Selecting Software Estimating Techniques that Fit the Software Process

Kal Toth

Portland State University

ktoth@cs.pdx.edu

Updated version of this paper

The original version of this paper was published in PNSQC 2009. In response to attendees and readers I updated the paper to clarify my usage of "incremental development" to encompass staged and concurrent development that could lead to independent product releases.

Abstract

When embarking on a new project, the software engineering manager will need to decide early on whether to follow a Waterfall, Agile, Prototyping, Incremental or some hybrid or variant of these software processes. To assess project feasibility, to secure budget, and to properly plan resources and schedules, responsible managers should also decide about their software estimating process - whether to us expert judgment with estimating rules-of-thumb; parametric techniques like COCOMO and Function-Points; or estimating databases populated with analogies and proxies from prior projects. The characterizing attributes of a given new project greatly influence software process and estimating choices. This paper provides context and guidance that will help the software practitioner understand the influences of project attributes on the selection of suitable software processes and on software estimating techniques when embarking on the next software project. This paper will also assist companies to decide how to best apply their resources to maintain a suitable software estimating infrastructure to support project planning and execution.

About the Author

Kal Toth is the Director of the Oregon Master of Software Engineering Program (OMSE) and Associate Professor in the Maseeh College of Engineering and Computer Science, Portland State University (PSU). A Professional Engineer (P. Eng) with a software engineering designation registered in British Columbia, he has a Ph.D. from Carleton University in electrical and computer systems engineering. Kal has over 25 years of management, technical, and consulting experience leading and working for a range of technology companies and organizations including Hughes Aircraft, Datalink Systems Corp., BC Software Productivity Centre, the CGI Group Inc., Intellitech Canada Ltd., National Defence (Canada), Communications Canada, and External Affairs (Canada). He has managed and participated in a technical capacity addressing project management, software quality, and information security aspects of air traffic control, e-commerce, distributed information, and packet-switching systems. At PSU he facilitates software engineering courses including software project management, software engineering principles and processes, software quality, and practicum projects.

Introduction

Software engineers and teams are constantly being asked questions like: “How long will this take?” and “How many developers are needed to get it done? Far too often, estimates are produced in an ad hoc manner and yield unreliable results, or results that are misunderstood and misused. The prudent software engineer will study the attributes of the software problem at hand and systematically apply their experience on similar projects to come up with useful estimates. They will also revise their estimates on an ongoing basis as project uncertainties firm up. The most experienced software professionals will apply software estimating methods that have worked consistently for them in the past. Mature software companies will support their projects by sustaining software estimating culture and infrastructure including proven estimating processes, tools and historical project data.

When practitioners work on similar projects, in similar domains, using similar software processes, and their company’s standard estimating process, they tend to produce fairly useful and consistent estimates. But what does a team or company do when they shift from Waterfall to Agile development? What happens if they transition from an estimating culture without systematic estimating to one where estimates are integral to the software process? What about the developer making a career move from the aerospace industry with a formal estimating culture to a company or industry sector in the world of commercial software products, e-commerce applications, enterprise information systems, embedded computing, or open source software? Such radical shifts may render a proven estimating technique to be inappropriate in the new domain. At the very least, several adjustments will be needed to re-tune and re-target the estimating technique being used to adapt it to the new development culture.

To make the necessary adjustments, software engineers first need to understand the ramifications of the estimating techniques they used in the prior context. Next they need to differentiate the new types of projects they expect to tackle from their previous ones (What’s different?). Finally they need to learn which aspects of their estimating technique(s) can be re-used, and which ones will need to be adjusted or re-worked to support the new domain. Armed with a suitable mapping of estimating techniques and development processes, the software professional should be able select the most appropriate estimating technique for the new domain, identify the gaps, and transition to the new or revised estimating approach.

One approach to tackle these challenges is to consider key project attributes that distinguish projects from each other. If we can use these attributes to characterize both software process selection and software estimating choices, it should be possible to relate software processes and estimating techniques to achieve “best fit” (see Figure 1). This should help companies and their software engineers focus on the most suitable range of estimating techniques for their context thereby improving estimating effectiveness.

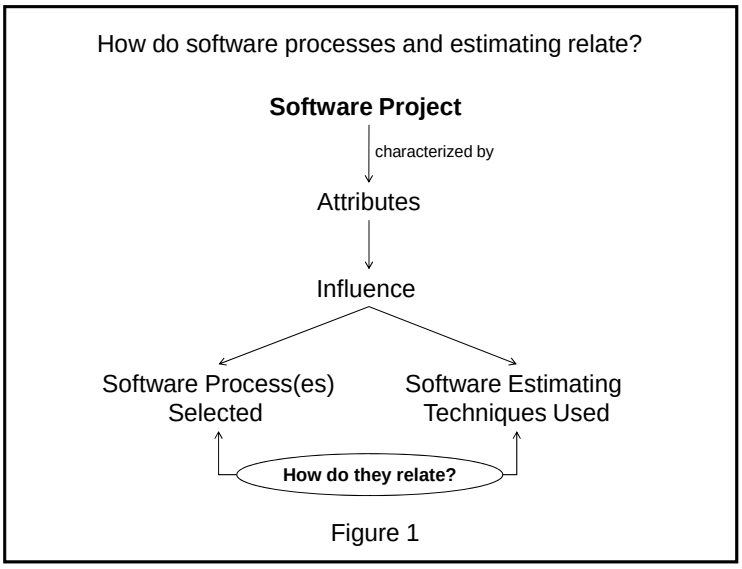

Figure 1

Central Questions Posed by this Paper

Development teams naturally tend to select the software process they are most familiar with. This may often be the correct choice, especially if the company tends to take on similar projects in a given application domain. However, many companies, especially larger ones, have projects with widely ranging process needs and they often have multiple software process cultures and competencies across their various software development groups.

As suggested in [1], the unique attributes of a new software project should be examined and related to the company's capabilities when deciding on the software process to be adopted. Once this choice is made, project planning can begin in earnest including the application of suitable processes to estimate effort and schedules. Given that software estimates are derived, one way or another, from work activities, it follows that the chosen software estimating technique(s) should also account for the unique attributes of the project.

Such relationships between software process and estimating for a given project are not widely acknowledged across the software engineering community. This paper explores this gap by addressing the following questions:

- What are the key attributes characterizing an arbitrary project?
- How do such factors influence process selection? [1]
- How do such factors influence software estimating choices?
- Which SW estimating techniques are most suitable at a given stage of software development?
- Which SW estimating techniques are most suitable for a given software process?

Key Assumptions Underlying this Paper

The primary goals of software estimating are to: confirm project feasibility, rationalize budget for the project, and use estimates to control the project (determine when a new or better estimate is required).

Ideally, projects should start with stable objectives and scope:

- Scope should not change very much, otherwise you are really tackling a different project; [2]
- Properly defined scope covers the breadth of project requirements and constraints;
- When required functions and features emerge, they should be consistent with objectives and scope;
- Requirements should not expand scope without stakeholder agreement (those controlling budgets).

Estimates are uncertain, but they should improve with time: as the project progresses, new knowledge yields better estimates which reduces estimating uncertainty; estimating should be iterative - new estimates are opportunities for re-shaping the project, re-targeting objectives and scope as required.

The Customer and Contractor must mutually embrace success:

Both parties must recognize that requirements are rarely certain or static and that resources (budget and schedule) are finite; they must agree to consider and accept trade-offs between requirements, budget and schedule; also agree to explore and understand impacts of key project influencing factors while meeting

[1] Project influencing factors can be used as "adjustment factors" or "drivers" when producing estimates

[2] If scope is pushed out significantly by requirements changes, the customer and contractor should be prepared to revise estimates, assess project impacts on budgets and schedules, and renegotiate scope or trade-off requirements to fit within previously agreed upon budgets and schedules.

mutual commitments; The Customer and Contractor are thereby mutually involved and committed to achieve success (a.k.a. “win-win”).

Software Estimating in a Nutshell

Software Estimating Basics

Most software estimating techniques start by estimating the “size” of the software product deriving effort and schedule estimates from software size. Historical and industry data are used to arrive at standard allocations of effort to such size measures; and schedules are derived using other guidance that distributes effort over the timeline or calendar. Several (but not all) techniques apply project adjustment or influence factors to account for various attributes of the project.

Software size represents the magnitude of the problem being tackled. The underpinning rationale is that effort and schedule should naturally correlate with the relative difficulty of the software problem being tackled. Software “size” is typically measured in terms of the number of features or “points” representing the amount of software to be constructed – for example, the number of functions, stories, use cases, objects, components, files, user interface widgets, lines-of-code (LOC), etc. Such points are typically categorized by relative complexity and other product attributes with effort estimates being associated with each type of point. Nominal effort estimates per point come from industry data or past company projects. The software estimator’s task, then, boils down to assessing the number of points of each applicable category and calculating the total estimated product size, effort and schedules.

Of course, many practitioners directly estimate software construction effort without explicitly relating their estimates to product size – deriving loading profiles and schedules from these effort estimates.

A critical aspect of software estimating is to determine where to start. Figure 2 illustrates this challenge. Should we first estimate size and then estimate effort and schedule? Or should we start with a fixed schedule and then determine how much software (size) and effort can be fit into that schedule. Or is it really the level of effort that should drive the estimate? In other words, should we estimate how much software can be built within allocated effort and then find a schedule that will best accommodate this effort and the amount of software to be built?

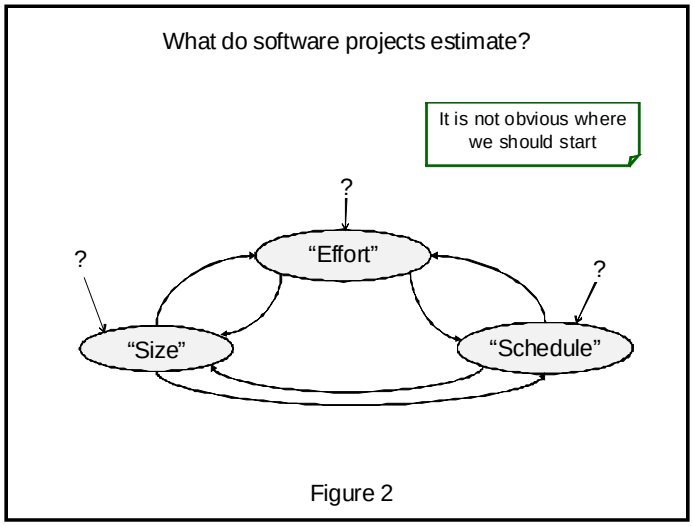

Figure 2

Using Historical Project Data

Estimates can be derived from historical industry and company data. Parametric models with estimating tools exist to support companies lacking their own historical project data. Tool vendors have collected, categorized and analyzed data from numerous projects across various application and technology domains. Their tools use empirical formulas to calculate estimates according to inputs that characterize the properties (influencing factors) of the software project to be estimated. The estimator’s main task is to identify and evaluate the new project’s influencing factors and provide these as inputs to the tool.

Historical databases of past projects can improve the quality of estimates and reduce estimating overheads, that is, the effort and time expended to produce good estimates. Such databases represent the collective software project knowledge of the company, thereby providing consistent and more certain estimates, than the other techniques mentioned above. Historical estimating biases (over and under estimating) can also be derived and used to adjust estimates. Furthermore, useful rules-of-thumb can be extracted from such databases and used to produce rough-cut estimates that match the company's project profiles. Estimating databases are populated with software components from prior projects. Referred to as software "analogies" and "proxies" they are "sized" and categorized by their project attributes. Statistical techniques (e.g. regression analysis) are applied to estimate and adjust size, effort and schedule from previous analogies and proxies.

Using Influencing Factors to Drive Estimates

It is fairly clear that the nature of the software product itself and the company's capabilities are the main factors that drive a given project estimate. But are there other useful factors? Luckily, we can draw upon parametric estimating techniques that employ detailed attributes of a software product, the chosen development environment, project team competencies, and other project aspects.

Take note that these parametric estimating techniques do not use the same or equivalent influencing factors (though they somewhat overlap), and many estimating techniques use them only superficially, or even disregard them altogether. For example, experienced software engineers often produce "fuzzy" estimates from their experience with similar prior projects together with prerequisite knowledge of the current project's objectives and scope - project influencing factors are considered implicitly - rarely explicitly. In other cases, rough estimates are derived from industry or company rules-of-thumb without addressing project attributes systematically.

If project influencing factors are not available or not used, the prudent estimator should understand and communicate to management and the customer that such estimates are very rough (uncertain) and that they should be accordingly used with caution.

Estimating Uncertainty

Software estimating should <u>not</u> be thought of as a one-time effort for a project. Estimates and plans should be updated iteratively during or at the completion of each significant development stage. Early on, estimates tend to be highly uncertain; as the project moves forward, additional knowledge and experience accumulate hence yielding tighter estimates (see Figure 3). Barry Boehm called this the "Cone of Uncertainty" [2].

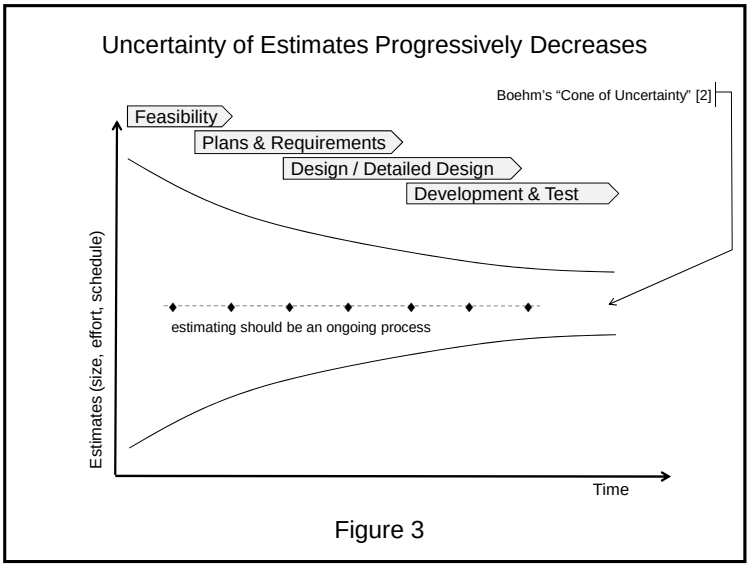

Figure 3

Recommended Readings on Estimating: Barry Boehm [2], Steve McConnell [3], and Mike Cohn [4].

Software Process Selection

What Influences Software Process Selection?

To understand how project factors influence software process selection, let's review the rationale that motivate the software practitioner to select one software process over another:

- **Waterfall development** is best suited for projects with high complexity, criticality, and maintainability requirements. Waterfall processes are fairly sequential allocating detailed analysis, specification, and design decisions in the initial stages of development to address performance, security, reliability, safety, maintainability, and other non-functional requirements including design constraints. This approach implies higher process ceremony than most of the alternative software processes.
- **Agile development** focuses on working code over detailed specification, analysis and documentation. Short iterations (often called "sprints") of fairly systematic development are advocated with close customer involvement, test-driven development, continuous integration, pair-programming, and ongoing refactoring. Agile processes are characterized by lower process ceremony than Waterfall.
- **Prototyping**, sometimes referred to as evolutionary development, is exploratory in nature tackling unknown or poorly understood aspects of the project - for example, user interface and technology uncertainties. Typically, software prototypes are developed using low process ceremony to build knowledge before transitioning the project into more systematic development following Waterfall and Agile development processes.
- **Incremental development** (a.k.a. staged development and release) aims at increasing project scalability by applying divide-and-conquer strategies to fashion independent development activities that may be executed concurrently or serially. Such independent development is achieved by using thorough analysis and design techniques to partition the requirements and/or the architectural design into loosely coupled components that may be independently addressed. Development increments may be governed by Prototyping, Waterfall or Agile approaches and may deliver separate releases.

Incidentally, none of these processes is "pure" – they don't adhere dogmatically to a given process. For example, Waterfall processes do not necessarily dictate strict sequencing, high ceremony processes, or massive documentation. Similarly, Agile does not mean abandoning proven design principles, coding standards, or test coverage analysis.

Aside: You might wonder why I left iterative and spiral software development off the above list of processes. My reasoning is that Prototyping, Agile, Waterfall and Incremental processes possess very distinctive properties. And both iterative and spiral development processes are really strategies that can be applied to fine-tune them. For example, an entire project could be broken down into iterations along the time-line where each iteration delineates several concurrent increments of development. Iteration is also a way of organizing the work within a given development increment to progressively explore software work products as they are being prototyped, or augment functional capabilities as new stories are formulated during Agile development, or execute mini-waterfall processes to progressively refine and mature a branch of the software development effort. Meanwhile, spiral development, which is very closely related to iterative development, allocates risk management activities to separate spirals where each could follow Prototyping, Agile or Waterfall processes to address specific risk mitigation objectives, albeit using different styles of development.

Software Process Ceremony is Driven by Project Influencing Factors

Figure 4 depicts how software processes relate process ceremony to the difficulty of the problem being tackled. Prototyping, Agile, Waterfall, and Incremental development are shown as software processes that cover a scale of increasing process ceremony addressing increasingly difficult (harder) software problems. For example, a project to develop a large, complex system to control a nuclear reactor, an aircraft navigation system, or an embedded medical device, may be best organized as a number of concurrent increments of development that incorporate Prototyping, Agile, and Waterfall processes across an iterative lifecycle. In other words, a hybrid high-ceremony process is planned out to address the needs of a very difficult software problem.

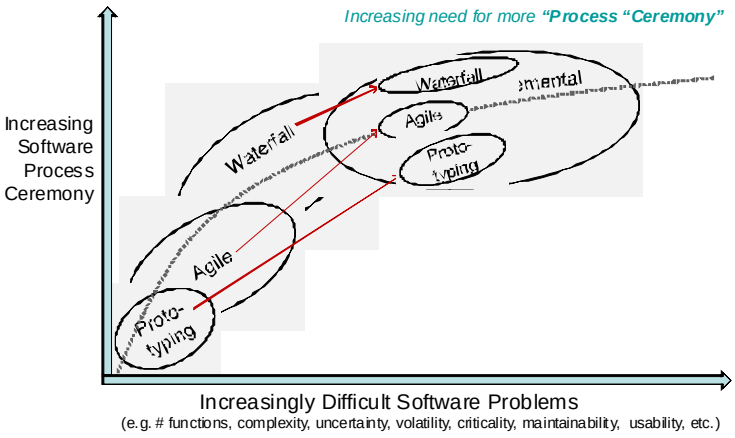

Figure 4

Note: See Toth [1] for insights into process selection, and Brook and Toth [5] on process ceremony.

Customer-Supplier "Flexibility"

Customer-supplier "flexibility" represents the degree to which the customer and supplier have agreed to constrain requirements, budget and schedule. For example, the budget might be agreed to be fixed while the requirements are flexible (e.g. allowed to vary within project scope), and the schedule is constrained (e.g. the schedule may be pushed out by some agreed period of time). Futrell et. al. in [6] recommend explicitly addressing such flexibility in the project plan using a matrix. Table 1 illustrates customer-supplier flexibility for 4 projects.

Customer-Supplier "Flexibility" Biases the SW Process

	Project A	Project B	Project C	Project D
Requirements	"Fixed"	"Fixed"	"Flexible"	"Flexible"
Budget	"Constrained"	"Flexible"	"Constrained"	"Fixed"
Schedule	"Flexible"	"Constrained"	"Fixed"	"Constrained"

Table 1

Project A: Consider Waterfall to ensure fixed requirements are addressed within the constrained budget. Estimating effort should focus on assessing whether an acceptable schedule is feasible;
Project B: Consider Waterfall to ensure fixed requirements are addressed within the constrained schedule. Estimating effort should assess whether an acceptable budget is feasible;
Project C: Consider Agile since time-to-market / product release is the top priority. Estimating should be "design-to-schedule" within an acceptable budget;
Project D: Consider Agile since budget is fixed. Estimate should be "design-to-budget" within an acceptable schedule.

How does flexibility affect estimating?

As discussed earlier, the goal of estimating is to assess the expected product "size", the effort to build the product, and the schedule within which to build the product. However, it is not always clear where to start, and it is not always possible to arrive at a reasonable size estimate. One interesting observation, is that a flexibility matrix focuses our attention on a natural starting point. Consider the following project flexibility possibilities:

Requirements are "fixed": This implies that the customer has a fairly fixed idea of functions, features and non-functional requirements that have been well-specified in a software requirements specification (SRS) or similar document. The customer has likely agreed to a fairly systematic change process as well. The main goal, therefore, is to estimate acceptable budget and schedule to implement the requirements. Some flexibility with respect to budget and schedule may be allowed. Such projects are considered "requirements driven".

Schedule is "fixed": This implies that the customer or marketing department is highly focused on delivery milestones and shipping dates to meet market urgency or operational needs. The main goal is to estimate which requirements can be implemented within the allocated schedule. They are also likely to allow changes in functions and features mid-stream to meet requirements. And some budget flexibility may be permitted. Such projects are referred to as "schedule driven" or "design-to-schedule".

Budget is "fixed": This implies that the customer expects the project to operate within a budget envelop that will directly or indirectly fix the total allowable effort. Exceeding the budget will likely lead to serious customer dissatisfaction and possibly even payment penalties. Such projects will endeavor to assess which functions and features can be feasibly implemented within the fixed budget. Depending on the customer's preferences, trade-offs between software size and schedule may be made before settling on an acceptable project posture. Such projects are considered "budget driven" or "design-to-cost".

Figure 5 illustrates these three project flexibility scenarios and reinforces the iterative aspect of estimating. For example, a design-to-cost (budget driven) project starts with budget and effort considerations but is likely to trade off various requirements and schedule options before settling on a combination of effort, requirements and schedule that will satisfy the budget constraint.

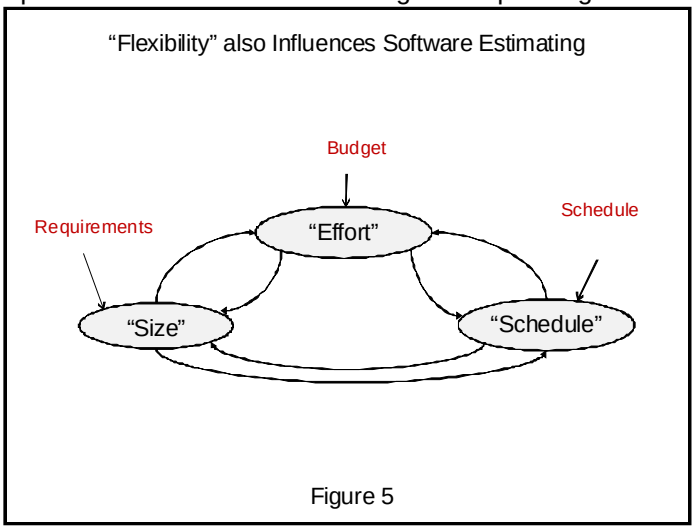

Figure 5

Common Estimating Techniques and Strategies

To explore the implications of project influencing factors, let us delve a bit further into the common estimating techniques (illustrated in Figure 6) focusing on key differentiating aspects:

Expert Judgment: Assumes an experienced software practitioner is familiar enough with the application domain to make a sound estimate. The expert is really calling on their personal recall of past software projects and project attributes. Such estimating is typically referred to as rough-cut or "fuzzy" estimating". When the base data is simply memory recall, we refer to it as "Guestimating".

Function-Point Estimating: Function and feature points and categories are derived from industry data. The estimator assesses the number and complexity of functions and features by category to arrive at a size estimate and uses estimating adjustment factors that characterize the requirements and the design.

Analogy-Based Estimating: Past project analogies are characterized by their attributes, actual software size, effort, and project schedules. The estimator searches for comparable analogies in the historical database to arrive at representative estimates.

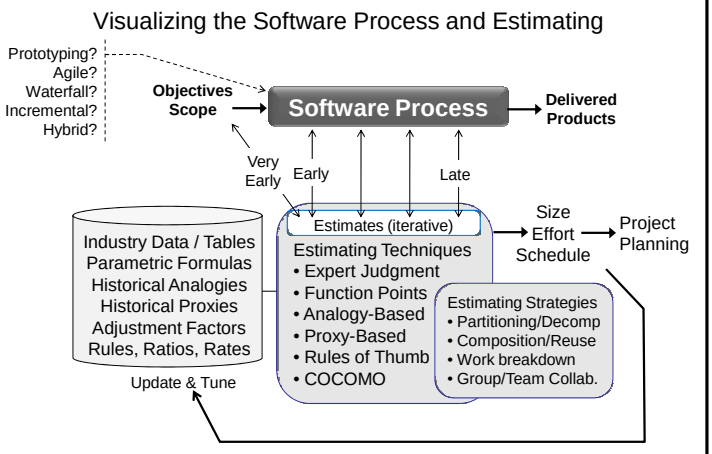

Figure 6

Proxy-Based Estimating: Software features (e.g. stories, objects, components, widgets, etc.) of past projects are counted, characterized, categorized, and maintained in the estimating database. The estimator assesses the number and size of each feature point in the new product, searches for comparable proxies, and extrapolates size, effort and schedules from this base data.

Estimating Using Rules-of-Thumb: Rules-of-Thumb include: software productivity rates (LOC/unit of time), possibly categorized; re-use factors such as the relative cost of reusing software components as a percent (%) of the original development effort; and effort distribution factors used to distribute rough-cut estimates over project activities and phases. Rules-of-Thumb may be derived from industry data or may be drawn from a company's software project history.

COCOMO Estimating: A fairly widely used parametric technique, COCOMO used empirical formulas based on regression analysis of industry data categorized by relative project complexity ("organic", "semi-detached", or embedded"). COCOMO uses estimating adjustment factors that characterize product, computing environment, personnel and project attributes. COCOMO uses an input size estimate in lines-of-code (LOC) to yield effort and schedule estimates. Several COCOMO variants and products are available.

Estimating Strategies for Refining Estimates

Design to Cost/Schedule: This is really a cost or schedule driven estimating technique. Through trial and error, size estimates yielding the desired cost or schedule are located. Software size may be derived from analogies, proxies or function-points.

Partitioning/Decomposition: Requirements may be partitioned into relatively independent parts that can be separately estimated. Once a stable design is developed it can be partitioned into loosely coupled parts and separately estimated.

Composition: This is actually a design technique that can be used to produce bottom-up solutions from components that can be estimated and aggregated to yield rolled-up estimates.

Group Estimating: Delphi estimating involves bringing together several software practitioners who use their expert judgment to converge on a collective estimate through consensus. Team members may also be called upon to support bottom-up estimating, a recommended strategy to secure commitment of team members to achieve results that meet their own estimates.

Project Influencing Factors

It should be clear by now that project attributes influence both the software process and the software estimating technique(s) we may elect to use. Top-level project flexibility attributes have been identified (requirements, budget and schedule) that influence how to begin strategizing about estimating. These factors also help us decide whether to use Waterfall or Agile development – but they tell us less about how to choose when to use Prototyping and Incremental development. The discussion about process ceremony should encourage us to consider incremental development strategies to scale-up by organizing hybrid development processes.

Meanwhile, we have examples of estimating techniques that leverage project attributes to account for project differences. COCOMO, for example, uses various product, personnel competency and other project attributes. And Function-Points uses product-specific attributes almost exclusively. Of course, many estimating techniques almost totally ignore such project attributes assuming the estimating expert will somehow bring them into play as required.

Figure 7 captures and synthesizes the fundamental observations made earlier about project flexibility, COCOMO, Function-Points, and personal experience. This figure presents what I believe to be a fairly complete "taxonomy" of project influencing factors. The hierarchy is first broken down by project flexibility drivers, and then further broken down by product, environment, personnel and project pressure / involvement factors. Several of the project influencing factors map to Function-Points and COCOMO adjustment factors.

Important Note: Although Function-Points and COCOMO provide guidance on assigning weights to their adjustment factors, they use different value ranges and aggregation methods. I have made no attempt herein to rationalize these scales. For the purposes of this paper, the reader should simply assume that each of the influencing factors may take on a range of values (e.g. small to large; few to many; low to high; etc) and that such factors should be used to adjust initial estimates upwards or downwards as appropriate.

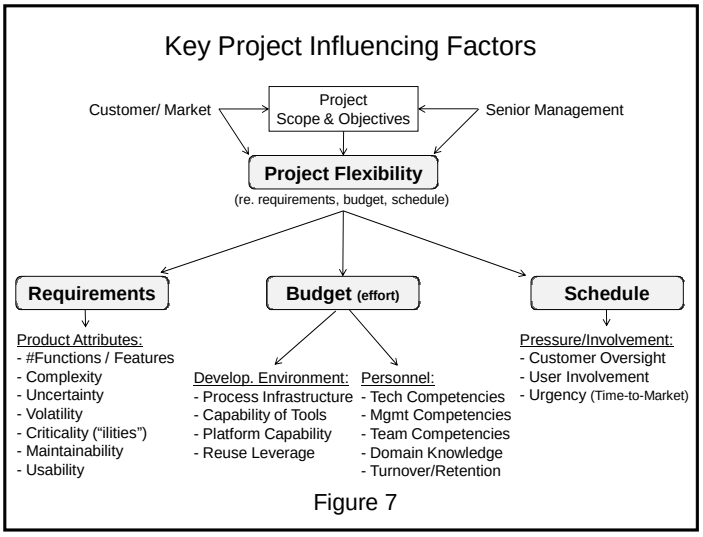

Figure 7

Project Flexibility

As discussed earlier, this represents customer-supplier flexibility regarding the project's requirements, budget and schedule within agreed objectives and scope. For example, the customer and supplier might agree to be flexible with respect to requirements, but fixed with respect to project schedule (completion date) while constraining total budget.

Product Attributes

Functions / Features (#): the relative number of functional requirements and features; from a few key functions to a very large number of functions, use cases, scenarios, user stories, etc.
Complexity: represents the relative complexity of the product in terms of the number of dependencies among functional requirements, features, external interfaces, and technologies.

Requirements Uncertainty: unambiguous, precise, and logically complete specifications denote low uncertainty; high-level, vague and incomplete requirements imply high uncertainty.
Requirements Volatility: the rate at which and the extent to which the customer and users are expected to make changes to required functions and features throughout the project.
Criticality: characterizes the security, reliability, safety, availability and performance requirements (a.k.a. "ilities") of the project – for example, web-sites, games, electronic toys are examples of low criticality applications; aerospace, military, and embedded applications are typically high criticality projects.
Maintainability: relates to the quality of the documentation and the modularity of the design – ad hoc, experimental prototypes do not need much documentation; highly maintainable systems have extensive and high quality requirements specifications, design documentation, and installation manuals.
Usability: represents the relative ease of use of the product's user interfaces and navigational controls.

Personnel / Team Factors

Technical Competency: analysis, requirements, design, coding, integration, testing, etc.
Management Competency: planning, estimating, directing, monitoring, reporting, negotiation, etc.
Team Competency: communications, collaboration, coordination, interpersonal, etc.
Application Domain Knowledge: knowledge and experience with domain functions and features.
Retention / Turnover: ability of the company and team to retain quality and proven personnel.

Development Environment Characteristics

Process Infrastructure: technical (eg coding) standards, spec templates, checklists, guidelines, etc.
Capability of Tools: extent and quality of requisite tools supporting reqts, design, coding, testing, etc
Platform Capability / Volatility: capability and maturity of O.S., DBMS, commercial apps, etc.
Reuse Leverage: extent and quality of reusable components and source code from prior projects.

Pressure and Involvement

Customer Oversight: the degree to which progress is made visible to customer representatives through reporting, documentation, and demonstrations.
User Involvement: the degree to which users get involved in developing and reviewing work products from requirements, through development, to product acceptance.
Schedule Pressure / Market Urgency: relative urgency expressed by the customer to deliver and release final products.

Key Project Factors Influence Software Process Selection

With reference to Table 2, the following summarizes the "key drivers" (only) that influence the selection of each software process (note: lesser project influencing factors have been set aside):

Prototype Development is driven by product complexity, unknown platform capability (technology risk). Also usability, requirements uncertainty and unfamiliar domain knowledge (requirements risks).

Agile Development is driven by schedule pressure (urgency to deliver to market). Volatility and uncertainty of requirements are also key Agile drivers. Agile is particularly dependent on technical competency and user involvement.

Waterfall Development is driven by criticality and maintainability requirements. Waterfall is also heavily dependent on process infrastructure, management skills, and customer oversight. Typically, the customer requires significant project visibility through documentation, reporting and in-progress demonstrations.

Incremental Development is driven by large-scale, that is, large numbers of functions, features, and complexity. An incremental process implies strong process infrastructure, customer oversight and project management competencies.

Key Factors Influence Software Process Selection

	Prototyping	Agile	Waterfall	Incremental
Product Requirements:				
• Functions / Components				♦
• Complexity	♦			♦
• Volatility		♦		
• Uncertainty	♦	♦		
• Criticality			♦	
• Maintainability			♦	
• Usability	♦			
Team / Personnel:				
• Technical Competency		♦		
• Management Competency			♦	♦
• Team Competency				
• Application Domain Knowledge	♦			
• Retention / Turnover				
Development Environment:				
• Process Infrastructure		◊	♦	♦
• Capability of Tools				
• Platform Capability / Volatility	♦			
• Reuse Leverage				
Pressure / Involvement:				
• Customer Oversight			♦	♦
• User Involvement		♦		
• Schedule Pressure / Market Urgency		♦		

Table 2

A Closer Look of Estimating and Influence Factors

Suppose we had a few highly expert estimators who are trained and experienced in all of the estimating techniques identified herein. These estimating gurus are also well-versed in facilitating estimates with developers. Suppose also that their company supports a "Super Estimating Model" (see Figure 8) with commercial Function-Points and COCOMO tools as well as industry tables and a corporate repository of historical analogies and proxies. Estimating has been facilitated by having the proxies and their nominal counts categorized and regressions of these proxies automated. Various rules-of-thumb have been derived and tuned from previous projects.

On top of this, the estimating gurus are quite familiar with the range of application domains and technologies within which the company specializes, namely, healthcare and medical informatics. These estimating specialists also recognize that this project, like most, will encounter some technology risks, application domain unknowns, and uncertain requirements. Also most of their projects need to

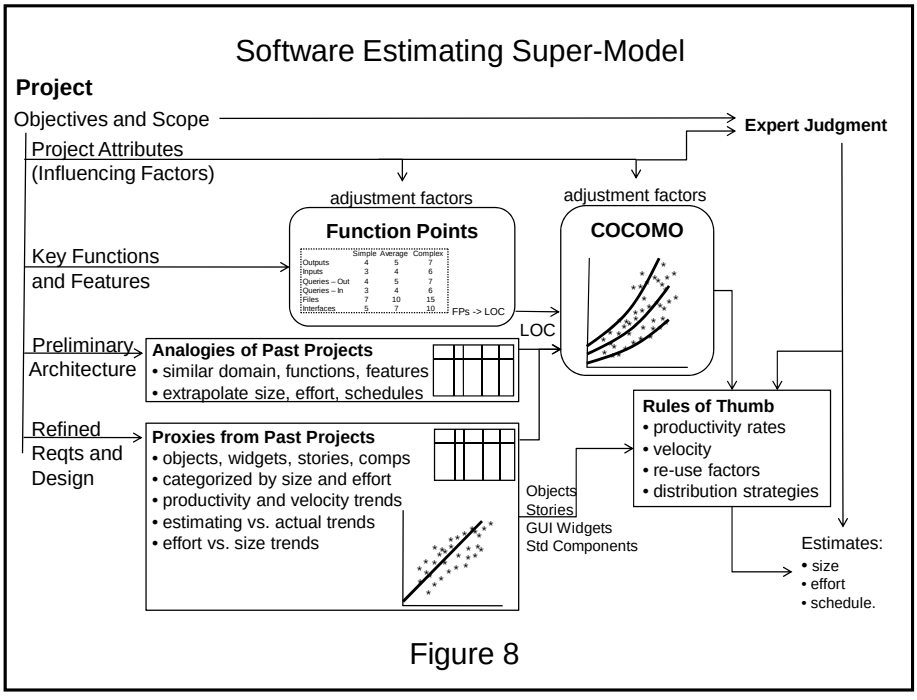

Figure 8

address some critical security, reliability and availability requirements. And they have a good supply of skilled developers at their disposal, some of whom have experience building security and reliability systems, and others who have good experience with Agile development.

Now suppose they are responsible for leading the software estimating effort on a new project that is similar to various elements of prior projects covered by the company's historical estimating database. Their estimating process would be characterized as follows:

A. They would apply the following estimating steps iteratively throughout the lifecycle of a given project from early feasibility, through requirements and design stages of development, to core software development stages
B. As they progress through the project from feasibility through to final delivery, they would:
    1. Validate project objectives and scope;
    2. Identify project influencing factors / attributes (hopefully most of them, but perhaps not all);
    3. Assign appropriate values for each project influencing factor;
    4. Obtain key functions and features that comprise the initial requirements baseline;
    5. Obtain preliminary architectural design plus additional functions and features as they become better understood;
    6. Obtain refined requirements and design details as they become better understood;
    7. Repeat step 6 as required until project completion;
    8. If major changes in scope are encountered and (hopefully) agreed to, many of the steps above will need to be re-visited to assess impacts and produce new estimates.

Software Estimating by Stage of Development

Given the above scenario, the estimating specialist would develop the software estimates iteratively along the timeline as indicated in Figure 9. Judgment with rules-of-thumb is useful during early project stages. Function-Points estimating is useful during initial project planning and requirements engineering; Analogy-based estimating is most useful in support of initial planning, requirements and preliminary design stages. Proxy-based techniques should be used to develop more accurate estimates in later project stages from detailed design through software development (construction) stages. Here are some additional pertinent observations:

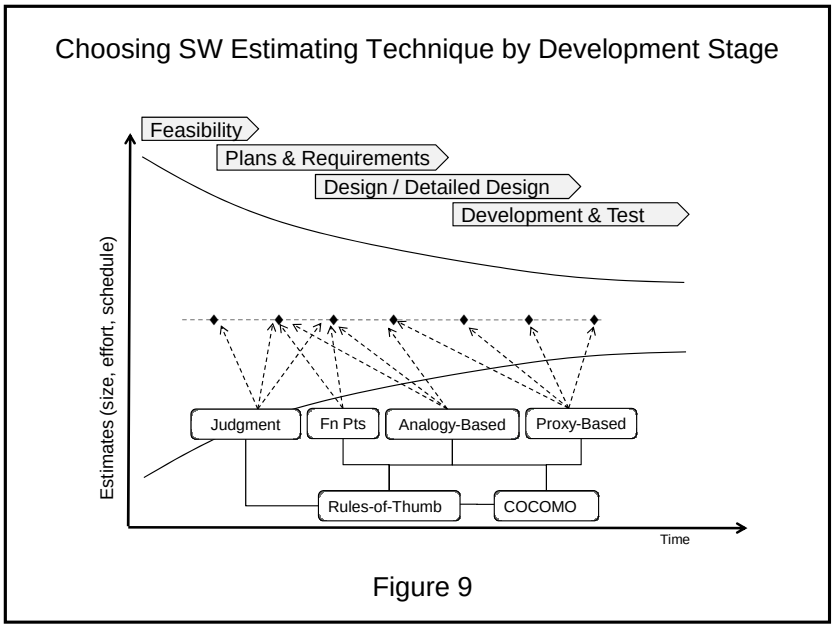

Figure 9

- ❑ Judgment and Rules-of-Thumb are commonly used together; but Judgment accuracy fades as the project moves forward; Rules-of-Thumb also fade with time;
- ❑ Function-Points needs special training;
- ❑ Analogy-based estimating is easier to use and support than Proxy-based estimating;

- ❑ COCOMO is useful throughout lifecycle; but COCOMO may over-estimate small projects;
- ❑ Analogy-Based and Proxy-Based require significant investments in maintaining a historical database;
- ❑ Rules-of-Thumb and Proxies need to be constantly tuned to realize good results.

Key Project Factors Influence Choice of Estimating Technique

Important Observations:

1. Table 3 identifies key project factors for comparing estimating techniques; unchecked factors have second-order influences and have been set aside (are unchecked);

2. Volatility, uncertainty, management competency, process infrastructure, customer oversight, and user involvement are not explicitly addressed by any of the estimating techniques. However, volatility, uncertainty and user involvement may influence the selected software process (Agile versus Waterfall for example). The estimator should consider these factors prior to completing an estimate.

Key Project Factors Influence Choice of Software Estimating Technique

	Judgment	Function Pts	Analogy/Proxy	COCOMO
Product Requirements:				
• Functions / Components	√	√	√	
• Complexity	√	√	√	√
• Volatility				
• Uncertainty				
• Criticality				√
• Maintainability	√	√		√
• Usability		√		
Team / Personnel:				
• Technical Competency	√			√
• Management Competency				
• Team Competency				√
• Application Domain Knowledge	√		√	√
• Retention / Turnover				√
Development Environment:				
• Process Infrastructure				
• Capability of Tools	√		√	√
• Platform Capability / Volatility		√	√	√
• Reuse Leverage	√	√	√	√
Pressure / Involvement:				
• Customer Oversight				
• User Involvement				
• Schedule Pressure / Market Urgency			√	√

Table 3

With reference to Table 3, estimating technique selections are driven as described below:

Judgment:

- ❑ Judgment can be used to yield effort estimates and/or size (LOC, stories, widgets, objects, …);
- ❑ Use Rules-of-Thumb to derive effort and schedule from size estimates;
- ❑ First consider number of functions and features, complexity, and technical competencies;
- ❑ Then consider applying the remaining maintainability, application domain, tools and reuse factors.

Function-Points:

- ❑ Works best when high level functions and their relative complexities are fairly well understood;
- ❑ Consider only if you have requisite training and practice so that you apply this technique correctly;
- ❑ Function-Points adjustment factors cover maintainability, usability, platform capability and reuse;
- ❑ If used together with COCOMO, most project influencing factors will be covered.

Analogy-Based and Proxy-Based:

- ❑ First consider prior projects with similar application domains, functionalities and complexities;
- ❑ Second consider past projects that used similar tools, were implemented on similar platforms, and were produced under similar schedule pressure;
- ❑ Proxy-Based estimating will need to choose the appropriate type of proxy and should incorporate reuse into estimates by applying rules-of-thumb;
- ❑ If COCOMO is used to derive effort and schedule estimates, many of the other influencing factors will be accounted for in the estimates.

COCOMO:

- COCOMO is capable of addressing a range of projects, including large scale and complex projects often with significant criticality and maintainability requirements;
- Use COCOMO to derive effort and schedule from LOC; or when other size measures like function-points can be converted into LOC;
- COCOMO, when carefully applied, should yield better estimates than Rules-of-Thumb estimates because this technique encourages the estimator to consider most of the project factors while Rules-of-Thumb do not cover such influencing factors;
- The estimator should consider adjusting the estimates for process infrastructure, customer oversight, and management competency which are not explicitly covered by COCOMO adjustment factors.

Recommendations / Guidance

The following top recommendations are first offered:

- When tackling a new project, the software practitioner should start by reviewing and annotating the project influencing factors presented in Figure 7. By creating and annotating a checklist of these factors, analysis will be focused on selecting the most appropriate software processes and estimating techniques thereby facilitating and rationalizing subsequent budgeting and planning decisions;
- Consider investing in a COCOMO tool – a relatively inexpensive but useful initiative. Although, COCOMO estimates can be rough-cut – especially when influencing factors are not well-understood, a COCOMO estimate using nominal values for these factors will provide a solid initial basis for new project planning and launch.
- Companies should also seriously consider establishing and maintaining an estimating database of historical software project data to support analogy-based, proxy-based and rules-of-thumb estimating. This would leverage several of the practical recommendations that follow.

The choice of software development process should be made as early as possible in a project. This can be done once the project's flexibility posture is decided and once key project drivers are understood.

With reference to Table 4, Prototyping is mainly driven by technical and requirements uncertainty and lack of knowledge; Agile processes mainly by requirements flexibility and schedule pressure; Waterfall development mainly by criticality and maintainability factors; and Incremental development by total project scale and complexity.

Expert Judgment combined with Rules-of-Thumb is a particularly useful and economic approach when little more than project objectives and scope are known, that is, during feasibility and early requirements formulation stages. However, the quality of estimates is highly dependent upon the domain experience and estimating skills of the software practitioner. Such estimates should be used mainly to assess project feasibility and rationalize budgets and schedules for new projects. They may be safely used to control small projects or those with constrained or fixed schedules or budgets and a flexible requirements posture (i.e. design-to-cost and design-to-schedule projects). However, when it comes to large, critical, complex and/or constrained projects, Expert Judgment with Rules-of-Thumb (only) estimating will not be good-enough, in most cases, to monitor and control budgets and schedules for fixed or highly constrained projects.

Because prototyping aims to explore technology, requirements and other risks and unknowns, it is not feasible to apply a systematic estimating technique to such activities. Prototyping, therefore, will require careful Expert Judgment to assess each project risk that could impact negatively on project feasibility, budget and schedule. Rules-of-Thumb will not apply. Such prototyping tasks will need to be broken-

down, time-boxed, resourced, and addressed early enough in the project schedule to avoid budget and critical path (schedule) problems. Clearly, adequate risk identification, risk evaluation and risk mitigation actions are essential project aspects to coordinate with estimating efforts.

Function-Points estimating is particularly useful during the requirements phase. However, the technique requires intensive training and is not widely practiced in non-MIS (management information system) domains.

Process and Estimating Choices are Driven by Key Project Influencing Factors

	SW Processes					Estimating Techniques			
	Prototyping	Agile	Waterfall	Incremental		Judgment	Function Pts	Analogy/Proxy	COCOMO
Product Requirements:									
• Functions / Components				♦		√	√	√	
• Complexity	♦			♦		√	√	√	√
• Volatility		♦							
• Uncertainty	♦	♦							
• Criticality			♦						√
• Maintainability			♦			√	√		√
• Usability	♦						√		
Team / Personnel:									
• Technical Competency		♦				√			√
• Management Competency			♦	♦					
• Team Competency									√
• Application Domain Knowledge	♦					√		√	√
• Retention / Turnover									√
Development Environment:									
• Process Infrastructure		◊	♦	♦					
• Capability of Tools						√		√	√
• Platform Capability / Volatility	♦						√	√	√
• Reuse Leverage						√	√	√	√
Pressure / Involvement:									
• Customer Oversight			♦	♦					
• User Involvement		♦							
• Schedule Pressure / Market Urgency		♦						√	√

Table 4

Analogy-Based estimating and Proxy-Based estimating require the company to create, maintain and tune a historical database of analogies and proxies. One recommended strategy is to augment the company's version control system with tools to categorize and count proxies. Mechanisms for collecting and associating development effort would also be required. Similarly, Rules-of-Thumb should be maintained in such a database and tuned as new project data is collected. This implies a corporate investment in requisite tools and support.

Analogy-Based estimating is applicable during early stages of a project while Proxy-Based techniques begin to be practical during the core construction phases of a project. Agile development will use story-points and track "velocity" to estimate the next iteration (or two) of development. Waterfall development, meanwhile, can exploit standard components as proxies during design stages, and GUI widgets and use-case-points during construction stages. For smaller and less critical projects, Rules-of-Thumb (like productivity factors and velocity) are likely good-enough to derive effort estimates from LOC, Function Point and story-point estimates.

COCOMO can be used successfully to produce effort and schedule estimates for both Agile and Waterfall projects given LOC size estimates have been provided as input, the correct COCOMO estimating equations are used, and suitable values for the adjustment (influencing) factors have been made. COCOMO is well-suited for medium to large sized projects with high complexity, criticality, and maintainability requirements using Waterfall or Incremental development.

Incremental development can benefit from all of these estimating techniques. For example, Analogy-Based estimates can be used to secure early budgets and other resources; a prototyping increment would use Expert Judgment; an Agile increment might use Proxy-Based story-points; and Waterfall increments could apply Analogy-Based estimating with COCOMO. A pre-requisite for such an approach would be to partition requirements, and possibly the preliminary design as early as possibly in the project.

Summary / Conclusion

The software practitioner, whether developing project plans, leading the estimating effort, or contributing bottom-up estimates, has a vital role to play in the preparation and ongoing control of the project. As professionals, they are accountable to senior management and customers for the software estimates that shape the project. They should therefore be intimately involved in the selection of software processes that directly affect the estimates.

The selection of software process and software estimating technique are driven by key project attributes, namely, project influencing factors. The relationships between software process and estimating technique are far from deterministic, however, they do appear to be coupled in practical ways that should inform the software practitioner and should be exploited in the conduct of their work.

To enhance software estimating effectiveness, the company should first decide whether to invest in building and maintaining a software estimating database which represents a significant investment in tools and effort. However, without such a repository in place, the company will not be able leverage Analogy-Based or Proxy-Based estimating techniques and the increased confidence in estimates they offer.

As an alternative, the company may consider acquiring Function-Points training and tools. An investment in COCOMO estimating tools, meanwhile, would be useful as an adjunct to Analogy-Based, Proxy-Based and Function-Points tools and infrastructure.

The fall-back, of course, is to rely on Expert Judgment and Rules-of-Thumb which yield good-enough early estimates to secure budgets, and to support design-to-budget or design-to-schedule projects. However, Expert Judgment and Rules-of-Thumb will not yield better estimates-to-complete for projects with relatively fixed requirements. On the other hand, story-point estimating may be a reasonable alternative for companies practicing Agile projects where it is only necessary to estimate the next iteration (or so) of development.

A given project's influencing factors will inform the software professional, especially with respect to project planning and estimating. The project influencing factors identified in this paper can be used as a checklist to help decide on the software process, choose an appropriate software estimating technique, and use project influencing factors to adjust software estimates. Such a checklist should be annotated and appended to the project's objectives and scope during the feasibility phase and updated as required throughout the project lifecycle.

A project's influencing factors should to be tracked and "tuned" from project to project. This implies conducting in-progress and post-project reviews comparing actual results to original estimates. Such reviews should assess the relative impacts of project influencing factors on the obtained results and fine-tune the criteria used to select values for each of these factors.

In summary, software practitioners and their companies are strongly encouraged to create a sound software process and estimating infrastructure. This paper has recommended, among other things, to

systematically apply project influencing factors, implement a historical database for analogy and proxy-based estimating, and employ a COCOMO estimating tool to anchor estimating activities.

Possible Further Study

Among the key influencing factors, Process Infrastructure, Customer Oversight, and Management Competency did not yield explicit intersections with software process selection, or software estimating techniques. However, COCOMO's equations for "embedded" projects may very well (partially) compensate for these omissions. It would appear that these attributes would be particularly relevant to large mission-critical projects and would therefore be used to necessarily inflate software estimates for such projects. These perspectives are left as an open area for further study. Nevertheless, these omitted factors should be systematically addressed by prudent software practitioners to adjust their estimates.

References

[1] Kal Toth, "Which is the Right Software Process for Your Problem?" The Cursor, Software Association of Oregon (SAO), April 2005

[2] Barry Boehm, "Software Engineering Economics", Prentice Hall, Englewood Cliff, N.J., 1981

[3] Steve McConnell, "Software Estimation: Demystifying the Black Art", Microsoft Press, 2006

[4] Mike Cohn, "Agile Estimating and Planning", Prentice Hall, 2005

[5] Dan Brook, Kal Toth, "Levels of Ceremony for Software Configuration Management", PNSQC, October 2007

[6] Futrell, Shafer, Shafer, "Quality Software Project Management", Prentice Hall, 2002