Special Topics in Software Estimation

Software Cost Estimating for Iterative/ Incremental Development Programs

Agile Cost Estimating

Outline

- Part I:
  - Software project estimation using functional size
  - Software projects
  - Software size
  - Software project estimation based on functional size
  - Historical data and ISBSG
  - Conclusions
- Part II:
  - Iterative and Incremental Development (IID) Programs
  - Agile Software Development Processes
  - Issues for Program Managers
  - Software Estimating Process
  - Summary

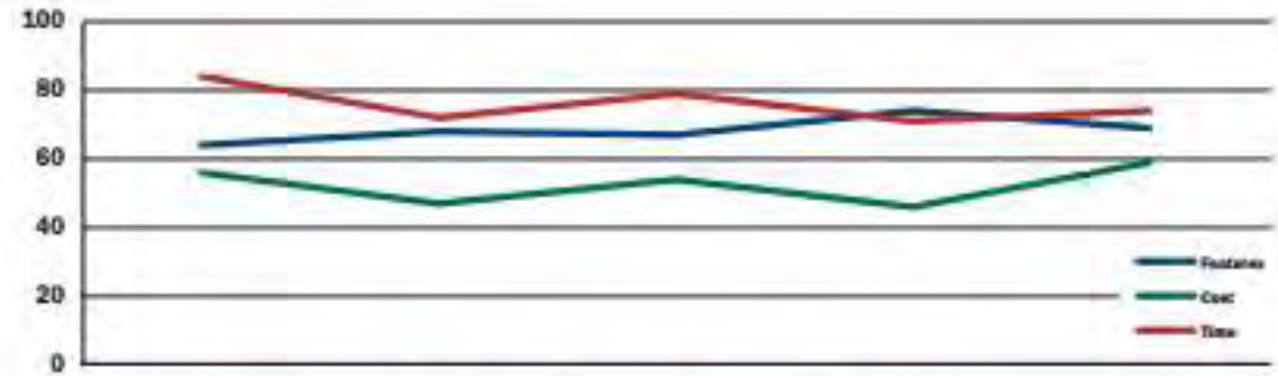
PART I

The problem

	2004	2006	2008	2010	2012
Successful	29%	35%	32%	37%	39%
Failed	18%	19%	24%	21%	18%
Challenged	53%	46%	44%	42%	43%

Project resolution results from CHAOS research for years 2004 to 2012.

Time and cost overruns, plus percentage of features delivered from CHAOS research for the years 2004 to 2012.



	2004	2006	2008	2010	2012
TIME	84%	72%	79%	71%	74%
COST	56%	47%	54%	46%	59%
FEATURES	64%	68%	67%	74%	69%

Software Industry

- **Software project industry: low maturity**
  - Low estimation maturity
  - No or little formal estimation processes
  - No or little use of historical data
  - Customers chose suppliers based on price, not reality
- **Lots of schedule and cost overruns**
  - Standish Chaos reports: Most projects fail or are at least unsuccessful
- **Low customer satisfaction rates**
  - In Europe: only slightly higher than the financial sector

Two main reasons

- **Unstable user requirements**
  - Starting the development too early in the project
  - Not enough time spent on requirements analysis
  - Users not involved or not involved enough

- **Unrealistic project expectations**
  - Usually: only expert estimates (optimistic)
  - Pressure to lower cost and deliver faster
  - End date is not estimated, but a given
    - Duration is an important cost driver!

Requirements

- Worst in class software development organizations spend 7,5% of the project budget on requirements

- Best in class software development organizations spend 28% of the project budget on requirements

- More effort spent on requirements increases project success!

Unrealistic expectations

- **Many projects are not estimated in a professional way**
  - Only expert estimates, no use of estimation models / historical data
- **Underestimation results in bad planning**
  - Development team too small
  - Duration too short
  - Unrealistic milestones
  - Project management with no grip on the project
  - Extra management attention, more meetings
  - Stress in the team → bad quality → more effort
  - Bad software, low maintainability

Why a realistic estimate?

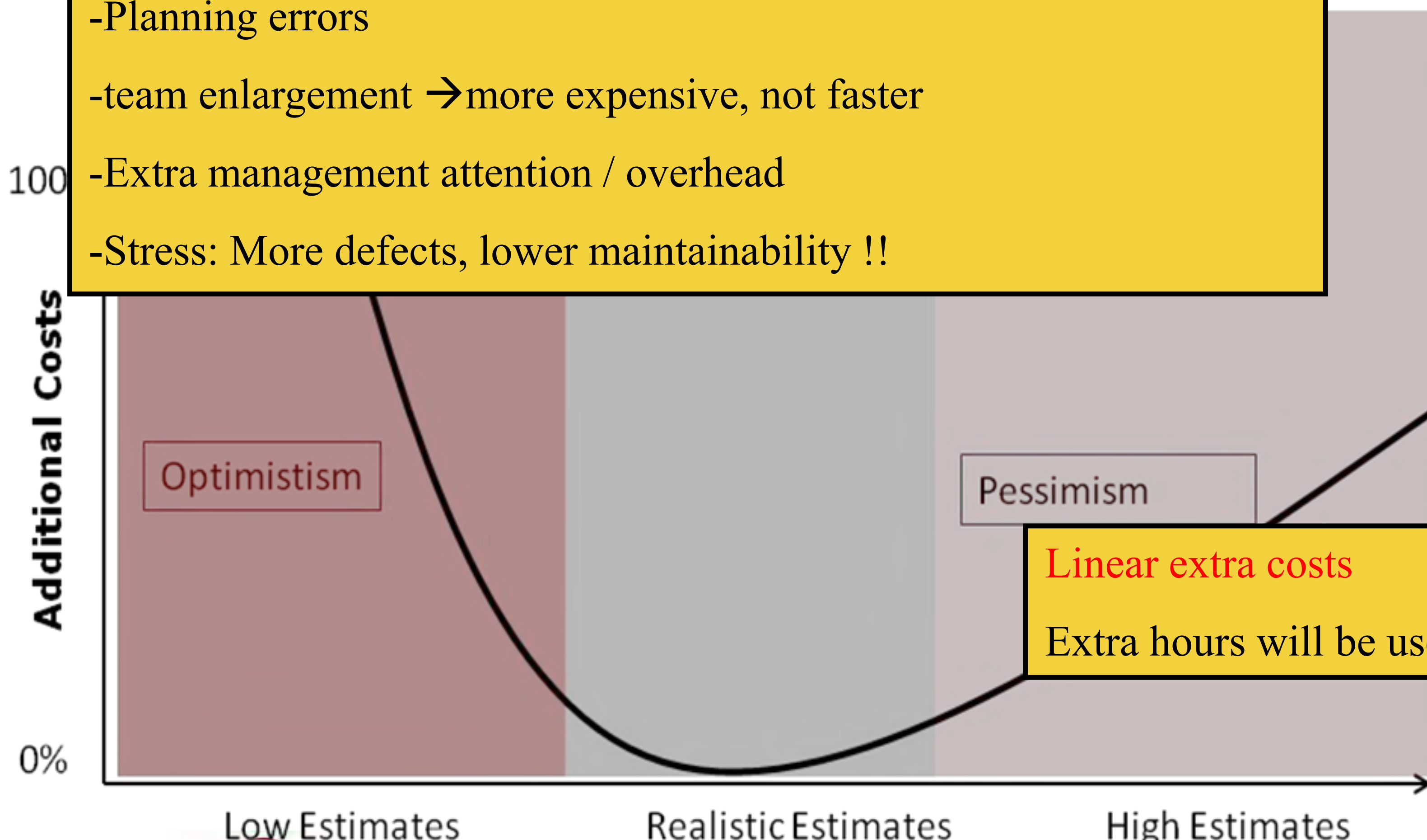
Non-linear extra costs

-Planning errors

-team enlargement →more expensive, not faster

-Extra management attention / overhead

-Stress: More defects, lower maintainability !!



Linear extra costs

Extra hours will be used

Realistic estimates

- **Use multiple estimation methods:**
  - Expert estimates (bottom up)
  - Parametric estimates (bottom up)
  - Challenge / Comparison
- **All estimates should be expressed in ranges!**
  - Low: 20000 hours / Likely 30000 hours / Max: 45000 hours
- **Reality check the estimates (own history / ISBSG)**
  - Hours per 1000 slocs
  - Hours per function point
- **Document/Baseline the Estimate**
  - Basis of Estimate (BoE)
  - AACE recommended practice 74R-13

Realistic estimates, why not?

IT industry – estimates are too optimistic

- Business/customer: pressure to lower price;
- Business/customer: pressure shorter time-to-market;
- Business/customer: incomplete requirements
- Business/customer: early fixed price/date quote

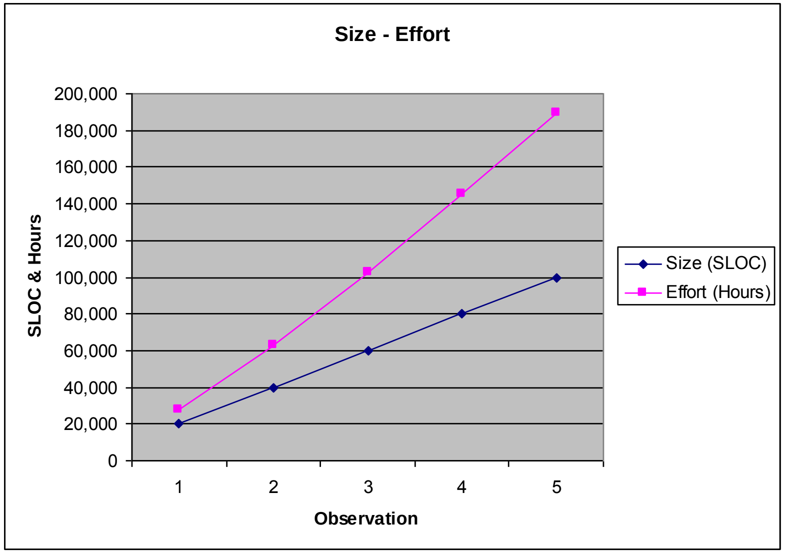
IT supplier

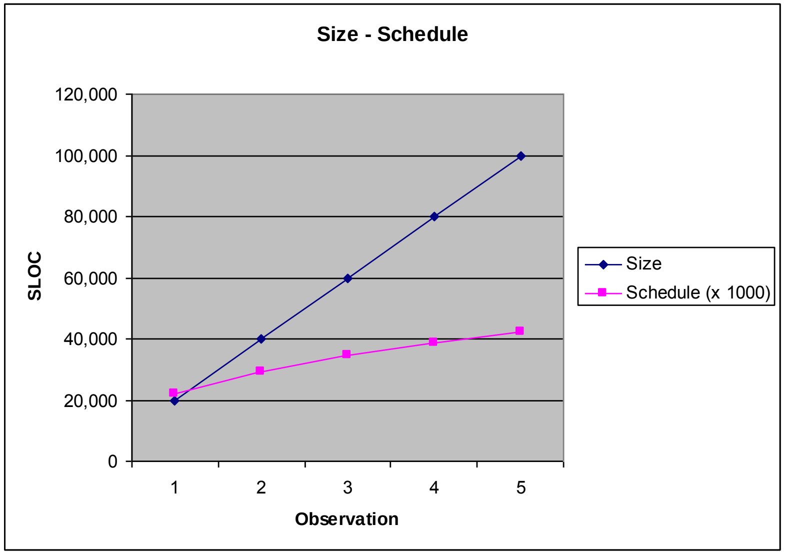
- Unclear what customer wants;
- Immature estimation techniques (only expert estimates);
- No idea about own performance and capabilities;
- Not defendable → easy to push back

Optimistic estimates are more rule than exception

Why Size Matters

- Effort & schedule vary in proportion to size (but not linearly!)
- Knowing size allows estimators to determine effort (cost) & schedule
- Better size estimates = better effort & schedule estimates
- *Software Size is <u>the main driver</u> of software development effort, cost, and schedule -- use the <u>best available</u> estimate of size, and use a range!*





Software Size

- Software size is the key input parameter for most estimation models
- In practice: hard to measure upfront. Not tangible
- Main types of software size:
  - Technical size, e.g. slocs, MB's, etc.
  - Functional size, e.g. function points, COSMIC FP

- Technical size must be estimated / guessed
- Functional size can be measured, if the functional requirements are known

Technical Size

- Most often used: source lines of code (slocs)
  - ‘Easy’ to measure after completion
  - More slocs is better?
  - Less slocs is better?
  - Does a price or a quote of a price per 1000 slocs make sense?

- Slocs can be used in some models, like COCOMO
- But size in slocs can’t be measured, only guessed
- Slocs are different between languages
- Slocs are different between code counting tools
- Result: the estimates based on slocs are probably not very accurate

Functional size

- Can be used early in the project, when functional requirements are known
- ISO standard measurement method to size the functional requirements (what does the software for the user):
  - IFPUG Function points
  - NESMA Function points
  - COSMIC Function points
- Objective, verifiable, repeatable, defensible !!
- Independent of the business or systems requirements

- More function points means more functionality: value!

Functional Size Measurement

- Measurement of the functional user requirements of a piece of software
  - What should the software do for the user?
  - Not ’how’ or ‘why’
  - Result: size of the software: number of function points (FP)
- Purposes:
  - Software Project estimation
  - Project Performance measurement
  - Scope management
  - Project Benchmarking
  - RFP Management: contracting ‘price/FP’

Functional Size Measurement Methods

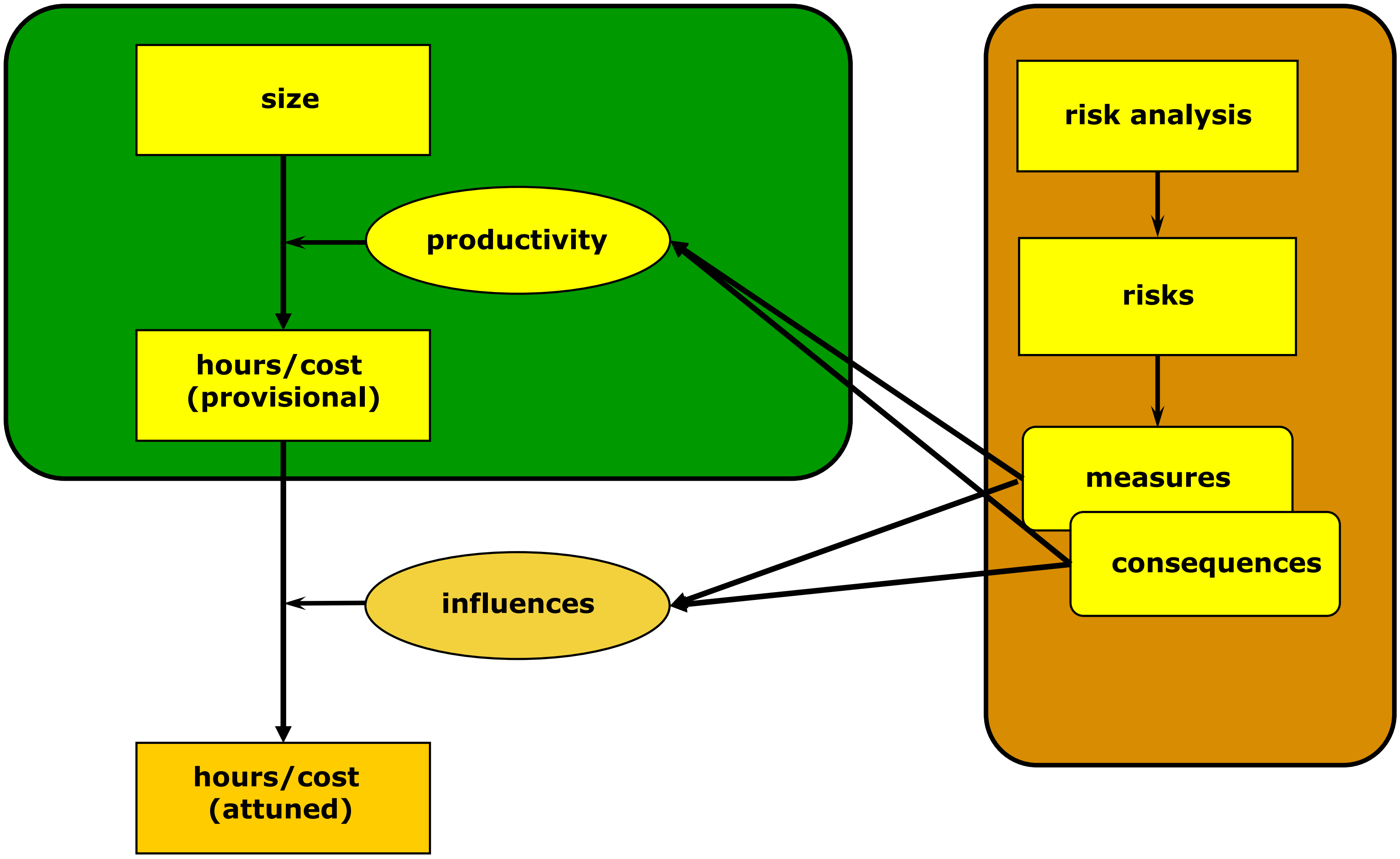
	FPA	COSMIC
Domain	Business applications	Business applications, Real-time applications, Infrastructure software
Data model required?	Required	Not required (but useful)
Measurement of separate components	Not possible	Possible
Size limit per function	Yes	No
Benchmarking data	Many	Some (ISBSG R11: 450)
Early sizing	Based on data model	Based on process model

Estimation models

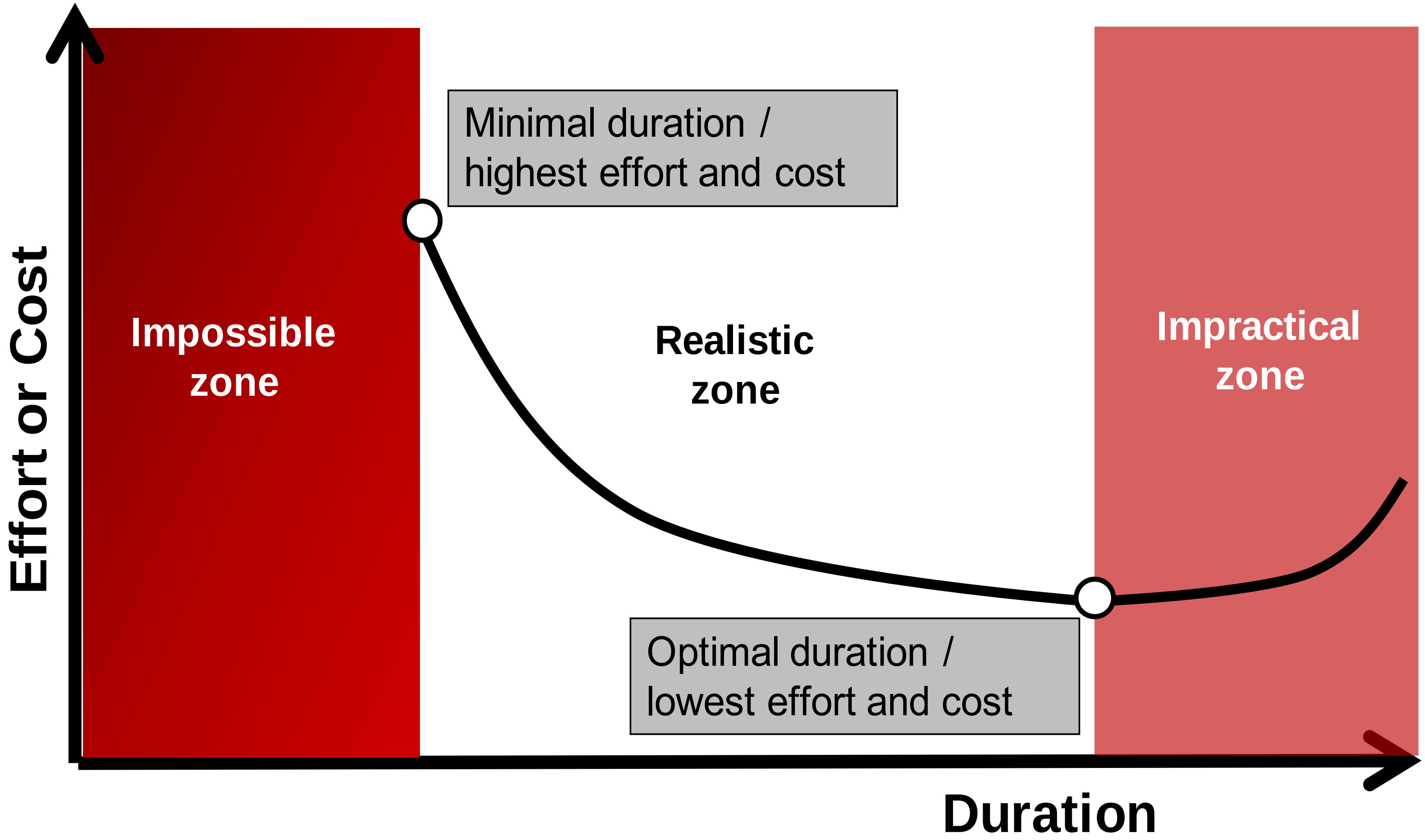
- Models that use functional size:
  - Galorath SEER-SEM
  - QSM SLIM
  - Price Systems – Trueplanning
  - Many more

- Parametric models are necessary for project estimation
  - To use historical data in new estimates
  - To understand the uncertainties and risk
  - To build scenarios
  - Communication to stakeholders
  - Non-linear influences of cost drivers

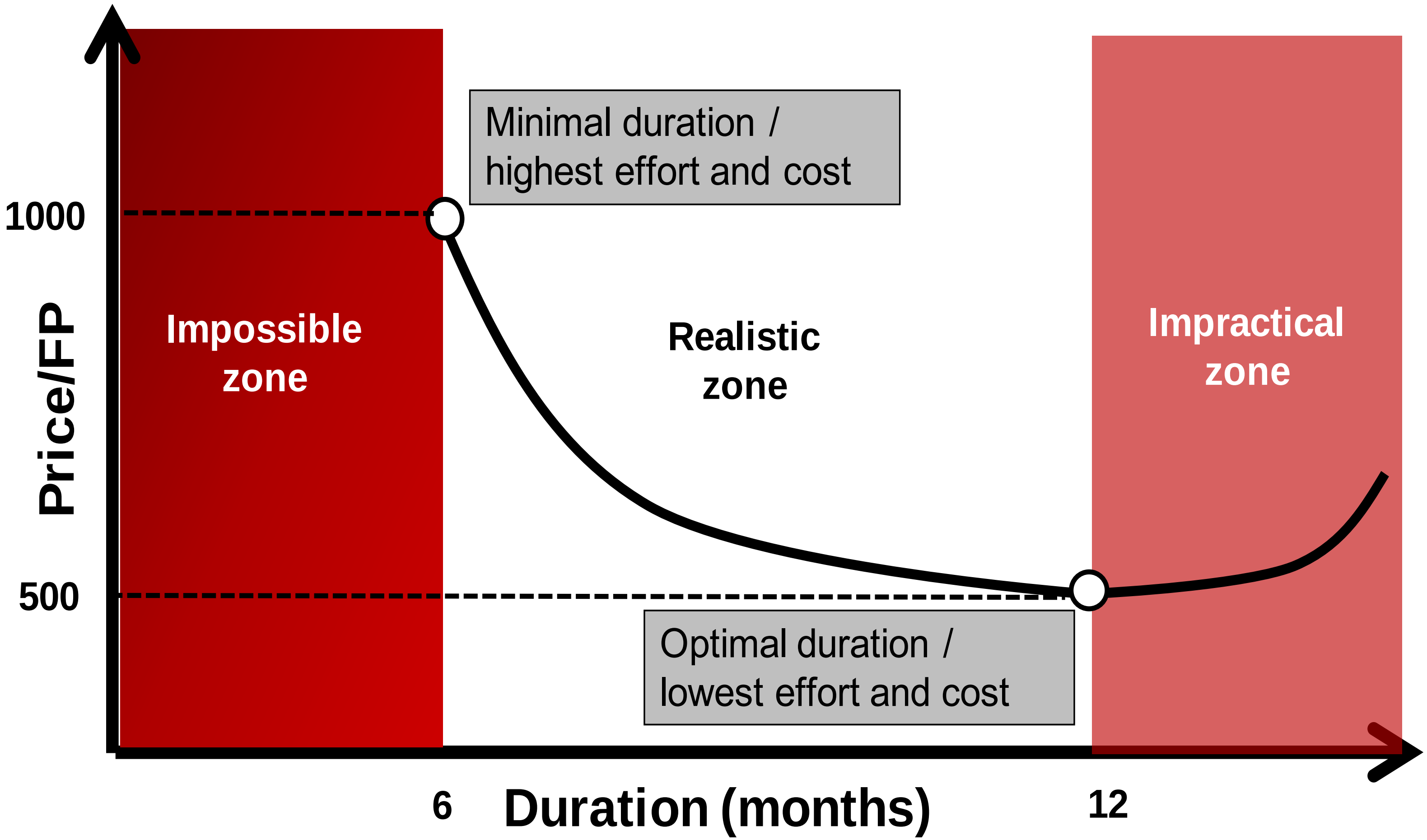
Basic estimation model



Effort / duration tradeoff



Example


Minimal duration /
highest effort and cost
1000
Impossible
zone
Realistic
zone
Impractical
zone
Price/FP
500
Optimal duration /
lowest effort and cost
6
Duration (months)
12

Historical data

- Parametric estimation models need historical data to estimate
- Preferred for estimation: data of the company itself
- For new types of projects or no data available: Industry data can be used

- Sources of industry data:
  - Data delivered with the models mentioned
    - QSM SLIM: trendlines based on slocs or FP
    - SEER-SEM: knowledge bases
  - Data by Benchmarking suppliers (Gartner, DCG, etc.)
  - Independent (International Software Benchmarking Standards Group)

ISBSG

- International Software Benchmarking Standards Group
- **Independent and not-for-profit;**
- Full Members are non-profit organizations, like IFPUG, NESMA, GUFPI-ISMA, FiSMA, QESP, DASMA, JFPUG, Swiss-ICT and CESI;
- Associate members: AEMES (Asociacion Espanola de Metricas de Software), ASSEMI (France);
- Grows and exploits two open repositories of software data (.xls):
  - New development projects and enhancements **(> 6000 projects**);
  - Maintenance and support **(> 1200 applications**).

- Everybody can submit project data
  - DCQ on the site / on request (.xls)
  - Anonymous
  - Free benchmark report in return

ISBSG

- Mission: "**To improve the management of IT resources** by both business and government, through the provision and exploitation **of public repositories of software engineering knowledge** that are standardized, verified, recent and representative of current technologies".

- All ISBSG data is
  - validated and rated in accordance with its quality guidelines
  - current
  - representative of the industry
  - independent and trusted
  - captured from a range of organization sizes and industries

Example: reality check

- A telecom company wished to develop a new Java system for the maintenance of subscription types;
- A team of experts studied the requirements documents and filled in the WBS-based estimation calculation (bottom-up estimate);
- They decide that an estimate of **5500 hours** and a duration of **6 months** should be feasible;
- The project manager decided not to believe the experts ‘on their blue eyes’ only and wished to carry out a **reality check**.

Reality Check: Effort

- An estimated FPA comes up with the expected size:
  - Min: 550 FP, likely 850 FP, Max 1300 FP
  - Implicit likely expert PDR: 5.500/850 = **6.5 h/FP**
- Selecting the most relevant projects in the ISBSG D&E repository show the next results:

		Functional Size		
	PDR (h/FP)	**550**	**850**	**1300**
Min.	3,2			
Percentile 10%	4,3			
Percentile 25%	**6,2**	**3.410**	**5.270**	8.060
Median	**8,9**	**4.895**	7.565	11.570
Percentile 75%	**12,9**	7.095	10.965	16.770
Percentile 90%	19,8			
Max.	34,2			
N	89			

5.500 hours seems **optimistic**

Reality Check: Duration

- Same analysis is possible
- Also, formulas have been published in the Practical Project Estimation book
- For instance:

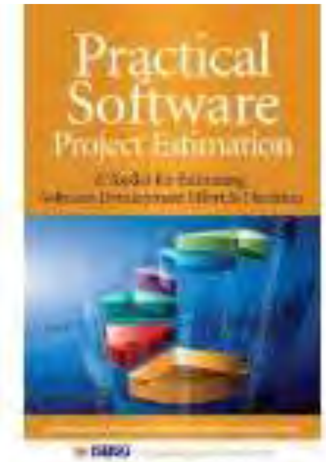


table C-2.2 Project Duration, estimated from software size only				
Functionele omvang	550	FP		
C uit tabel	0,507			
E1 uit tabel	0,429			
Duration =	C * Size^E1			
Duration =	**7,6**	elapsed months		

	550	850	1300
Duration	7,6	9,2	11,0

Results

- Expert estimate was assessed optimistic
- Adjusted Estimate:
  - Effort: 8000 hours
  - Duration: 10 months

- This turned out to be quite accurate! Without the adjustment, the project probably would have failed.

- The project manager now always carries out reality checks and is ‘spreading the word’.

Conclusions Part I

- Many software projects fail due to bad estimation practices;
- Estimates based on functional size are likely to be most accurate, because they are based on past performance data and objective measurement;
- Parametric estimation models are needed to understand the risks and to calculate scenarios;
- The availability of historical data is essential to all types of estimation!

PART II

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 using Agile software development
- In practical experience the terms iterative, incremental and agile may be used interchangably

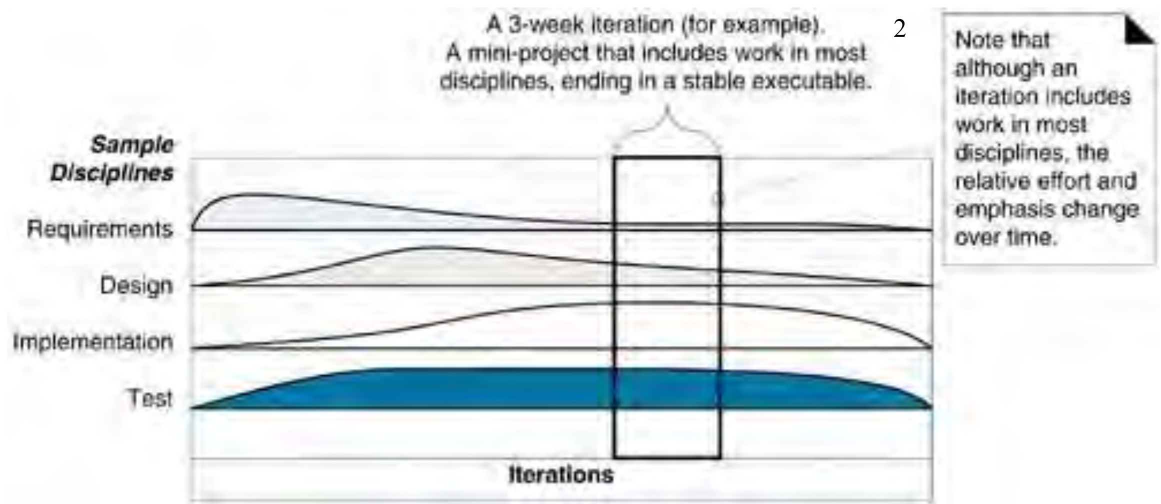
While Incremental/Agile programs say they do not have development metrics, I have almost always found them in the development room

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 [1]
  - 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
- 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 self-contained mini project, activities in many disciplines (requirements analysis, testing, etc.) occur within a single iteration



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

(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

Typical IID Problems – SLOC Count

		Code Counting Organization and SLOC Counts				
UCC Categories	Contractor Categories	Support Contractor 2011	Support Contractor 2012	Development Contractor 2011	Government 2011	Government 2012
Common		2,395	2,451	-	-	-
Connectors.	Connectors	52,511	34,012	70,385	55,438	27,627
Feature Packages	Feature Packages	5,887	8,173	49,277	7,468	18,836
Core Infrastructure	Core Infrastructure	36,133	19,276	162,011	461	211,228
Information Services	Information Services	23,245	-	11,432	25,256	-
Presentation	Presentation Infrastructure	14,523	-	-	51,813	-
Tools		35,743	-	-	1,813,456	1,813,948
	Task Services	-	-	-	-	-
In-House Dev	In-House Dev	-		-	1,852,357	-
Total		170,437	63,912	293,105	3,806,249	2,071,639

Through analysis, we were able to somewhat reconcile these large differences

Typical IID Problems (continued)- Gathering Historic Data

Estimated S/W Development Costs through the Completion of "X" Increments

	Contractor 1		Contractor 2		In-House		
	Increment Development	Agile Development	Increment Development	Agile Development	Increment Development	Agile Development	Totals
Inc a.	$ 411,600	$ -	$ 411,600	$ -	$ 100,000	$ -	**$ 923,200**
Inc b	$ 1,032,402	$ -	$ 1,108,939	$ -	$ 100,000	$ -	**$ 2,241,341**
Inc c	$ 1,711,706	$ 538,398	$ 1,664,882	$ 296,508	$ 549,322	$ 218,400	**$ 4,979,216**
Inc c Ext 1	$ -	$ 812,672	$ -	$ -	$ -	$ -	**$ 812,672**
Inc c, Ext 2	$ -	$ 186,242	$ -	$ -	$ -	$ -	**$ 186,242**
Totals	**$ 3,155,708**	**$ 1,537,312**	**$ 3,185,421**	**$ 296,508**	**$ 749,322**	**$ 218,400**	**$ 9,142,671**

Software Maintenance as a % of Develoment Costs

	Factor	Annual Maint.	$/FTE	FTEs *
Low	5%	$ 457,134	$ 213,600	3
Most Likely	10%	$ 914,267	$ 179,412	6
High	13%	$ 1,188,547	$ 155,141	8

One could suggest that these problems are common to all Software Intensive Programs

ICEAA CEBoK

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[5] 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 [6]
- 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

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

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

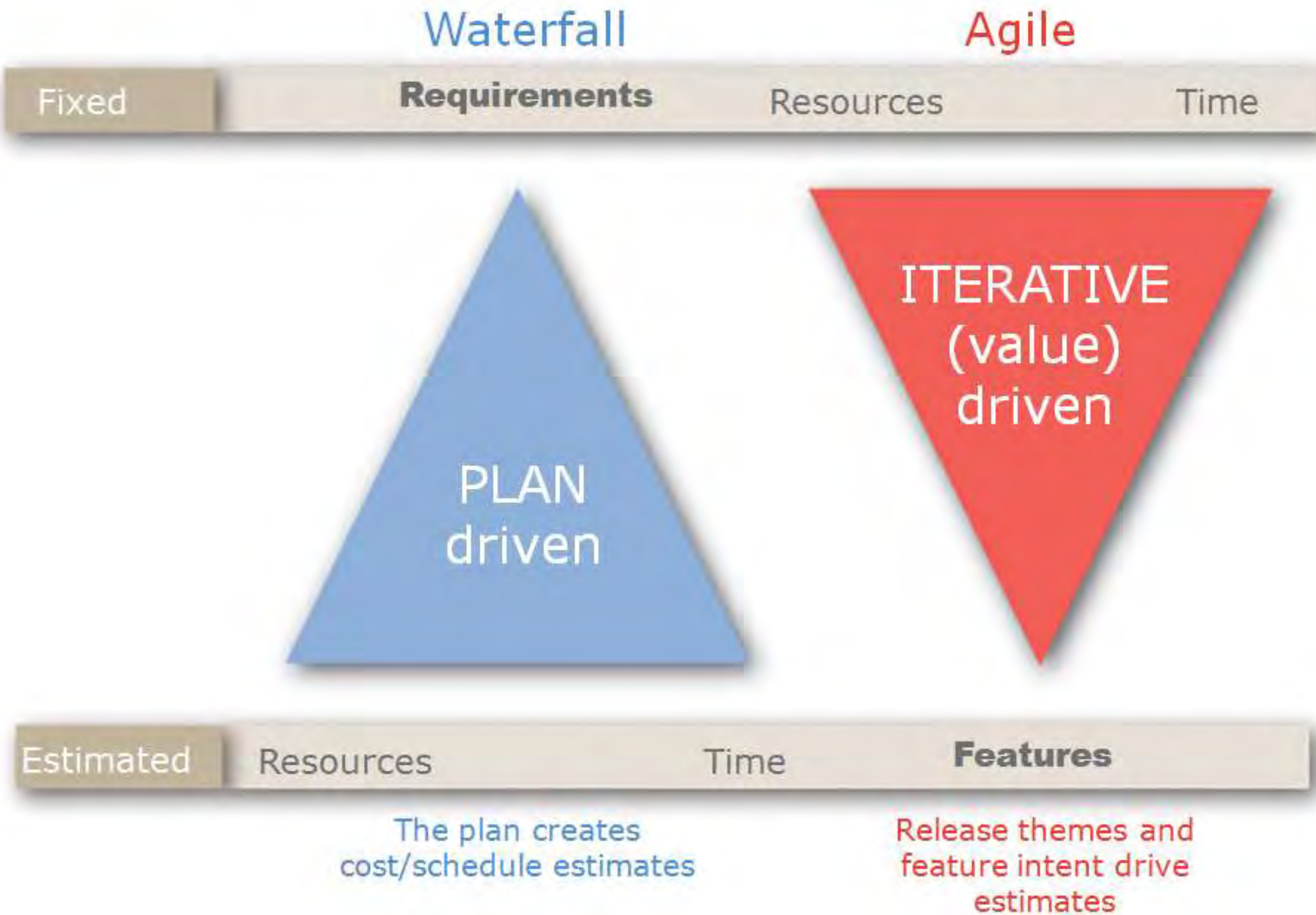
Welcome to Agile

- 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



9

ICEAA CEBoK

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

ICEAA CEBoK

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*)

ICEAA CEBoK

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

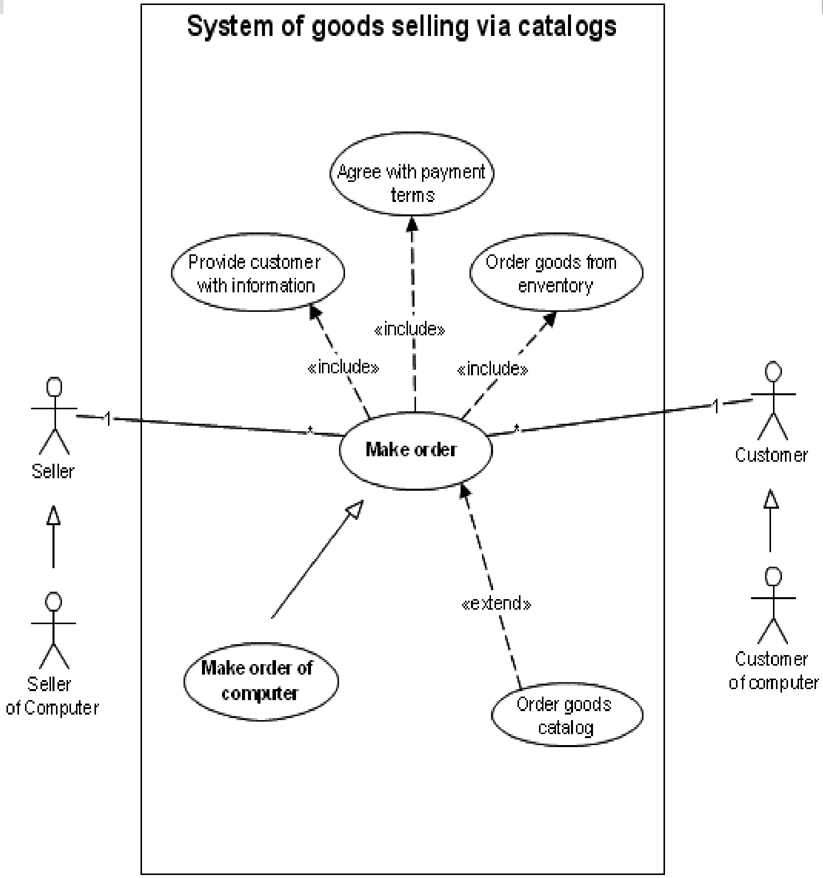
The weakness of this approach is justifying the team size, number of sprints, sprint length and total required to meet the requirement

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.

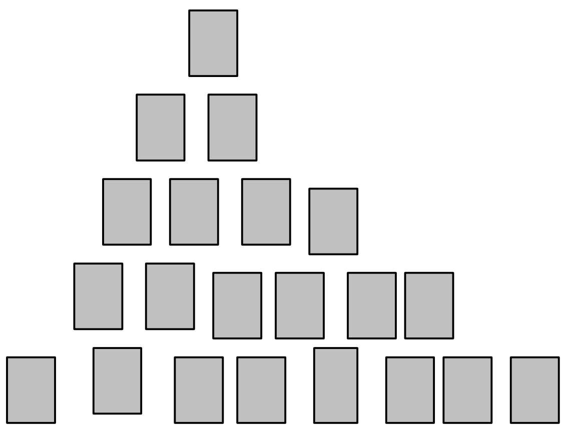
What is a Use Case Point?

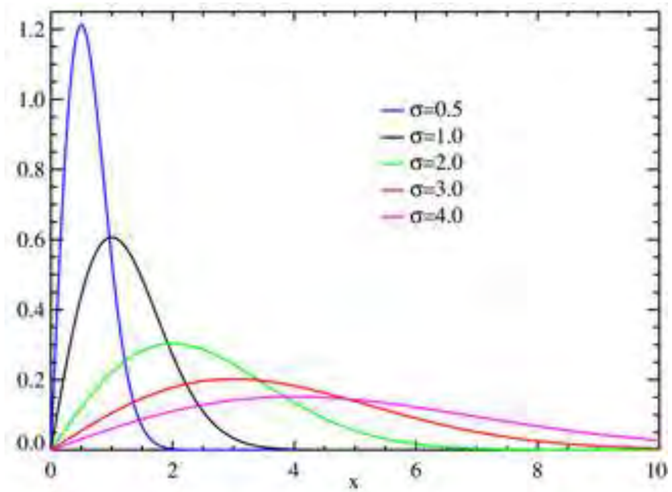


- A weighted count of actors and use cases
  - Actor weight is classified as:
    - 1 – Simple: highly-defined and elemental, such as a simple API call
    - 2 – Average: protocol-driven interaction, allowing some freedom
    - 3 – Complex: potentially complex interaction
- Use Case weight is classified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

Moving to Automated Models

- Step 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

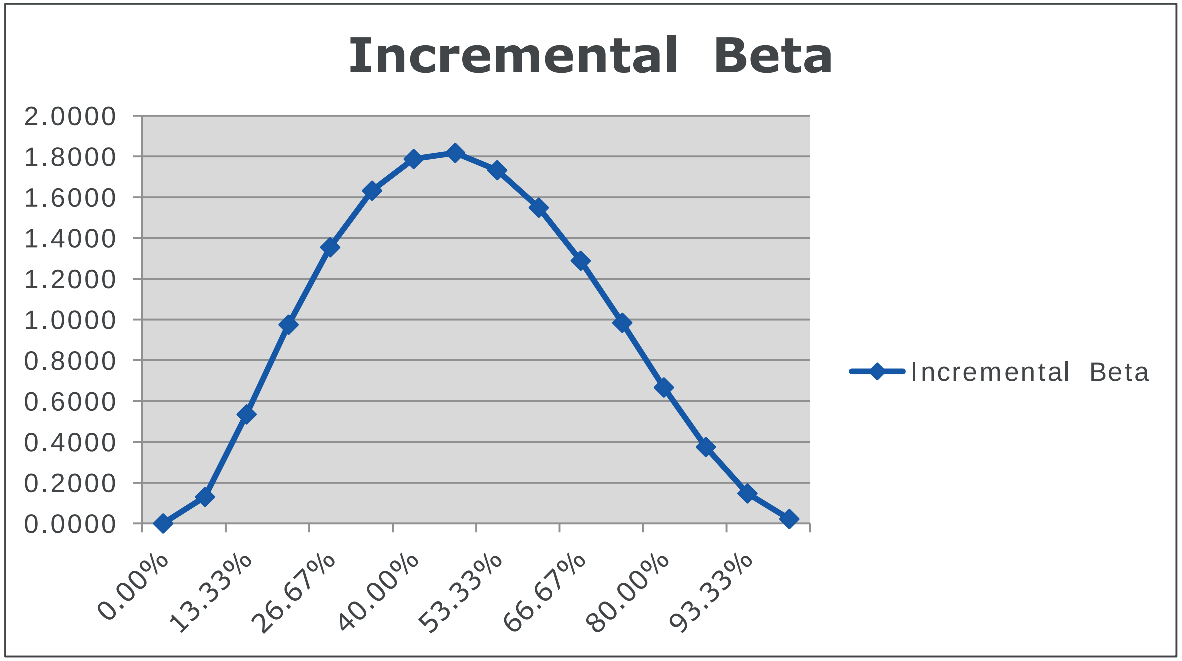
Scale	Complexity Description	Effort Multipliers
- -	Significantly less complex	0.7
-	Slightly less complex	0.9
=	Complexity equivalent to Increment X	1.0
+	Slightly more complex	1.3
+ +	Significantly more complex	1.7

- 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
- 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

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.*

5

ICEAA CEBoK

Contact Information

- Dr. Harold van Heeringen
  - Email: harold.van.heeringen@sogeti.nl
  - @haroldveendam
- Bob Hunt
  - Email: BHunt@Galorath.com
  - Phone: 703.201.0651