Software Cost Estimating for Iterative and Incremental Development Programs

Date: 14 February 2012

Bob Hunt - Dulos Inc

Jon Kilgore – Kalman and Company

Jennifer Swartz - Kalman and Company

Dulos, Inc.

Exceptional Service

Outline

- Iterative and Incremental Development (IID) Programs
- Agile Software Development Processes
- Issues for Program Managers
- Software Estimating Process
- Summary

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, etc.

Software Development

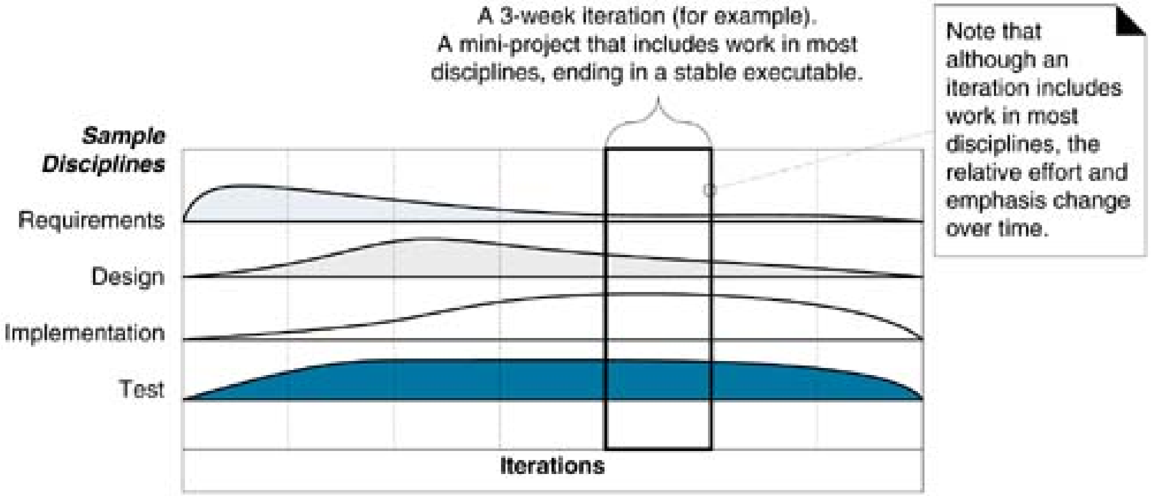
- Iterative Drive programs, by nature, start with a less well-defined metric
  - Therefore, they may require alternative estimating approaches
- This briefing will focus on the challenges of estimating an iterative program

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
- In many defense programs, **increments** are 9-12 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

Each Iteration/Sprint is a Mini Project

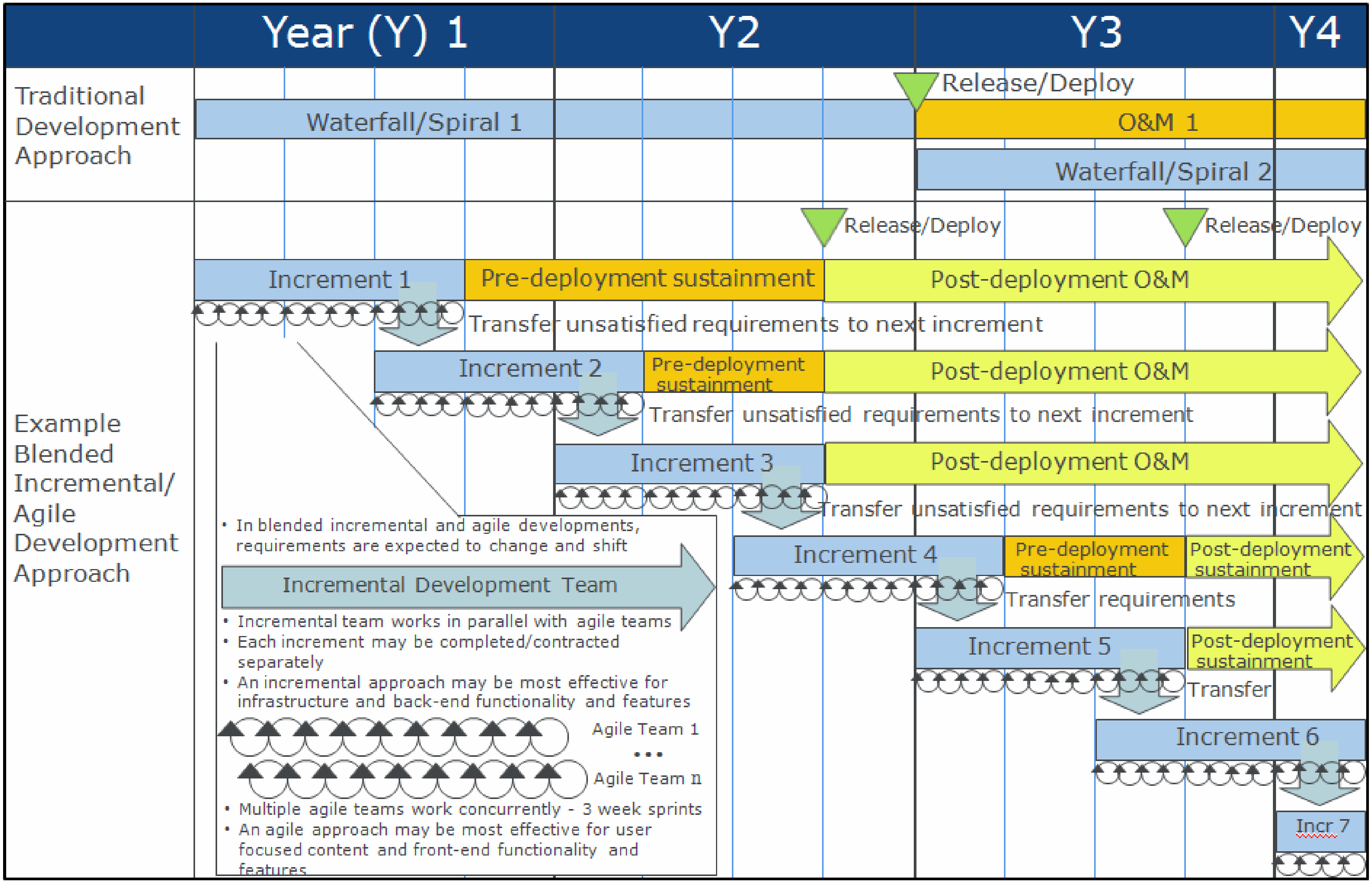
- Each iteration/sprint includes production-quality programming, not just, for example, requirements analysis
  - The software resulting from each iteration/sprint is not a prototype or proof of concept, but a subset of the final system
- More broadly, viewing an iteration as a self-contained mini project,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


A 3-week iteration (for example).
A mini-project that includes work in most disciplines, ending in a stable executable.
Note that although an iteration includes work in most disciplines, the relative effort and emphasis change over time.
Sample Disciplines
Requirements
Design
Implementation
Test
Iterations

[2]

Software Lifecycle Comparison

3


Year (Y) 1
Y2
Y3
Y4
Traditional Development Approach
Release/Deploy
Waterfall/Spiral 1
O&M 1
Waterfall/Spiral 2
Example Blended Incremental/ Agile Development Approach
Release/Deploy
Release/Deploy
Increment 1
Pre-deployment sustainment
Post-deployment O&M
Transfer unsatisfied requirements to next increment
Increment 2
Pre-deployment sustainment
Post-deployment O&M
Transfer unsatisfied requirements to next increment
Increment 3
Post-deployment O&M
Transfer unsatisfied requirements to next increment
Increment 4
Pre-deployment sustainment
Post-deployment sustainment
Transfer requirements
Increment 5
Post-deployment sustainment
Transfer
Increment 6
Incr 7
Incremental Development Team
Agile Team 1
• • •
Agile Team n

- In blended incremental and agile developments, requirements are expected to change and shift
- Incremental team works in parallel with agile teams
- Each increment may be completed/contracted separately
- An incremental approach may be most effective for infrastructure and back-end functionality and features
- Multiple agile teams work concurrently - 3 week sprints
- An agile approach may be most effective for user focused content and front-end functionality and features

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

What is Agile Software Development?

- 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[5]

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”[6]

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 [7]

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

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

Radical 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])

Radical 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
  - It sounds very much like what we called "rapid prototyping"

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!

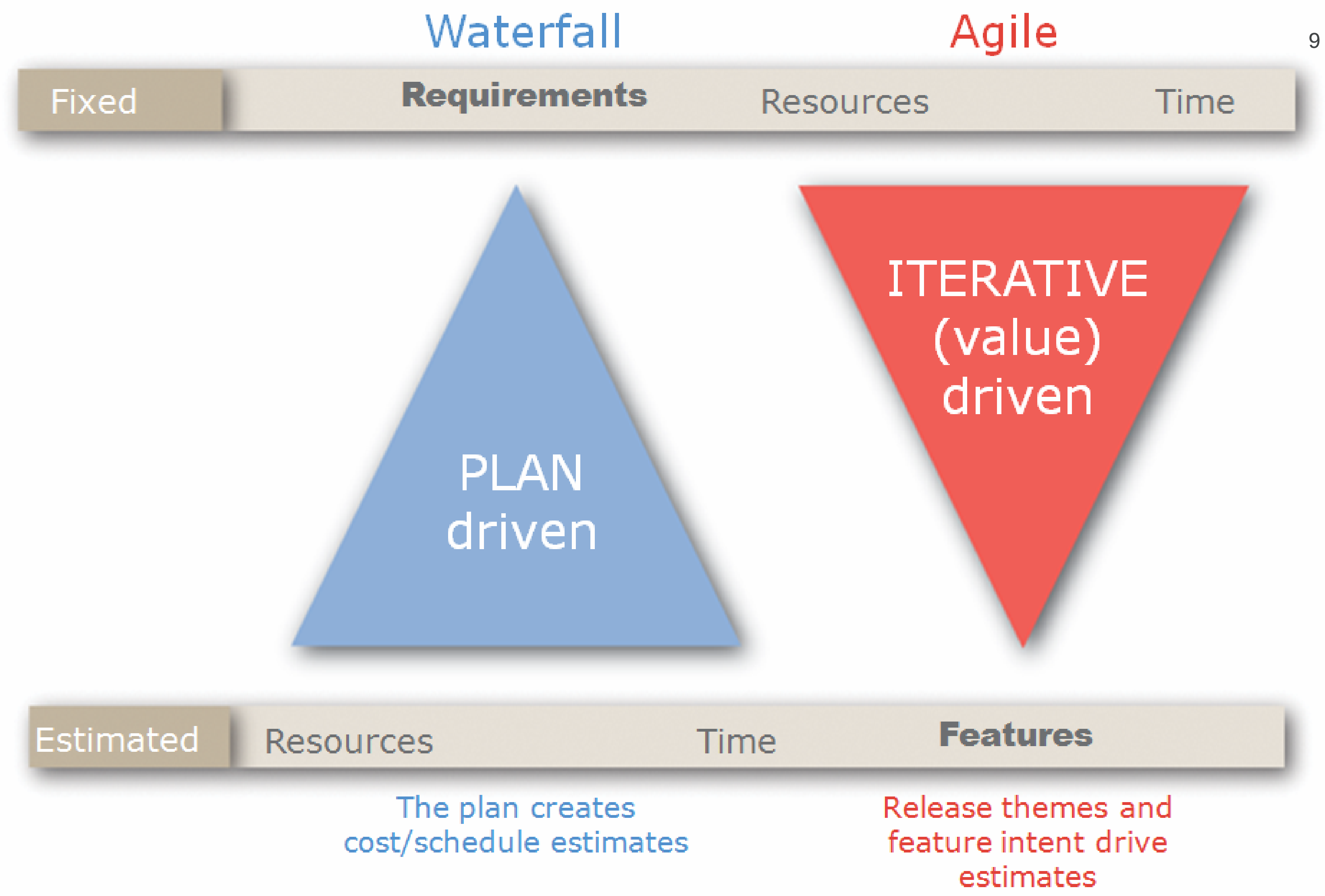
What do they have in common?

- 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

What do they have in common?

- 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



Waterfall

Agile

Fixed **Requirements** Resources Time

PLAN driven

ITERATIVE (value) driven

Estimated Resources Time **Features**

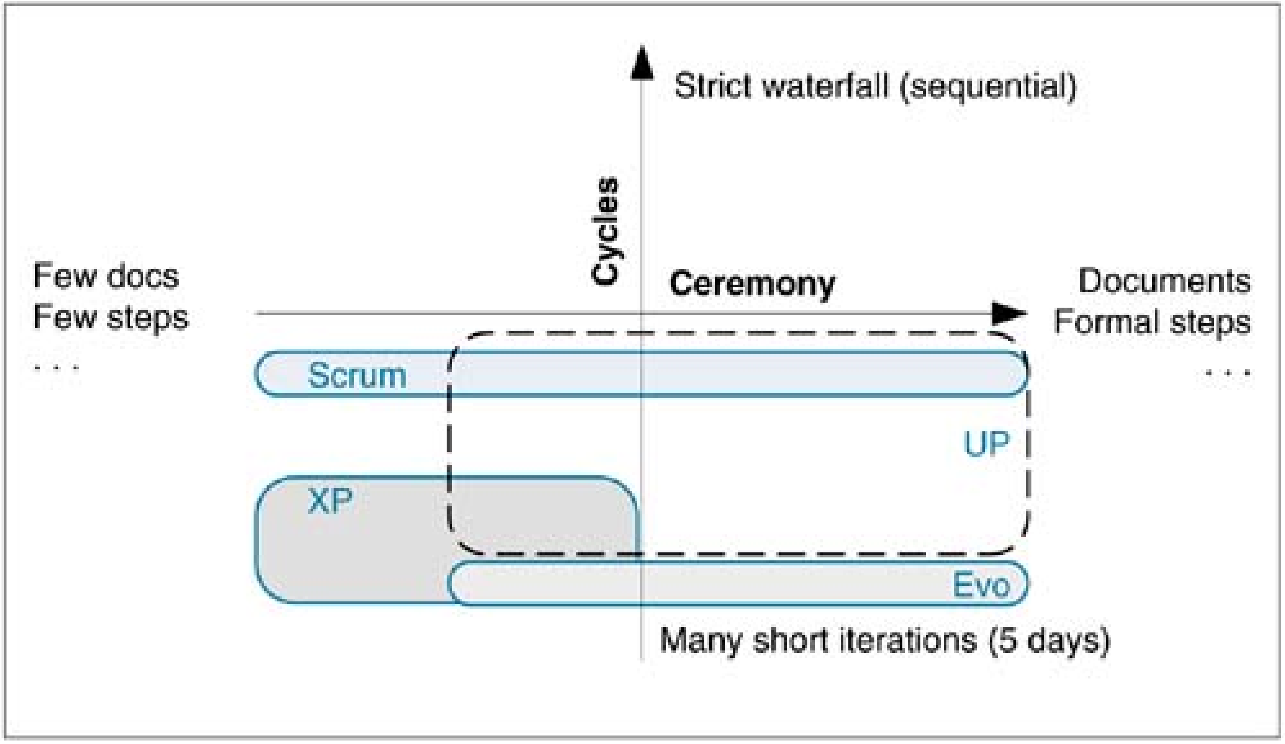
The plan creates cost/schedule estimates

Release themes and feature intent drive estimates

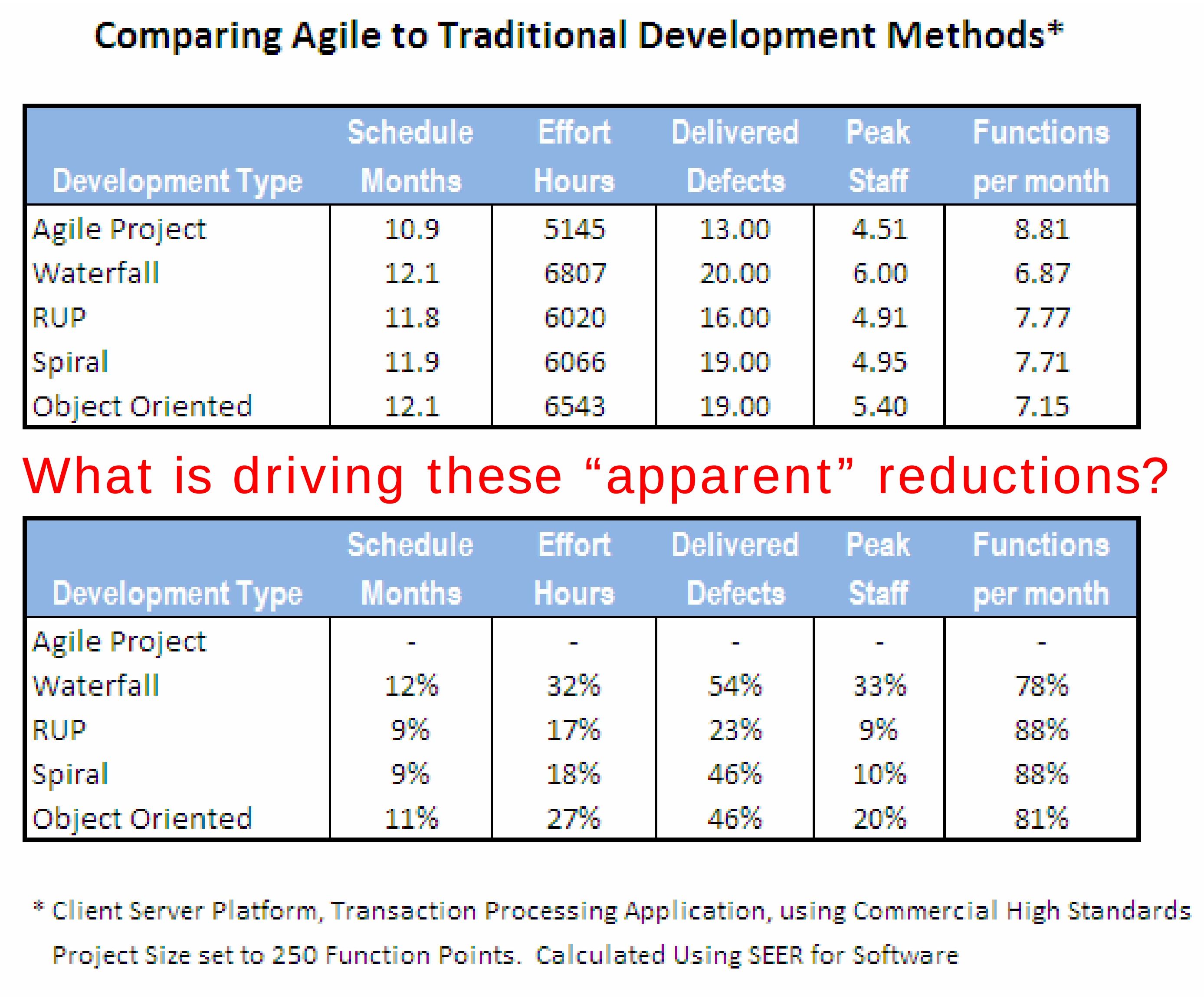
Agile Processes vs. the Waterfall Process

- The figure below presents various agile development processes terms of cycle and ceremony as compared to traditional waterfall process

10



What do the Models Say?



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

Scrums and Sprints

- Scrum Size:
  - 1-10 people
- Sprint Length:
  - 1-6 weeks
- Story Points per Sprint:
  - 6-9 Use Case Pointer per Sprint

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

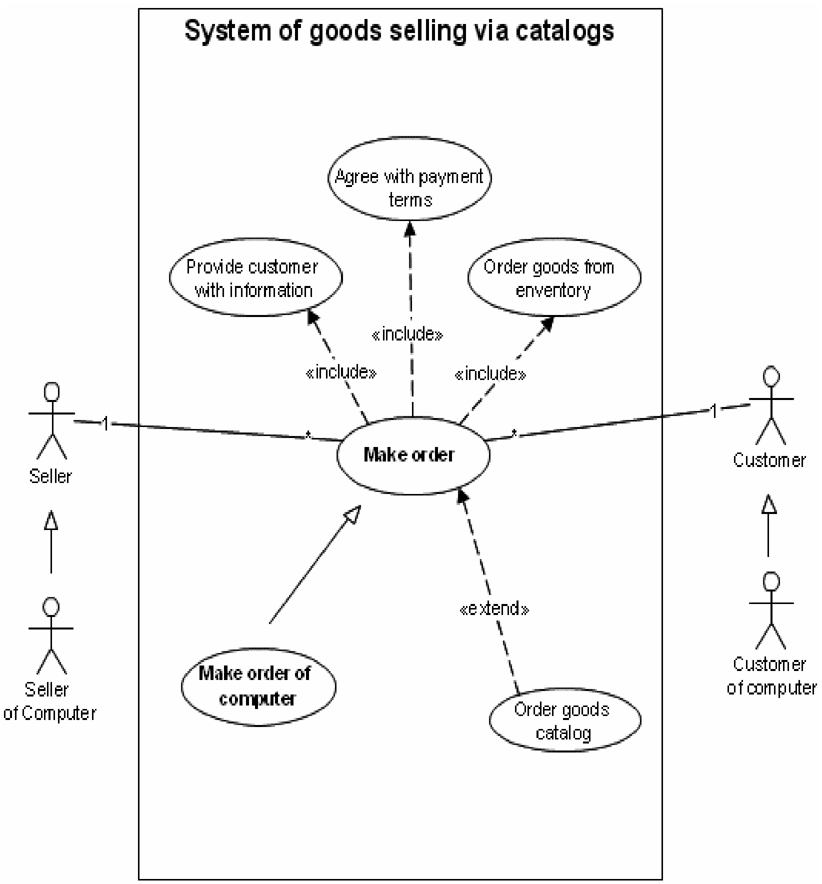
Process 1: Build-Up Approach

- When a program is comprised completely of agile sprints, we can use industry norms 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

Process 2: Structured Approach based on "Velocity"

- When Use Case came into vogue, an estimating approach developed around that concept
  - Use Case Points or Story Points were used to express the requirement
  - This process was first documented in the Schneider and Winters Model (see the appendix slides)
  - Process 2 can be summarized by:
  - 1. Express requirements in Use Case Points (UCP) or Story Points (SP)
  - 2. Use a process to rank (UCP/SP): small, medium, large, Fibonacci sequence, planning poker
  - 3. Estimate and/or document the velocity (number of story points per time period) at which the scrum team can work
  - 4. Spread the sprints over time to develop time-phased estimate

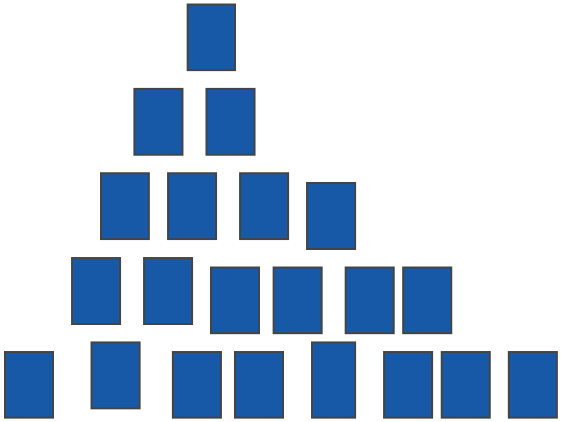
What is a Use Case Point?

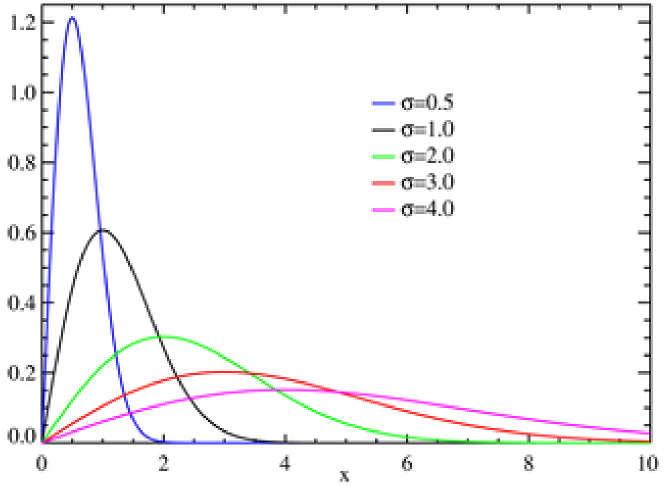


- A weighted count of actors and use cases
  - Actor weight is classified as:
    - 1 – Simple: highly-defined and elemental, such as a simple API call
    - 2 – Average: protocol-driven interaction, allowing some freedom
    - 3 – Complex: potentially complex interaction
- Use Case weight is classified as:
  - 5 – simple: 3 or fewer transactions
  - 10 – average: 4-7 transactions
  - 15 – Complex: more than 7 transactions

Moving to Automated Models

- Step 4 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, 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

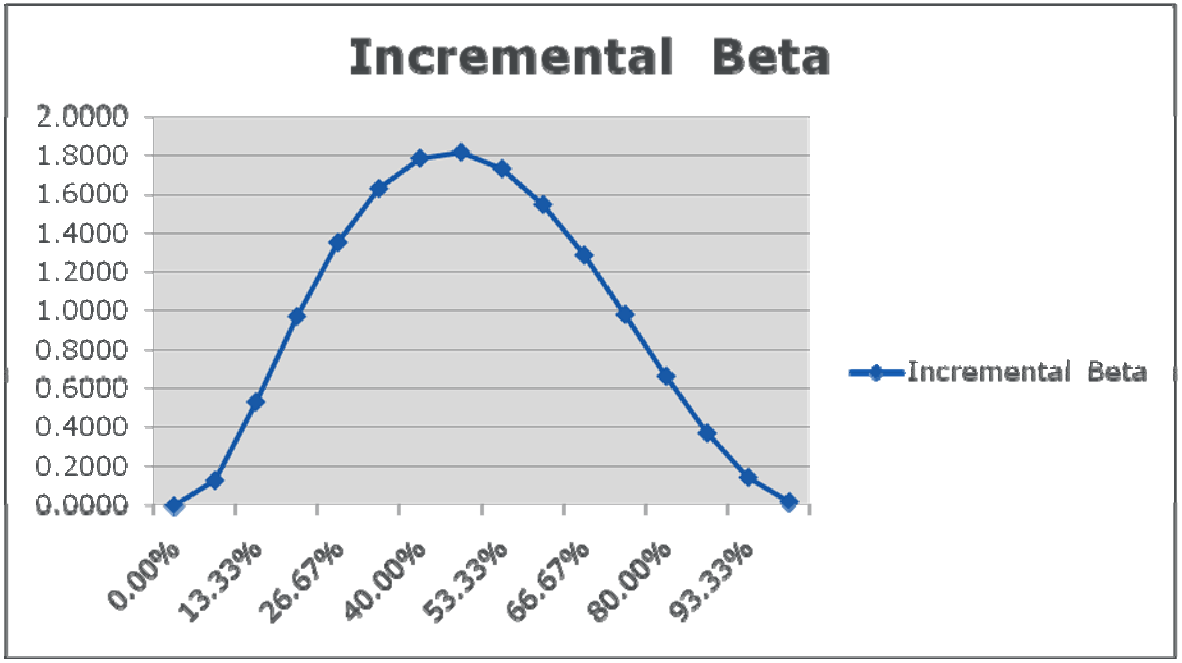
Scale	Functional Description	Effort Multipliers
- - -	Significantly less functionality to be delivered	0.5
- -	Moderately less functionality to be delivered	0.7
-	Slightly less functionality to be delivered	0.9
=	Functionality equivalent to Increment X	1.0
+	Slightly more functionality to be delivered	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”[12]

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

5

Dulos, Inc.

Exceptional Service

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. 5
- Smith, G., & Sidky, A. (2009). *Becoming Agile in an Imperfect World.*

APPENDIX SLIDES

History of Estimating Software via Use Case Points

- Mid-1990s: Rumbaugh, Booch, and Jacobson of Rational Software Corporation developed the Unified Modeling Language (UML) as notation and methodology for developing object-oriented software
- UML was incorporated into the Rational Unified Process (RUP) by Rational Software
- Within UML is the concept of defining the requirements for software products with Use Cases
- Rational Software Corporation created a software project estimating technique based on Use Case Points and including statistical and weighted modifiers

History of Estimating Software via Use Case Points

- Karner’s technique is now incorporated into RUP
  - Use Cases, as defined by UML, describe what the actors want the system to do and have proven to be an easy method for capturing the scope of a project early in its lifecycle
  - Use Cases may allow a consistent artifact to base an early project estimate

Schneider and Winters Model – Applying Use Cases

Weighting Actors for Complexity

Actor Type	Description	Quantity	Weight Factor	Subtotal
Simple	Defined API	3	1	3
Average	Interactive or protocol-driven interface	2	2	4
Complex	Graphical user interface	1	3	3
Total Actor Points				10

Weighting Use Cases for Complexity

Use Case Type	Description	Quantity	Weight Factor	Subtotal
Simple	Up to 3 transactions	3	5	15
Average	4 to 7 transactions	2	10	20
Complex	More than 7 transactions	1	15	15
Total Use Cases				50

- Add the Actors total to the Use Cases total to determine the Unadjusted Use Case Points (UUCP) = 60

Schneider and Winters Model – Applying Use Cases

- Weighting technical factors is an exercise to calculate a Use Case Point modifier, called the Technical Complexity Factor

Weighting Technical Factors

Technical Factor	Factor Description	Weight Factor	Project Rating	Subtotal
T1	Must have a distributed solution	2	5	10
T2	Must respond to specific performance objectives	1	3	3
T3	Must meet end-user efficiency desires	1	5	5
T4	Complex internal processing	1	5	5
T5	Code must be reusable	1	3	3
T6	Must be easy to install	.5	3	1.5
T7	Must be easy to use	.5	3	1.5
T8	Must be portable	2	0	0
T9	Must be easy to change	1	5	5
T10	Must allow concurrent users	1	0	0
T11	Includes special security features	1	5	5
T12	Must provide direct access for third-parties	1	0	0
T13	Requires special user training facilities	1	3	3
Total TFactor				42

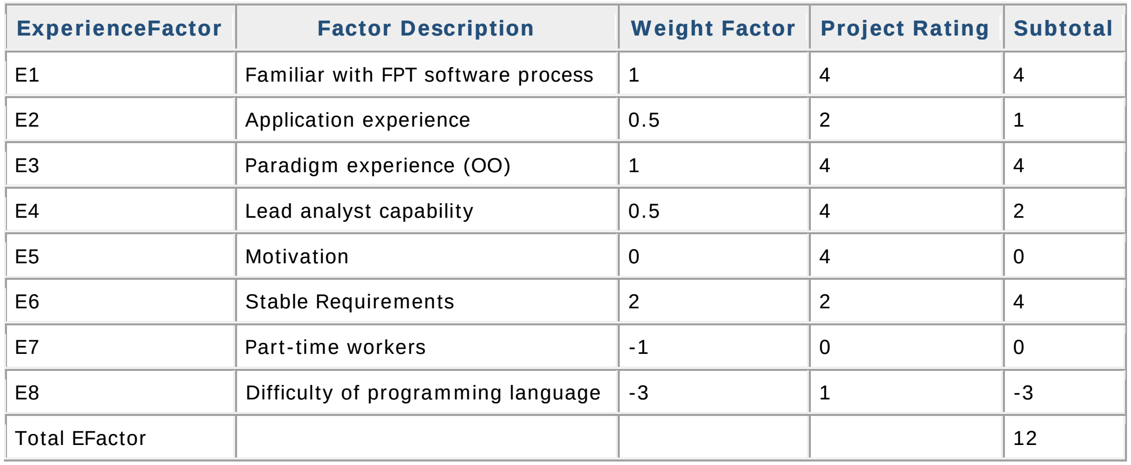
Schneider and Winters Model – Applying Use Cases

- (Weighting Factor) x ∑(Tlevel) = TFactor
  - The TFactor does not directly modify the UUCP
- To calculate TCF, multiply TFactor by 0.01 and then add 0.06
  - (0.01 x TFactor) + 0.6 = TCF
  - (0.01 x 42) + 0.6 = 1.02 TCF
- Calculate the size of the software (Use Case) project by multiplying UUCP by TCF
  - UUCP x TCF = Size of Use Case (SzUC)
  - 60 x 1.02 = 61.2
  - Note: Reusable software components should not be included in this estimate
  - Identify the UUCP associated with the reusable components and adjust the SzUC accordingly

Schneider and Winters Model – Applying Use Cases

- The experience level of each team member can have a great effect on the accuracy of an estimate
  - This is called the Experience Factor (EF)

Weighting Experience Factors



ExperienceFactor	Factor Description	Weight Factor	Project Rating	Subtotal
E1	Familiar with FPT software process	1	4	4
E2	Application experience	0.5	2	1
E3	Paradigm experience (OO)	1	4	4
E4	Lead analyst capability	0.5	4	2
E5	Motivation	0	4	0
E6	Stable Requirements	2	2	4
E7	Part-time workers	-1	0	0
E8	Difficulty of programming language	-3	1	-3
Total EFactor				12

- To calculate EF, go through the preceding table and rate each factor from 0 to 5
  - ∑(Elevel) x Weighting Factor = Efactor
  - Calculate the EF by multiplying the Efactor by -0.03 and adding 1.4 = 1.04

Schneider and Winters Model – Applying Use Cases

- To calculate the Use Case Points, multiply SzUC by EF
  - SzUC x EF = UCP
  - 61.2 x 1.04 = 63.648
- An alternate calculation:
  - UUCP x TCF x EF = UCP
  - 60 x 1.02 x 1.04 = 63.448

Schneider and Winters Model – Applying Use Cases

- Now that we have estimated the Use Case Point, where do we go from here?
  - Use the Use Case Point count to directly estimate man-hours
  - Use the Use Case Point count to directly estimate size

Contact Information

- Bob Hunt
  - Email: BobHunt@DulosInc.com
  - Phone: 703.201.0651
  - Jon Kilgore
  - Email: jon.kilgore@kalmancoinc.com
  - Phone: 757.262.7462
  - Jennifer Swartz
  - Email: jennifer.swartz@kalmancoinc.com
  - Phone: 330.416.8450