estimate

estimate • analyze • plan • control

Estimating Agile Software Development

Software Cost Estimating for Iterative/ Incremental Development Programs
Agile Cost Estimating

ICEAA June 2015

SEER by GALORATH

- Software versus Hardware estimating
- Fundamental assumptions of current Software Estimating Models
- Iterative and Incremental Development (IID) Programs/Agile Software Development Processes
- Size Metrics
- Software Estimating Processes
- Issues for Program Managers
- Summary

Hardware vs Software Estimating

- We make Software Estimating "seem" to be different, but -
- Both use multiple estimating techniques
- Both use similar techniques:
  - Expert judgment – utilize SME inputs for a task
  - Analogy – estimate based on past examples
  - Parametric – mathematical relationships
- Top down – estimate total cost and allocate
- Bottom up – estimate each detailed tasks/CSCIs

Most estimates use a blend of methods

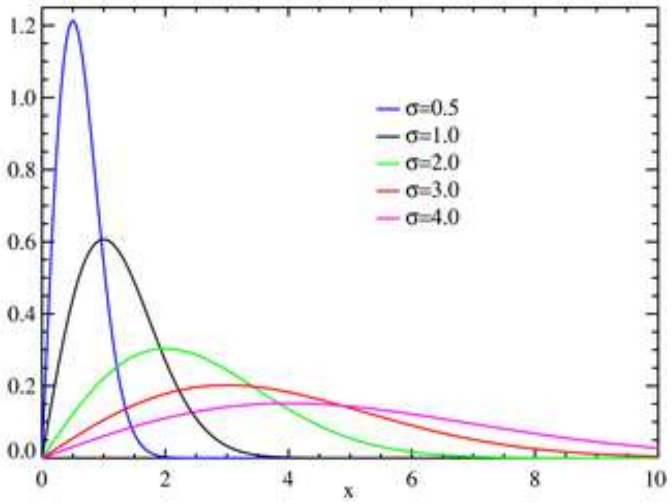
Fundamental Assumptions

Fundamental assumptions of current Software Estimating Models

- There is a fixed relationship between size and effort

$$Effort^{0.5} \times Time = \frac{Size}{Technology}$$

- Total effort can be distributed by a mathematical model



Software Development

- While there are many approaches to Software Development, they can generally be placed into 2 categories:
  - Plan Driven – following a version of the Waterfall Development Process
  - Iterative Driven – following a version of the Agile Development Process
- Plan Drive programs have an assumption of some reliable/realistic size metric, for example:
  - Source Lines of Code (SLOC)
  - Function Points
  - Use Cases, User Stories, Web Pages

Software Development

- Iterative Drive programs, by nature, start with a less well-defined metric
  - Therefore, they may require alternative estimating approaches
- This briefing will focus on the challenges of estimating an iterative program using Agile software development
- In practical experience the terms iterative, incremental and agile may be used interchangably

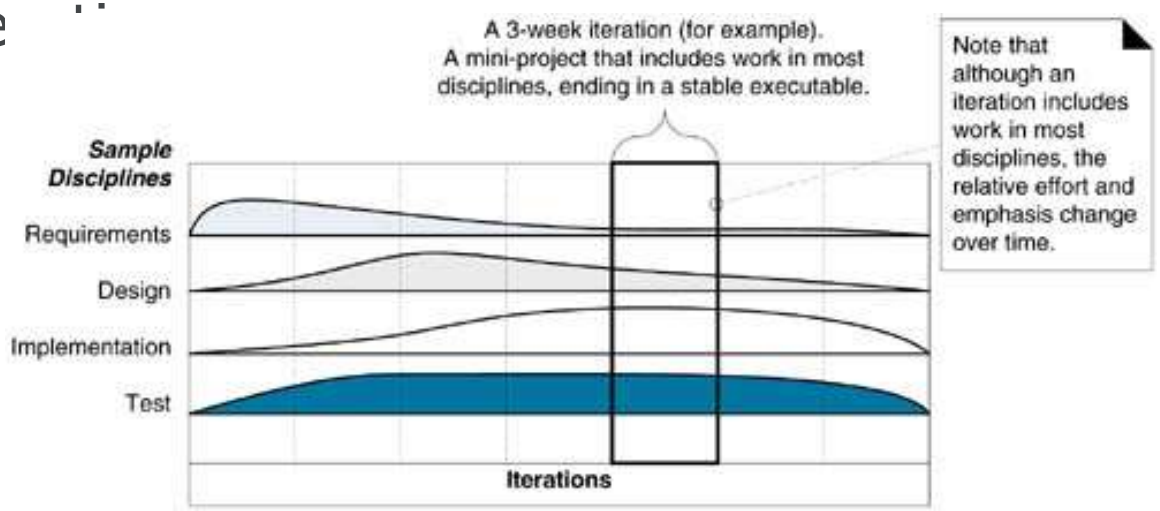
While Incremental/Agile programs say they do not have development metrics, I have almost always found them in the development room

IID Programs' Key Terms

- **IID** is an approach to building software in which the overall lifecycle is composed of iterations or sprints in sequence
  - Each Iteration is a self-contained mini project
  - It grew out of the increased application of Agile Development techniques
- In many defense programs, **increments** are 6 -12 months in length and each increment is composed of multiple **iterations/sprints** of 1-6 weeks[1]
- Time-boxing is the practice of fixing the iteration or increment dates and not allowing it to change
- This approach is gaining favor in large federal programs

Each Iteration/Sprint is a Mini Project

- Each iteration/sprint includes production-quality programming, not just, for example, requirements analysis
  - The software resulting from each iteration/sprint is not a prototype or proof of concept, but a subset of the final system
- More broadly, viewing an iteration as a[1] self-contained mini project, activities in many disciplines (requirements analysis, testing, etc.) occur within a single ite[2]



IID

- Although IID is in the ascendency today, it is not a new idea
  - 1950s "stage-wise Model" – US Air Defense SAGE Project
  - IBM created the IID method of Integration Engineering in the 1970s
- IID Programs tend to be less structured in the beginning, and therefore reliable estimates of cost and schedule may not be available until 10-20% of the project is complete[4]

(in a recent program I saw a cost variance during the first 4 increments of 45% per size metric)

- The current emphasis on agile software development processes maps directly into the IID Concept

What is Agile Software Development?

SEER by GALORATH

- In the late 1990s, several methodologies received increasing public attention
- Each had a different combination of old, new, and transmuted old ideas, but they all emphasized:
  - Close collaboration between the programmer and business experts
  - Face-to-face communication (as more efficient than written documentation)
  - Frequent delivery of new deployable business value
  - Tight, self-organizing teams
  - And ways to craft the code and the team such that the inevitable requirements churn was not a crisis

Manifesto for Agile Software Development

- "We are uncovering better ways of developing software by doing it and helping others do it
- Through this work, we have come to value:
  - Individuals and interactions over processes and tools
  - Working software over comprehensive documentation
  - Customer collaboration over contract negotiation
  - Responding to change over following a plan
- That is, while there is value in the items on the right, we value the items on the left more"

Principles behind the Manifesto

- Principles of Agile Developers:
  - Our highest priority is to satisfy the customer through early and continuous delivery of valuable software
  - Welcome changing requirements, even late in development
    - Agile processes harness change for the customer's competitive advantage
  - Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale
  - Business people and developers must work together daily throughout the project
  - Build projects around motivated individuals
    - Give them the environment and support they need, and trust them to get the job done
  - Working software is the primary measure of progress

Principles behind the Manifesto

- Principles of Agile Developers (continued):
  - The most efficient and effective method of conveying information to and within a development team is face-to-face conversation
  - Agile processes promote sustainable development
    - The sponsors, developers, and users should be able to maintain a constant pace indefinitely
  - Continuous attention to technical excellence and good design enhances agility
  - Simplicity, the art of maximizing the amount of work not done, is essential
  - The best architectures, requirements, and designs emerge from self-organizing teams
  - At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly [8]

Common Myths about Agile

SEER by GALORATH

Myth	Reality
Silver bullet / magic	Actually very hard work!
Has no planning / documentation / architecture	Just the minimum possible
Is undisciplined or a license to hack	Disciplined, business driven work
Is new and unproven / just a fad / not being used by industry leaders	Not anymore. Many large and small organizations using it
Only good for small projects	Also used successfully on medium and large projects

Differences of Agile and Non-Agile

Agile	Non-agile
Prioritize by value	Prioritize by *dependency*
Self-organizing teams	*Managed* resources the minimum possible
Team focus	*Project* focus
Evolving requirements	*Frozen* requirements
Change is natural	Change is *risky*

- Recent observations regarding the utilization of Agile development approaches within the Federal Government:
  - May work best when the project is more requirements-driven than schedule-driven
  - Beginning to see common usage in Department of Defense (DoD) unclassified (e.g. Marine Corps) and classified programs (e.g. Naval Reconnaissance Office [NRO])

Differences of Agile and Non-Agile

- Recent observations regarding the utilization of Agile development approaches within the Federal Government (continued):
  - Being talked about within emerging National Aeronautics and Space Administration (NASA) projects
  - Being used in DHS
  - It sounds very much like what we called "rapid prototyping"
  - More common than is being recognized

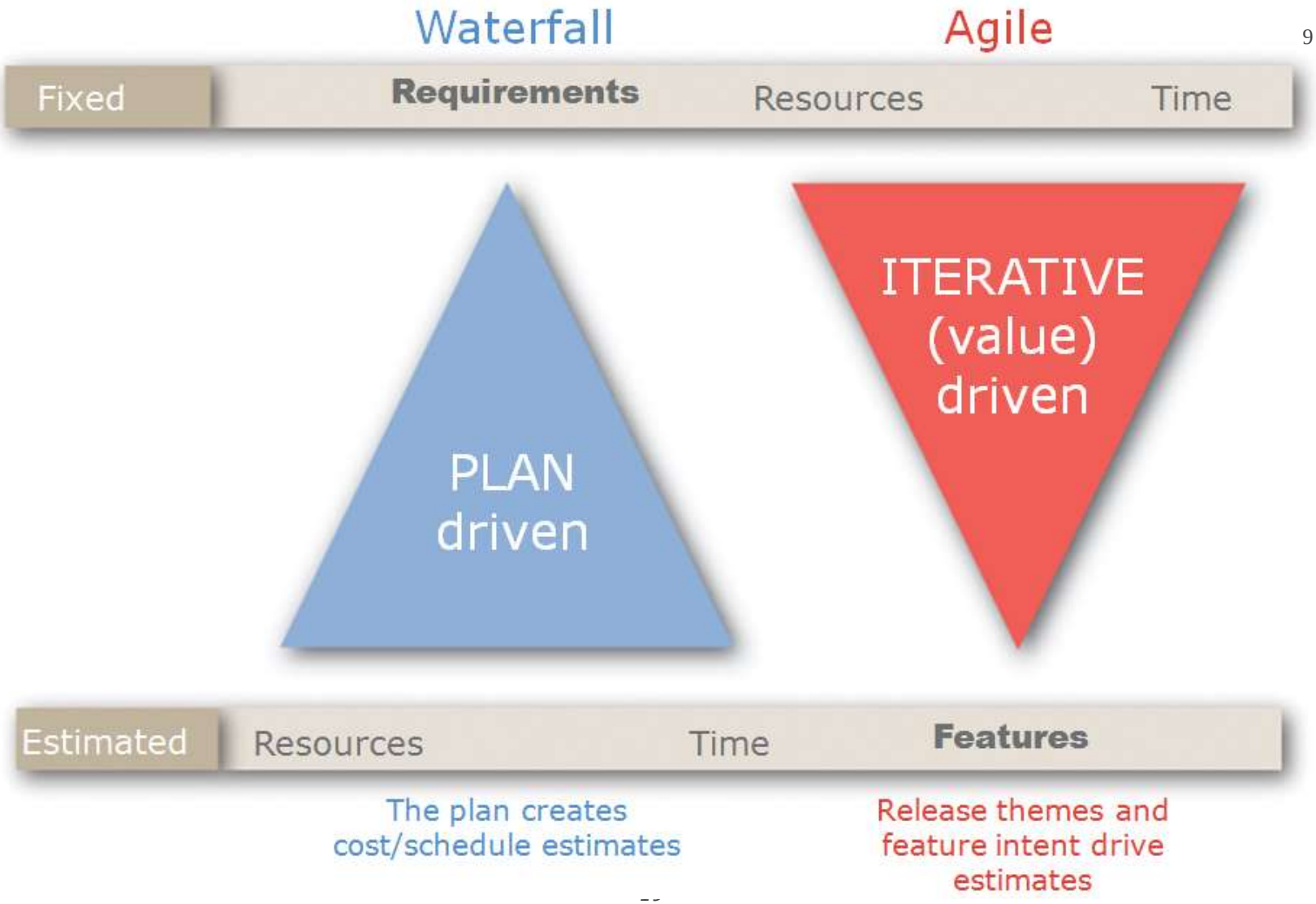
Welcome to Agile

- What is an agile development approach?
- Depends on the *flavor*:
  - Agile Modeling
  - Lean Development (LD)
  - Adaptive Software Development (ASD)
  - Exia Process (ExP)
  - Scrum
  - eXtreme Programming (XP)
  - Crystal methods
  - Evolutionary – EVO
  - Feature Driven Development (FDD)
  - Dynamic Systems Development Method (DSDM)
  - Various Unified Processes (UP): agile, essential, open
  - Velocity tracking, and more!

What do they have in common?

- Agile projects are focused on key business values
  - What does the client really, really, *really* want?
  - Deliver what the client wants at the end of the project, not what the client wanted at the beginning of the project
- They all contain a project initiation stage (aka planning)
  - Project scope, constraints, objectives, risks are all officially documented
- Short (very short) development of chunks of features/stores/requirements/needs/desires (aka sprints)
- Constant feedback
  - The one place where we can actually find short meetings
- Customer participation is MANDATORY or no-go!
- Refactoring; as in, do it again and this time get it right, or better

The Agile Paradigm Shift



What do the Models Say?

SEER by GALORATH

Comparing Agile to Traditional Development Methods*

Development Type	Schedule Months	Effort Hours	Delivered Defects	Peak Staff	Functions per month
Agile Project	10.9	5145	13.00	4.51	8.81
Waterfall	12.1	6807	20.00	6.00	6.87
RUP	11.8	6020	16.00	4.91	7.77
Spiral	11.9	6066	19.00	4.95	7.71
Object Oriented	12.1	6543	19.00	5.40	7.15

What is driving these "apparent" reductions?

Development Type	Schedule Months	Effort Hours	Delivered Defects	Peak Staff	Functions per month
Agile Project	-	-	-	-	-
Waterfall	12%	32%	54%	33%	78%
RUP	9%	17%	23%	9%	88%
Spiral	9%	18%	46%	10%	88%
Object Oriented	11%	27%	46%	20%	81%

* Client Server Platform, Transaction Processing Application, using Commercial High Standards Project Size set to 250 Function Points. Calculated Using SEER for Software

Other Current Research

Empirical evidence indicates development costs may be reduced by 10 to 20 percent for Iterative Driven Programs. In a "The Raytheon Agile Journey" a presentation by Cindy Molin (Director, SW Engineering) and Katherine (K) Sementilli (Deputy, SW Engineering), Raytheon Missile Systems on June 22, 2012 the following efficiencies based on agile development are observed (based on over 250 projects and over 5 million ELOCs):

Agile Development Results

- 20% of Raytheon SW Engineering Development Productivity
- 25% productivity increase Agile vs Non-Agile
- 10% variability reduction Agile vs Non-Agile
- 50% faster for Agile vs Non-Agile
- Time on task for an average work day 30% more for Agile vs Non-Agile

Scrums and Sprints

* I have Use Case, Feature Point, and other metrics for specific agile development programs, but I am not sure they are transferable

- Scrum Size:
  - 1-10 people (have seen up to **20**)
- Sprint Length:
  - 1-6 weeks (have seen up to **13** weeks) *(13 conveniently give 4 sprints per year*)
- Story Points* per Sprint:
  - 6-9 Story Points per Sprint
- There seems to be a real avoidance of using Function Points or SLOC in many of these efforts.
- (*But trust me a size metric exists somewhere within the development community*)

Size Metrics

- Very few software development projects start without some concept/metric of how long and how much time the development will take
- The Estimator's challenge is to "quiz" that out of the metric
- I have always felt more confident in my answer when the estimation team's size metric/proxy and the development team's effort proxy were the same
- I believe there is more estimation error from poor sizing than from estimation models or processes
- Common size metrics – a “can” be valid

SLOC	Function Points
Use Case	Stories

- Agile Software Development seem ot often avoid SLOC and Function Point metrics

The Rise of Story Points

A "User Story" is a simple statement about what a user wants to do with a feature and the value the user will gain from that feature.

- The User
- What the user wants to do with a feature
- Value gained by the user
- Acceptance Criteria
- Front and back of a card

Story Points represent a value given to a user story that is used to measure the effort required to implement the sto

It is a number that represents story size based on how hard a story is to implement.

Story Point Velocity

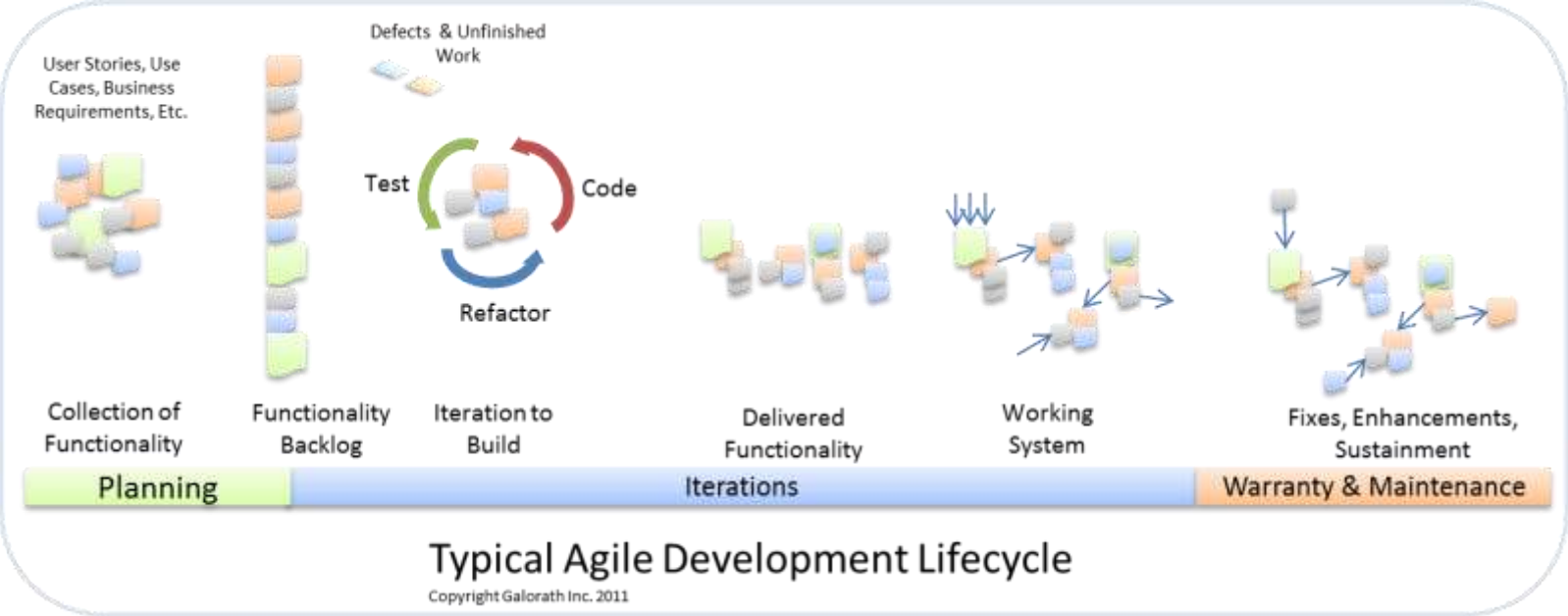
- Velocity measures how much of something can be achieved over a fixed period of time; e.g. how many Story Points are completed during a Sprint
  - You could do this at the User Story level but you have no relative measure between User Stories
- Velocity is a "team" measurement – not the individual
- Iteration Duration / Completed Total SP = Velocity
- Iterations needed = Total SP / Velocity
- Don't change the duration and use the same result

Quiz: A project has a total size of 365 Story Points. The team has a velocity of 25/SP per iteration

How many iterations will the project take?

Story Points Velocity Can Be Used to Determine Duration/Iterations

- What does the customer want…. A Schedule!
- But to provide one you need to know the **teams** Story Point "velocity"
- If the schedule is too long then some functionality could be removed – or other adjustments made



Some Drawbacks To Story Points

- Story Points are team dependent!
  - Members of different teams will have different levels of experience leading to different perspectives related to how hard a story is.
- Points don’t easily scale across different projects.
  - How one team’s points can vary.
- "Inflation" can occur as soon as the second sprint
  - Teams often blame not delivering the Story due to a faulty count. That is usually not the reason!
  - As a consequence a natural inflation appears during the next count.

Disadvantages

- Difficult to explain
  - May resort to comparing Poodles to Great Danes
- Teams have different interpretations when working independently
  - Story Points need to be scored as a team
- Tendency to skip over detailed iteration planning by assuming a velocity * SP = work
  - Still need to break the User Story into tasks and estimate the capacity for the sprint
- Takes a while to trust the results
  - But this is common for all new sizing measures!

Advantages To Story Points (a few)

- Prevents Managers from converting a SP into a Calendar day (They have to know the velocity)
- Promotes cross-functional behavior (teams can compare similar things)
- Points do not decay (when compared to ideal days)
- They are a “Pure” size measure relative to other known things
- People are pretty good at generating a valid relationship of size

Four Estimating Processes

- **Process 1: 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)
- **Process 2: Structured approach** based on established "velocity" – most often used internally by the developer since detailed/sensitive data are available to them
- **Process 3: Automated Models approach** based on a size metric – which may be difficult to quantify
- **Process 4: Factor/Complexity approach** based on data generated in early iterations

A Word About 2014 Rates

- Developers and Tester - $70 to $200 per hour, median team rate about $125
- Agile Coach - $100 to $200 per hour, average about $150
- Business Analyst - $125
- Average Team Rate of about $115

WARNING: THESE ARE BROAD AVERAGE I HAVE FOUND THIS YEAR

Process 1: Build-Up Approach

When a program is comprised completely of agile sprints, we can use industry norms or program plans to develop an estimate

- Process 1 is defined as:
  - SS x Sp time x # Sprints
    - SS (normally 1-10 people) x Sp time (normally 0.25 to 1.25 months) x # Sprints
    - Frequently used by independent estimators since actual data are often unavailable
    - Remember to factor in time for demonstrations/user feedback
    - Can develop a point estimate and a range
    - Works well for small programs

The weakness of this approach is justifying the team size, number of sprints, sprint length and total required to meet the requirement

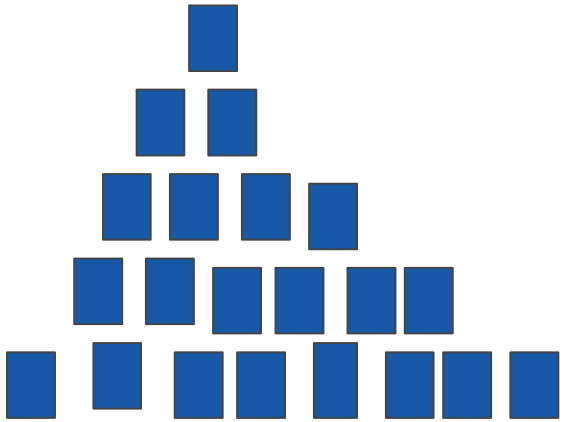
Process 2: Structured Approach based on "Velocity"

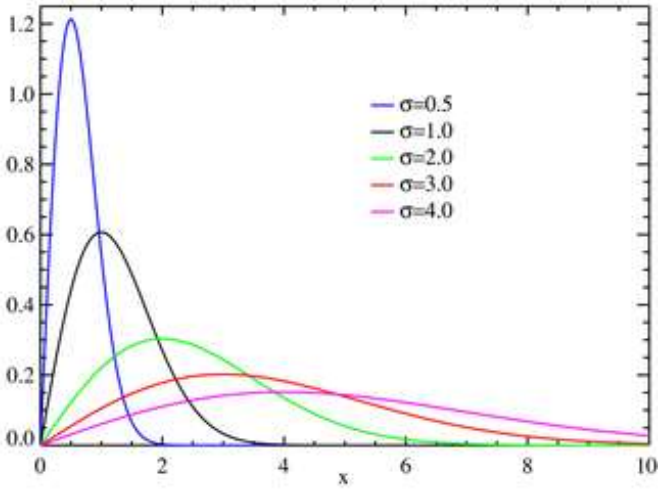
- Process 2 can be summarized by:
  1. Express requirements in the same size metric used by the developer; normally Features, Feature Points, Use Case Points, Story Points, … What the size metric is unimportant as long as it is consistently used across this program*
  2. (optional). Use a process to rank the size metric: small, medium, large using something like Fibonacci sequence, planning poker
  3. Estimate and/or document the velocity (number of size metrics per time period) at which the Agile team has worked
  4. Estimate and/or document the historic cost per size metric for the Agile team
  5. Spread the sprints over time to develop time-phased estimate

* I would hope that over time we could develop standards for agile development across the various size metrics and programs. However, since these metric often do not conform to a "standard" this is an elusive task. But an average over several early interactions may be very accurate for a specific [program.

Moving to Automated Models

- Step 5 of the previous slide suggested you time-phase the Sprints
  - When you do this, the results often resemble the Rayleigh Function used in modern software models





- This observation leads to the third estimating process

Process 3: Automated Model Approach

- The “Parameter” settings within automated models can be adjusted to estimate costs and schedule for complex/large projects
  - The “environmental factors” in SEER, PRICE, SLM, and COCOMO II have been adjusted to reflect Agile practices and therefore Iterative Development
  - Remember, the size metric is still the key cost driver, which is even less certain in agile programs than traditional ones

Process 4: Factor/Complexity Approach

- In a normal IID program, the initial program estimate must be based on broad parameters with wide ranges – analogy to previous programs and/or generic models
- Specific iterations/sprints can be estimated using the agile estimating processes previously presented
- The real question is: how do we estimate the cost of future Increments (time boxes)?
- The following slides present Process 4 Factor/Complexity Approach

Process 4: Factor/Complexity Approach

- Step 1: Select a Baseline Increment (often the last successful increment) for the program
- Step 2: Carefully analyze this baseline increment – this analysis could be based on SLOC, function points, features, requirements, dollars, or some other metric
- Step 3: For each new increment, compare the expected functionality and complexity of the new increment to the baseline (or last successful) increment

  - Notional functional and complexity factors are presented on the next slide

Process 4: Factor/Complexity Approach

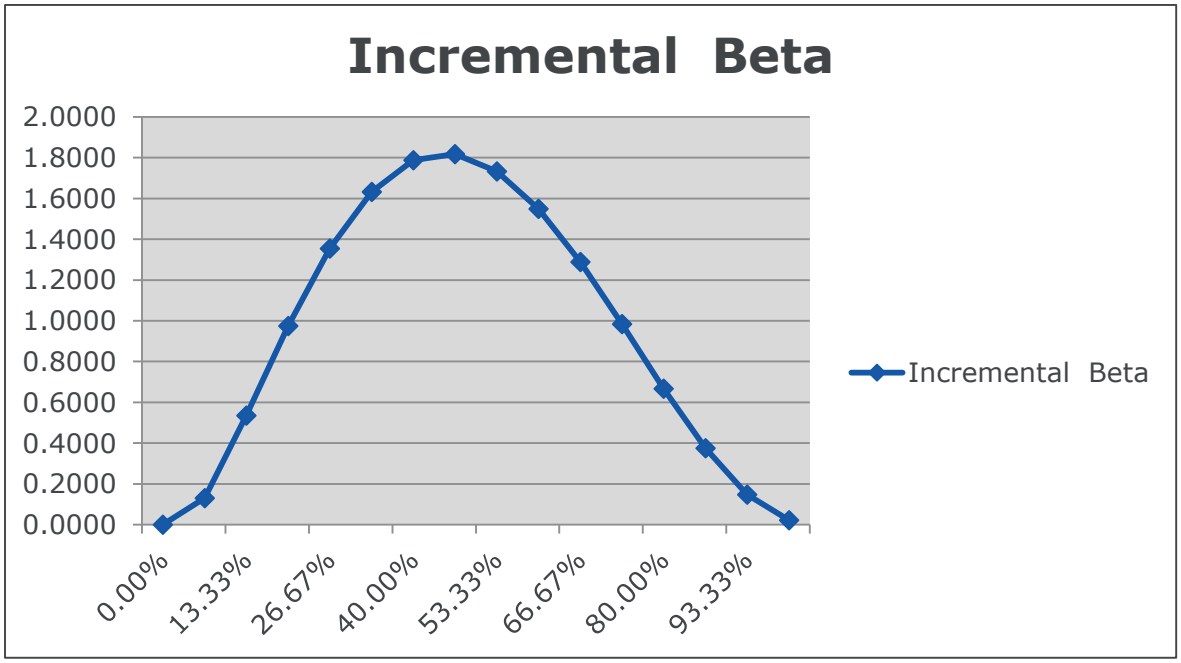
Scale	Functional Description	Effort Multipliers
- - -	Significantly less functionality to be delivered	0.5
- -	Moderately less functionality to be delivered	0.7
-	Slightly less functionality to be delivered	0.9
=	Functionality equivalent to Increment X	1.0
+	Slightly more functionality to be delivered	1.3
+ +	Moderately more functionality to be delivered	1.7
+ + +	Significantly more functionality to be delivered	2.0

Scale	Complexity Description	Effort Multipliers
- -	Significantly less complex	0.7
-	Slightly less complex	0.9
=	Complexity equivalent to Increment X	1.0
+	Slightly more complex	1.3
+ +	Significantly more complex	1.7

- These initial set of factors came from the environmental factor from traditional software cost models
- Step 4: Because each Increment is a mini project, use a Rayleigh or simple Beta Curve (such as a 60/50 Beta curve) to phase costs
- However, do not be surprised if you encounter programs that are truly operated and manages as Level of Effort (LOE)

Process 4: Factor/Complexity Approach

- Step 5: The project can define the length of each increment – likely between 4 and 14 months



Issues for Project Management

- Cost and Schedule modelers usually want well-defined program requirements and size metrics early in the lifecycle – the nature of IID programs argues against this
  - IID programs tend to be less structured in the beginning, and therefore reliable estimates of cost and schedule may not be available until 10-20% of the project is complete 11
- Initial contracts tend to be Fixed Price or LOE
  - This does not imply poor value to the project
  - It does imply that key "value-added" metrics may not be identified or collected
- "Time Boxing" tends to resolve the individual scheduling issues, but not the total program length issue
  - A specific cost estimating strategy is required to accurately plan for resources

Issues for Project Management

- If a program has too many planned Increments (10 or more), it may not be a well-defined program and could spin out of control or just become an LOE research project
- Establishing and monitoring metrics becomes critical
- "To be able to adopt an empirical approach to project management and control, we must be able to objectively demonstrate and measure how much progress the project has made in each iteration
  - Possible ways to measure progress include:
    - Number of products and documents produced
    - Number of lines of code produced
    - Number of activities completed
    - Amount of budget/schedule consumed
    - Number of requirements verified to have been verified implemented correctly"

Schedule Analysis

- Due to the short length of increments (generally 9-12 months) and continuity between increments, phasing the costs within a specific increment is less important
- However, the "million dollar questions" for incremental and agile programs (where requirements definition and documentation are less detailed, and the development is more flexible/emergent) are:
  - What will the program look like at Initial Operational Capability (IOC)?
  - How many increments will it take?
  - How long is each increment going to last?
- Cost estimators are going to have to adjust, and examine these programs as a schedule analyst might to produce credible lifecycle estimates

Summary

- Fixed Price and/or LOE contracts in the early phases should be written so that key "value-added" metrics are collected and reported during each increment
- Estimators may have to employ a variety of software estimating methodologies within a single estimate to model the blended development approaches being utilized in today's development environments
  - An agile estimating process can be applied to each iteration/sprint
  - Future Increments can be estimated based on most recent/successful IID performance
- Cost estimators will have to scrutinize these programs like a schedule analyst might to determine the most likely IOC capabilities and associated date
  - The number of increments are an important cost driver as well as an influential factor in uncertainty/risk modeling

Summary

- All of the estimation methods are susceptible to error, and require accurate historical data to be useful within the context of the organization
- When developers and estimators use the same "proxy" for effort, there is more confidence in the estimate

Recommended Reading

- “The Death of Agile” blog
- “Agile Hippies and The Death of the Iteration” blog
- Story Point Inflation

5

Endnotes

- *1, 2, 4, 10, 11:* Larman, C. (2010). *Agile and Iterative Development: A Manager's Guide.*
- *3:* Kilgore, J. (2012). Senior Associate, Kalman & Company, Inc.
- *5, 6, 7, 8:* Agile Alliance. (2012). *Agile Alliance*. Retrieved 2012, from http://www.agilealliance.org
- *9:* Coaching, T. L. (n.d.). Rally Software Scaling Software Agility.
- *12:* Bittner, K., & Spence, I. (2006). *Managing Iterative Software Development Projects.* Addison-Wesley Professional.

Additional References

- Cohn, M. (2009). *Succeeding with Agile Software Development using Scrum.*
- Dooley, J. (2011). *Software Development and Professional Practice.*
- Gack, G. (2010). *Managing the Black Hole.*
- George, J., & Rodger, J. (2010). *Smart Data (Enterprise Performance Optimization Strategy).*
- Royce, W., Bittner, K., & Perrow, M. (2009). *The Economics of Iterative Software Development: Steering Towards Better Business Results.* Addision Wesley Professional.
- Smith, G., & Sidky, A. (2009). *Becoming Agile in an Imperfect World.*

Contact Information

- Bob Hunt
  - Email: Bob.Hunt@GalorathFed.com
  - Phone: 703.201.0651