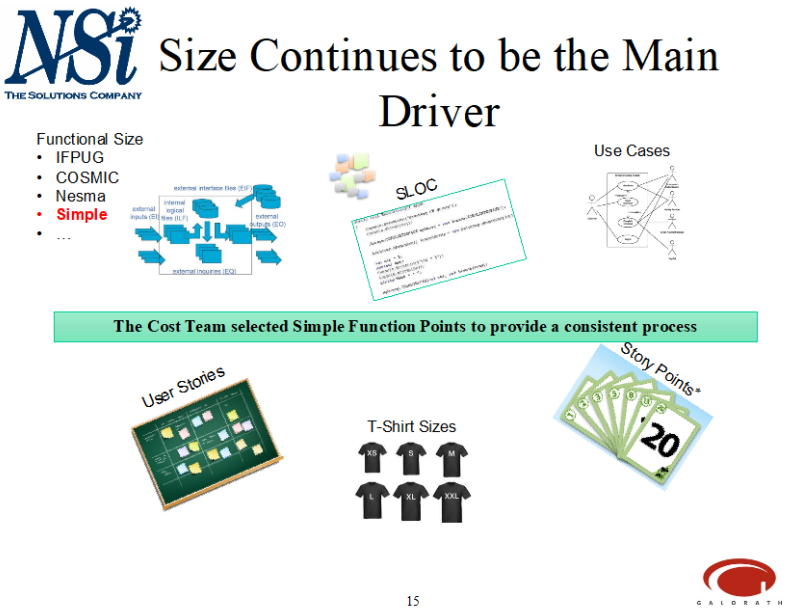

Figure 7- Software Size Metrics

6.4 Figure 8 -Simple Function Point Analysis Validation, outlines recent SFP validation efforts.

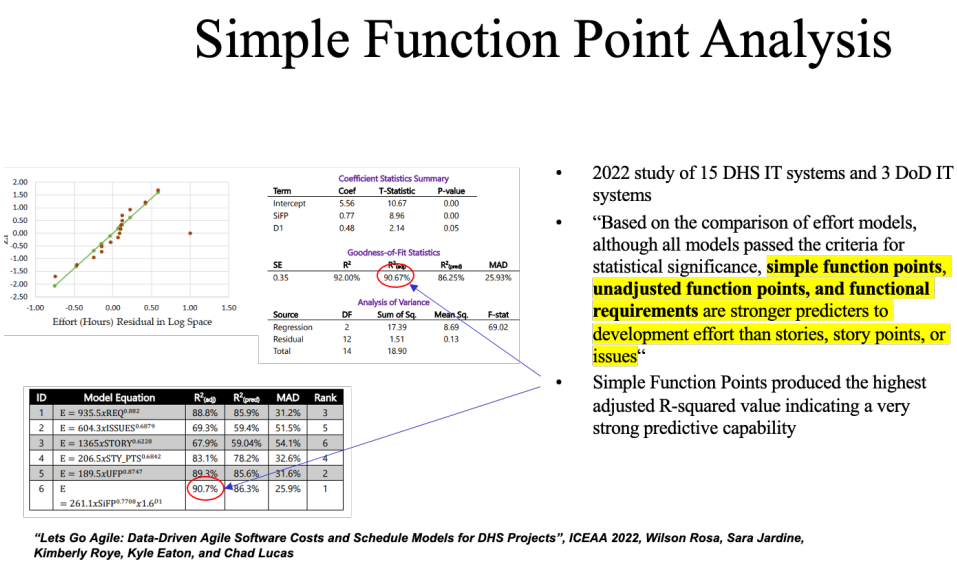

Figure 8 -Simple Function Point Analysis Validation

6.5 The Simple Function Point methodology was developed by Italian researchers and acquired by IFPUG in 2019

(https://www.ifpug.org/ifpug-acquires-the-simple-function-points-method). SFP can be performed quickly and early in a program's lifecycle using existing documents. The methodology focuses on two elementary processes – Transactions and Logical Data Groups. A comparison between traditional Function Point analyses and Simple Function Point analyses is presented below in Figure 9 – Traditional Functions Point Analysis versus Simple Function Point Analysis.

IFPUG Components	Low	Average	High
External Inputs	3	4	6
External Outputs	4	5	7
External Inquiries	3	4	6
Internal Logical Files	7	10	15
External Interface Files	5	7	10

SFPA Components	Weighting Factor
Transactions (Create, Update, Delete, Report, Read)	4.6
Logical Data Groups (Saves)	7

Figure 9 – Traditional Functions Point Analysis versus Simple Function Point Analysis

6.6 SFP analysis was appropriate for this program for several reasons.

6.6.1 SFP is currently used by other large DoD agile programs such as the Army's Integrated Personal and Pay System (IPPS-A).

6.6.2 The SFP analysis reduced the time necessary to complete the function point assessment by as much a 50%.

6.6.3 Simple Function Points are very appropriate for early-stage agile development because the sizing is based on software functionality, which is captured in early development stages.

6.6.4 Simple function points provide a consistent methodology to estimate size.

6.7 A sample of how SFP was applied to the roadmap elements is presented in Figure 10 - SFP Example, below.

Work Package	Description	Transactions	Data Stores	SFPs	Notes
Registration and management of services	Provide utility to manage registration and management of services. This service will listen for registration information such as name, IP address. and TCP/IP ports from services as they come on line. It will also provide this data to services that request it.	4	1	25.4	The description implies the creation of a new process/utility. This would require the ability to Read, Write, Delete, and Change the underlying data for a total of 4 elementary processes. Also, the ability to save/store the data is required. 4 transactions and 1 Data store total. 4 Transactions at 4.6 SFP plus and additional 7 SFP for the Data store = 25.4 SFPs
Big Data AI	Develop machine learning routines to analyze post mission data to create post mission brief.	2	0	9.2	The machine learning rountines do not involve the user. However, the analysis and creation of a briefing do require user interaction. This would be an update and report transaction, respectively. 2 Transactions at 4.6 SFP per transaction = 9.2 SFPs
Route - C2_ISR	Add additional attributes and other features to reach full C2-ISR level of functionality	0	0	0	Changes/Updates to reach a standard. NFR

Figure 10 - SFP Example

6.8 Some of the over 1300 “Roadmap” elements (basic work packages) have a zero SFP count. Items such as documentation or meeting a certain developmental standard do not require end user interaction and, as such, are not functional. There is, however, effort associated with these requirements as they add complexity to the overall work effort. Most software cost estimating models account for these hours from the parametric estimating equations derived for their historical data base. SEER estimates are further characterized by the development standards used that tune estimate to capture the required documentation and QA efforts. Non-model users might utilize the Software Non-functional Assessment Process (SNAP) to size the non-functional effort.

6.9 The team’s software estimate included:

6.9.1 System Requirements Design- Create initial system requirements and Decide which functions are allocated to software.

6.9.2 Software Requirements Analysis- Detailed software requirements analysis and synthesis.

6.9.3 Preliminary Design- Subdivide software into packages or functions, define data flows between different program components, and map design to software requirements.

6.9.4 Detailed Design- Further define software is down to single decision points.

6.9.5 Coding and Unit Test- Software programming and unit level testing performed by programmers.

6.9.6 Component Integration and Test- Integrate software units into cohesive software components, component-level testing, and integrate components into a cohesive program.

6.9.7 Program Test- Formal testing to determine compliance with requirements.

6.9.8 System Integration and Test- Program-level software-to-software integration, and software-to-hardware integration (software labor only; estimate hardware portion in SEER HW)

6.9.9 Not included in the software development estimate but included on the complete life cycle cost were Maintenance, Hardware, and Installation/Distribution/Fielding.

6.10 The estimate breaks out labor by category: Management, Business Analyst, Architect / Designer, Data Analyst (Architect), Coders / Programmers, Quality Assurance & Test Personnel, Configuration & Release Manager, and Quality Control Lead.

6.11 The summary results are presented in the Figure 11 – Summary Results and Cost Comparison. The model used our estimate of Functional Size (over 25K SFP) to estimates hours and then costs. Ultimately, there were about 100 hours per function point. Application of a streamlined rate resulted in an average cost per function point at about $16K. Full agile cost is higher than Hybrid because full agile assumes a higher degree of volatility than Hybrid agile. The 17% difference between the original estimate and the SFP estimate gives you confidence in both respective estimates.

Cost Estimate Summary	Full Agile	Hybrid Agile	IPT Estimate	% Delta Full	% Delta Hybrid
HOURS	**2,363,461**	2,063,210	**1,876,031**	**26%**	10%

7.0 Summary Results/Lessons Learned

7.1 Sizing remains a key cost driver.

7.2 Simple Function Point analysis works well for early agile programs.

7.3 Simple Function Point analysis: was consistently applied across all elements based on the element description.

7.4 The T-Shirt size methodology *may* be inconsistent since it is dependent largely on team turn-over rate and maturity.

7.5 At the top-level a 17% difference seems very reasonable and give the team confidence in the SFP and T-Shirt sizing estimate.

7.6 The functional size could be run through other models and backfired to more familiar software (e.g. NEMO, COCOMO, etc.).

7.7 An analysis of selected elements could be used to build a NAVAIR Function Point/Cost database that would be helpful in more agile development programs.

7.8 An automated tool to “rack and stack” elements would be very helpful for updates to the estimate and quick turn platform specific needs.