NSi
THE SOLUTIONS COMPANY

Sizing Agile Software Development Programs

\

Authors

Bob Hunt, NSI
Heather Meylemans, NAVAIR
Denton Tarbet, NSI
Chad Lucas, NSI
Rainey Southworth, NSI

NSi
THE SOLUTIONS COMPANY

Outline

- AGILE DEVELOPMENT Overview
- FUNDAMENTALS OF SOFTWARE ESTIMATION
- TYPES OF SIZING
- PHYSICAL SIZING
- RELATIVE SIZING
- FUNCTIONAL SIZING
- CONCLUSIONS

THE SOLUTIONS COMPANY

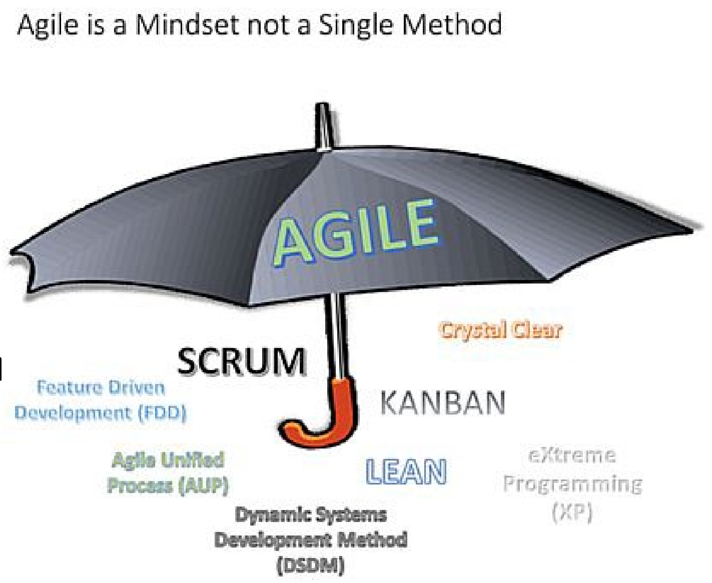
What Is Agile

- “Agile,” includes all forms of Agile and iterative development.
- Stories, features, story points, and feature points used to reflect the same concept
  - Recognizing that a “feature” typically may be used in a different context than a “story.”
  - Specifically, in large federal programs, “features” generally represent a larger concept than “stories.”
  - We believe the application of estimating, management, and tracking practices can significantly and positively impact the success and cost of federal programs.
- Two classes of federal agile software development programs
  - Programs that are evolving on an incremental basis that generally follow the commercial Agile practice
  - Large “transformational” programs creating completely new capabilities.
    - In these “transformational” programs a “Hybrid-Agile” approach is often applied with longer sprints and larger conceptual stories/features.

THE SOLUTIONS COMPANY

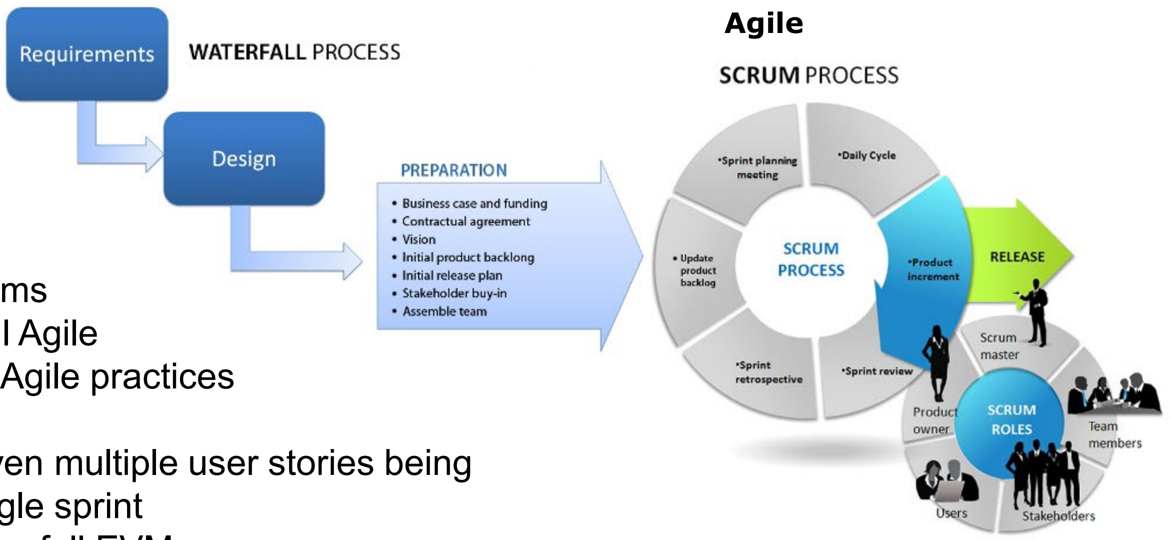
Agile is a Mindset*

- Agile refers to the methods and best practices for organizing projects based on the values and principles documented in the Agile Manifesto.
- No one way to implement Agile
  - **Kanban**
  - **Scrum**
  - **Extreme Programming (XP)**
  - **Feature-driven development**
  - **Dynamic Systems Development Meth**
  - **Crystal**
  - **Lean**
  - **Adaptive Project Framework**


Agile is a Mindset not a Single Method
AGILE
Crystal Clear
SCRUM
Feature Driven Development (FDD)
KANBAN
Agile Unified Process (AUP)
LEAN
eXtreme Programming (XP)
Dynamic Systems Development Method (DSDM)

* From David DeWitt of Galorath

Practical Applications of Agile
Full or Hybrid Agile (Water-Scrum-Fall) Development



Two classes of Federal programs

- Incremental programs – Full Agile
  - Follow the commercial Agile practices
    - Small user stories
    - Single sprint, or even multiple user stories being completed in a single sprint
  - Generally, not applying a full EVM process
- Transformational programs – Hybrid Agile
  - Creating completely new capabilities
  - "Hybrid-Agile" approach applied
    - Longer sprints
    - Larger conceptual stories/features
    - Full EVM process.

Testing and Sustainment (sometimes in the Sprint sometimes a separate activity)

Agile Software Development Metrics*

- Attempts to quantify the cost of software failures don't agree on percentages, but they generally agree that the number is large.
- The Standish Chaos Report is probably the most well-known of these studies. It defines success as projects delivered within budget, on schedule, and with expected functionality.
  - 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**
  - **Waterfall only has a 49% success rate.**
  - Agile projects are nearly **1.5X more successful** than Waterfall.
  - **Scrum is the most popular Agile framework**, with 61% of respondents from 76 countries reporting that they use it
- Agile is better; but still not great

*Jack Flynn, 16 Amazing Agile Statistics (2023),

Estimating Methodologies

Methodology 1: Many Agile programs are fixed price (labor rates times quantity)

Methodology 2: Simple Build-up approach based on averages can be defined as:
Sprint Team Size (SS) x Sprint Length (Sp time) x Number of Sprints (# Sprints)

Methodology 3: Structured approach based on established "velocity" – most often used internally by the developer since detailed/sensitive data are available to them – need several iterations

Methodology 4: Automated Models (NEMO, SEER, COCOMO, TruePlanning, SLIM, …) approach based on a size metric (Physical, Relative, or Functional size)

- Assume there is a fixed relationship between size and effort, e.g. Effort =*A**(Size Metric)^*B*C* where A is a constant, B is the non-linear scaler, and C is a combination of Environmental factors
- Results are then modified by current trends and analyses
- Total effort can be distributed by a mathematical model; e.g. Weibull, Rayleigh

Methodology 5: Analogy/Factor/Complexity approach based on data generated in experience or actual iterations, e.g. T-Shirt, *…*

Fundamentals of Software Estimation

- In the late 1960's and early 1970's, analytic equations based on Lines of Code data were derived by Putnam, Jenson, Boehm, Galorath, and others.
- There was general agreement that effort was a function of size;
  - Effort (months)=A*(size of the program)**B
  - Early COCOMO formula, E=3.2*(KSLOC)**1.05

  (Today the exponent varies in commercial models from about 0.9 to 1.2).
- Over time databases, software tools, productivity factors, and complexity factors have significantly effected the fundamental estimation equations and the models have become more complex.
- Automated models are adjusted to account for Agile practices.
- SEER and TruePlanning examples are presented in the backup.

Model Estimation Summary

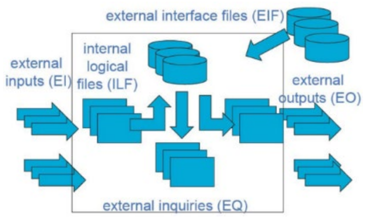
- Size is a key input
- Over time factors related to complexity and productivity have become a greater influencer in the models.

Three Broad Categories of Software Size

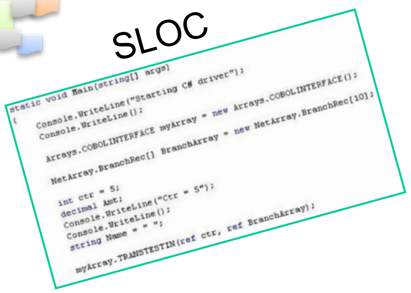
- Physical Size – Source Lines of Code (SLOC)
  - An objective measure
  - Highly dependent on language and programmer skill
  - Generally rejected by Agile developers since developed and designed for the Waterfall development method.
  - SLOC counts can be automated reliably for historical data collection.
- Relative Effort Size
  - Story Points, T-Shirt Sizing
  - Relative measure determined by Software Developers
  - These measures are generally familiar to Agile Teams
- Functional Size
  - Objective Size measure, standardized
  - Can be independently estimated
  - There are multiple Functional Sizing Metrics

Size Continues to be a Driver in Software Effort Estimation

Functional Size
- **IFPUG**
- **COSMIC**
- **NESMA**
- **Simple**
- …

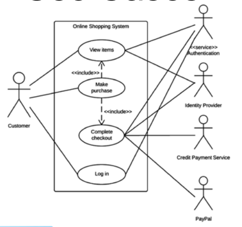


SLOC



```
static void Main(string[] args)
{
    Console.WriteLine("Starting C# driver");
    Console.WriteLine();

    Arrays.COBOLINTERFACE myArray = new Arrays.COBOLINTERFACE();

    NetArray.BranchRec[] BranchArray = new NetArray.BranchRec[10];

    int ctr = 5;
    decimal Amt;
    Console.WriteLine("Ctr = 5");
    Console.WriteLine();
    string Name = " ";

    myArray.TRANSTESTIN(ref ctr, ref BranchArray);
```

Use Cases



Story Points*

User Stories

T-Shirt Sizes

BUT

Environmental, Productivity, Complexity, and other effort drivers are critical.

Physical Sizing

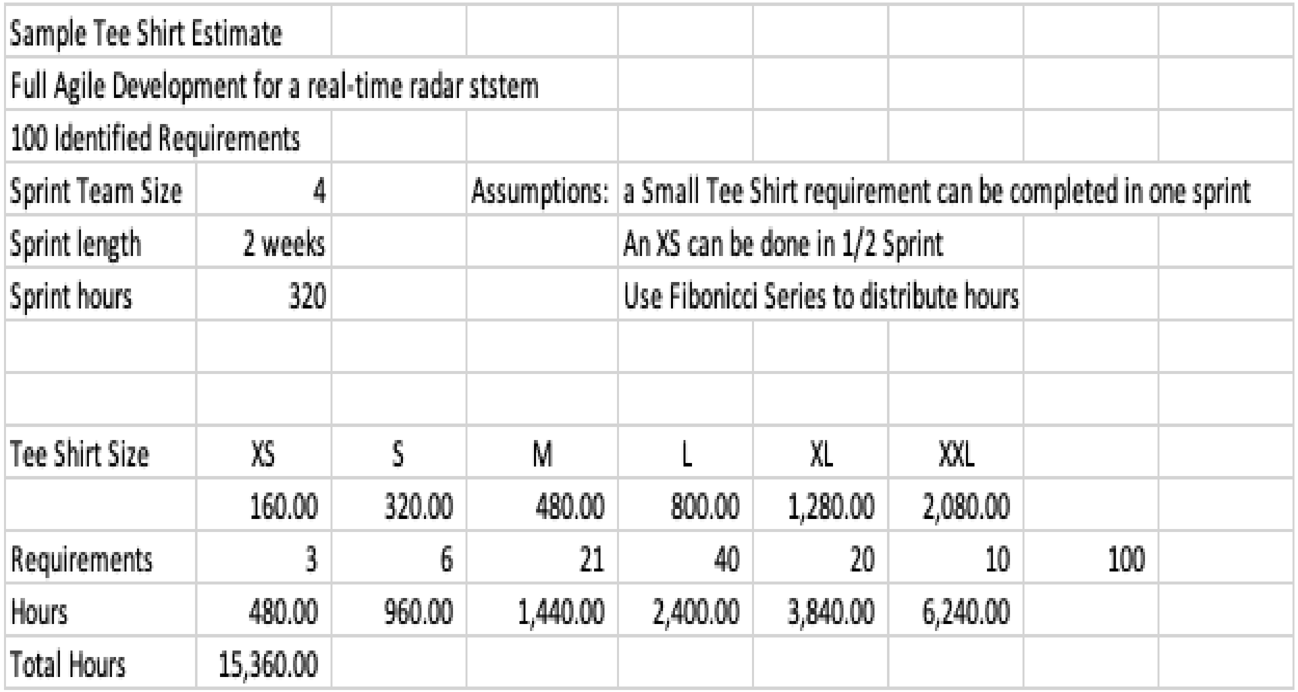
- **Source Lines of Code (SLOC):** the total number of lines of source code in a project – KSLOC, ESLOC, … (be sure your code measure is the same one used in the model)
- Can use code counters like USC's UCC or the Government (UCC-G)
- SRDR is a good data source
- **Advantages:**
  - Accepted and is used in many automated models like COCOMO
  - SLOC is easily quantified
  - SLOC is being used today to successfully estimate and manage Agile programs
- **Disadvantages:**
  - Different programming languages, programmer experience, and automated tools effect the code count
  - When platforms and languages are different, LOC can be difficult to normalize
  - For new programs, SLOC must be estimated; usually by analogy to similar programs
- Size is normally estimated as low, most likely, and high number
- Therefore, a distribution can be developed to estimate at the desired confidence level, e.g., the 70% level

Relative Effort Size

- **Relative Effort Size** is determined by the development team
- Common relative measure are Story Points, Feature Points, Epics, T-Shirt Size, Use Case, User Stories, …
- The effort associated with each of these measures is based on expert opinion or analogy from previous work
- **Advantage**
  - These are metrics that software developers are familiar and comfortable with
  - They are typically project or team specific
  - They have been successfully applied (example on the following slides)
- **Disadvantage**
  - There is often no "formal"/consistent methodology
  - They often only reflect effort directly related to coding
  - It is difficult to extrapolate from project to project and especially for organization to organization due to process improvements and old data
  - Limited historical databases

Successful Relative Effort Size Example

- In a recent large Federal Agile development program, the T-Shirt hour estimate was compared to SEER-SEM hour estimate – when the estimates were reconciled there was a 5% difference in the total hours estimated
- A consistent methodology was applied.



Sample Tee Shirt Estimate								
Full Agile Development for a real-time radar ststem								
100 Identified Requirements								
Sprint Team Size	4		Assumptions:	a Small Tee Shirt requirement can be completed in one sprint				
Sprint length	2 weeks			An XS can be done in 1/2 Sprint				
Sprint hours	320			Use Fibonicci Series to distribute hours				
Tee Shirt Size	XS	S	M	L	XL	XXL		
	160.00	320.00	480.00	800.00	1,280.00	2,080.00		
Requirements	3	6	21	40	20	10	100	
Hours	480.00	960.00	1,440.00	2,400.00	3,840.00	6,240.00		
Total Hours	15,360.00							

- RECENT USAF SME ESTIIMATES:S = Small (320 Hours); M = Medium (1600 Hours); L = Large (2880 Hours), and XL = Extra Large (6400 Hours)

THE SOLUTIONS COMPANY

Functional Size

- Functional Size Measurement (FSM) is a technique for measuring software in terms of the **functionality it delivers**
- Functional Size is primarily used at the planning stage for input into project resource estimation calculations for cost, effort and schedule
- There are multiple Functional Sizing Metrics - **COSMIC** - **IFPUG/SFP (SiFP)** – **NESMA**
- There are minimal differences between the various size counts
- International Function Point User Group (IFPUG) has been the most common functional size measure in the U.S.
- Recently, IFPUG adopted Simple Function Points (SFP or SiFP) and this measure is quickly gaining traction in the estimating community

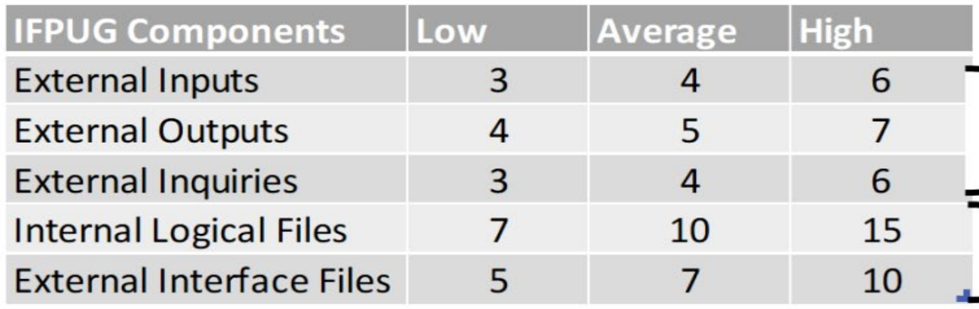
THE SOLUTIONS COMPANY

Functional Size

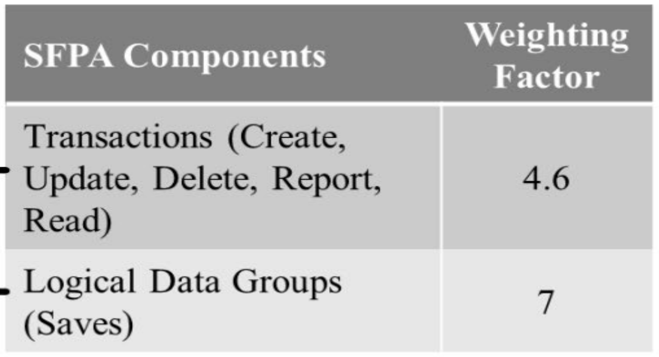
- **Advantages**
  - Independent of the technology
  - Estimated from statements of early requirements
  - Objective, repeatable and verifiable
  - Enables benchmarking
- **Disadvantages**
  - Can require people with the expertise to carry out this activity.
  - Can take some time and has associated costs

Simple Function Points SFP

- The Simple Function Point method estimates a software's functional size based on quantifying its business functions / transaction types, system interfaces, and other functional requirements from high-level acquisition documentation
- The SFP method developed by Italian researchers and acquired by IFPUG in 2019 (https://www.ifpug.org/ifpug-acquires-the-simple-function-points-method)
- The SFP count can be performed quickly and early in a program's lifecycle using existing documents
- Focuses on two elementary processes – transactions and logical data groups



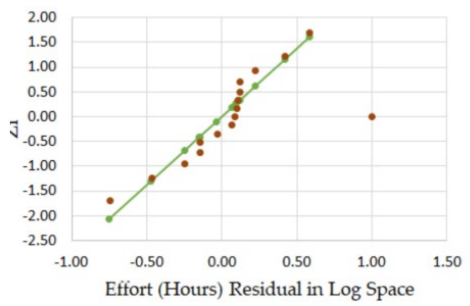
IFPUG Components	Low	Average	High
External Inputs	3	4	6
External Outputs	4	5	7
External Inquiries	3	4	6
Internal Logical Files	7	10	15
External Interface Files	5	7	10



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

THE SOLUTIONS COMPANY

Simple Function Point Analysis (Validated by a DHS Study referenced below)



Coefficient Statistics Summary

Term	Coef	T-Statistic	P-value
Intercept	5.56	10.67	0.00
SiFP	0.77	8.96	0.00
D1	0.48	2.14	0.05

Goodness-of-Fit Statistics

SE	R^2	$R^2_{(adj)}$	$R^2_{(pred)}$	MAD
0.35	92.00%	90.67%	86.25%	25.93%

Analysis of Variance

Source	DF	Sum of Sq.	Mean Sq.	F-stat
Regression	2	17.39	8.69	69.02
Residual	12	1.51	0.13	
Total	14	18.90		

ID	Model Equation	$R^2_{(adj)}$	$R^2_{(pred)}$	MAD	Rank
1	$E = 935.5xREQ^{0.882}$	88.8%	85.9%	31.2%	3
2	$E = 604.3xISSUES^{0.6879}$	69.3%	59.4%	51.5%	5
3	$E = 1365xSTORY^{0.6228}$	67.9%	59.04%	54.1%	6
4	$E = 206.5xSTY\_PTS^{0.6842}$	83.1%	78.2%	32.6%	4
5	$E = 189.5xUFP^{0.8747}$	89.3%	85.6%	31.6%	2
6	$E = 261.1xSiFP^{0.7708}x1.6^{D1}$	90.7%	86.3%	25.9%	1

- 2022 study of 15 DHS IT systems and 3 DoD IT systems
- "Based on the comparison of effort models, although all models passed the criteria for statistical significance, **simple function points**, **unadjusted function points, and functional requirements** are stronger predicters to development effort than stories, story points, or issues"
- Simple Function Points produced the highest adjusted R-squared value indicating a very strong predictive capability

"Lets Go Agile: Data-Driven Agile Software Costs and Schedule Models for DHS Projects", ICEAA 2022, Wilson Rosa, Sara Jardine, Kimberly Roye, Kyle Eaton, and Chad Lucas

THE SOLUTIONS COMPANY

Military Requirements

- Capers Jones, Estimating Software Costs, Second Edition, page 389

  "Military software requirements are usually the most precise and exacting of any class of software. This is due to the long-standing requirement of traceability…. Although these military requirements documents are large and sometimes ambiguous, the specificity and completeness of the military software requirements makes it easier to derive function point totals than for any other kind of software application."

- Might question this for emerging DoD programs. In those programs capabilities are defined, developmental requirements are derived by developer to best provide the capability and then demonstrations are provided to validate the capability.
- Most models that backfire use the Unadjusted (raw) function point count and allow the estimate to take care of the non-functional (SNAP) requirements.

Automated FP Counters

- Automated Function point counters use AI (Natural Language Processing (NLP) and a robust rules set to reduce the time it takes to inspect and estimate the functional size of each requirement.
- The DHS CADE Simple FP model, SISe, has a list of 143 keywords.
- Two examples are ScopeMaster and Cadence, the actual lists are product sensitive and are revised annually.
- The automated counting models do the initial heavy lifting, but some manual review is required.
  - Often a large number of military requirements show a “zero” function count when there is obviously work required.
  - A recent large Federal program had a manual function count of over 25,000 and the initial automated count was about 18,000.
  - When the ”zero” elements were evaluated the two counts came within 8%

THE SOLUTIONS COMPANY

Cost to Calculate Function Points

- A “Certified” function point counter (IFPUG, COSMIC, Nesma) is estimated at $100 to $200 an hour.
- Assume 15 IFPUG FP/Hr (from Capers Jones)
- Then the cost per function point is between $7 per and $14 Per Function Point
- Therefore a 10,000 IFPUG count should cost between $70 K and $140 K
- Experience shows that a SFP count takes significantly less time than a full IFPUG count, and an automated counter would reduce the time even further.

NSi THE SOLUTIONS COMPANY

Functional Size Issues

- Once the Function Point Count (FPC) is established, the count must be converted in effort hours/person months, etc.
  - International Software Benchmarking Standards Group (ISBSG) offers a good commercial database (it may contain only successful projects)
  - Currently no large Federal/National Security database is available. (The SRDR is beginning to collect this data.)
  - Analogy to similar programs
  - Most commercial models "backfire" the FPC into SLOC.
- When the FPC is completed, some requirements will have a "0" FPC. 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.
  - Non-model users might utilize tools like SNAP to account for these non-functional hours.

THE SOLUTIONS COMPANY

Backfiring FP to SLOC

- There are multiple data sources (Capers Jones, Galorath QSM, Unison) that offer backfiring tables.
- Correct backfiring requires different metrics for each type of FP (Applied Software Measurement 3rd Edition,, Capers Jones, page 80)
- Below is a **partial** table from QSM

	QSM SLOC/FP Data			
Language	Avg	Median	Low	High
ABAP (SAP) *	28	18	16	60
ASP*	51	54	15	69
Assembler *	119	98	25	320
Brio +	14	14	13	16
C *	97	99	39	333
C++ *	50	53	25	80
C# *	54	59	29	70
COBOL *	61	55	23	297

- For 3rd Generation Languages, 50 SLOC per FP is often used

Summary/Recommendation

Summary

- All size metrics can provide meaningful estimates when used appropriately
  - The team documented the successful utilization of SLOC, FP, and T- Shirt sizing in recent Agile Development Programs
- It is important to ensure an Apples-to-Apples comparison is made
- Successful models (like the Navy's NEMO) took a basic estimating model (COCOMO) and built a specific database to estimate their programs
- No direct FP to hours model exists, although Boehm Center for Systems and Software Engineering (BCSSE) at USC is working on a COCOMO III release addressing the conversion from Function Points to hours

Recommendations

1. Develop a model with a formulation something like; Effort = A(FP)^b)C

   Where: A = a "complexity" modifier
   FP = number of function points
   b = a derived exponent
   C = a "productivity" modifier

2. Develop/validate a National Security FP database

Backup

- BACKUP SLIDES

THE SOLUTIONS COMPANY

SEER by Galorath

Model is based on the Jensen SEER/Sage equation

$$S_E = C_{TE} * (K)^{1/2} * t_d$$

Where: S_E = Product Size in ESLOC

C_{TE} = Effective Technology Constant (based on cost driver inputs)

K = total software life-cycle effort in person-years

t_d= development time in years

Software development effort = 0.3945 * K

SEER Effort Formula:	$K = (S_E / (C_{TE})(t_d))^2$
SEER Schedule Formula:	$t_d = D^{-0.2} (S_E / C_{TE})^{0.4}$

Where:

D = Staffing Complexity Constant (i.e., How hard is it to get the staff required to code this type of software?). Note that the default value for D = 15

Unison Cost Engineering TruePlanning for Software

TruePlanning® CER:

$$Effort\ (Hours) = A * Size\ \hat{}B$$

Where:

A = influence of the drivers in the model (Functional Complexity, Technology, People, Reuse, Organizational Productivity, etc. - 33 numerical cost drivers + 10 nominal cost drivers)

Size = software size in specified Size Units (SLOC, IFPUG FP, COSMIC FP, etc.)

B = economy or diseconomy of scale as a function of Organizational Productivity (B ranges between 1.077 and 1.117)

Example: Military Avionics in SLOC with default values: $Effort = 0.252 * Size$ 1.107

$$Schedule\ (Months) = C * Effort\ \hat{}0.33$$

Where:

C = efficiency/inefficiencies in developing the software based on model inputs (Functional Complexity, Technology, People, Organizational Productivity, etc., 33 numerical drivers and 19 nominal drivers – C ranges from 0.41 to 0.9)