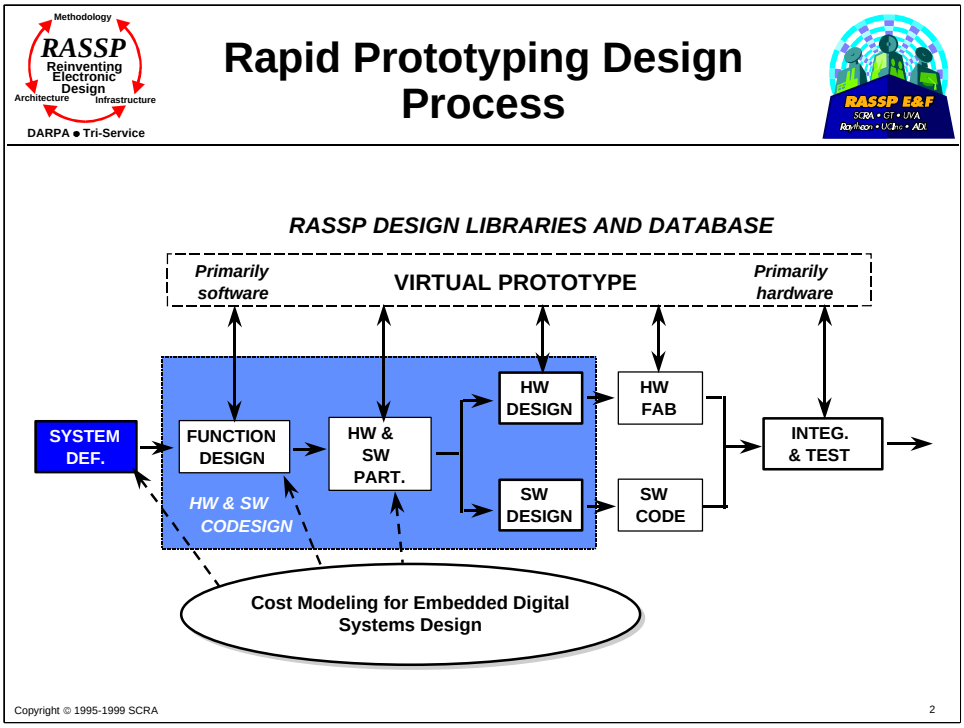
Cost Modeling for Embedded Digital Systems Design

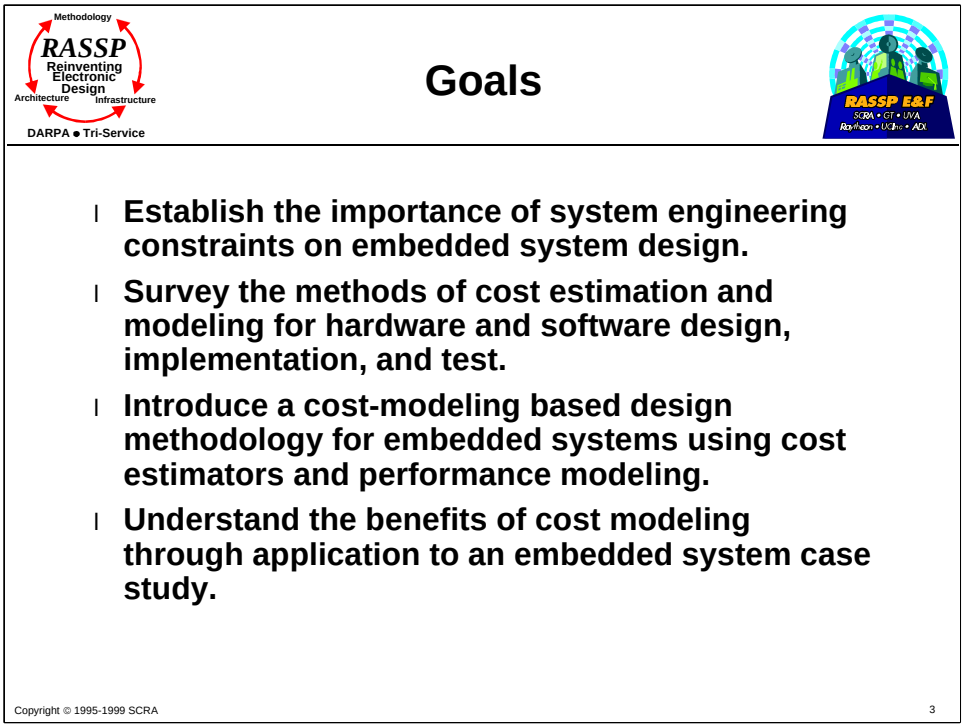
RASSP Education & Facilitation Program

Module 57

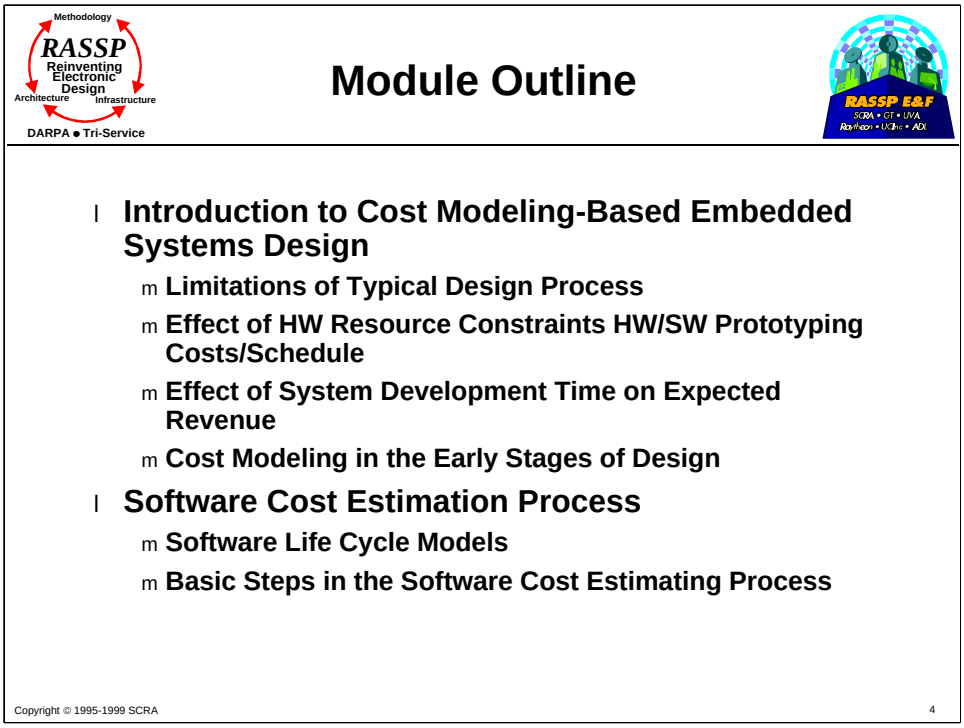
Version 3.00



The Rapid Prototyping Design Process is applicable to all modules in the E&F program. This slide indicates where in the process Cost Modeling for Embedded Digital Systems Design fits.



- Prior use of cost models in software engineering was used to estimate the cost of software development. It has seldom been used to drive the design methodology nor generate architectures for candidate implementations. This module describes how cost modeling can be used in the system synthesis process.



- Embedded systems are unique in the codesign requirements for hardware and software. This unique relationship has an interesting impact on the cost modeling process.

Module Outline (Cont.)

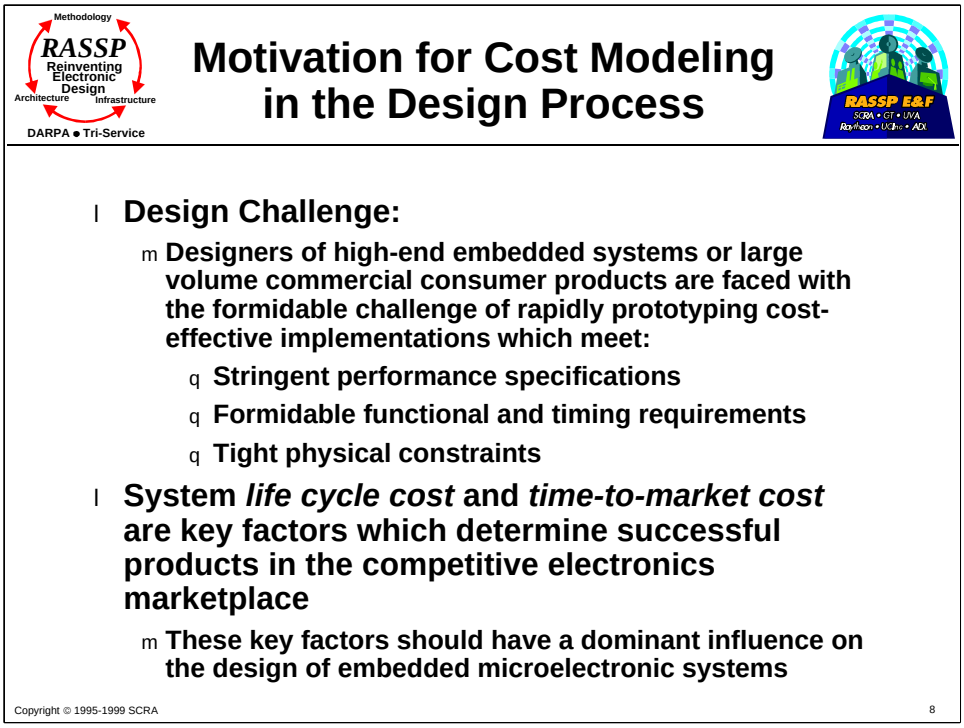
- **Parametric Software Cost Models**
  - **COCOMO**
  - **REVIC**
  - **COCOMO 2.0**
- **Parametric Hardware Cost Models**
  - **Full Custom**
  - **Gate Array**
  - **FPGA**

Module Outline (Cont.)

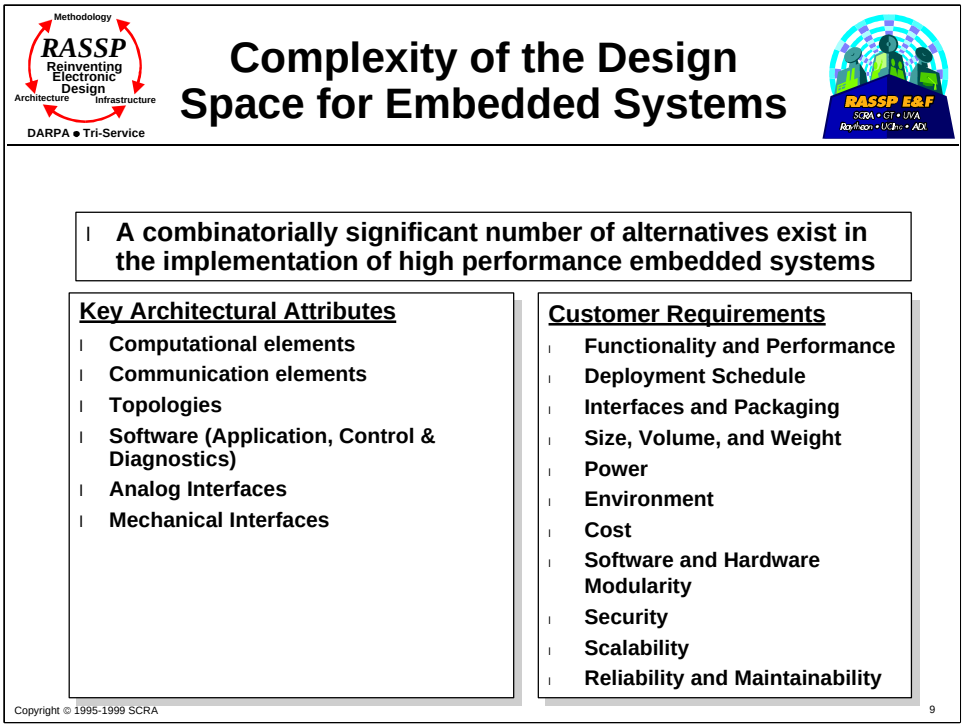
- **Applications of Cost Modeling in the RASSP Design Process**
  - **Classifications of System-Level Design & Test Methodologies**
  - **Example Automated Cost-Driven Architecture Selection/Sizing Model**
  - **Case Study: Synthetic Aperture Radar (SAR) Image Processor**
  - **Results: Detailed Cost Analysis for SAR Benchmark**
- **Summary and Major Issues**

Module Outline

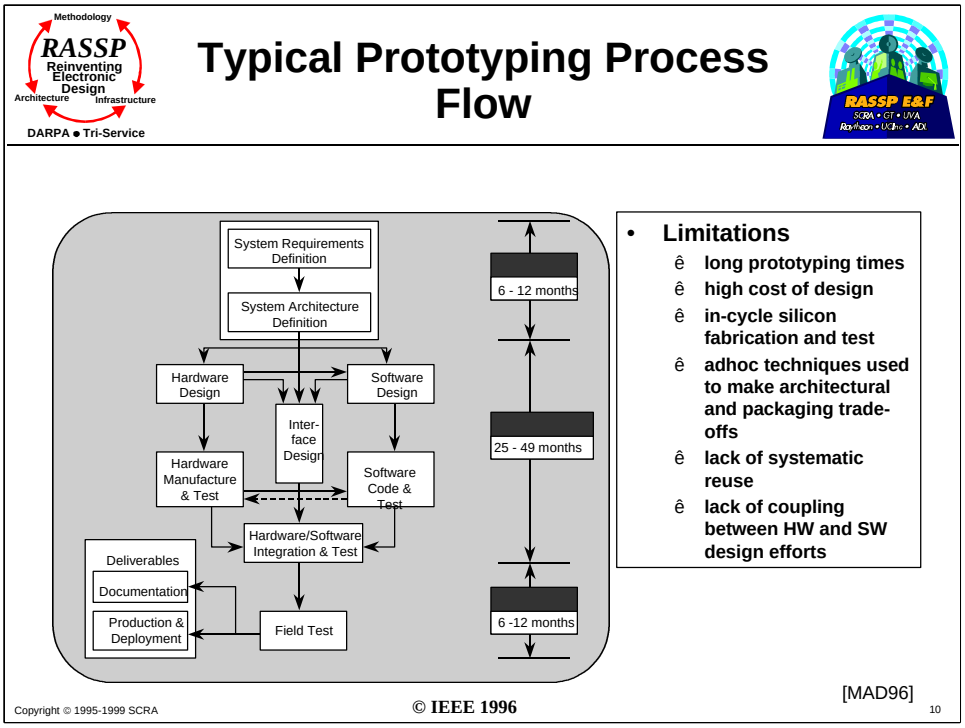
- **Introduction to Cost Modeling-Based Embedded Systems Design**
- **Software Cost Estimation Process**
- **Parametric Software Cost Models**
- **Parametric Hardware Cost Models**
- **Applications of Cost Modeling to the RASSP Design Process**
- **Summary and Major Issues**



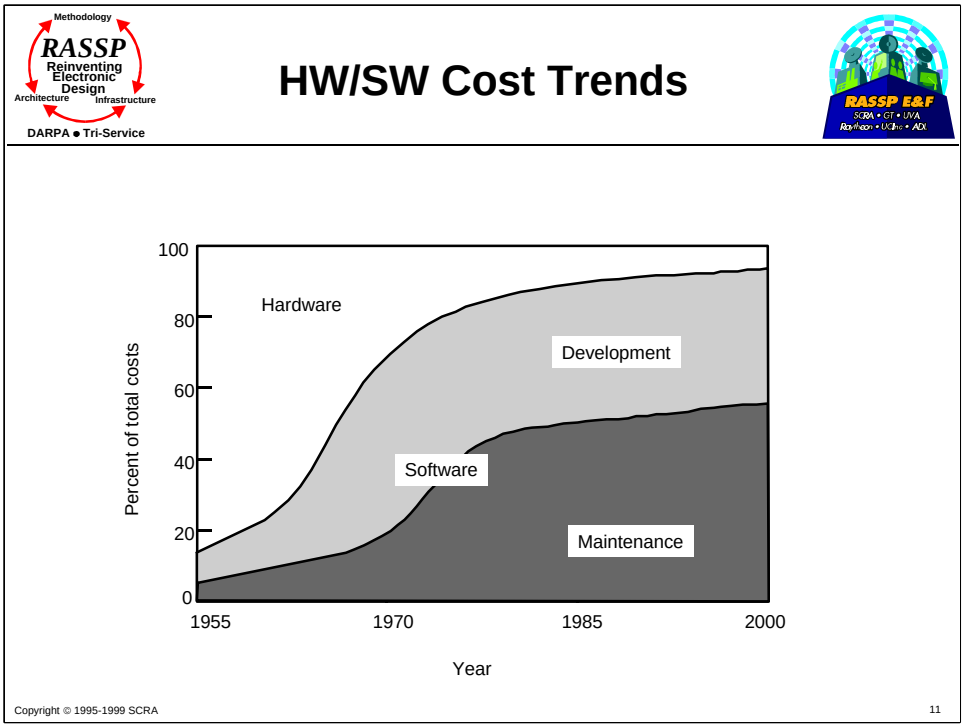
- Embedded system design places considerable importance on timeliness of the product, and its low cost. Due to the interacting nature of the hardware design and the software design phases in embedded system design, cost modeling assumes a greater role, and often drives the design process. For example, a low cost strategy may require the use of off-the-shelf parts for a system with a limited production run and a short time to market. On the other hand, for large production volumes with a sufficient design time available, cost modeling would, in all likelihood, prescribe a custom solution for the design of the product.



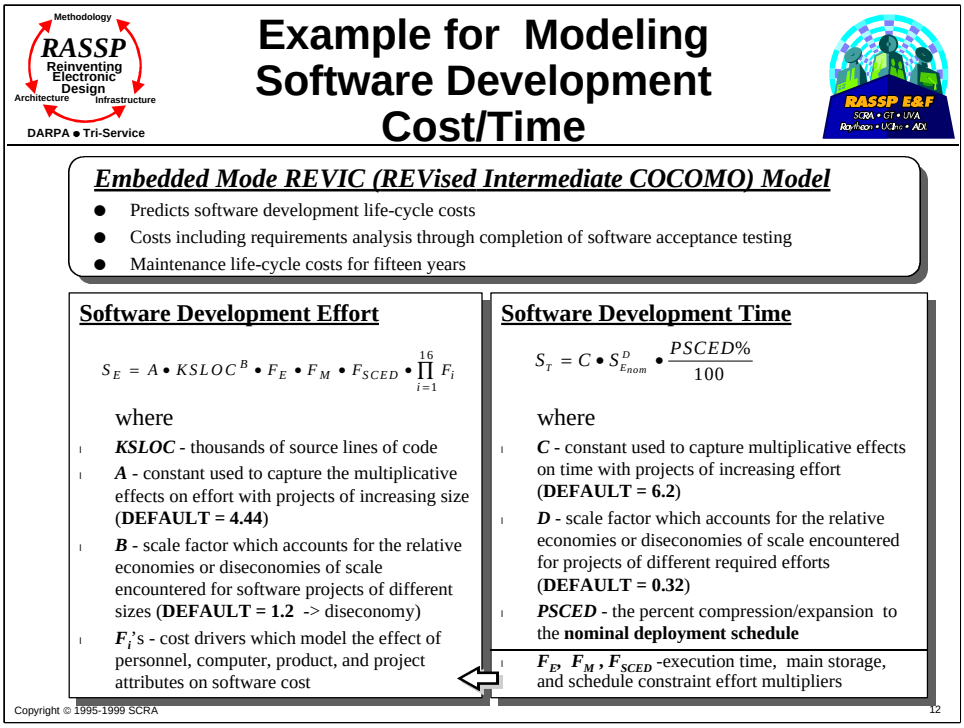
- Example embedded systems range from automotive anti-lock brake systems on the low-end to high performance radar and video systems on the high-end.
- System complexity has quickly grown from millions of operations per second to billions of operations per second.
- Systems which could once be implemented in hardwired or uniprocessor architectures, must now consist of arrays of programmable multiprocessors to meet performance requirements.
- The design task is further complicated by the requirement that these systems meet stringent form factor constraints.



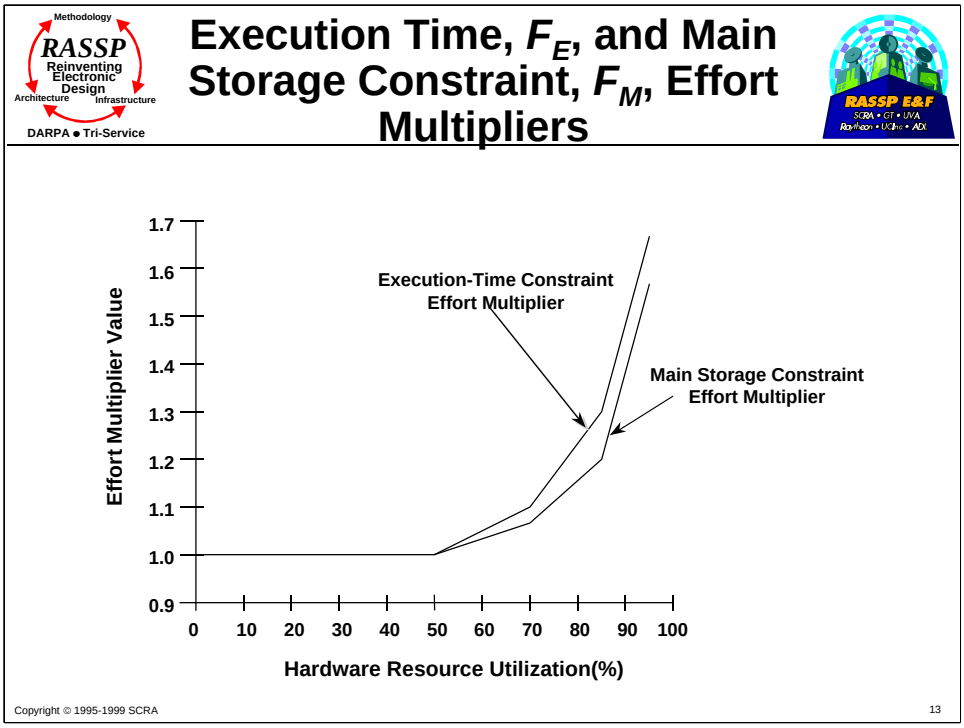
- The above figure illustrates a typical design methodology for high performance embedded systems.
- Approach is plagued with long prototyping times, high cost of design, and limited architectural exploration.
- Written requirements are used to specify system functionality and constraints which, in turn, foster ambiguity in their interpretation.
- Hardware subsystems and application software are not integrated until late in the design process.
    - Significant design flaws go undetected until the integration step.
    - Design errors are very costly because hardware/software integration does not occur until after hardware has been fabricated.
- Development costs range from $20 million to $100 million.
- Prototyping times range from 37 to 73 months.



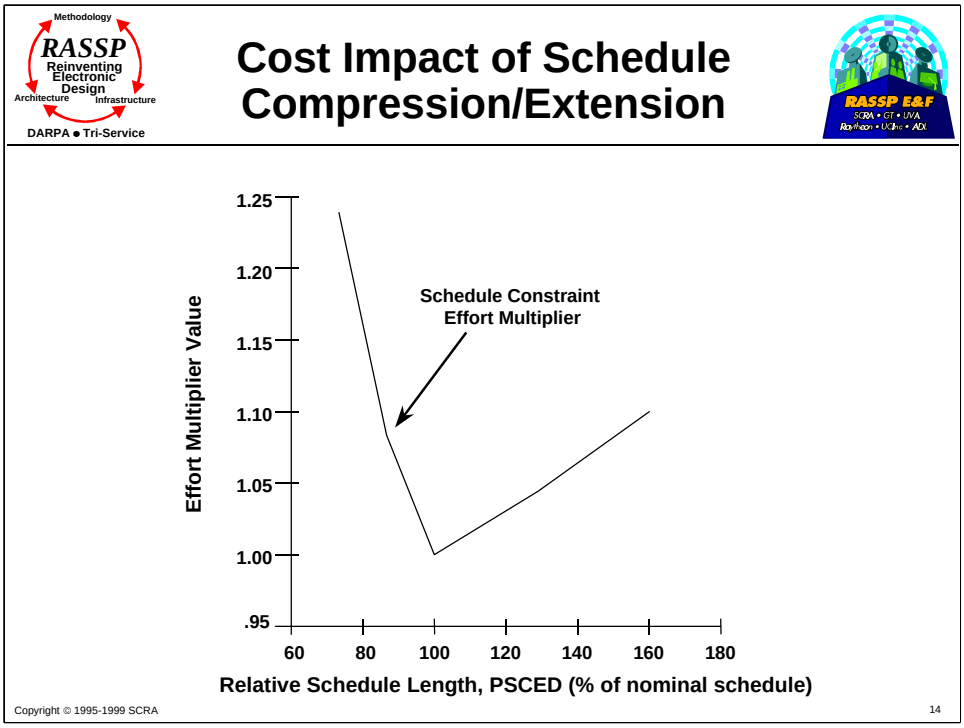
- Over the past 40 years, the percentage of total system costs attributable to software has increased drastically as the use of COTS hardware is becoming more common. Recent systems are to the right where a system has less than 20% hardware cost, as opposed to over 80% cost for the software component.
- This software growth creates a tremendous challenge for the software engineering profession. The challenge is twofold:
  - To significantly increase software development productivity
  - To increase the efficiency of software maintenance



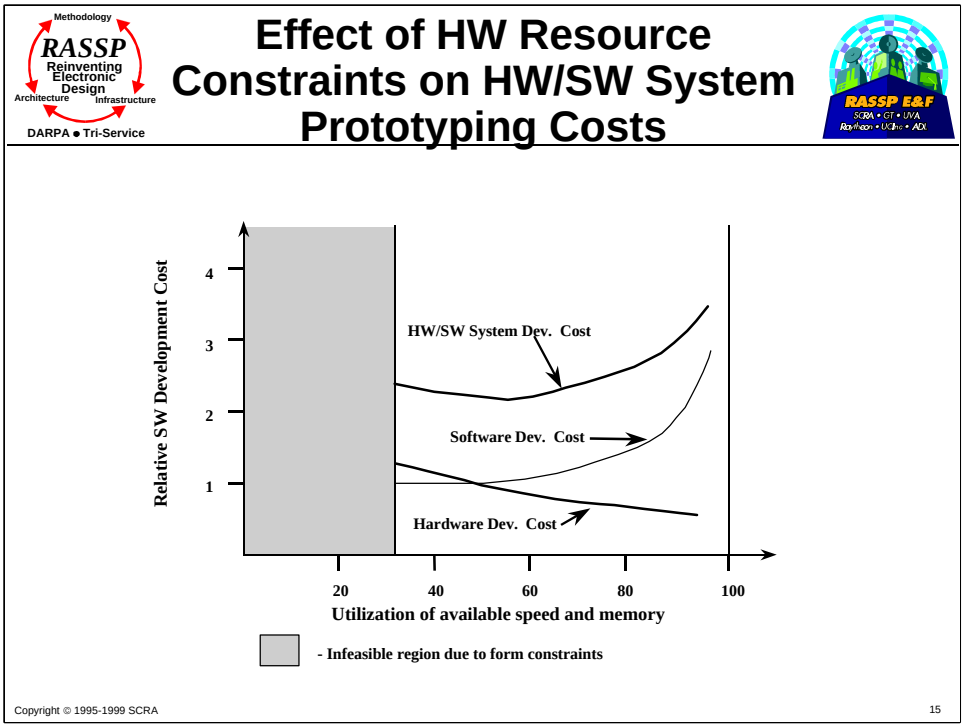
- For more information, see [AF95].
- It is noted that the effort (in man months) is proportional to a power of the number of lines of source code. The exponent depends on the size of the project. There are more than a dozen multiplier coefficients, F, that modulate the values predicted by the source code. These factors or coeffcients reflect the particular characteristics of the target application (real-time or non real-time), or the platform (embedded or mainframe), the software team (expert, novice), and the methodology used (object oriented, etc).
- The software development time depends on the software effort, and the amount of schedule compression required. Given a certain schedule for a certain amount of effort, if we are required to compress that schedule, we may have to assign a different set of resources (people and tools) to the task. In some cases, adding resources can actually delay the task.



- This graph shows the relation between the execution time and main storage constraint effort multipliers and the CPU and memory utilizations, respectively.
- The effort multipliers significantly increase as the resource utilization rises above 50%.
  - The impact is extreme as the utilization approaches 95%.
  - This is because of the extreme difficulty in designing code when the margins for error are very small.
- The effort multipliers scale the software development effort.
- For more information, see [BOEHM81].



- When faced with the prospect of long development schedules which will cause a product to be delivered to market late, many managers attempt to compress the schedule by throwing more people (e.g. person-hours) at the problem.
- This graph shows that schedule compression has the adverse effect of greatly increasing the schedule constraint effort multiplier value, thereby increasing the overall software development cost.
- This increase in cost is probably due to the larger labor force, which in turn increases communication problems, thereby adding errors and inefficiencies within the team.
- For more information, see [BOEHM81].



- Traditional system-level design and test methodologies attempt to minimize hardware costs for the purpose of minimizing overall system costs.
- Accomplished by maximizing hardware resource utilization, which leads tight execution time and memory budgets
- However, the above graph illustrates an often overlooked software prototyping principle [MAD95].
  - Various parametric studies based on historical project data show that software is difficult to design and test if 'slack' margins for HW CPU and memory resources are overly restrictive.
  - Software developers must interact directly with the operating system and/or hardware in order to optimize code to meet system requirements
  - Integration and test phase is particularly increased because resource constraints usually are not pushed until all software pieces come together
  - In systems in which most hardware is simply commercial-off-the-shelf (COTS) parts, the time and cost of software prototyping and design can dominate the schedule and budget.
  - If physical constraints permit, the hardware platform can be relaxed to achieve significant reductions in overall development cost.

RASSP Methodology — Reinventing Electronic Design — Architecture — Infrastructure — DARPA • Tri-Service

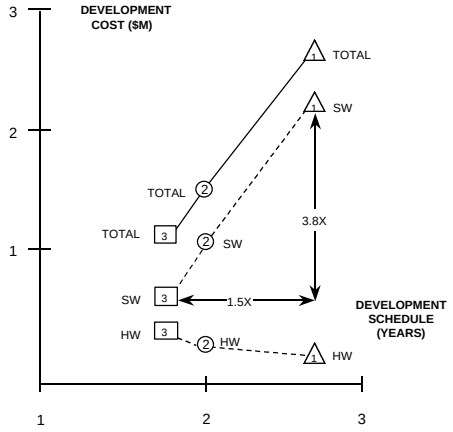
Design Trade-off Example:
SAR Processor with COTS Multiprocessor Cards

RASSP E&F — SCRA • GT • UVA — Raytheon • UCSC • ADI

① MINIMUM HARDWARE COST
- LOWEST PRODUCTION COST
- MINIMUM SIZE, WEIGHT, & POWER
- NOMINAL MEMORY & COMPUTATION MARGINS

② REDUCED DEVELOPMENT COST & TIME
- SAME NUMBER OF CARDS AND LIFE CYCLE COST AS ①
- RAM ADDED TO IMPROVE MEMORY MARGIN, ALLOW USE OF ADVANCED SOFTWARE DEVELOPMENT TOOLS AND METHODOLOGY

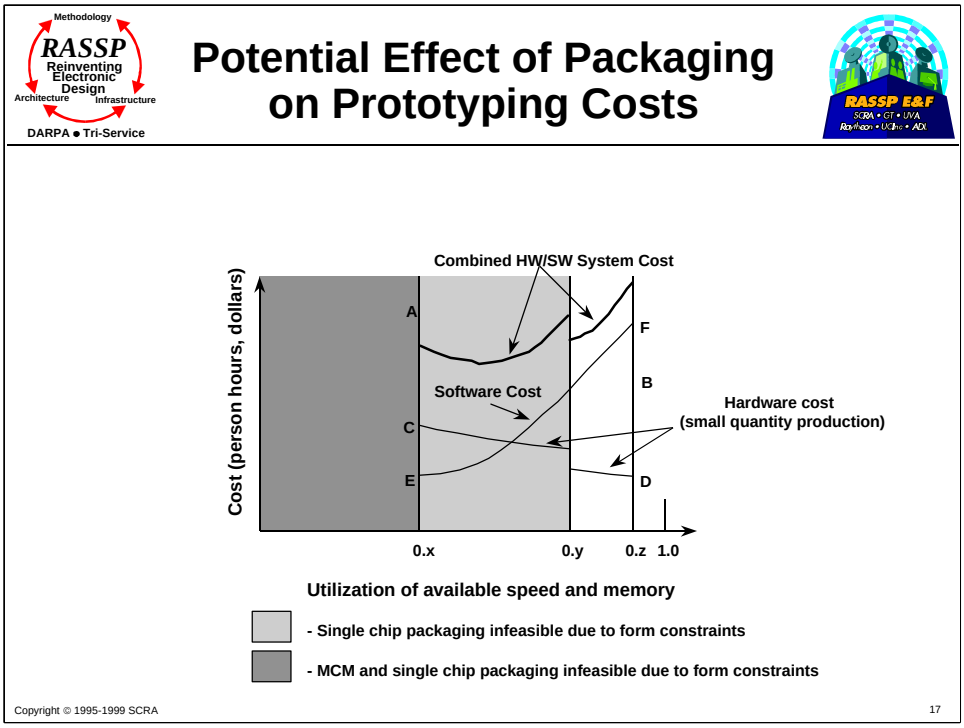
[3] MINIMUM DEVELOPMENT COST & TIME
- PROCESSORS ADDED TO ②
- IMPROVED COMPUTATION MARGIN FURTHER EASES SOFTWARE DEVELOPMENT



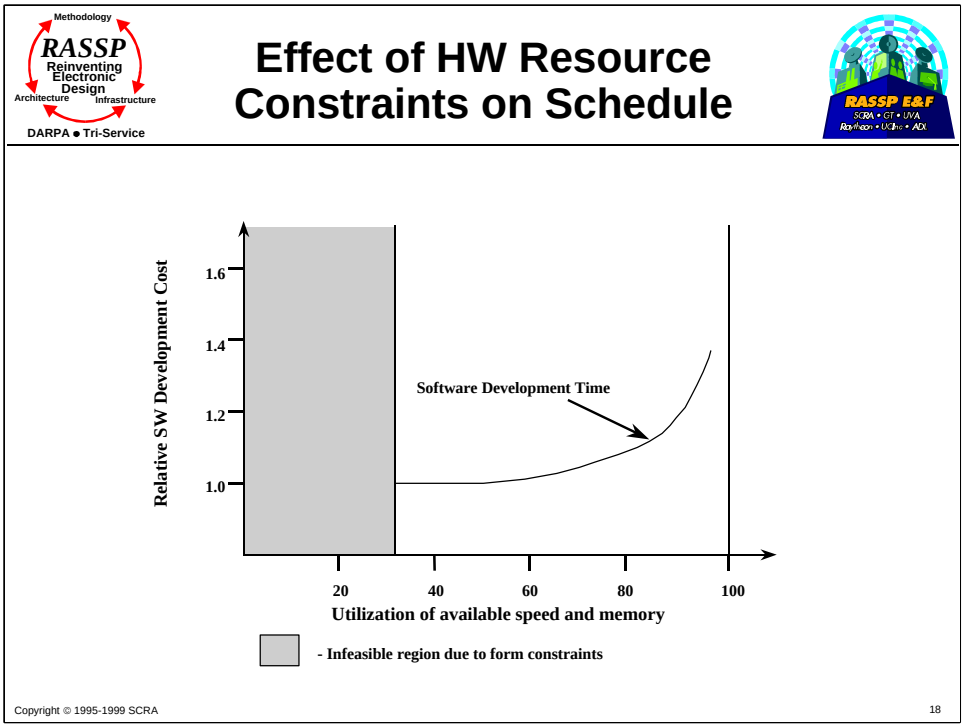
[AND95]

16

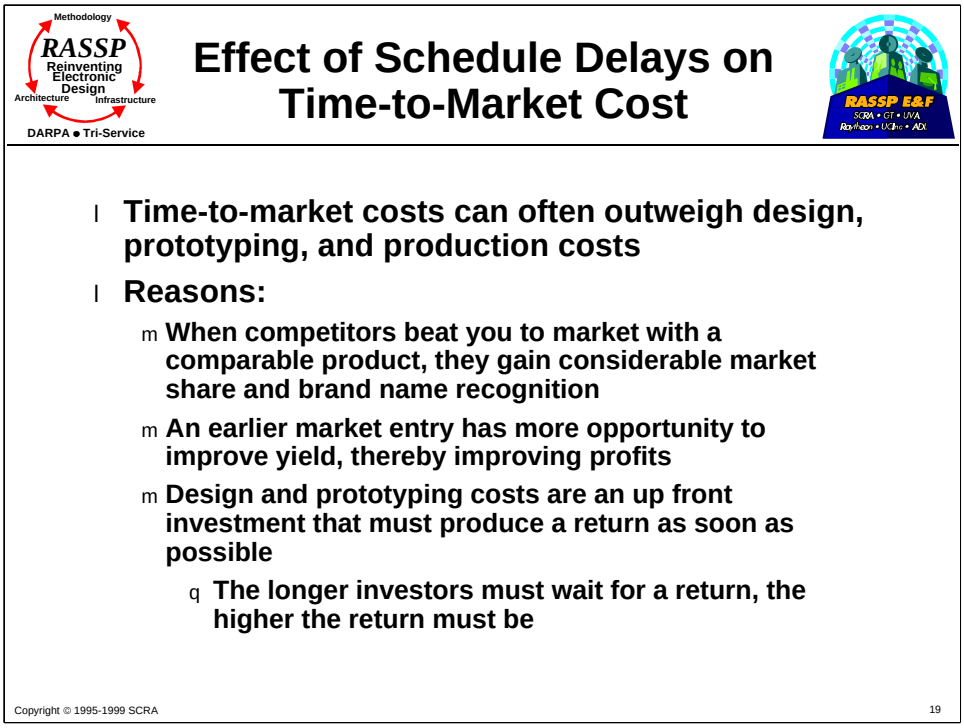
- In all three cases, use of a commercial off-the-shelf multiprocessor card solution is assumed, but details of the processor and memory margins differ.
- If the designer focuses entirely on minimizing hardware cost, the software development cost is nearly four times that of a design which seeks to minimize the overall development cost and time.
- The curves compare development cost and time for a synthetic aperture radar processor under three different assumptions regarding computation and storage requirements.
- The minimum hardware cost implementation uses only six MCV6-4x4m cards with a 88% execution time utilization and 86% memory utilization.
- Resulting hardware component cost is $100,000 plus the cost of the six cards - $281,000. Software cost development cost and time are $2,360,000 an 32 months, respectively. (total cost - $2,640,000)
- The reduced development cost/time implementation uses six MCV6-4x8m cards, thereby decreasing memory utilization to 43% allowing for the use of advanced software development tools and methodology.
- Software development cost and time decrease to $1,030,000 and 24 months; while hardware cost slightly increases to $315,000. (total cost - $1,350,000)
- The minimum development cost/time implementation uses eleven MCV6-4x4m cards with less than 50% memory and processor utilization.
- This further reduces software development cost and time to $620,00 and 21 months; while hardware increases to $432,000. (total cost - $1,050,000)



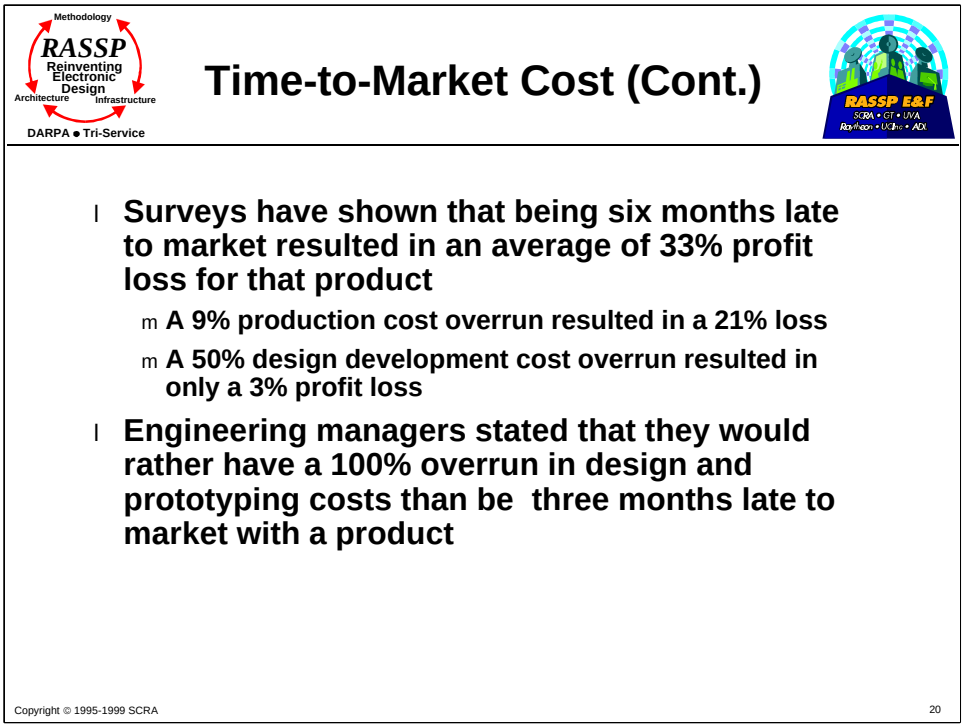
- Multi-chip module technology allows for increased packaging density over single-chip packaging
- This increased packaging density can allow for more slack to be added to the hardware architecture without violating system-level form factor constraints
- This added slack margin can possibly lead to significant software cost reductions
- However, the reduction in software cost is traded off against the increase in production costs due to MCM manufacturing.



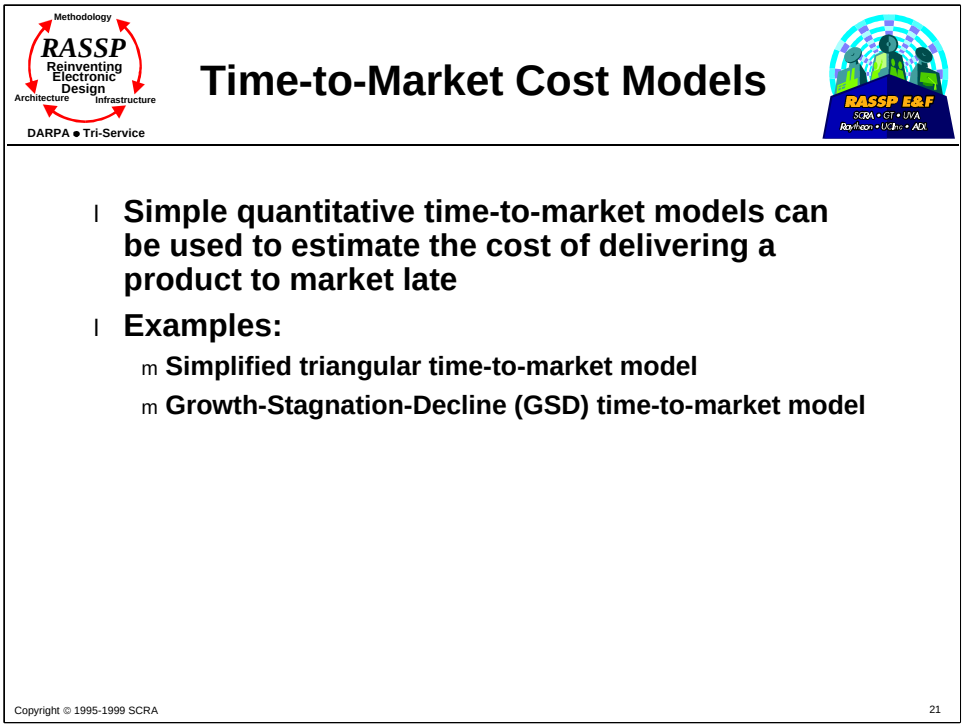
- This graph shows that hardware constrained architectures can also significantly increase system development time, especially when COTS hardware components are being used.
- If physical constraints permit, the hardware platform can be relaxed to achieve significant reductions in overall development cost .
- This increased software development time can be attributed to the same factors which lead to an increase in software development cost.



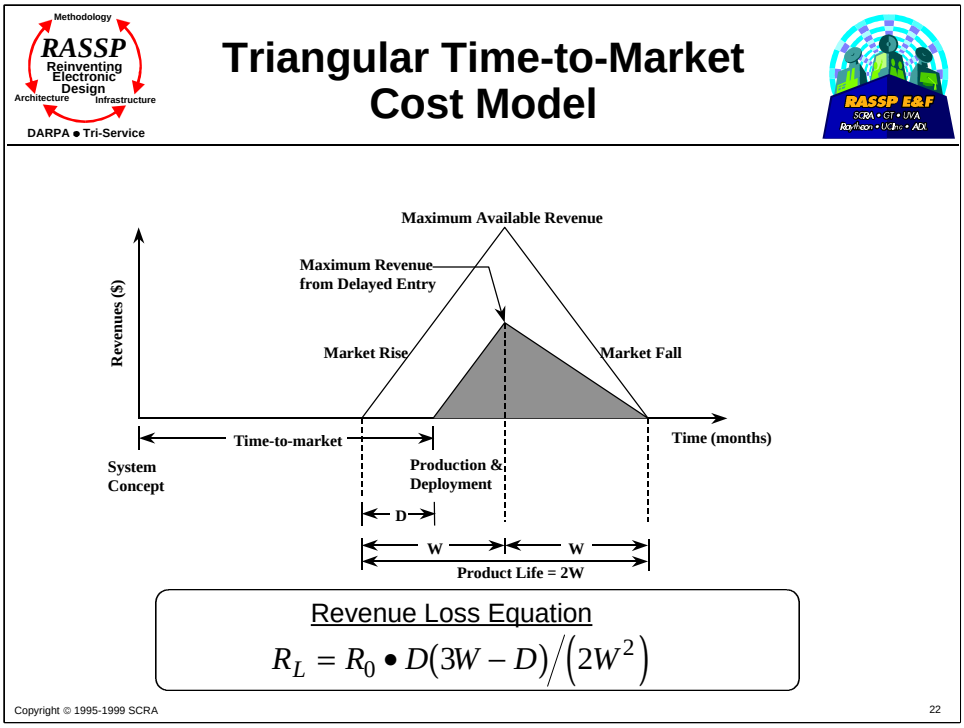
- The time to market is a primary driver in the commercial industry. A number of models have attempted to predict the impact of late delivery on the profitability of the product. We do not endorse any particular model, but show that any model can be included within the methodology. The quality of results would depend on the validity of the model.



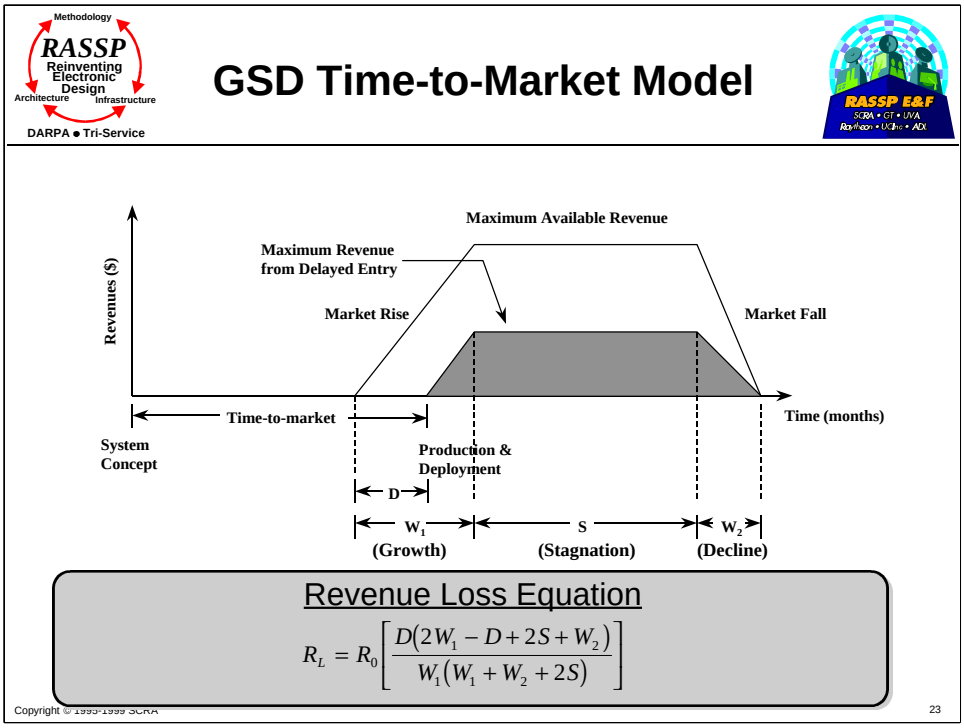
- For more information, see [DOANE93].



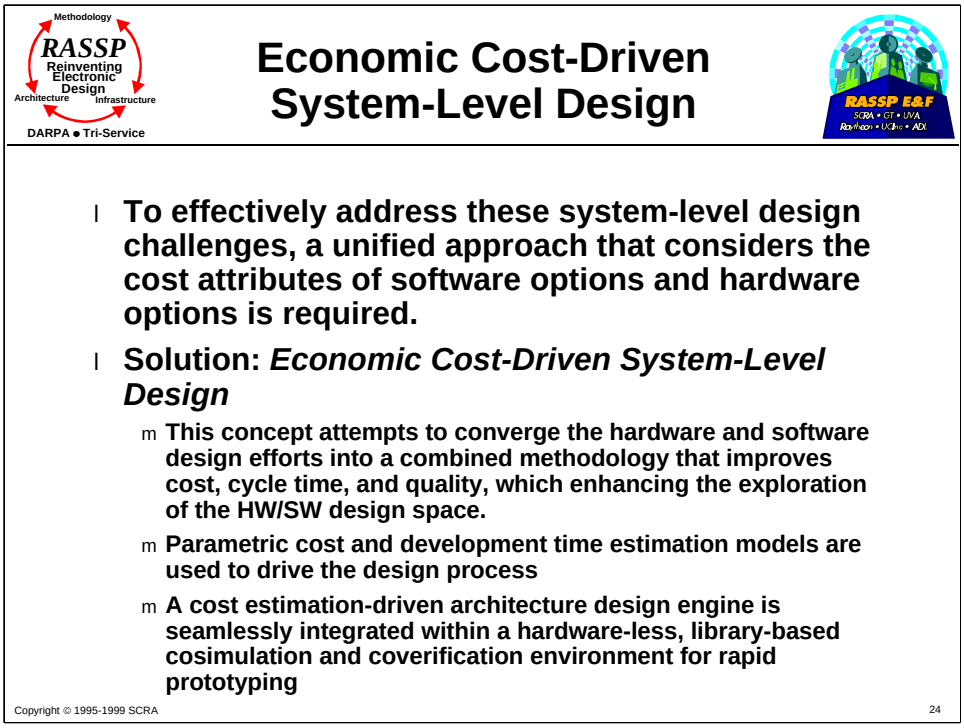
- Note that different models apply to commercial and Department of Defense applications. While commercial models stress the importance of time to market, defense application may stress lifetime costs of the system.
- We do not propose that any specific model is better than others, but that using a cost model is advantageous if it is accurate.



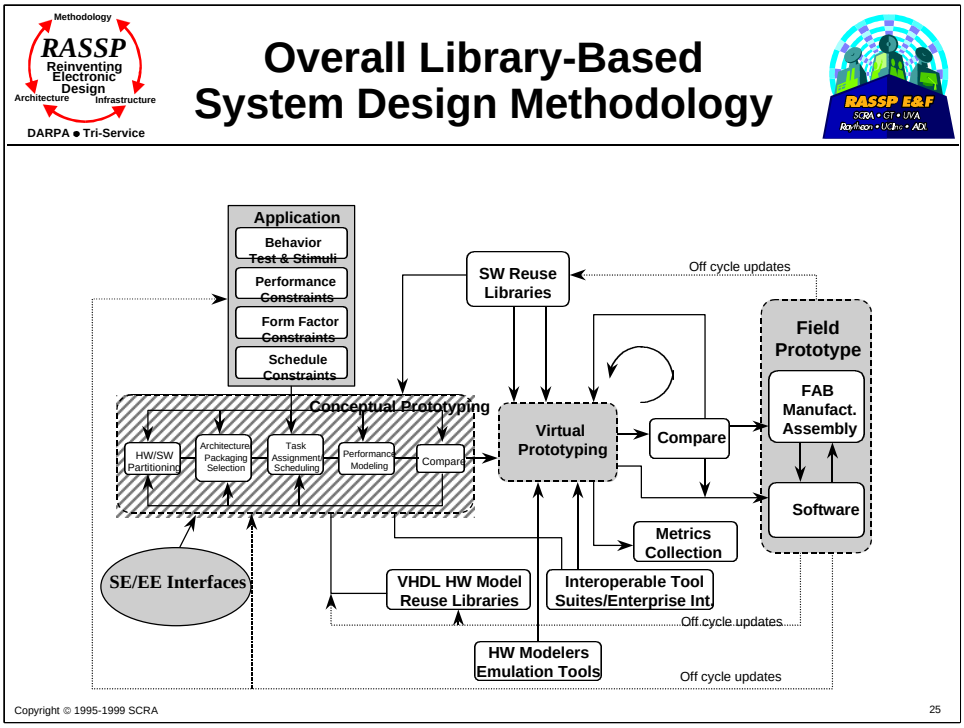
- The above graph illustrates the effect of delivering a product to market *D* months late.
- The non-shaded region of the demand window (triangle) signifies the lost of revenue, R_L, due to late entry in the marketplace.
- In order to maximize revenues, the product must be on the market by the start of the demand window.
- If the product life cycle (length is demand window) is short, being late to market can spell disaster.
- The revenue loss equation:
  - R_0 refers to the expected product revenue if it were on time
  - *D* is the delay (months) in delivering a product to market
  - *2W* is the length of the product life cycle(months)
- For more information, see [LIU95].



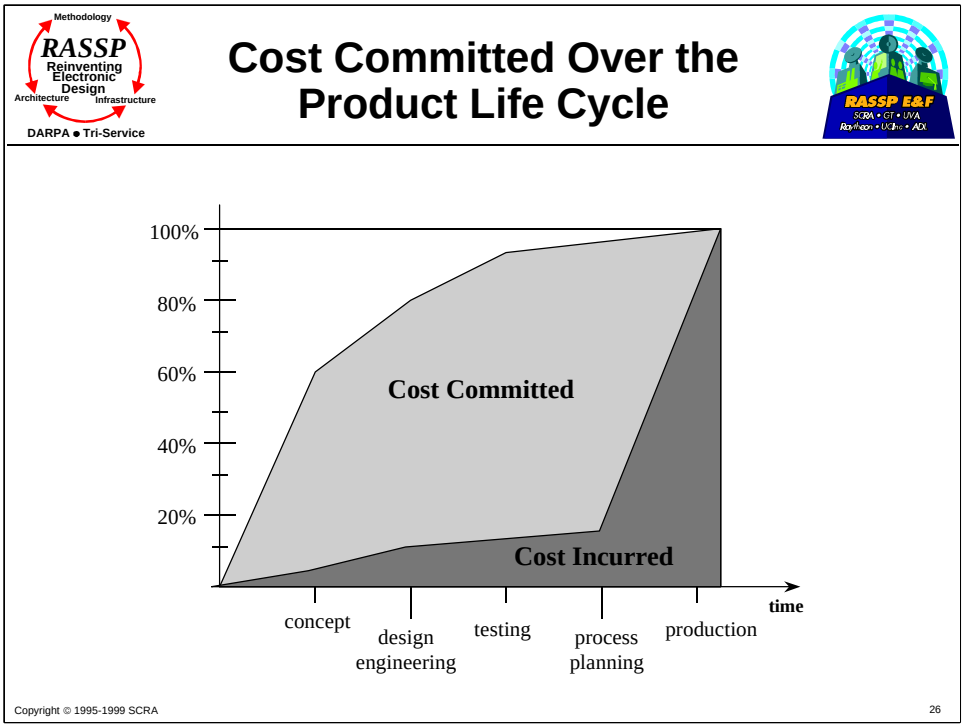
- This model partitions the product lifetime into three stages:
  - a market growth window W_1
  - a period of no growth (stagnation) S
  - a market decline W_2
- Market growth phase: The product begins to gain market acceptance and sales tend to grow rapidly as the product reaches mass market.
- Stagnation: As the product matures, sales are largely limited to repeat customers, since the majority of potential customers have already made their first choices. During this phase, there is no growth in the market.
- Market decline: As technology advances and superior products are launched, product sales will begin to decline. This downward trend in sales will continue as the market declines, thereby forcing managers to phase out the product.
- The non-shaded region of the curve represents the lost revenue R_L.
- R_0 is the total expected revenue if the product were on time, i.e D = 0.
- To maximize revenues, the product must be on the market by the product deployment deadline.
- For more information, see [LEVITT92].



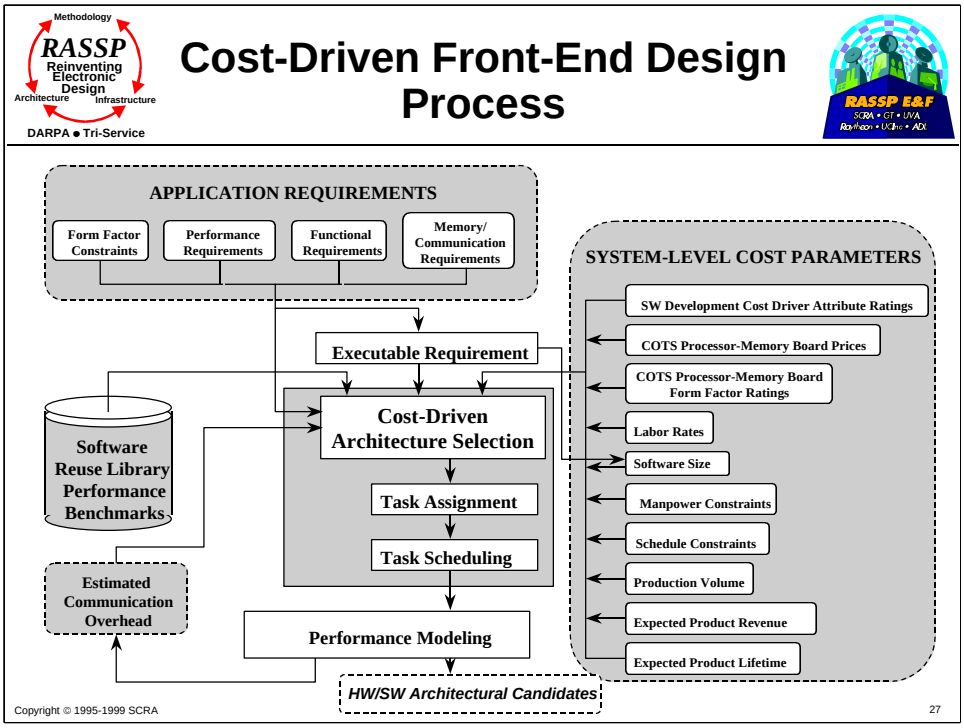
- With product cycles having durations of a few months, cost modeling can be quickly verified over the developmental cycle, and thus accurate cost estimates can be derived for future products



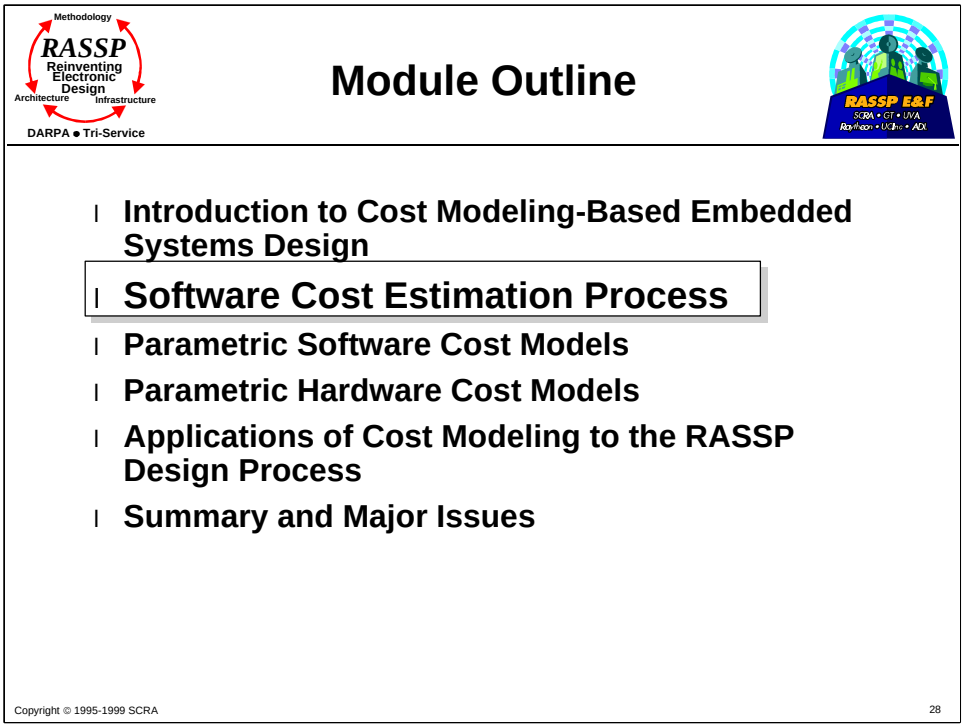
- The above diagram illustrates the design process [MAD95]. The application specification serves as input to the conceptual prototyping stage.
- Design and schedule constraints as well as behavioral specifications drive the conceptual design process.
- The conceptual design stage presents the best opportunity to utilize cost modeling techniques to develop minimal cost designs.
  - 80% of a product's final cost is determined by decisions made in the first 10% of the design cycle [SM94].
  - At this stage, high-level architectural trade-offs are made which only require rough cost estimates
    - Many universal (non-calibrated) parametric cost models are claimed by their developers to provide cost estimates within 20% of actuals 70% of the time which is sufficiently accurate to make high level design trade-offs.
    - These cost models are used to perform HW/SW partitioning, architecture selection, and packaging selection.
  - VHDL performance models are used to verify that these candidate architectures meet performance requirements.
- For more information, see [DEB97] and [MAD95].



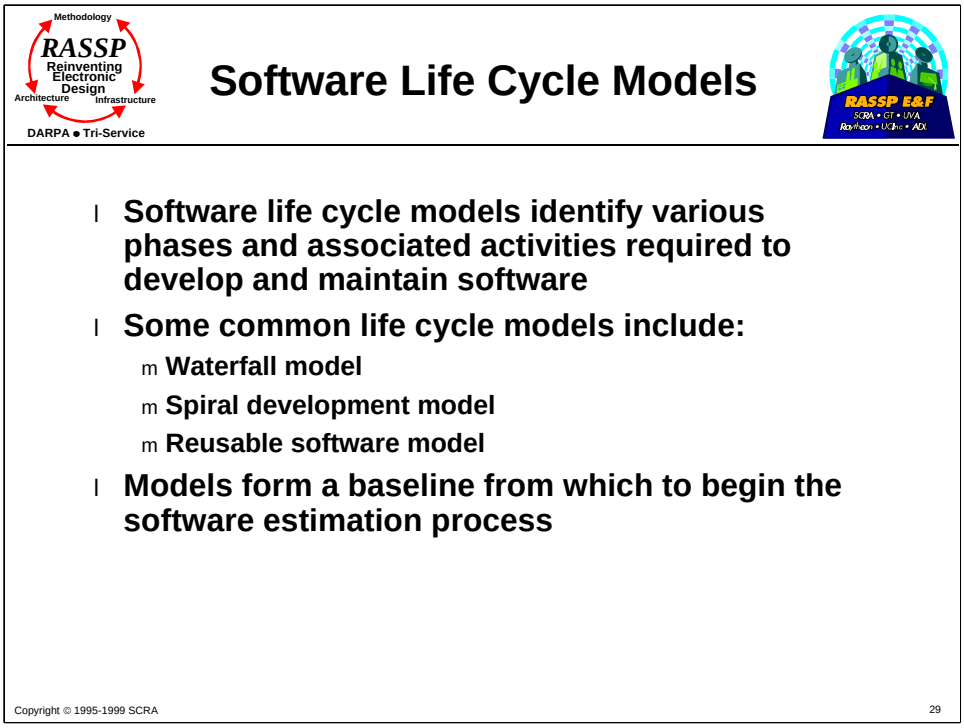
- Cost-effective embedded microsystems could benefit more from emphasizing cost-related issues during the early stages of design, than in the later stages.
- The figure depicts the cost committed versus the cost incurred over the product life cycle.
- The figure shows that a major portion of the projected life cycle cost for an electronic product stems from the consequences of decisions made during the early stages of design.
- Cost-effective products can only be produced by applying a high degree of cost emphasis during the early planning stages of design.
- For more information, see [*Business Week* 4-30-90].



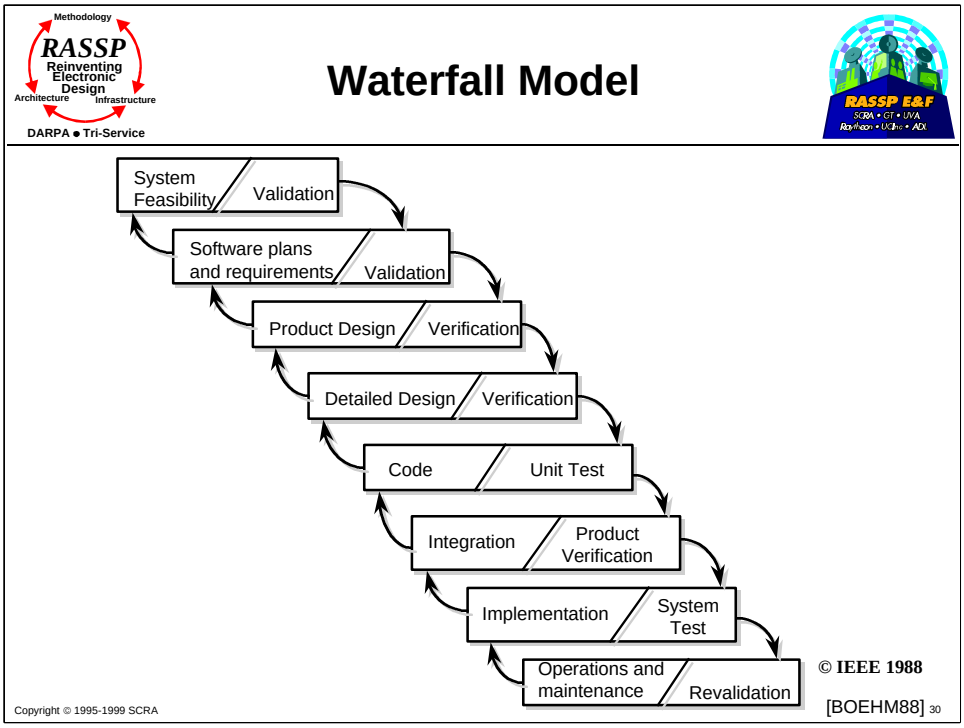
- The above diagram identifies where parametric cost models can be used in the front-end design process.
- Application requirements, system-level cost parameters, and DSP software library performance benchmarks serve as input to this design stage.
- Architectural trade-offs are made based on parametric cost estimates of software development, time-to-market losses, and maintenance.
- Example design objectives:
  - Choose the architectural platform which minimizes life cycle cost and/or maximizes profits while satisfying system-level design constraints.
  - Efficiently span the HW/SW design space
- The application task graph is then mapped to the architectural platform and scheduled.
- The performance of the resulting HW/SW architecture(s) is then verified using VHDL performance models.
- For more information, see [DEB97].



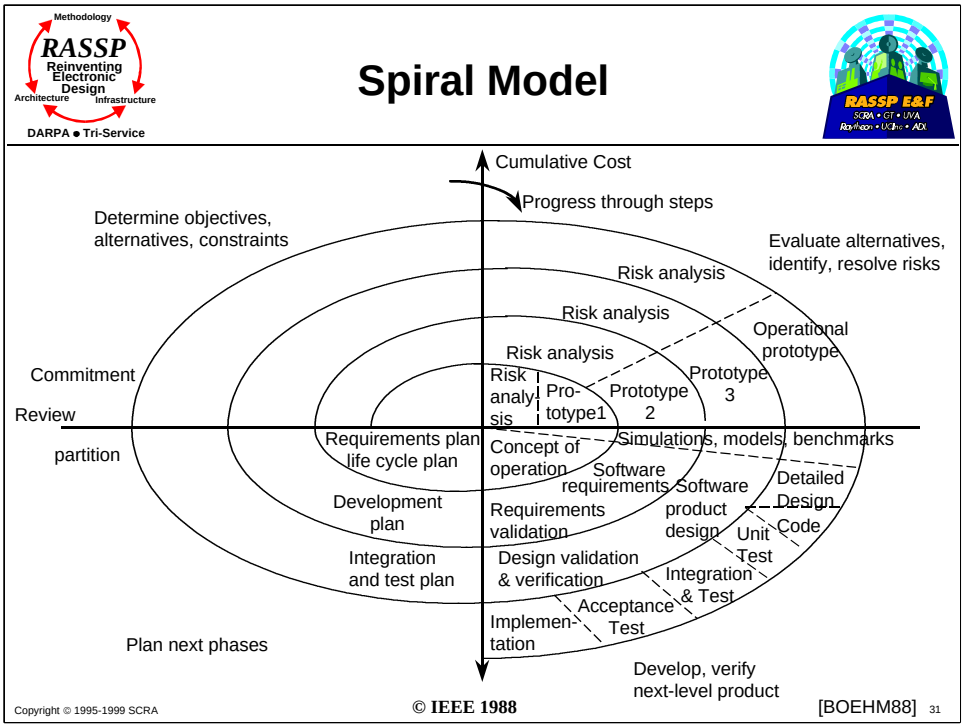
- We now present some methods that are used to make quantitative calculations of the software effort, the parameters of a model, and its dependence on the design methodology.



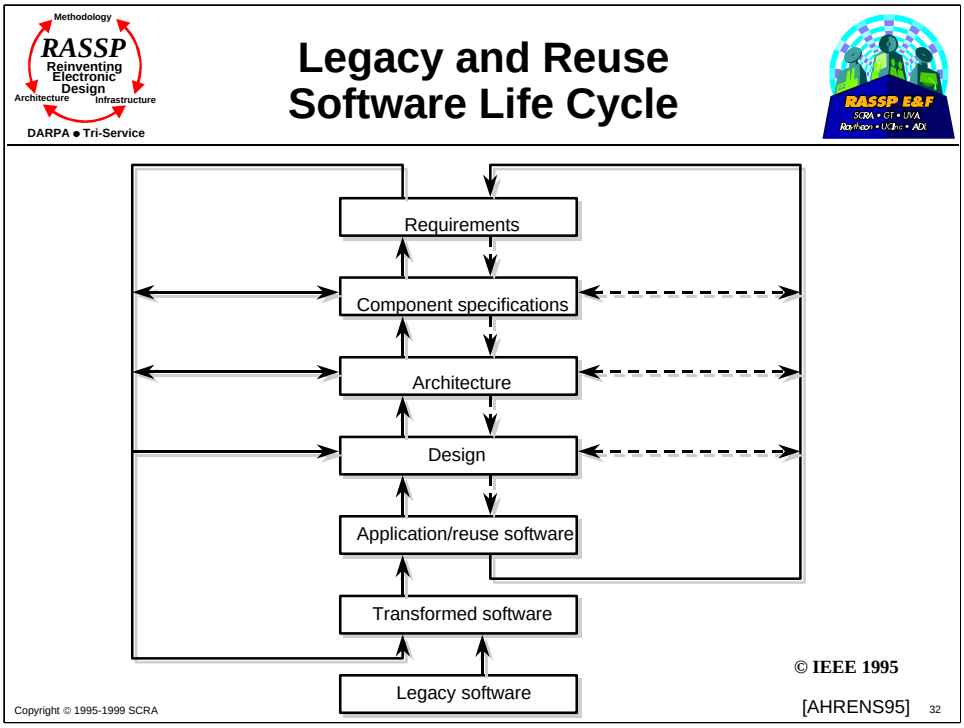
- **Note:** Extensions to the waterfall model cover:
  - incremental development
  - parallel development
  - program families
  - accommodation of evolutionary changes
  - formal software development and verification
  - stagewise validation and risk analysis
- Detailed explanation of waterfall model in next slide



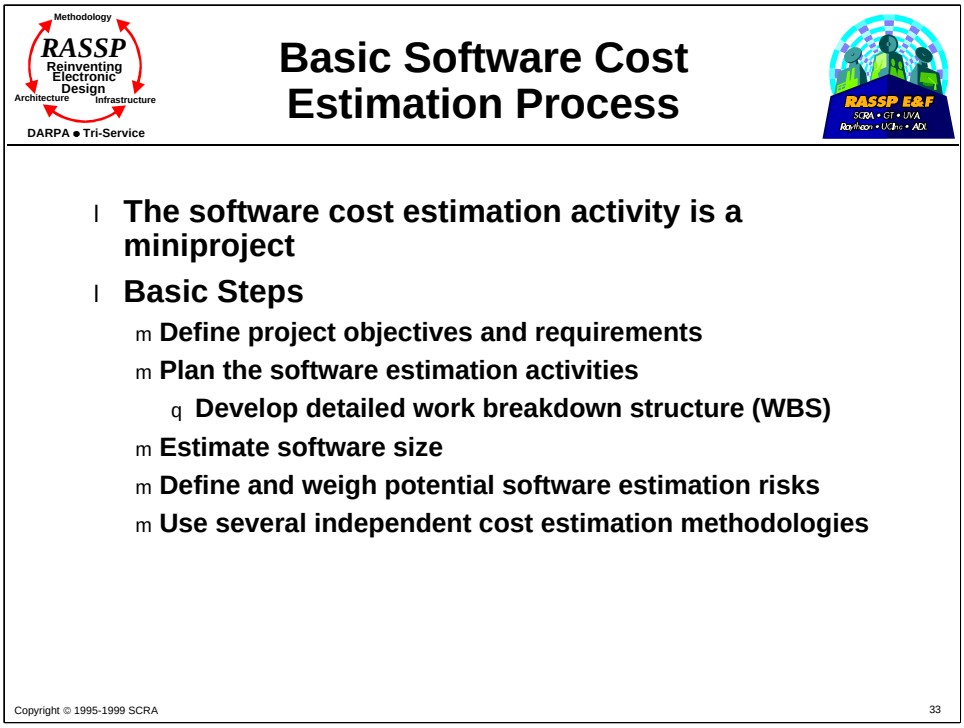
- Characteristics of the waterfall model
  - Emphasis on fully elaborated documents as completion criteria for early requirements and design phases
  - Iterations of earlier phase products are performed in the next succeeding phase
  - Each phase is culminated by a verification and validation activity



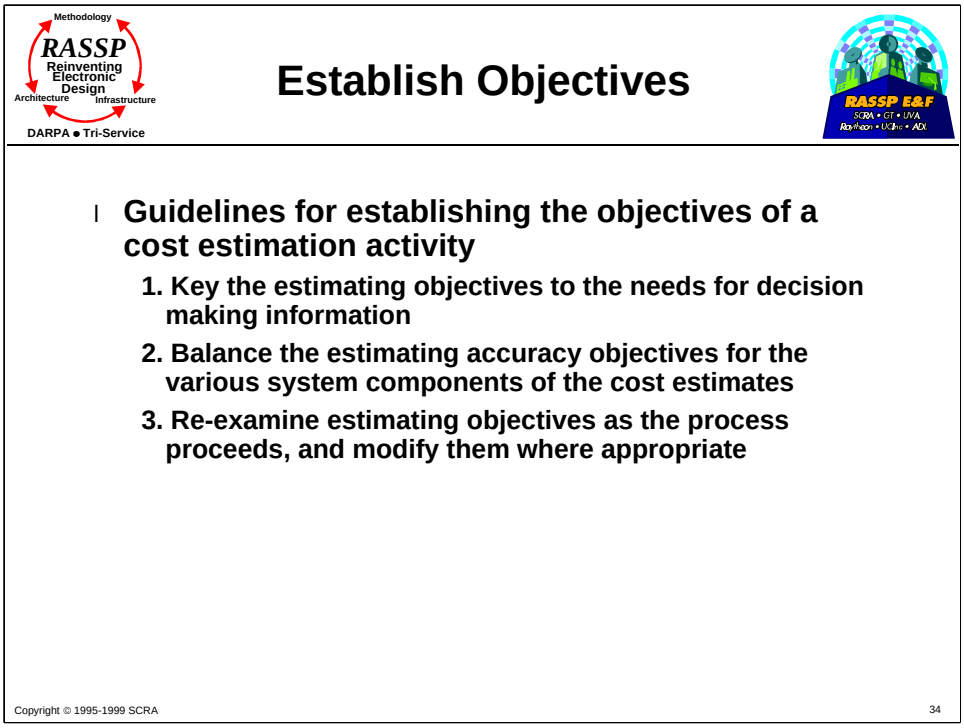
- Spiral Model of the software process-
  - Each cycle involves a progression that addresses the same sequence of steps for each portion of the product and for each of its levels of elaboration, from an overall concept of operation document down to the coding of each individual program



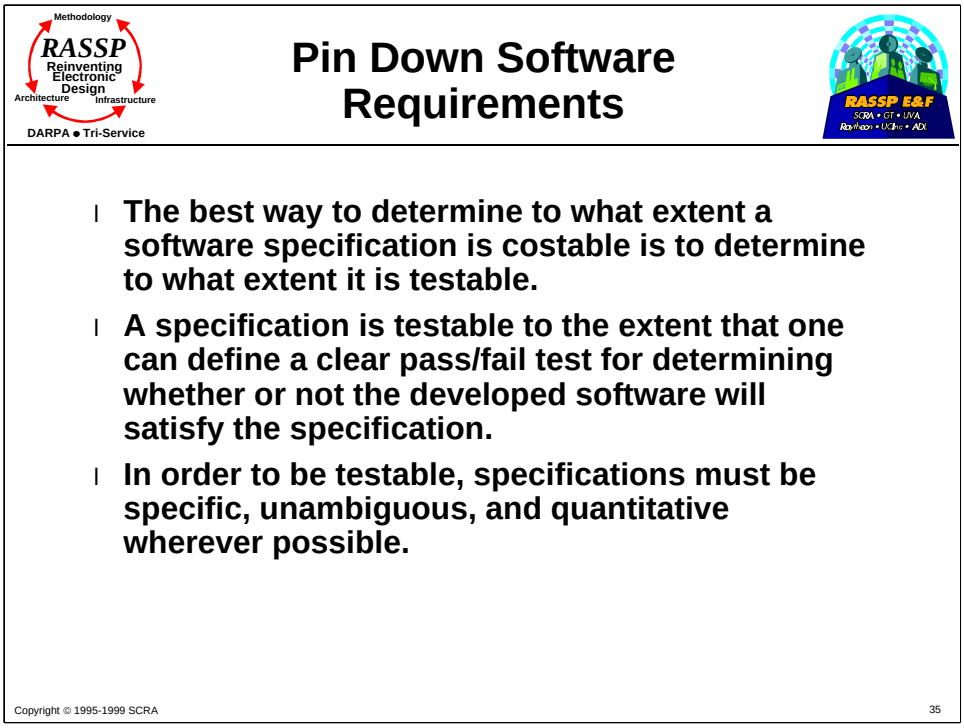
- Requirements - Domain or application software requirements defined in terms of functionality, capabilities, performance, user interface, inputs, and outputs
- Component specifications - Domain or application software requirements specified in terms of capabilities of hardware and software components and interfaces. This state is exemplified by the software specifications in DOD Military Standard 498, “Software Development and Documentation,” December 1994.
- Architecture - The hierarchy of software components , rules for component selection, and interfaces between components
- Design - Program interfaces, control flow, and logic defined in greater detail
- Application/reuse software - In application software, a unique software product; in software reuse, a library of adaptable resuable software components. The reuse software components are tested, verified, and validated
- Transformed software - Legacy software restructured and translated, if needed, into a modern programming language
- Legacy software - Application software created in a previous traversal of a software life cycle



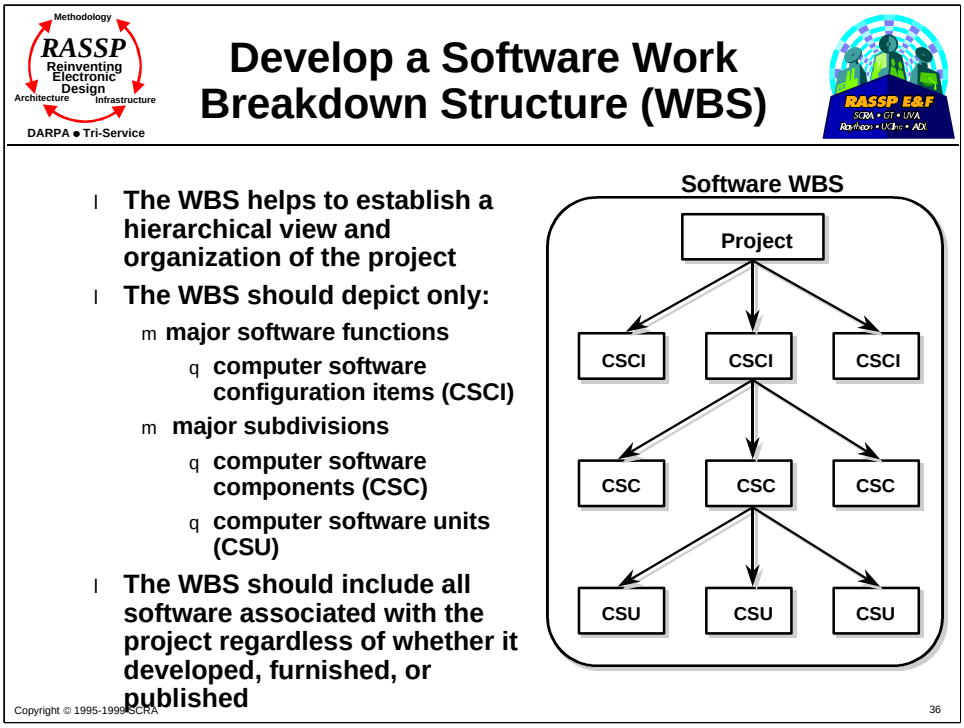
- What this slide suggests is a methodology for the cost modeling activity itself and methods for starting it within an organization.
- These various sub-activities are discussed in the following slides.



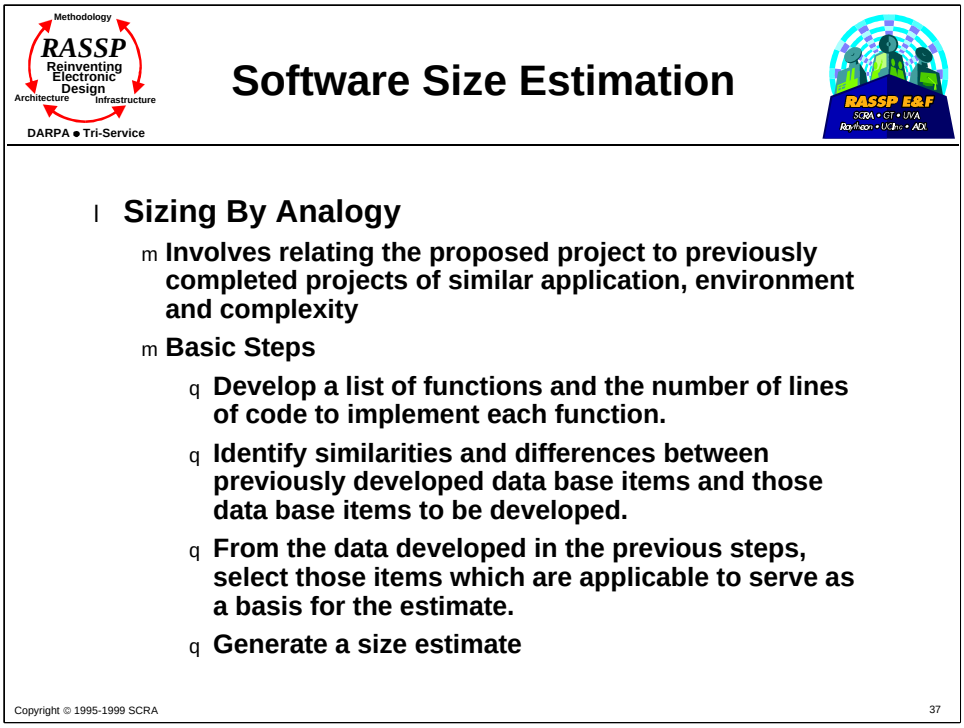
- The main factor that helps to establish cost-estimation objectives is the current software life-cycle phase.
  - Corresponds to the level of knowledge of the software whose costs is being estimated, and also to the level of commitment that will be made as a result of the estimate
  - Guideline #1 refers to absolute estimates for labor or resource planning, relative estimates for either/or decisions, generous or conservative estimates to heighten confidence in the decision.
  - Guideline #2 recommends that the absolute magnitude of the uncertainty range for each component be roughly equal - assuming that such components have equal weight in the decision to be made.
  - A further implication of guideline #3 is that budget commitments in the early phases should cover only the next phase. Once a validated product design is complete, a total development budget may be established without too much risk.



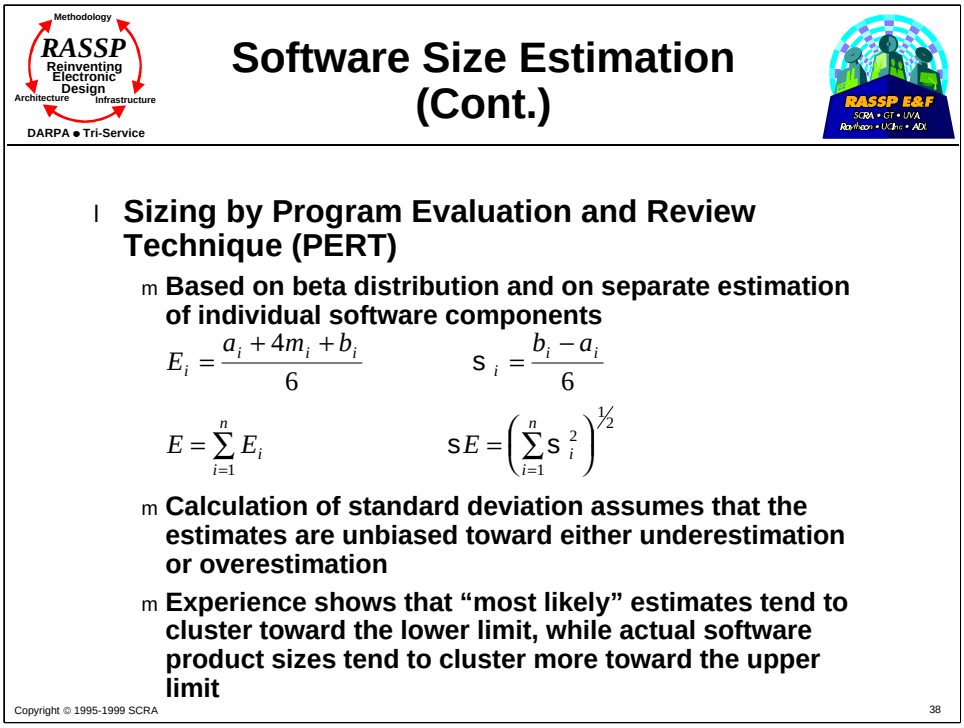
- Examples of testable specifications
  - The software shall compute aircraft position within the following accuracies:
    - + or - 50 ft in the horizontal plane
    - + or - 20 ft in the vertical plane
  - The system shall respond to:
    - Type A queries in <= 2 sec
    - Type B queries in <= 10 sec
    - Type C queries in <= 2 min
    - where Type A, B, and C queries are defined in the specification



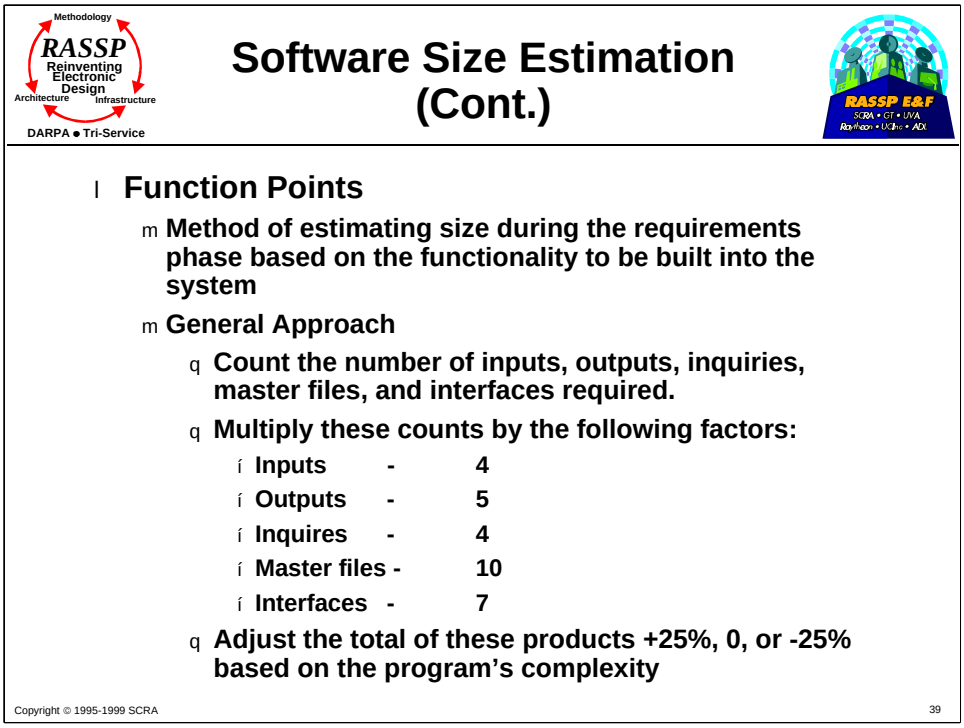
- In order to facilitate the planning of software cost estimation activities, a work breakdown structure should be developed.
  - Plan should detail the purpose, products, schedules, responsibilities, procedures, required resources, and assumptions made.
- The WBS provides a framework for specifying the technical objectives of the program by first defining the program in terms of hierarchically related product oriented elements and the work processes required for their completion.
- Each element of the WBS provides logical summary points for assessing technical accomplishments, and for measuring the cost and schedule performance accomplished in attaining the specified technical objectives.
- The WBS should be worked out in as much detail as feasible.
  - The more detail to which estimating activities are carried out, the more accurate the estimates will be.
  - The more the functions that software must perform are contemplated, the less likely the costs of some of the more obtrusive components of the software will be missed
  - The software size estimate, the major software cost driver, is computed by making estimates for the software WBS elements.



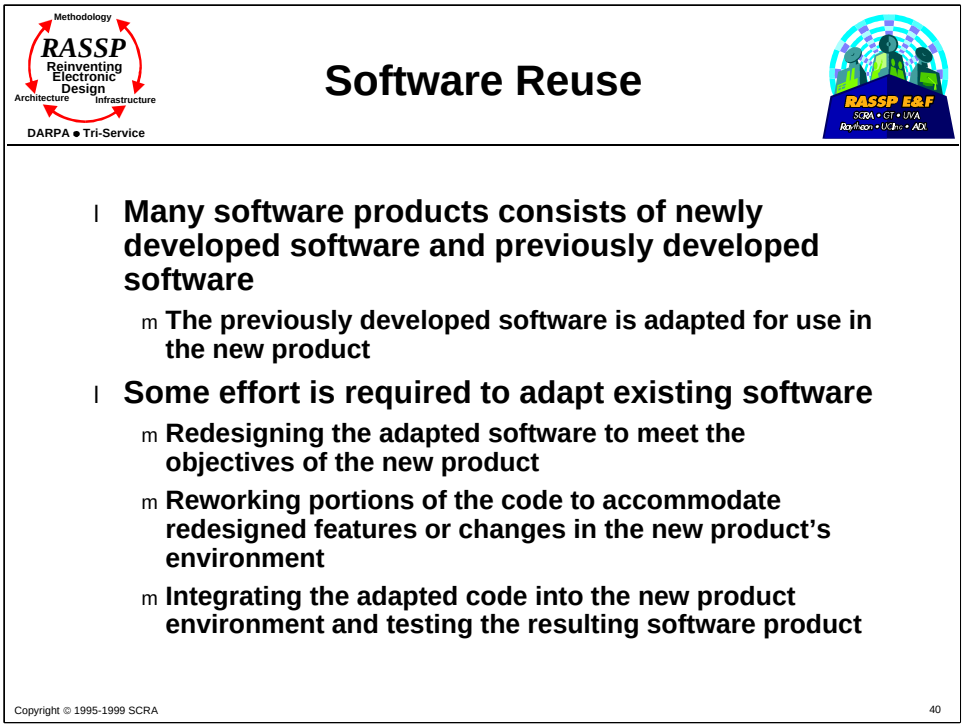
- A manual estimate in thousands of source lines of code (SLOC) or function points to the lowest level of detail possible (bottom-up) for each major function within each CSCI based on experience with a similar application and historical data.
- Software includes application code, operating system, control, diagnostics, and support software.
- The accuracy of the derived estimate will obviously depend on the completeness and the accuracy of the data used from the previous projects
- Actual data from completed projects are extrapolated to estimate the proposed project's software size.
- The strength of this method is that the estimates are based on actual project data and past experience.
- Limitations
  - Difficult to identify differences between completed projects and the proposed project
  - Accuracy of available historical information may be suspect
  - Similar projects may not exist
  - Some projects have no historical precedents
- For more information, see [SEPO96].



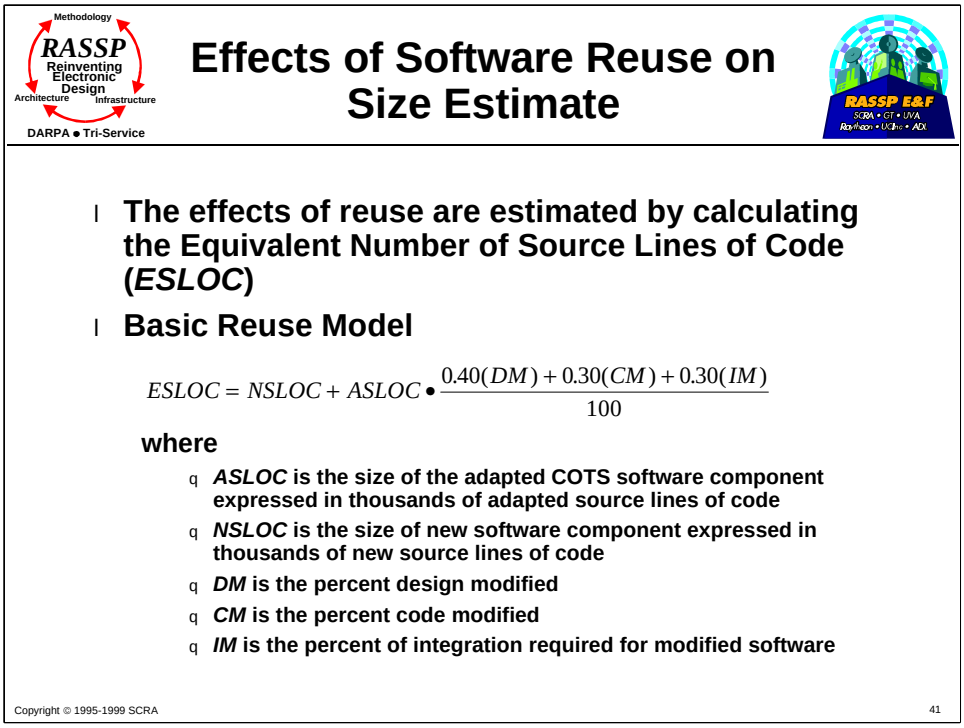
- PERT requires a nominal or expected size estimate plus a standard deviation(I.e., the lowest possible size and a highest possible size to reflect the uncertainty of the nominal estimate) to be developed.
- The spread between the lowest and the highest estimates may be as much as 30-50% in the early phases of a project, e.g., the Concept phase.
- Symbol definitions
  - a_i = The lowest possible size of the software component
  - b_i = The highest possible size of the software component
  - m_i = the most likely size of the component
  - E_i = the expected size of the software component
  - σ_i = the standard deviation of the software component estimate
  - E = the estimated total software size
  - σE = the standard deviation of the total estimate
- For more information, see [BOEHM81].



- In function point analysis, the number and complexity of inputs, outputs, user queries, files, and external interfaces of the software to be developed are determined.
- Initial application requirements statements are examined to determine the number and complexity of the various inputs, outputs, calculations and databases required.
- Points based on established values are assigned to each of these counts and then added to arrive at an overall function point rating for the product.
- This function point number is directly related to the number of end-user business functions performed by the system.
- Using data from past projects, it is possible to estimate the size of the software needed to implement these function points (typically about 100 source language statements are needed for each function point) and the labor needed to develop the software (typically about 1 to 5 function points per person-month).
- This approach is helpful in estimating size very early in a software product's development.
- For more information, see [SEPO96] and [DOD95].

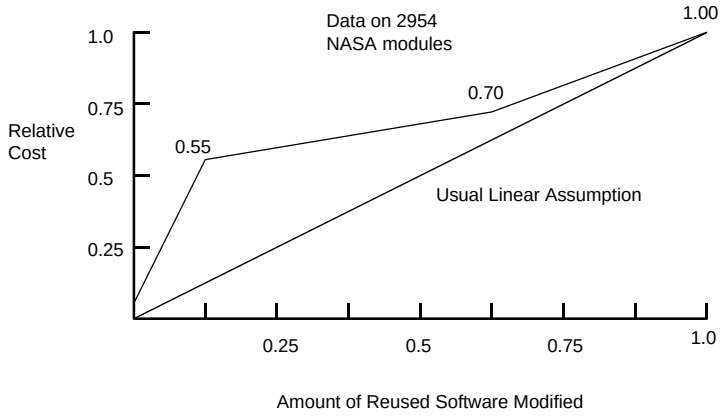


- Software reuse has been proposed many times in the 80s and 90s as a means to reduce cost of software development, and many ideas are currently being carried over to the hardware and embedded design areas.



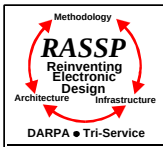
- **ASLOC**: the number of source lines of code adapted from existing software to form the new product.
- **DM**: the percentage of the adapted software's design which is modified in order to adapt it to the new objectives and environment.
- **CM**: the percentage of the adapted software's code which is modified in order to adapt it to the new objectives and environment.
- **IM**: the percentage of effort required to integrate the adapted software into an overall product and to test the resulting product as compared to the normal amount of integration and test effort for software of comparable size.
- For more information, see [BOEHM81].

Nonlinear Reuse Effects



[BOEHM95]

- The reuse cost function is nonlinear in two ways:
  - It does not go through the origin. There is generally a cost of about 5% for assessing, selecting, and assimilating the reusable component.
  - Small modifications generate disproportionately large costs. This is primarily due to two factors: the cost of understanding the software to be modified, and the relative cost of interface checking.



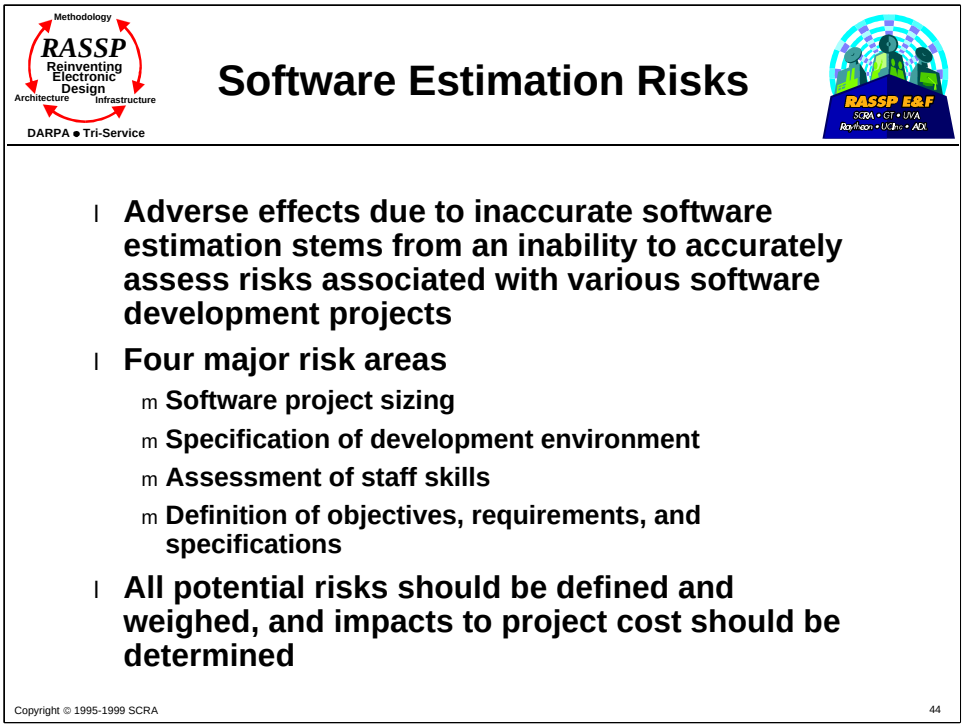
Nonlinear Reuse Model

- **Nonlinear reuse estimation model**

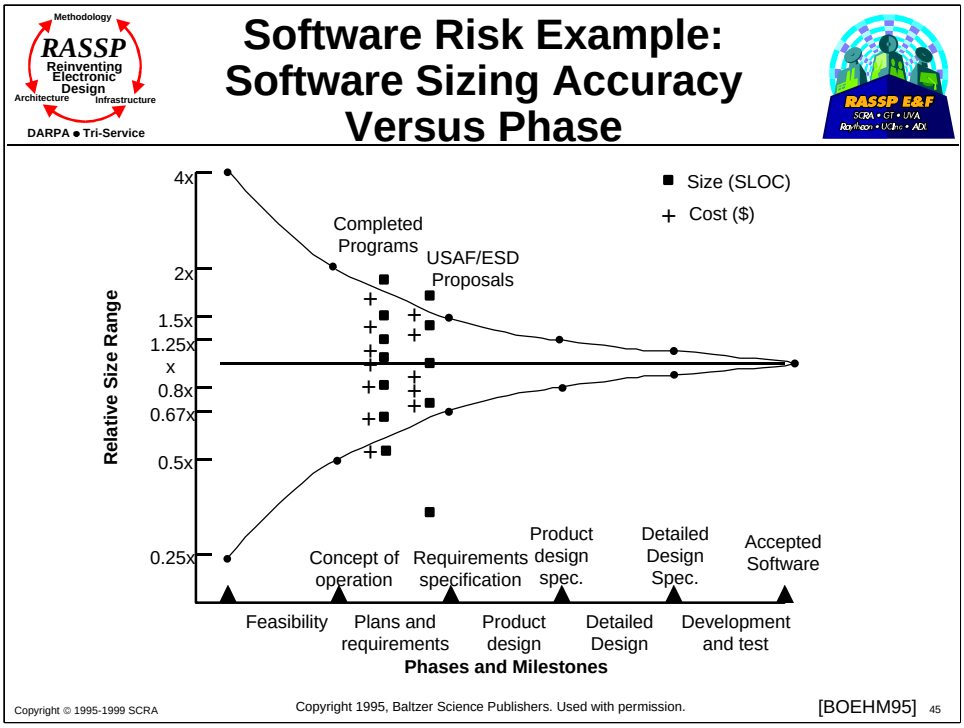
$$ESLOC = NSLOC + ASLOC \times \frac{(AA + SU + 0.4 \times DM + 0.3 \times CM + 0.3 \times IM)}{100}$$

  - **where**
    - ***SU* is the software understanding increment which is expressed as a percentage (ranges from 10-50%).**
    - ***AA* deals with the degree of assessment and assimilation needed to determine whether a fully-reused software module is appropriate to the application and to integrate its description into the overall product description (ranges from 0-8%).**
    - **The remaining variables in the equation are the same as those used in the original COCOMO.**

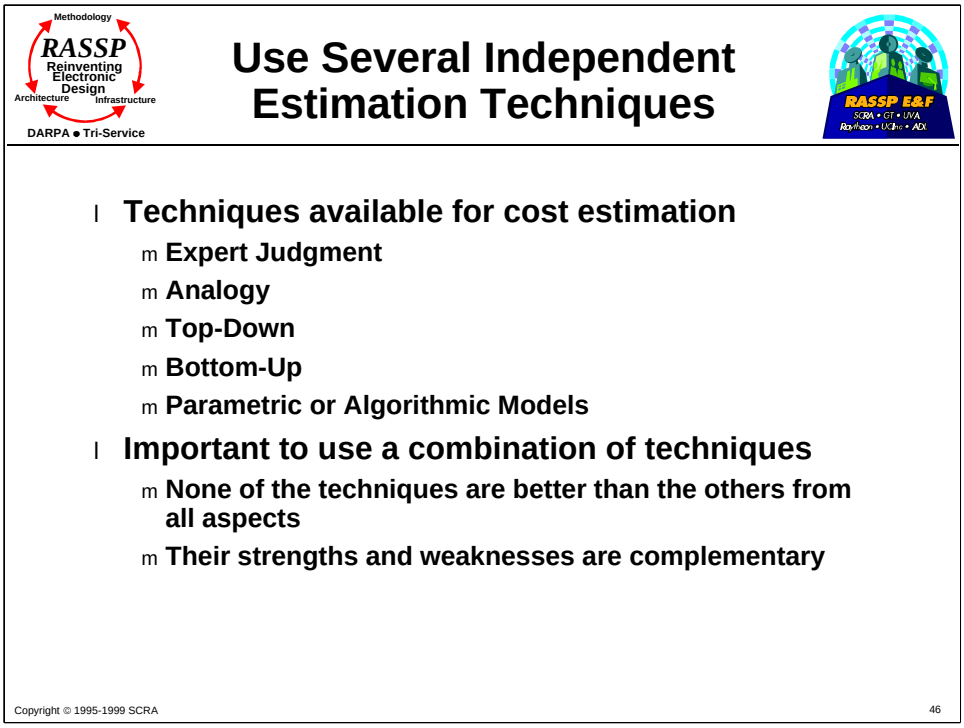
- The software understanding increment is based on the level of the adapted software's structure, application clarity, and self-descriptiveness
- Software structure ratings
  - *Very Low*: very low cohesion, high coupling, spaghetti code
  - *Low*: Moderately low cohesion, high coupling
  - *Nominal*: Reasonably well-structured; some weak areas
  - *High*: high cohesion, low coupling
  - *Very High*: Strong modularity, information hiding in data/control structures
- Application clarity ratings
  - *Very Low*: no match between program and application world views
  - *Low*: Some correlation between program and application
  - *Nominal*: Moderate correlation between program and application
  - *High*: Good correlation between program and application
  - *Very High*: Clear match between program and application worldviews
- Self-Descriptiveness ratings
  - *Very Low*: Obscure code; documentation missing, obscure or obsolete
  - *Low*: Some code commentary and headers; some useful documentation
  - *Nominal*: Moderate level of code commentary, headers, documentations
  - *High*: Good code commentary and headers; useful documentation; some weak areas
- Levels of AA effort include: none; basic module search and documentation; some module Test and Evaluation (T&E), documentation; considerable module T&E, documentation, extensive module T&E, documentation
- For more information, see [BOEHM95].



- The inability to accurately size the software project results in poor implementations, emergency staffing, and cost overruns caused by underestimating project needs.
- The inability to accurately specify a development environment which reflects reality results in defining cost drivers which may be inappropriate, underestimated or overestimated.
- The improper assessment of staff skills results in misalignment of skills to tasks and ultimately miscalculations of schedules and level of effort required.
- The lack of well defined objectives, requirements, and specifications, or unconstrained requirements growth during the software development life cycle results in forever changing project goals, frustration, customer dissatisfaction, and ultimately, cost overruns.

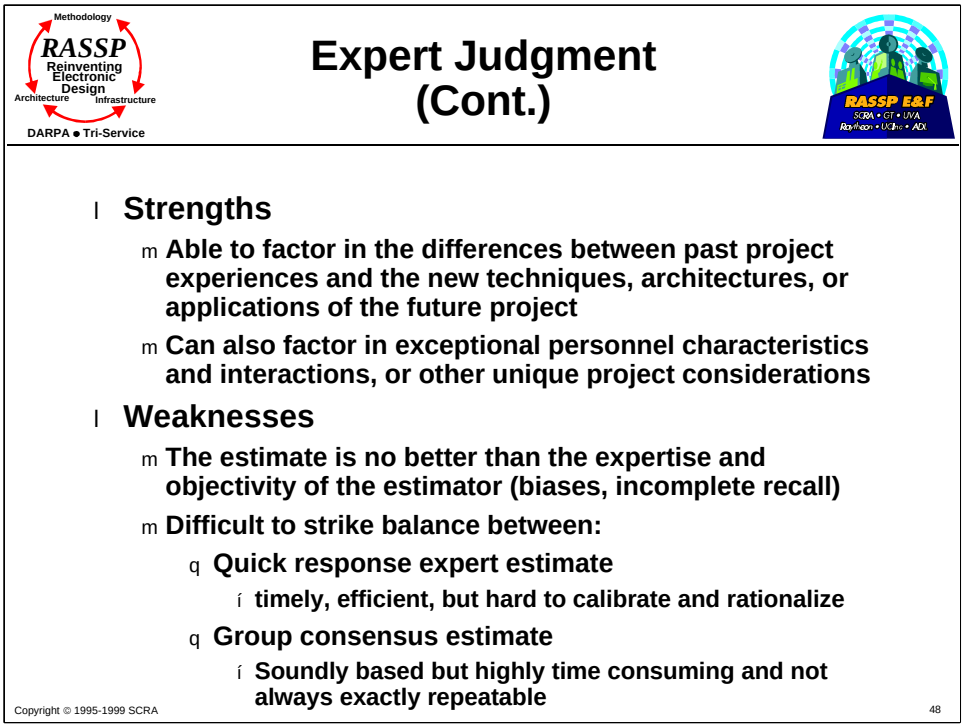


- This figure indicates the effect of project uncertainties on the accuracy of software size and cost estimates.
- In the very early stages, one may not know the specific nature of the product to be developed to better than a factor of 4 (25% -400%).
- As the life cycle proceeds, and product decisions are made, the nature of the product decisions are made, the nature of the products and its consequent size are better known, and the nature of the process and its consequent cost drivers are better known.
- Once written requirements have been specified, software costs/size can be estimated with a factor of 1.5 (67% - 150%) from project actuals.
- The earlier “completed programs” size and effort data points shown above are the actual sizes and efforts of seven software products built to an imprecisely-defined specification.
- The latter “USAF/ESD proposals” data points are from five proposals submitted to the U.S. Air Force Electronics Systems Division in response to a fairly thorough specification.
- SLOC - source lines of code

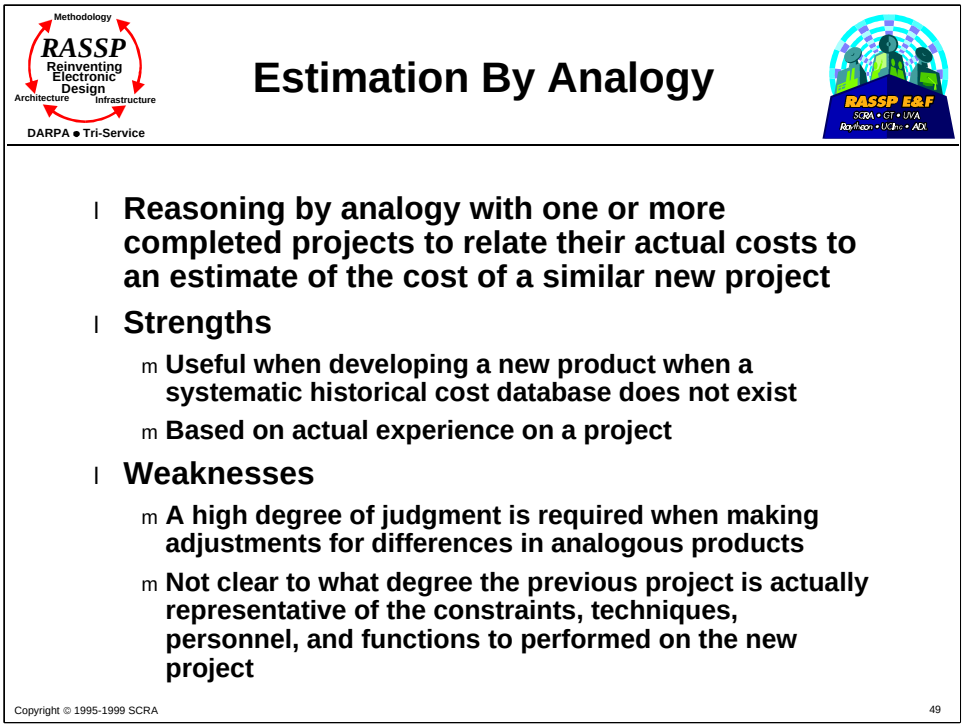


- More than one software estimation methodology should be used for comparison and verification purposes.
  - One method may overlook system level activities such as integration, while another method may have included this, but overlooked some key post-processing components.
  - For more information, see [BOEHM81].

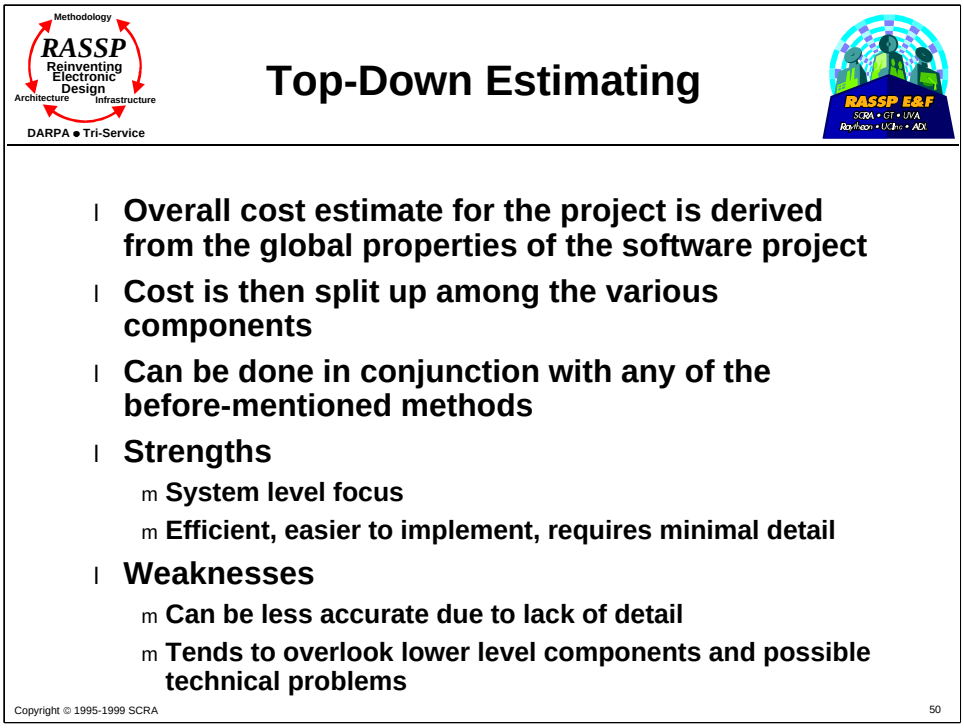
- Wideband Delphi Technique
  - Coordinator presents each expert with a specification and an estimation form.
  - Coordinator calls a group meeting in which the experts discuss estimation issues with the coordinator and each other.
  - Experts fill out forms anonymously
  - Coordinator prepares and distributes a summary of the estimates on an iteration form.
  - Coordinator calls a group meeting, specifically focusing on having the experts discuss points where their estimates varied widely.
  - The experts review the summary and submit another anonymous estimate on the form.
  - The last three steps are repeated until a consensus is reached.



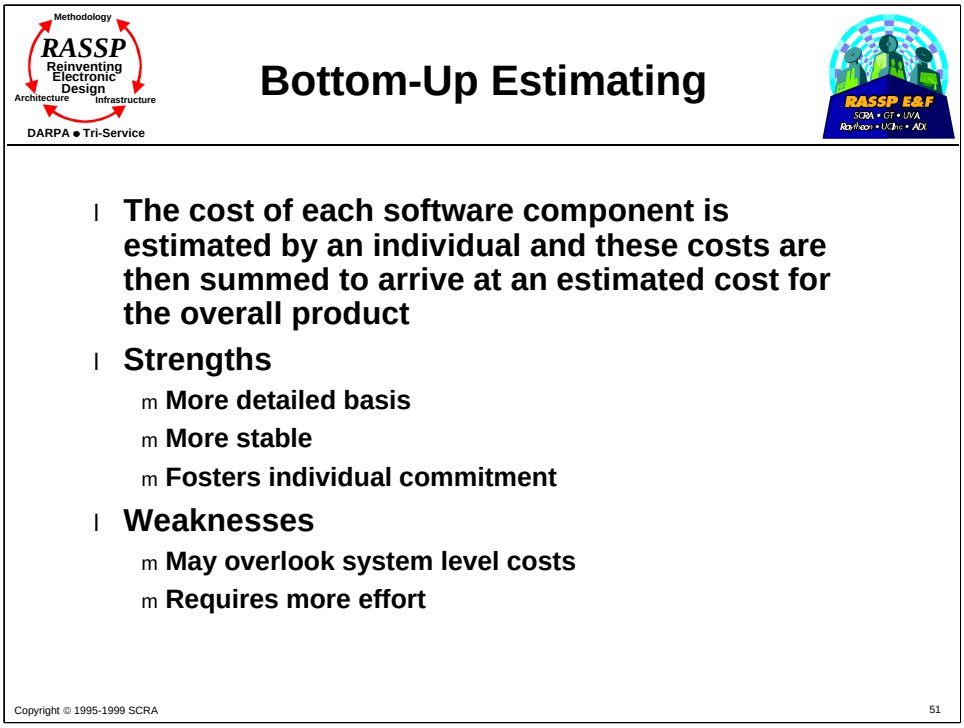
- While this technique appears ad hoc, it is widely used basis for costing products, market segments, and proposals.



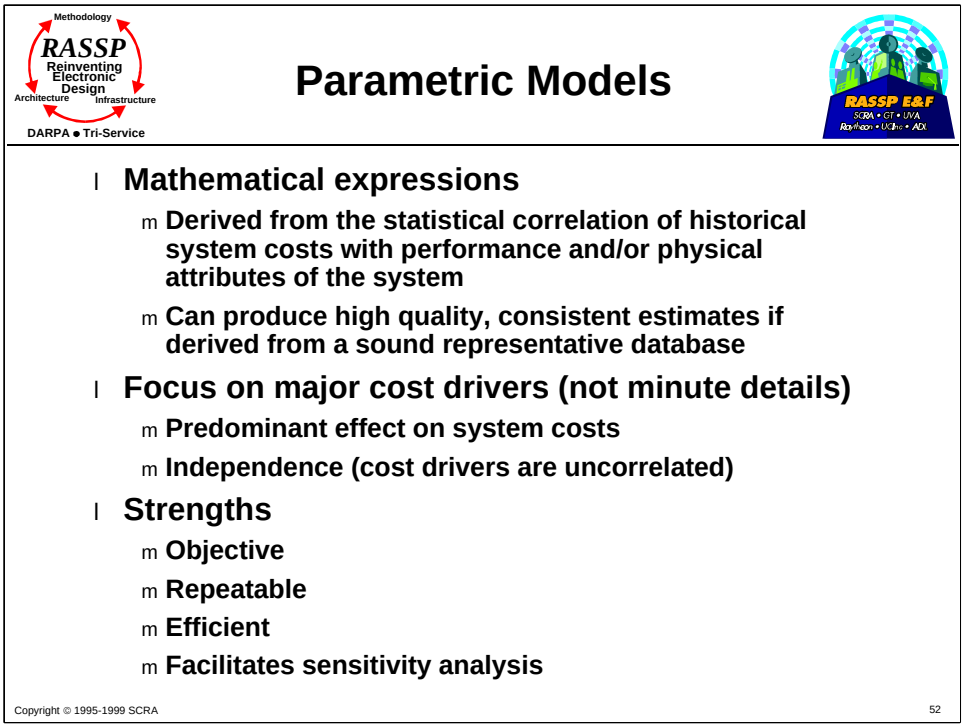
- If an organization has a legacy of similar projects, or has a database that records historical cost information, it may serve as a good starting point for the baseline cost of a new similar activity.



- Detailed explanation of strengths
  - The estimate is based on previous experience on entire completed projects, it will not miss the costs of system level functions such as integration, users' manuals, configurations management, etc.
  - Detailed explanation of weaknesses
  - This method often does not idenitify difficult low level technical problems that are likely to escalate costs.
  - It sometimes misses components of the software to be developed.
  - It provides no detailed basis for cost justification and iteration.
  - It is less stable than a multicomponent estimate, in which estimation errors in the components have a chance to balance out.



- Detailed explanation of strengths
  - Estimate will be based on a more detailed understanding of the job to be done
  - Each estimate will be backed up by the personal commitment of the individual responsible for the job
  - Estimates are more stable because the estimation errors in the various components have a chance to balance out.
- Detailed explanation of weaknesses
  - Tends to overlook many system level costs (integration, configuration management, quality assurance, project management) associated with software development.
  - As a result, these results are often underestimated.



- Weaknesses
  - Calibrated to previous projects
    - Questionable as to what extent these previous projects are representative of future projects using new techniques and technology
    - Unable to deal with exceptional conditions
    - exceptional personnel
    - exceptional project teamwork
    - exceptional matches (or mismatches) between the project personnel and the job to be done
- However, any estimating approach can be impacted by these drawbacks, and care should be exercised when accounting for such concerns.

Parametric Models (Cont.)

- **Linear models**

$$Effort = a_0 + \sum_{i=1}^{n} a_i x_i$$

- **Multiplicative models**

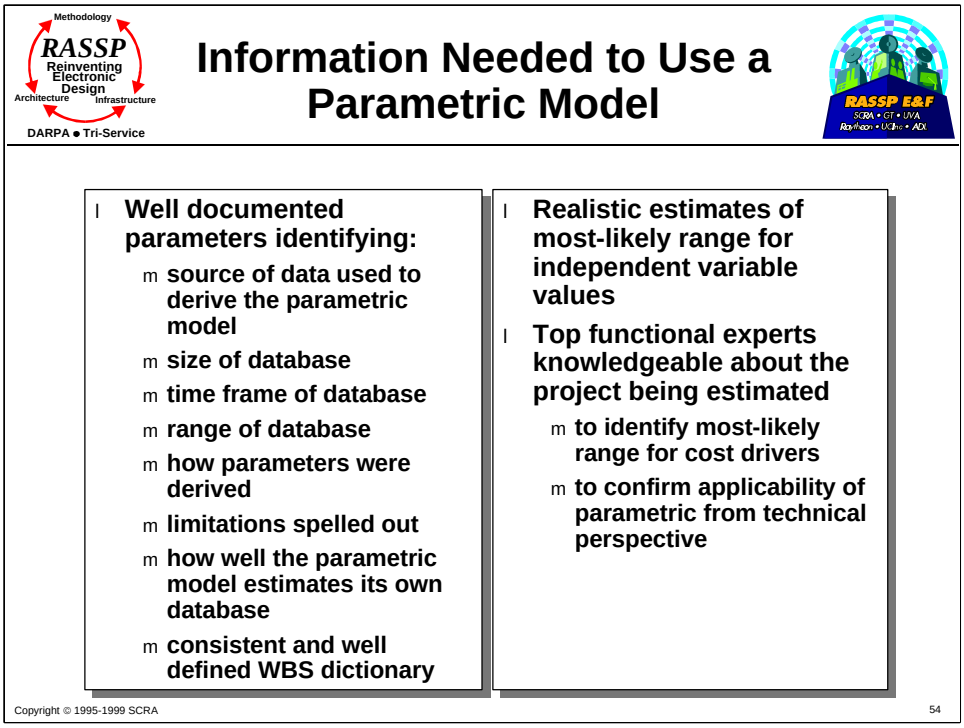
$$Effort = a_0 \prod_{i=1}^{n} a_i^{x_i}$$

- **Analytic models**

$$Effort = f(x_1, \ldots, x_n)$$

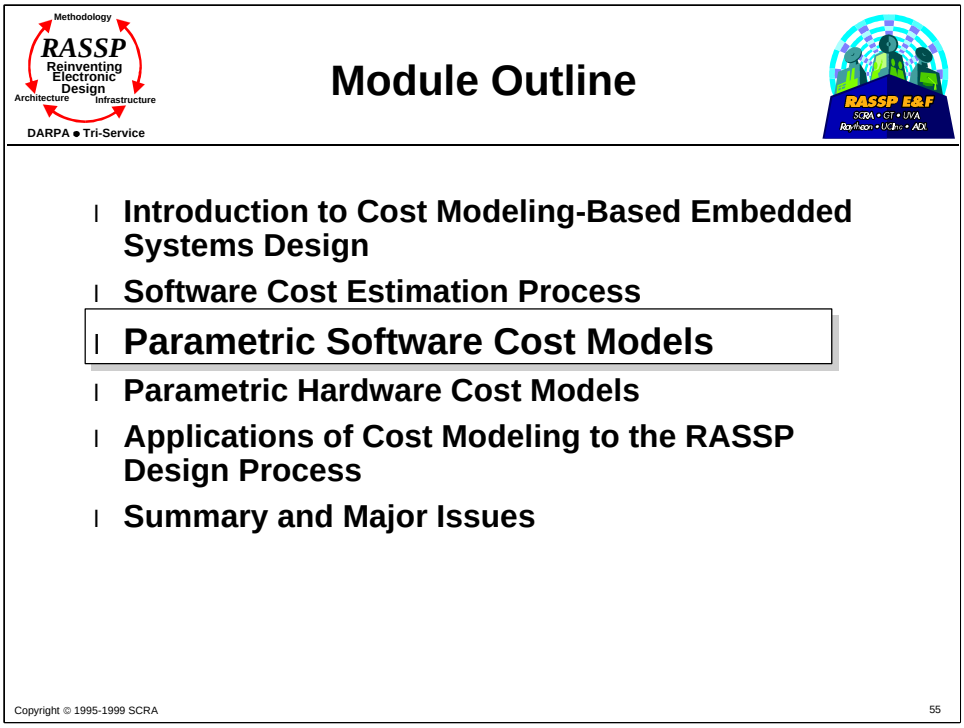
- **Tabular models**
  - **Contains tables which relate the values of cost driver variables to multipliers used to adjust the effort estimate**
- **Composite models**
  - **Incorporate a combination of linear, multiplicative, analytic, and tabular functions to estimate software effort as a function of cost driver variables**

- These models have the forms shown above where $x_1, \ldots, x_n$ are the cost driver variables, and $a_0, \ldots, a_n$ are a set of coefficients chosen to provide the best fit to a set observed data points. Development cost is obtained by multiplying the effort quantity by a constant cost for labor.
- Model coefficients are determined by using curve fitting techniques such least squares best fit (LSBF) method, multiple regression techniques, or curvilinear regression techniques.

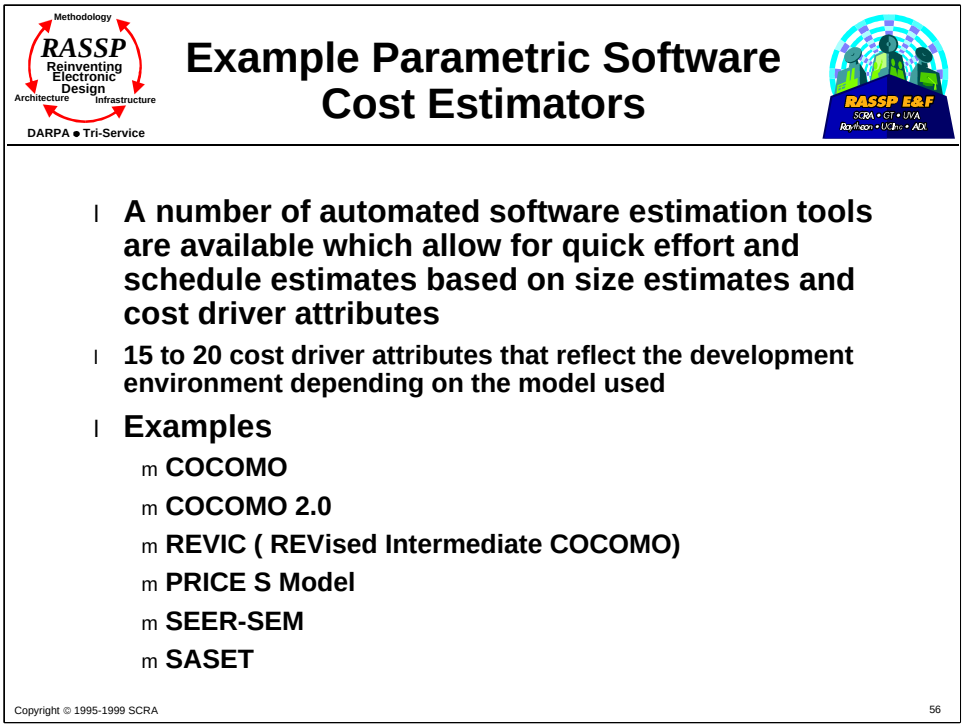


- These prescriptions provide an insight into how accurate the proposed parametric model is, and also provide some directions on its effective use and limitations. One should not use cost models without information on their underlying assumptions.

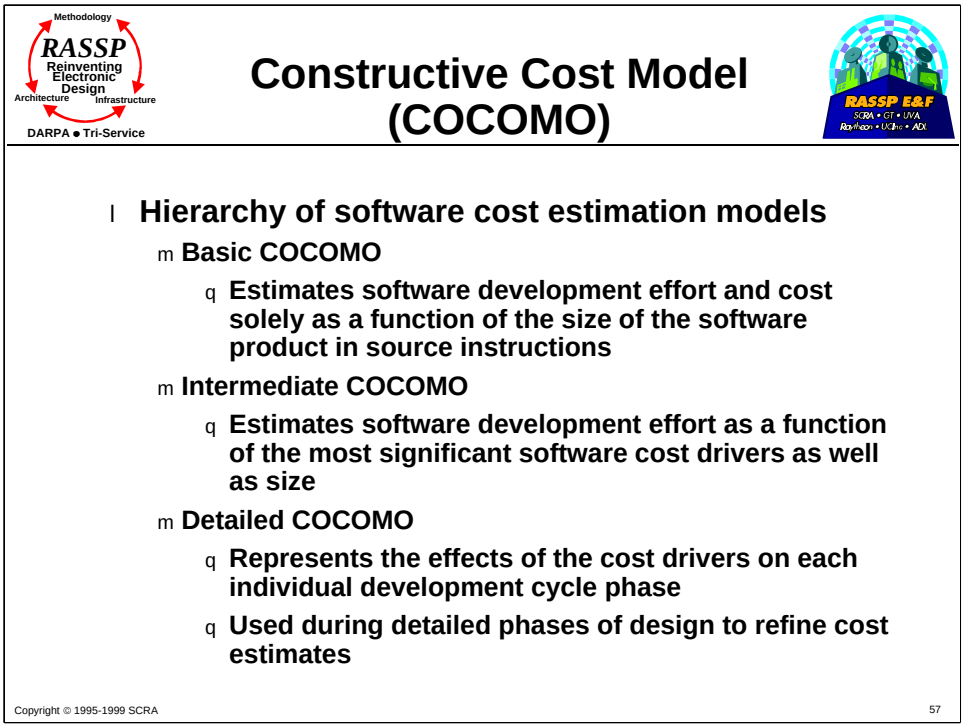
- For more information, see [DOD95].



- We examine a number of commercial cost models from the software arena, and see how they are applicable to embedded system design.

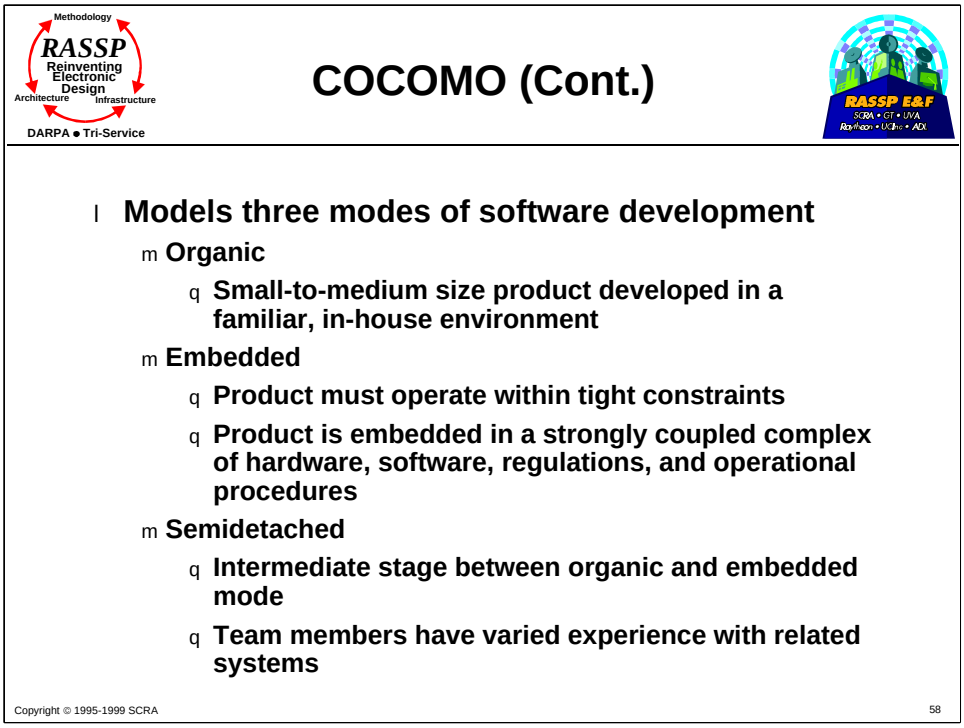


- SEER-SEM: System Evaluation and Estimation of Resources - Software Estimation Model (SEER-SEM) provides software estimates with knowledge bases developed from many years of completed projects.The knowledge base allows estimates with only minimal high level inputs. User only needs to select the platform (I.e. ground, unmanned space, etc.), application (I.e. command and control, diagnostic), development methods (I.e. prototype, incremental), and development standards (I.e. 2167A/498).
- SASET: The Software Architecture, Sizing and Estimating Tool (SASET) is a forward-chaining, rule-based expert system utilizing a hierarchically structured knowledge database. SASET uses a five-tiered approach for estimating including class of software, source lines of code, software complexity, maintenance staff loading, and risk assessment.
- COCOMO, COCOMO 2.0, and REVIC are public domain software cost estimation tools. Therefore, the parametric software cost/schedule equations are made available for these tools, thereby facilitating their use in automated design tools.
- PRICE S is a commercial tool. Its exact parametric equations are not made available. Therefore, for the purposes of this module, PRICE S can only be used for making manual design trade-offs.
- Others include SLIM, SOFTCOST-R, SoftEst and SYSTEM-4

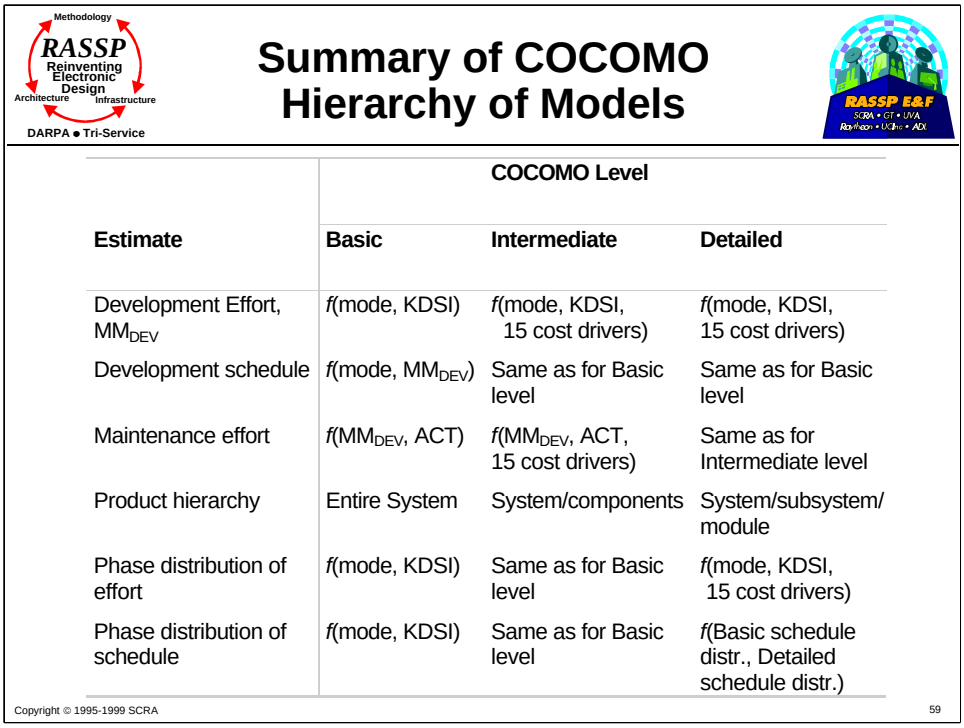


- The development period covered by COCOMO cost estimates begins at the beginning of the product design phase and ends at the end of the integration and test phase.
- The three models, Basic, Intermediate, and Detailed COCOMO, represent greater accuracy based on an increasing amount of input information provided to each class of models.
- In the early stages of the design, Basic COCOMO may be sufficient for an initial estimate, but as the project advances, advanced models may be required for increased accuracy of prediction.
- For more information, see slide 59.

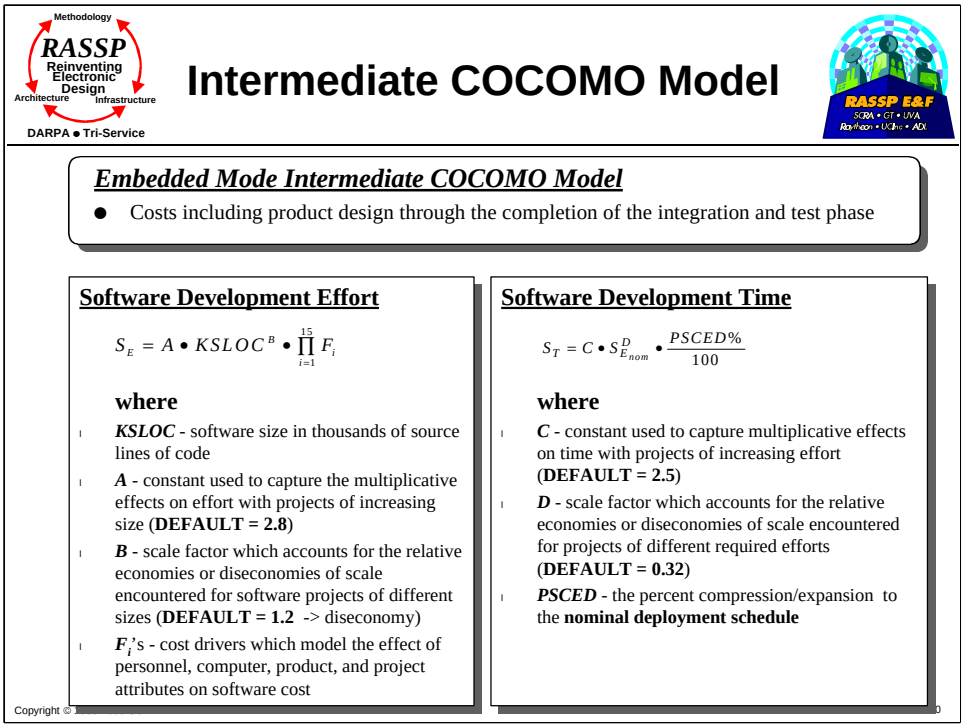
[COCOMO]



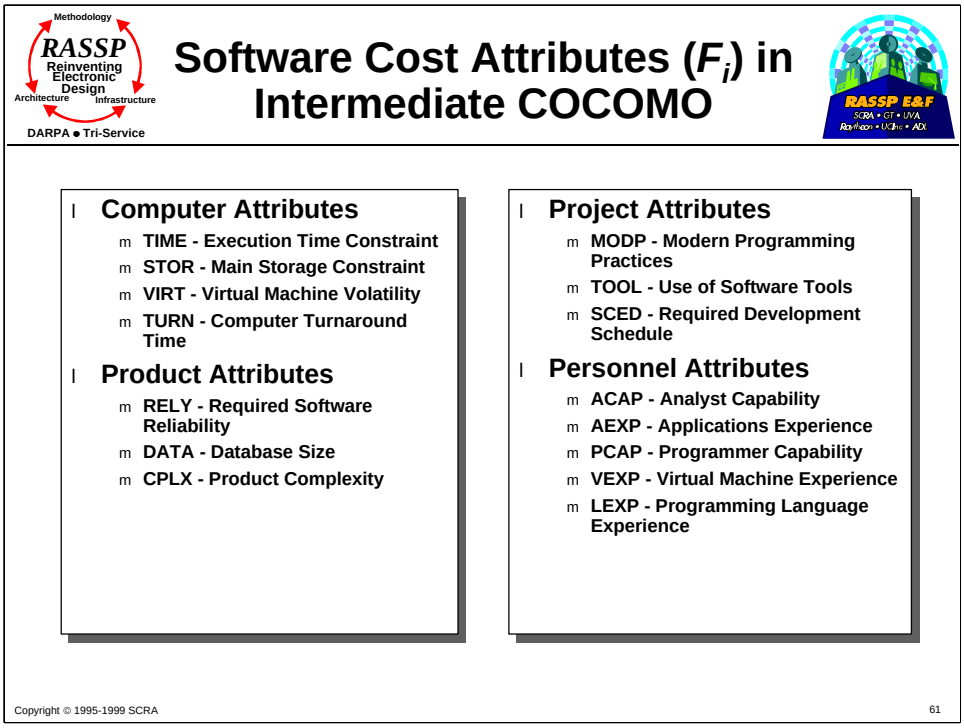
- The embedded mode of development is most applicable for the software costing for embedded digital systems.
- The focus will be on this mode of development for making embedded systems design trade-offs.
- The reader may wish to determine the applicability of these models to the class of products being developed.



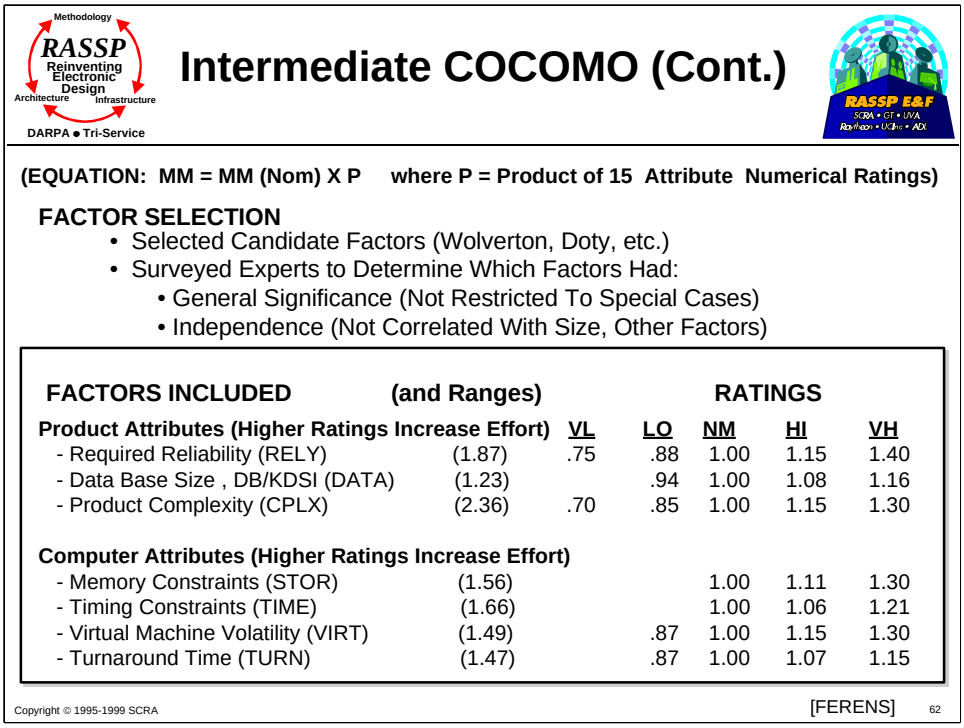
- Basic COCOMO:
  - Good for quick, early, rough order of magnitude estimates of software costs.
  - Accuracy is very limited, because the model lacks factors to account for differences in hardware constraints, personnel quality and experience, use of modern tool and techniques, and other project attributes known to have significant influence on software costs.
  - Basic COCOMO estimates its own database within a factor of 1.3 of the actuals only 29% of the time and a factor of 2 of the actuals only 60% of the time.
  - Accuracy is not good enough for making design trade-offs.
- Detailed COCOMO:
  - Used during detailed design to refine cost estimates
  - Employs a three level hierarchical decomposition of the software product (module level, subsystem level, system level).
  - Uses phase sensitive effort multipliers which accurately reflect the effect of the cost drivers on the phase distribution of effort
  - Cost models are used to make design trade-offs during the early stages of design. The level of information required by this model will not be available during conceptual design.
- Intermediate COCOMO:
  - Incorporates an additional 15 cost drivers to account for much of the software project cost variation.
  - Estimates are within 20% of the project actuals 68% of the time with respect to the COCOMO database.
  - There is enough information available about the development environment and software size (KDSI) to effectively use this model during the early stages of product design.
  - This model will be explored in more detail.



- The embedded model intermediate COCOMO includes integration and test costs, and is more involved than the basic model.
- The general form of equations is still similar across the COCOMO models.



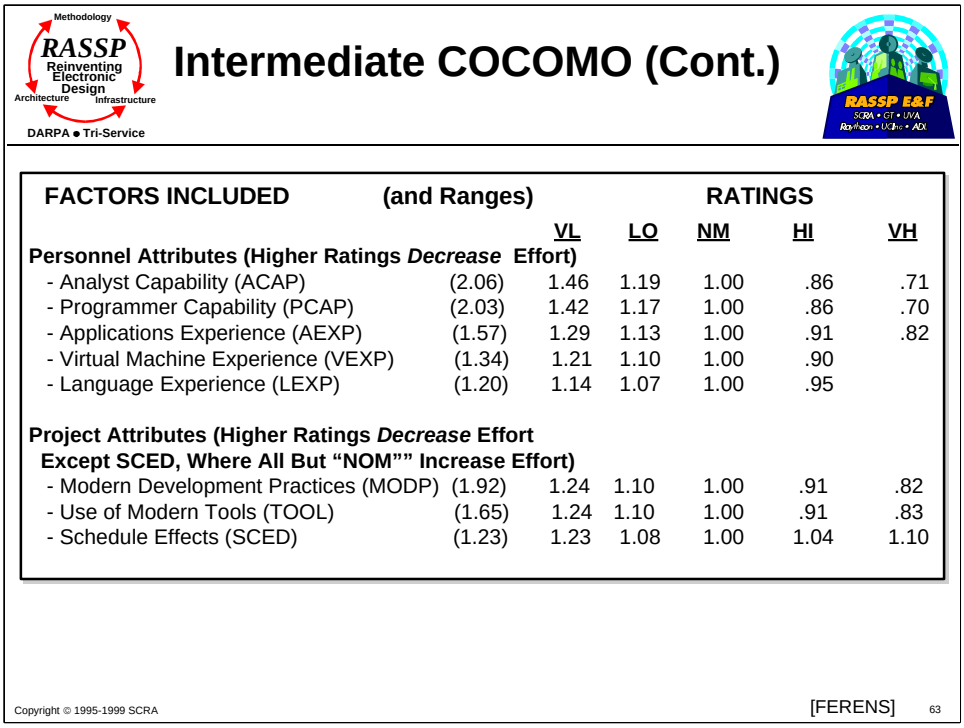
- These attributes allow customization of the cost model to company-specific, product-specific, organization skill-specific, and project specific attributes, usually through a multiplicative factor that varies around the value 1.0.



- **TIME-** the cost driver rating is a function of the degree of execution time constraint imposed upon a software subsystem. The rating is expressed in terms of the percentage of available execution time expected to be used by the subsystem and any other subsystems consuming the execution time resource.
- **STOR** - the rating is a function of the degree of main storage constraint imposed on a software subsystem. Main storage refers to main memory or DRAM storage. The rating is expressed in terms of the percentage of main storage expected to be used by the subsystem and any other subsystems consuming the main storage resources.
- **VIRT -** the rating is a function of the level of volatility of the virtual machine underlying the subsystem to be developed. For a given software subsystem, the underlying virtual machine is the complex of hardware and software (OS, DBMS, etc.) that the subsystem calls upon to accomplish its tasks.
- **TURN** - rating is a function of the turnaround time for computers in which software is being developed on after a failure.
- **RELY** - refers to the level of software reliability required. For example, RELY will receive a low rating if the effect of a software failure is simply the inconvenience incumbent upon the developers to fix the fault. However, for a very high rating, the effect of a software failure can be the loss of human life.
- **DATA** - refers to the amount of data to be assembled and stored in nonmain storage (that is, tapes, disks, etc.) by the time of software acceptance. The ratings are defined in terms of the ratio between the data base size in bytes and the software size.
- **CPLX** - increasingly complex operations correspond to the module complexity ratings of very low, low, nominal, high, very high, and extra high. The ratings are a function of the types of operations performed by the module: control, computation, device-dependent, or data management operations.

[Boehm81]

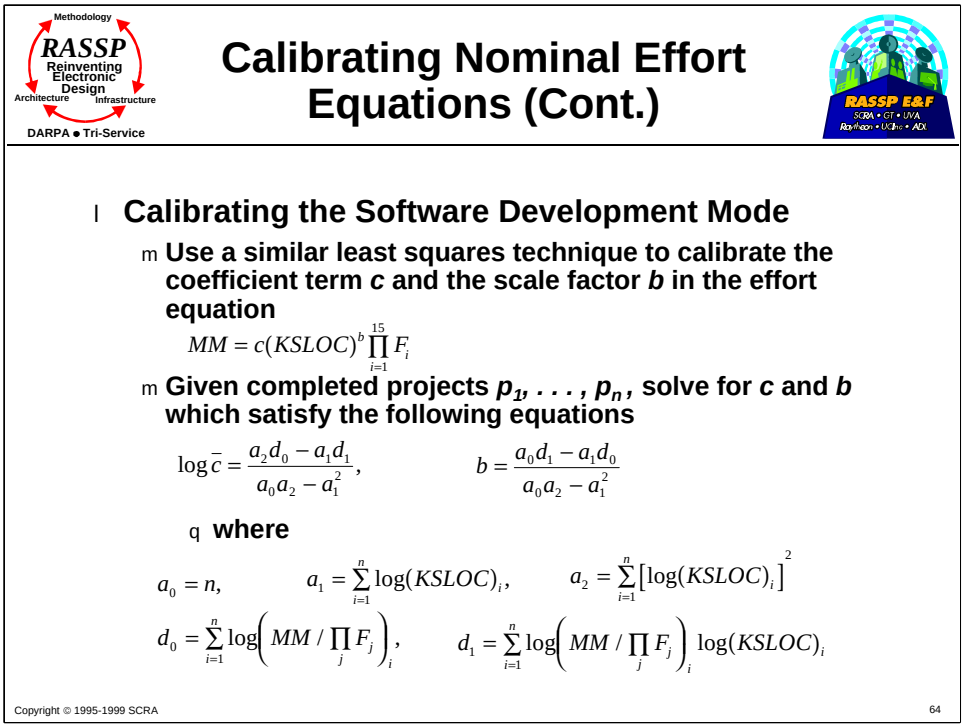
[COCOMO]



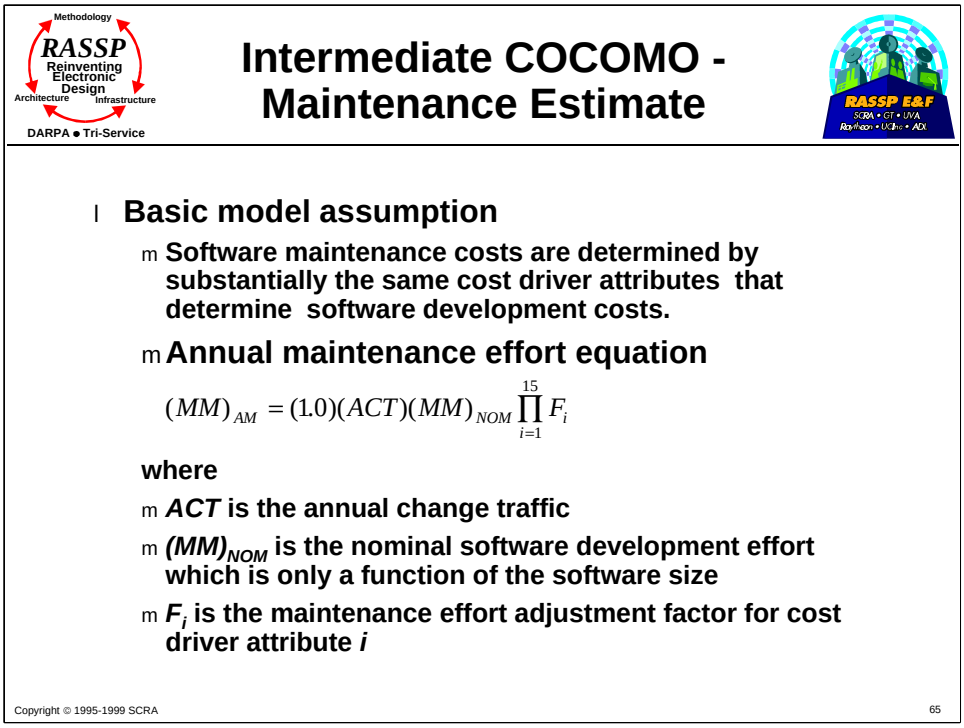
- **MODP**- the rating is a function of the degree to which modern programming practices (e.g. top-down requirements analysis and design, reuse, object-oriented methodologies, etc.) are used in developing software.
- **TOOL** - the rating is a function of the degree to which software tools are used in developing the software subsystem. Tool usage can range from basic microprocessor tools (assemblers, linkers, etc.) to advanced maxicomputer tools (instruction set simulators, cross compilers, integrated development environments, etc.)
- **SCED** - the rating is a function of the level of schedule constraint imposed on the project team developing a software subsystem. The ratings are defined in terms of the percentage of schedule stretchout or acceleration with respect to a nominal schedule for a project requiring a given amount of effort. A schedule acceleration below 75% (very low rating) of nominal is considered impossible by COCOMO.
- **ACAP** - the rating is a function of the level of capability of the software analysts. For a very low rating, a software analyst would problem rate in 15th percentile. In turn, an analyst with a very high rating would rate in the 90 percentile with respect to other software analysts.
- **AEXP** - the rating is a function of the level of experience the project team has with the applicaton. Less than 4 months would receive a very low rating. Greater than 12 yrs would receive a very high rating.
- **PCAP** - the rating is a function of the level of capability of the programming team.
- **VEXP** - the rating is a function of the level of experience the project team has with the virtual machine. A very low rating corresponds to no experience. A very high rating corresponds to greater than 2 years experience.
- **LEXP** - the rating is a function of the level of experience the programming team has the programming language.

[Boehm81]

[COCOMO]



- The constant-term calibration process is much more stable than the development mode recalibration process.
- Cautions which should be observed when recalibrating COCOMO, particularly recalibrating the development mode
  - Make sure the project data are as consistent as possible.
  - If the project data represent different modes, perform a separate recalibration for each mode.
  - If the sample size of project data is less than 10 projects, pick a standard COCOMO development mode and recalibrate its constant term, rather than recalibrating an overall development mode.
  - Make sure that the projects used for calibration are representative of the projects whose costs will be estimated with the recalibrated model.



- Software maintenance includes:
  - Redesign and redevelopment of smaller portions (less than 50% new code) of an existing software product
  - Design and development of smaller interfacing software packages which require some redesign (of less than 20%) of the existing software product
  - Modification of the software product's code, documentation, or data base structure
- Software maintenance excludes:
  - Major redesign and redevelopment (more than 50% new code) of a new software product performing substantially the same functions
  - Design and development of a sizable (more than 20% of the source instructions comprising the existing product) interfacing software package which requires relatively little redesign of the existing product
  - Data processing system operations, data entry, and modification of values in the data base

SOFTWARE SUPPORT COSTS - COCOMO

COCOMO-M

- Equation: $MM_A = MM_{NOM}$ x ACT x P_M
  - MM_A is Annual Person-Months
  - MM_{NOM} is Nominal Person-Months (From COCOMO)
  - ACT is Annual Change Traffic (% Code Changed per Year)
  - P_M is Product of Maintenance Multipliers (SCED Not Used; RELY and MODP Values Change)

REVIC

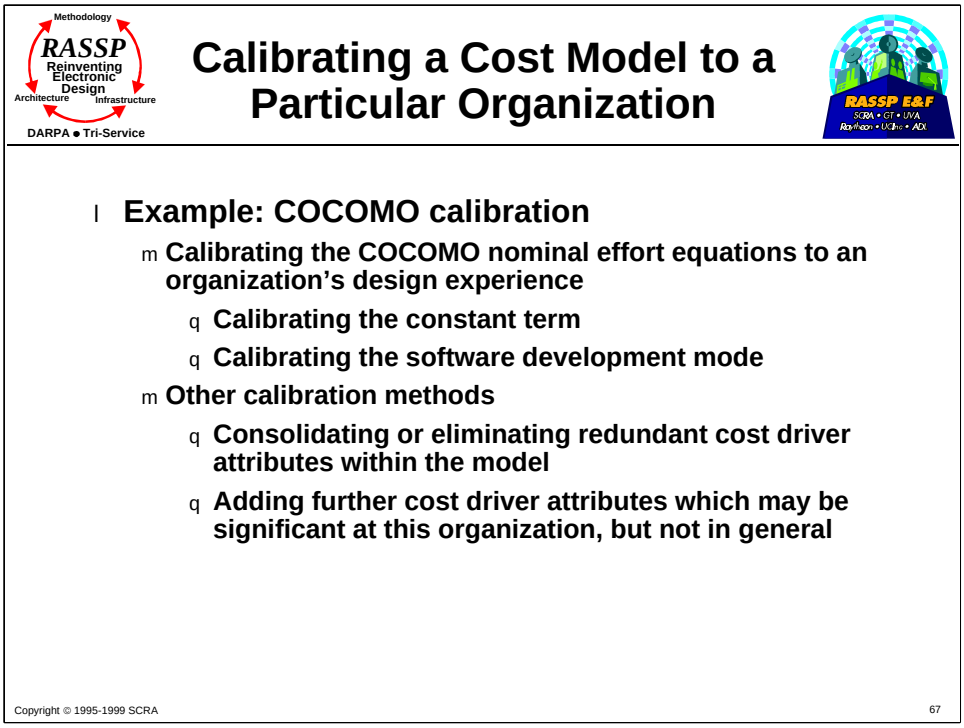
- Like COCOMO-M, Except (ONLY) 15-Year Support Costs
  - Highest In Years 1 and 2, Then Steady-State

COCOMO II

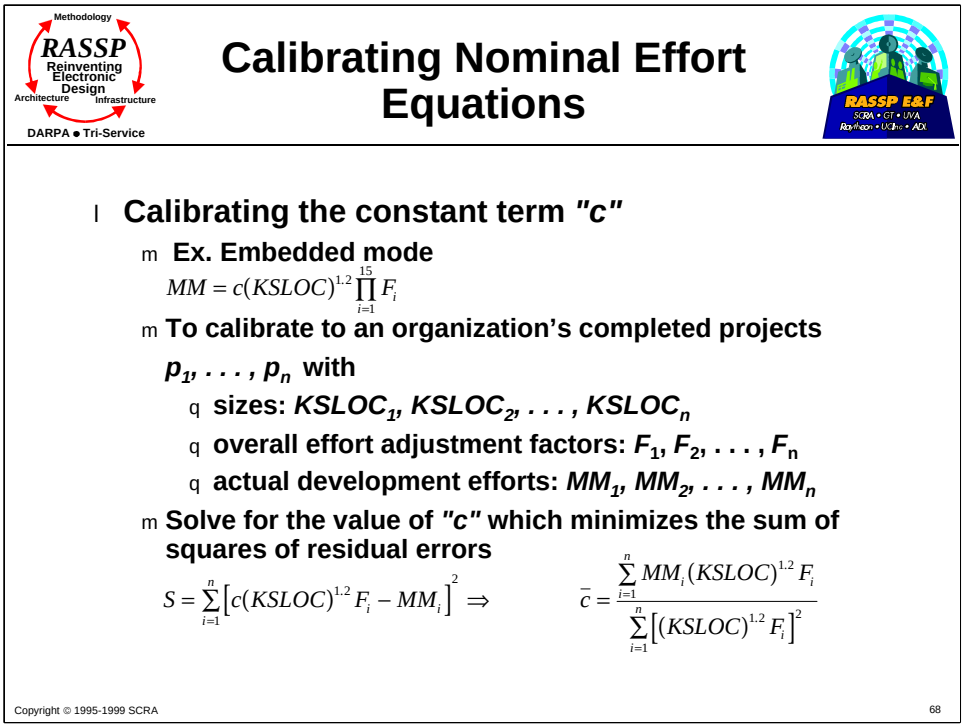
- Equation is $MM_M = A \times Size_M \times P_M$
  - MM_M is "Maintenance Man Months"
  - $Size_M$ = (Base Code Size) x MCF x MAF
  - MCF = (Size Added + Size Modified) / Base Code Size
  - MAF = 1 + ((SU/100)(UNFM))

[FERENS] 66

[COCOMO]

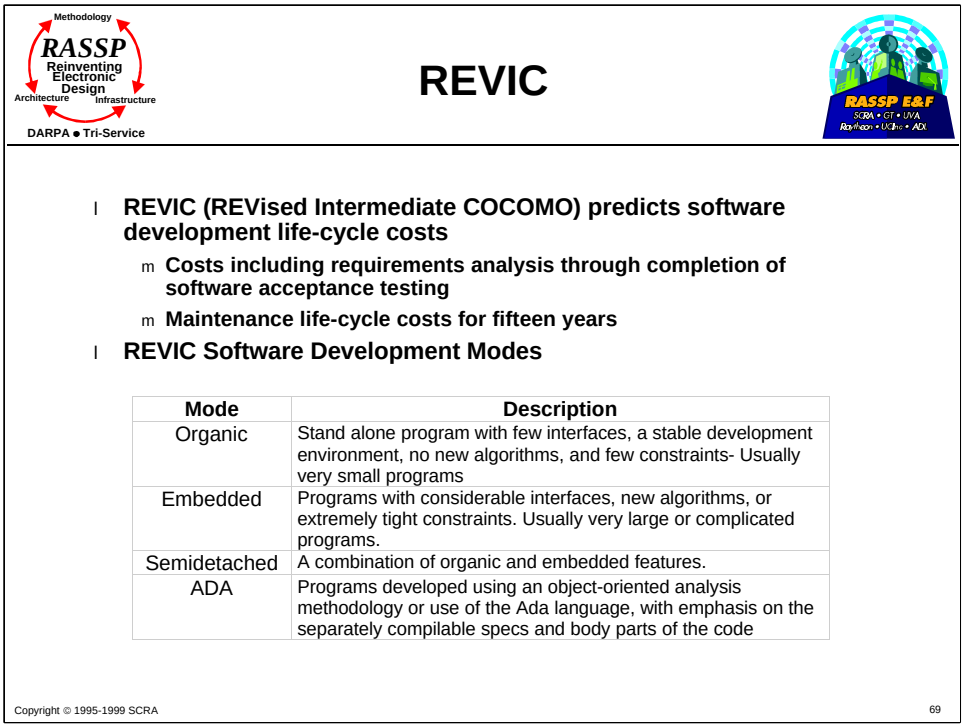


- The same calibration technique can be used for calibration to a specific application.

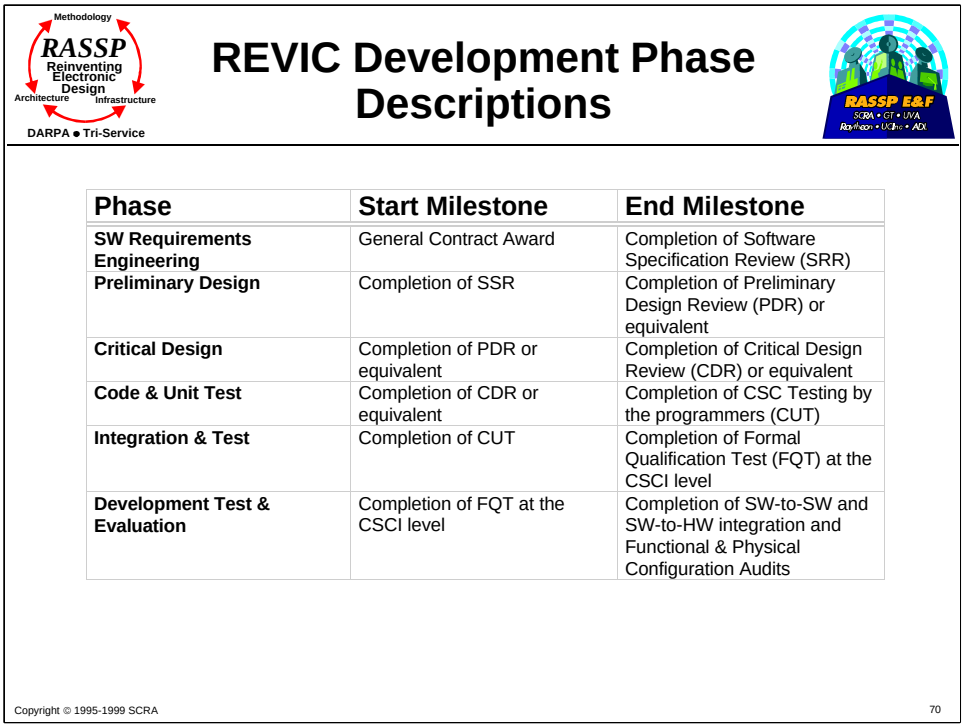


Reasons for calibrating the nominal effort equations

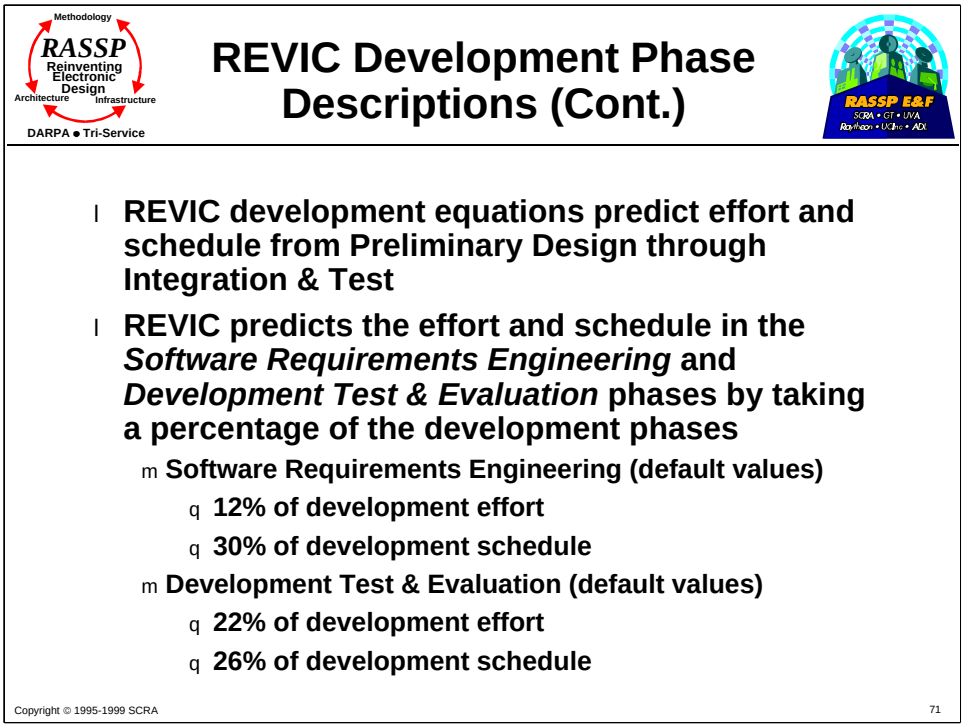
- An organization may consistently judge the COCOMO cost driver attribute ratings by different standards than were used in calibrating COCOMO.
- An organization may employ consistently different definitions of "delivered", "source instructions," "development," or "man-months" than those used in COCOMO.
- An organization's usual development mode may be somewhere in between the standard COCOMO development modes, in which case a special nominal effort equation can be calibrated to the organization's experience.



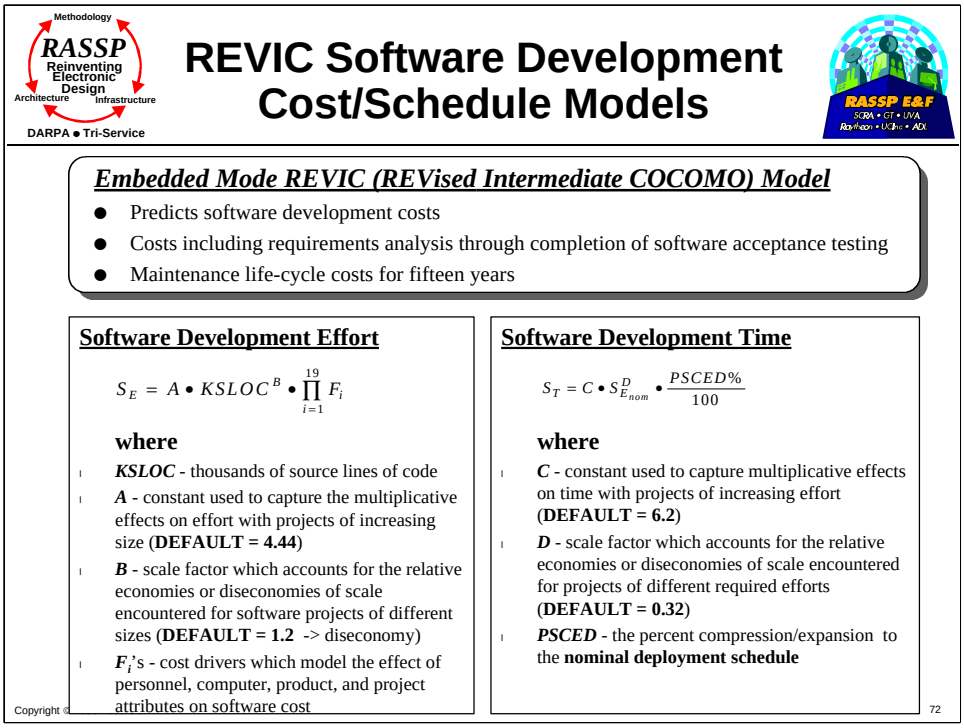
- As with intermediate COCOMO, the embedded mode of software development is most applicable to embedded systems design.
- REVIC is suitable for military systems because it also includes lifetime costs of the product.



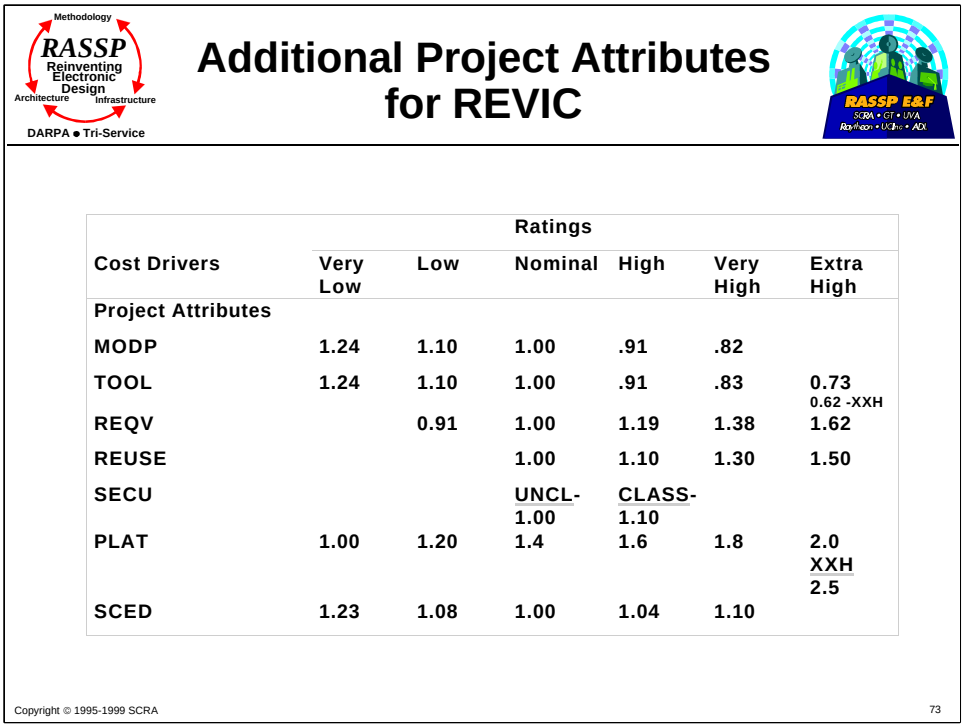
- These phases are consistent with the development phases of software in current practice as described in the RASSP Methodology module (Module 29).



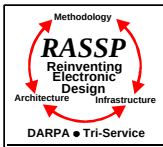
- Default values of REVIC rely on past historical information derived from several Air Force projects in the 1980s. An organization can refine these models by over-riding the default values.



- The REVIC Embedded Mode model is an extension of the underlying COCOMO (Intermediate model) with extensions to include lifecycle costs and also includes some addition parameters, such as reuse.



- Cost Attributes not included in COCOMO:
  - REQV - the rating refers to the level of requirements volatility in a project.
  - REUSE - the rating refers to level of required reusability of the software being developed. (Developing reusable software)
  - SECU - the rating refers to whether the software is being developed for a classified security application or not.
  - PLAT - the rating refers to the level of specification in the application platform. A very low rating refers to ground systems, where a extra high rating refers to manned space systems.
  - For more information, see [AF95].



REVIC Maintenance Phase Description

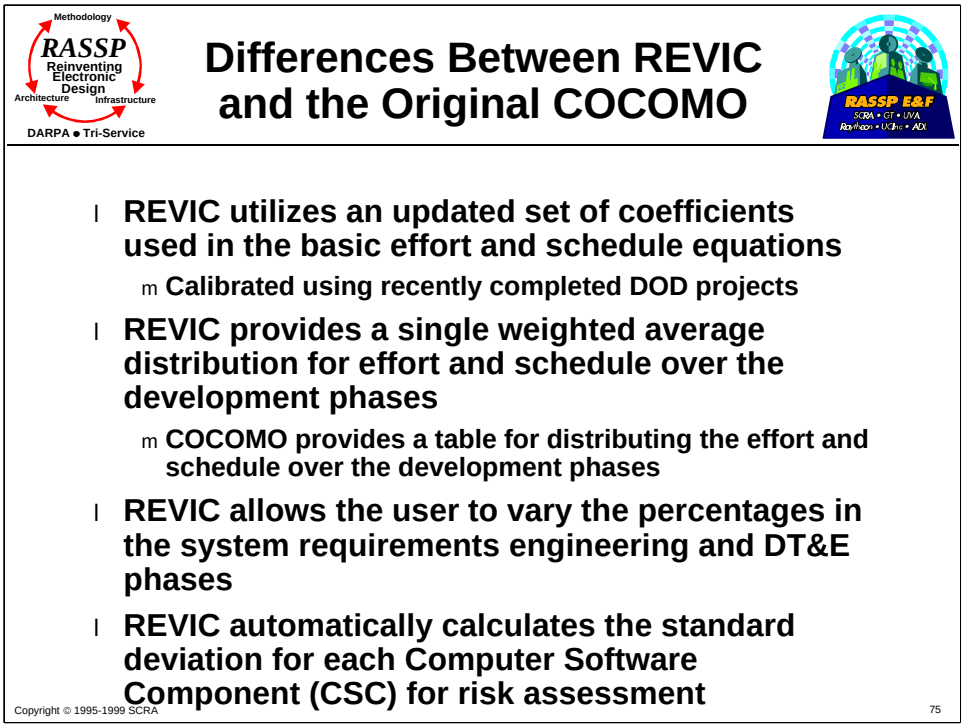
- **REVIC estimates the maintenance of a software product over a fifteen year period**
- **REVIC assumes the presence of a transition period after the delivery of the software**
  - **Residual errors are found before reaching a steady state condition providing a declining, positive delta to the ACT during the first three years**
  - **Maintenance Equation**

$$MM_{am} = (MM_{nom})\Pi(MF_i)ACT$$

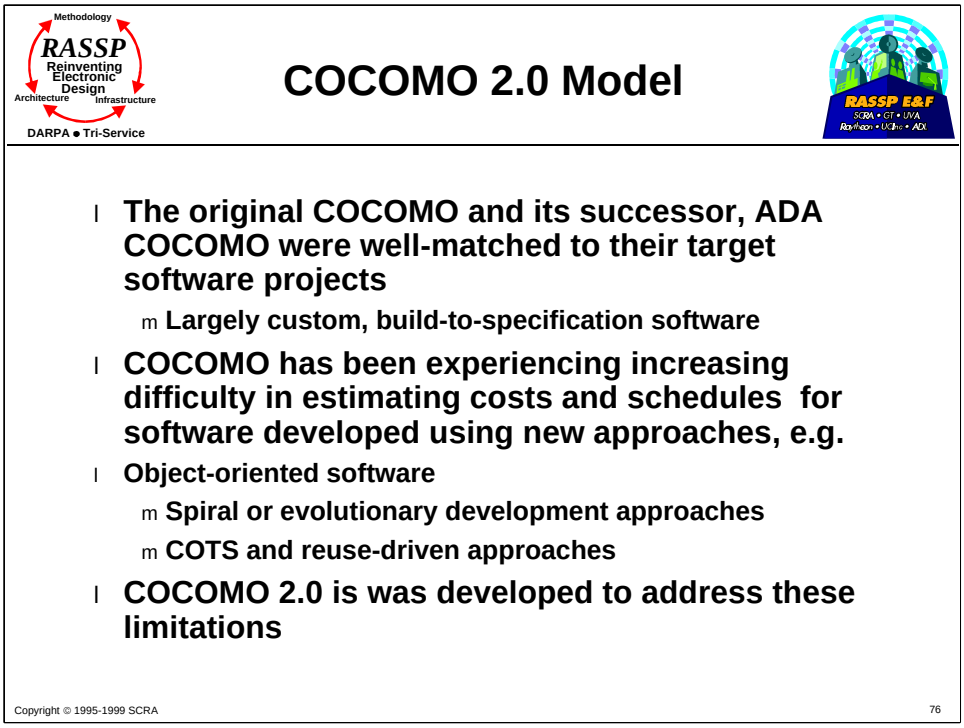
where

  - ***MM*$_{nom}$ is the nominal development effort**
  - ***ACT* is the annual change traffic (default value = 15%)**
  - ***MF*$_i$ is the set of maintenance adjustment factors**

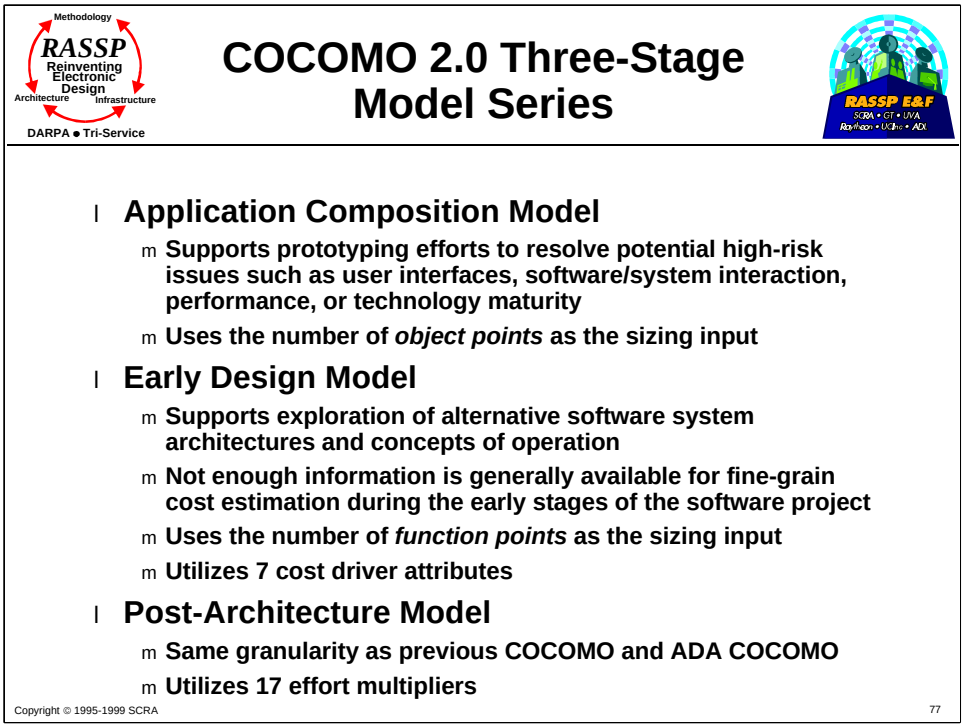
- The annual change traffic describes the changes in the product each year attributed to maintenance.



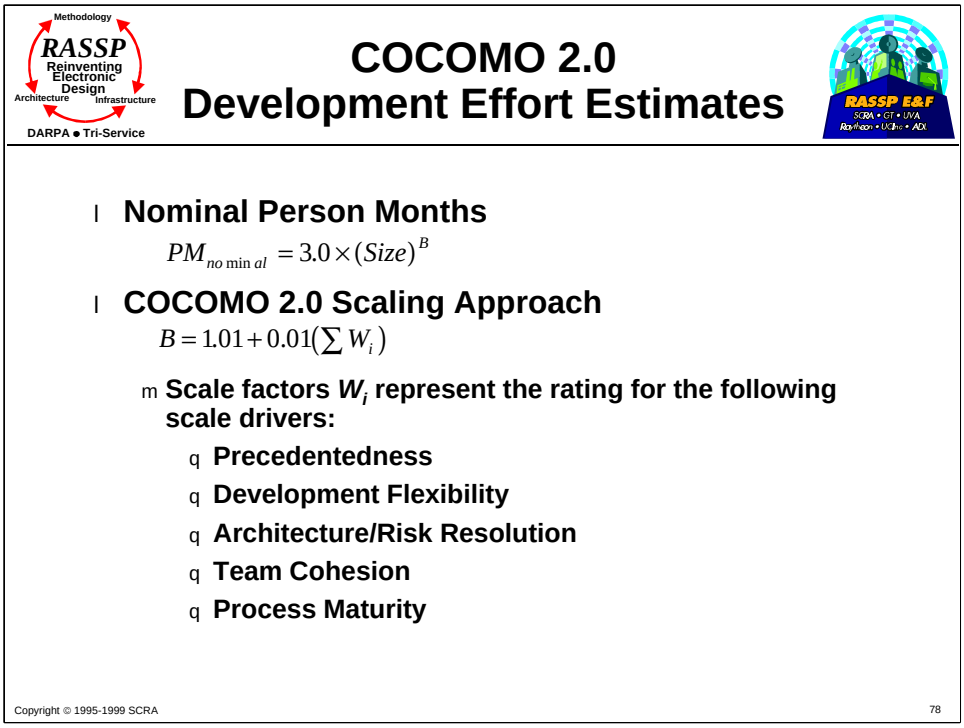
- DT & E refers to development test & evaluation.
- REVIC's coefficients have been calibrated using recently completed DOD projects.
  - Least squares minimization techniques were used to determine the coefficients as in COCOMO.
- REVIC provides estimates within +/- 5% of the projects in its database.



- The limitations of COCOMO are described in this slide, primarily due to adoptions of newer development methodology, such as object oriented design or autocoding and code generation. For more information, see [BOEHM95].



- The earliest phases or spiral cycles uses Application Composition capabilities for prototyping.
- The next phases or spiral cycles will generally involve exploration of software architectural alternatives or incremental development strategies which is support by the Early Design model.
- Once a software lifecycle architecture has been developed, more accurate information will be available on cost driver inputs, thereby enabling more accurate estimates. The Post-Architecture model is used to support this stage.
- Object points are a count of the screens, reports and third-generation-language modules developed in the application, each weighted by a three-level (simple, medium, difficult) complexity factor.
- Early Design Model Cost Drivers:
  - Personnel Capability (PERS)
  - Product Reliability and Complexity (RCPX)
  - Required Reuse (RUSE)
  - Platform Difficulty (PDIF) - a combined effort multiplier for the hardware architectural attributes. (execution time and main storage constraints and platform volatility)
  - Personnel Experience (PREX)
  - Facilities (FCIL) - combination of software tool usage and multisite development cost driver attributes
  - Schedule constraint (SCED)
- The Post-Architecture model is most useful making design trade-offs because it allows for differentiation of execution time constraints and memory constraints, unlike the early design model, thereby allowing the designer to trade-off the number of processors and amount of memory.



- The software size is expressed in thousands of source lines of code.

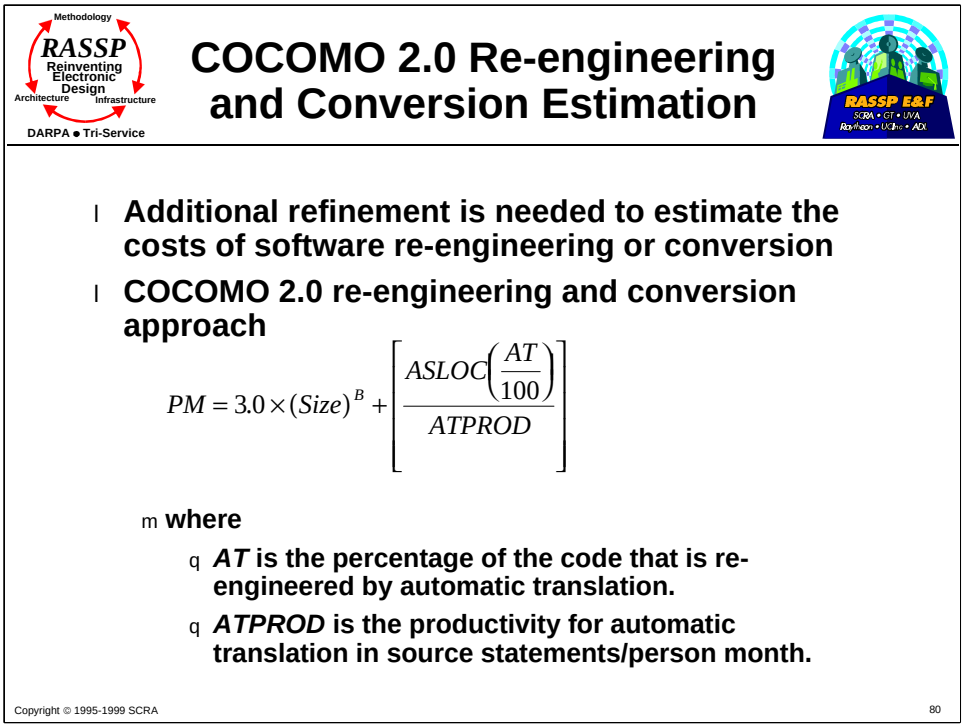
Breakage Percentage

- **COCOMO 2.0 uses a breakage percentage to adjust the effective size of the product based on the requirements volatility in a project.**
- **Breakage Percentage**
  - **The percentage of code thrown away due to requirements volatility**
- **COCOMO 2.0 adjustment (not used in Application Composition)**

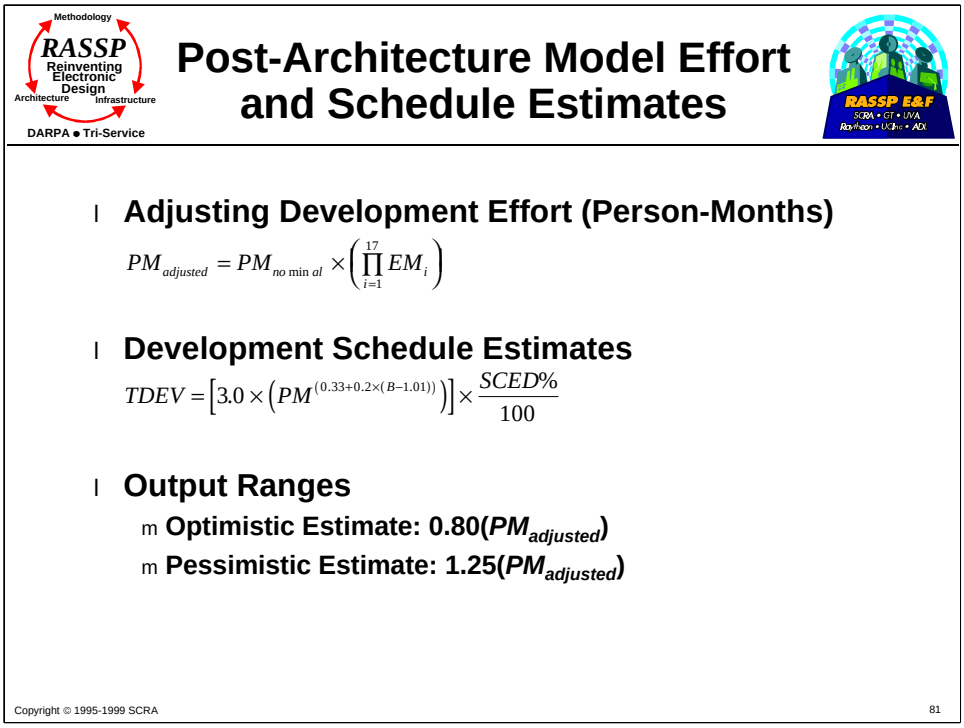
$$PM_{no\,min\,al} = 3.0\left[\left(\frac{1+BRAK}{100}\right)Size\right]^{B}$$

  - **where**
    - ***BRAK* is the breakage percentage**

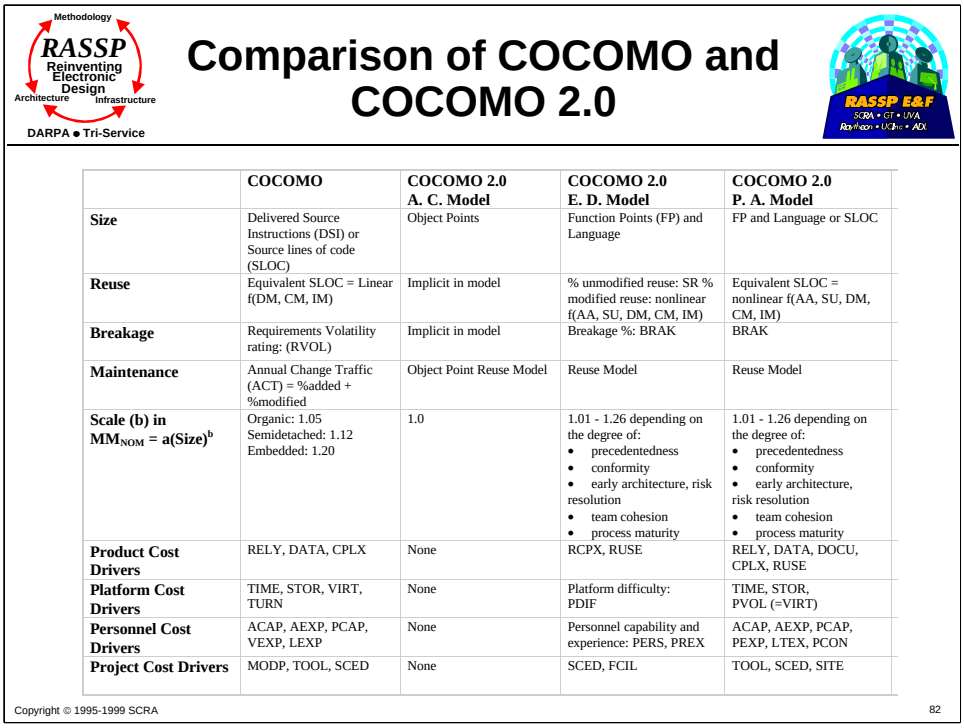
- COCOMO 2.0 assumes that project requirements can change over the lifetime of the product, and this is factored into the lifetime costs.



- The major difference between reuse and re-engineering and conversion is the efficiency of automated tools for software restructuring.
- Automatic translation can lead to very high values for the percentage of code modified, but very little corresponding effort, thereby requiring a separate approach to deal with this type of reuse.

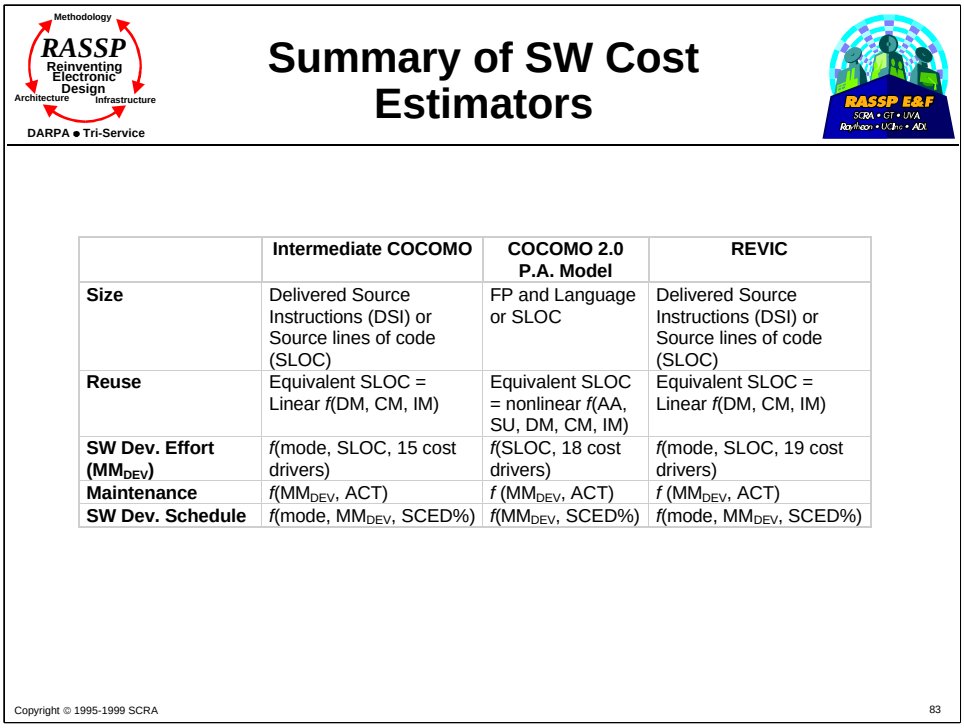


- Post-Architecture Model Effort Multipliers
  - Required Software Reliability (RELY)
  - Data Base Size (DATA)
  - Product Complexity (CPLX)
  - Required Reusability (RUSE)
  - Documentation match to life cycle needs (DOCU)
  - Execution Time Constraint (TIME)
  - Main Storage Constraint (STOR)
  - Platform Volatility (PVOL)
  - Analyst Capability (ACAP)
  - Programmer Capability (PCAP)
  - Applications Experience (AEXP)
  - Platform Experience (PEXP)
  - Language and Tool Experience (LTEX)
  - Personnel Continuity (PCON)
  - Use of Software Tools (TOOL)
  - Multisite Development (SITE)
  - Required Development Schedule (SCED)



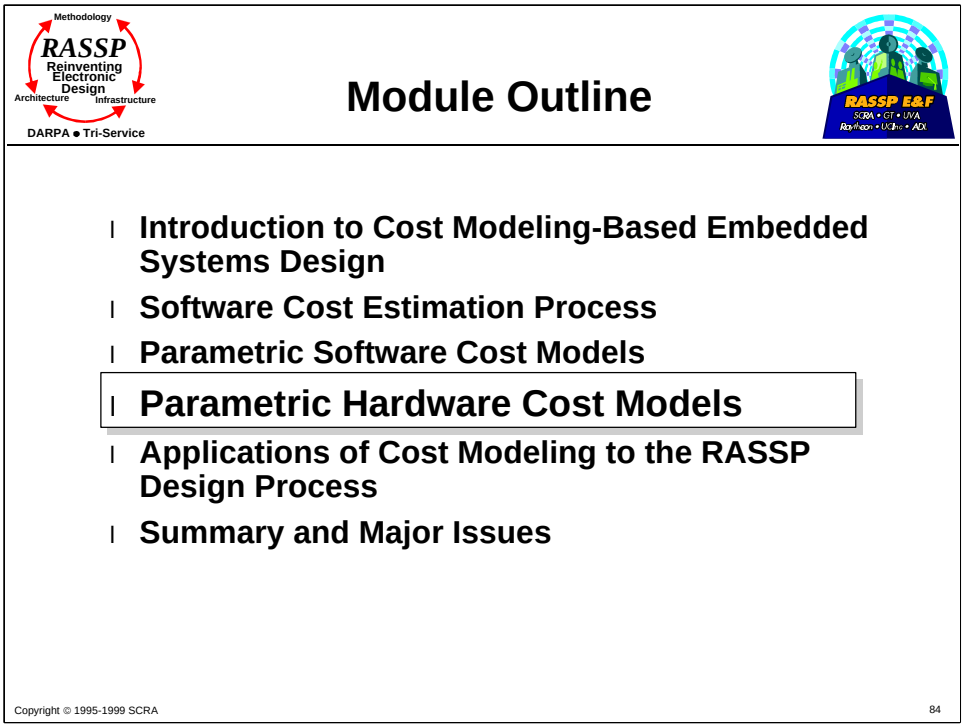
	COCOMO	COCOMO 2.0 A. C. Model	COCOMO 2.0 E. D. Model	COCOMO 2.0 P. A. Model
Size	Delivered Source Instructions (DSI) or Source lines of code (SLOC)	Object Points	Function Points (FP) and Language	FP and Language or SLOC
Reuse	Equivalent SLOC = Linear f(DM, CM, IM)	Implicit in model	% unmodified reuse: SR % modified reuse: nonlinear f(AA, SU, DM, CM, IM)	Equivalent SLOC = nonlinear f(AA, SU, DM, CM, IM)
Breakage	Requirements Volatility rating: (RVOL)	Implicit in model	Breakage %: BRAK	BRAK
Maintenance	Annual Change Traffic (ACT) = %added + %modified	Object Point Reuse Model	Reuse Model	Reuse Model
Scale (b) in $MM_{NOM} = a(Size)^b$	Organic: 1.05 Semidetached: 1.12 Embedded: 1.20	1.0	1.01 - 1.26 depending on the degree of: • precedentedness • conformity • early architecture, risk resolution • team cohesion • process maturity	1.01 - 1.26 depending on the degree of: • precedentedness • conformity • early architecture, risk resolution • team cohesion • process maturity
Product Cost Drivers	RELY, DATA, CPLX	None	RCPX, RUSE	RELY, DATA, DOCU, CPLX, RUSE
Platform Cost Drivers	TIME, STOR, VIRT, TURN	None	Platform difficulty: PDIF	TIME, STOR, PVOL (=VIRT)
Personnel Cost Drivers	ACAP, AEXP, PCAP, VEXP, LEXP	None	Personnel capability and experience: PERS, PREX	ACAP, AEXP, PCAP, PEXP, LTEX, PCON
Project Cost Drivers	MODP, TOOL, SCED	None	SCED, FCIL	TOOL, SCED, SITE

- All the models described so far are compared in this table. For more detailed information, see [BOEHM95] and [COCOMO].

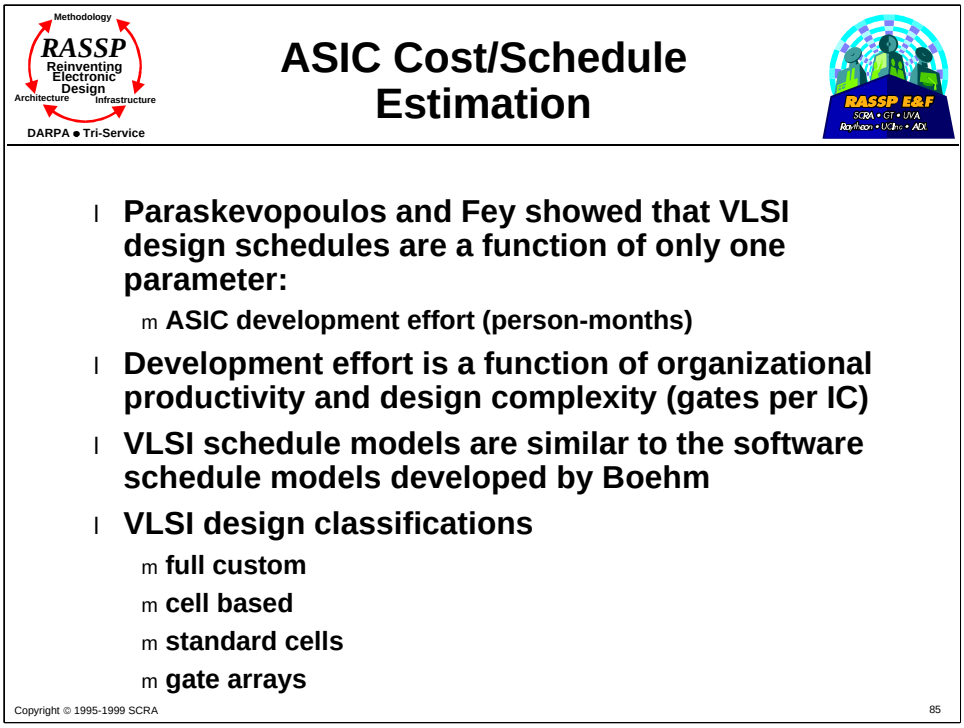


	Intermediate COCOMO	COCOMO 2.0 P.A. Model	REVIC
Size	Delivered Source Instructions (DSI) or Source lines of code (SLOC)	FP and Language or SLOC	Delivered Source Instructions (DSI) or Source lines of code (SLOC)
Reuse	Equivalent SLOC = Linear *f*(DM, CM, IM)	Equivalent SLOC = nonlinear *f*(AA, SU, DM, CM, IM)	Equivalent SLOC = Linear *f*(DM, CM, IM)
SW Dev. Effort (MM_{DEV})	*f*(mode, SLOC, 15 cost drivers)	*f*(SLOC, 18 cost drivers)	*f*(mode, SLOC, 19 cost drivers)
Maintenance	*f*(MM_{DEV}, ACT)	*f* (MM_{DEV}, ACT)	*f* (MM_{DEV}, ACT)
SW Dev. Schedule	*f*(mode, MM_{DEV}, SCED%)	*f*(MM_{DEV}, SCED%)	*f*(mode, MM_{DEV}, SCED%)

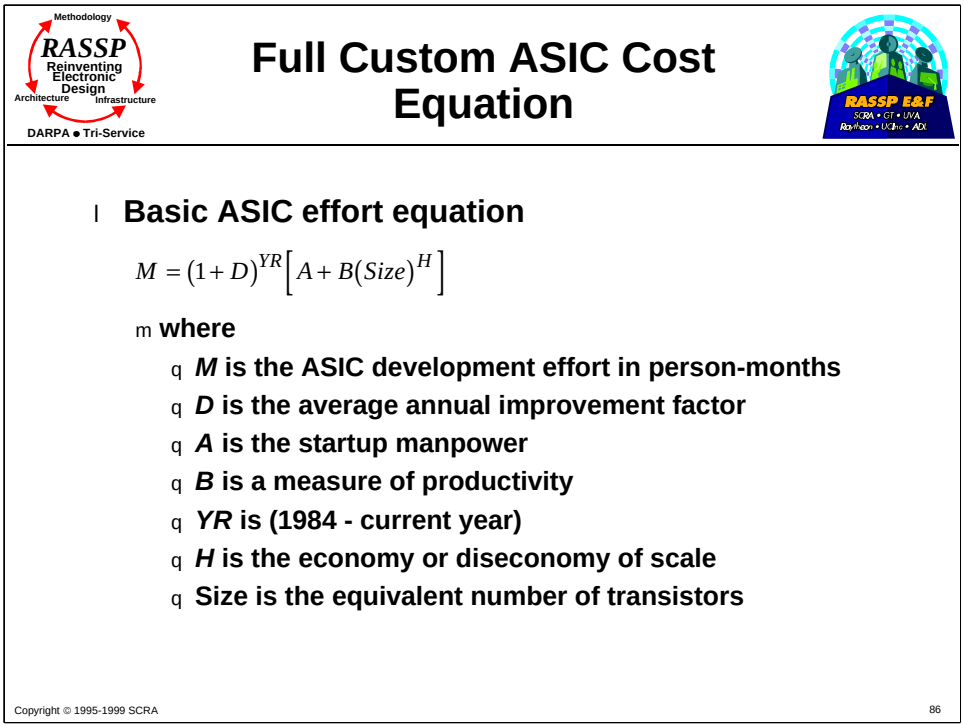
- Continuation of the comparison of cost models.



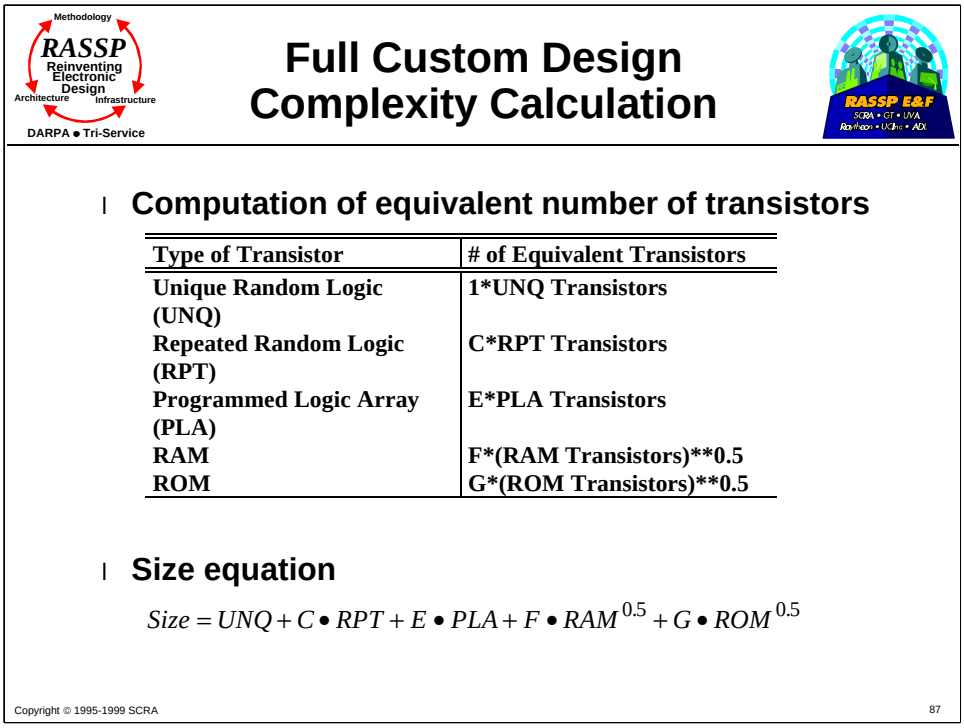
- Work has been done in the area of hardware cost estimation as well, but is not as comprehensive or general as the research work done in the area of software cost modeling. We survey some well-known hardware cost models.



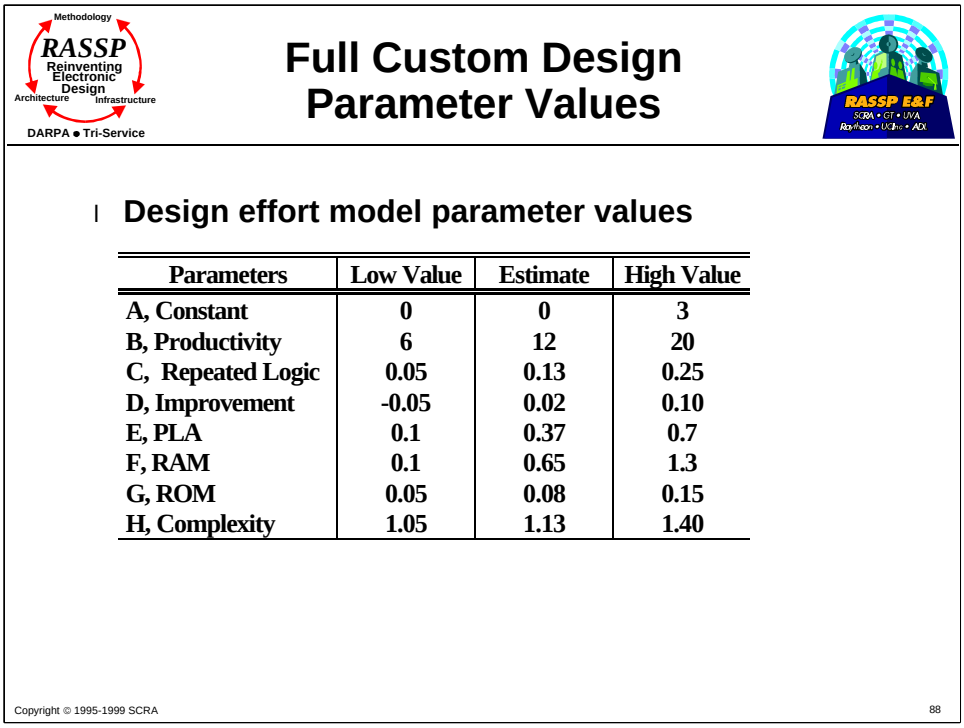
- The design effort and schedule include the following phases
  - logic design and verification, including chip architecture;
  - circuit design and verification, including timing and simulation;
  - layout and verification; and
  - test design, testing, and debugging
- VLSI classifications:
  - Full custom: the device is designed at the transistor level. Transistor performance and area are optimized
  - Cell based: the circuit is partially built of predefined blocks of cells. Most are newly created for the current design.
  - Standard cells: ideally, the device is designed from predefined libraries of cells. Only cell interconnect is optimized.
  - Gate arrays: the device is designed from gates and predefined cells. Only interconnect is optimized.
- For more information, see [PF87].



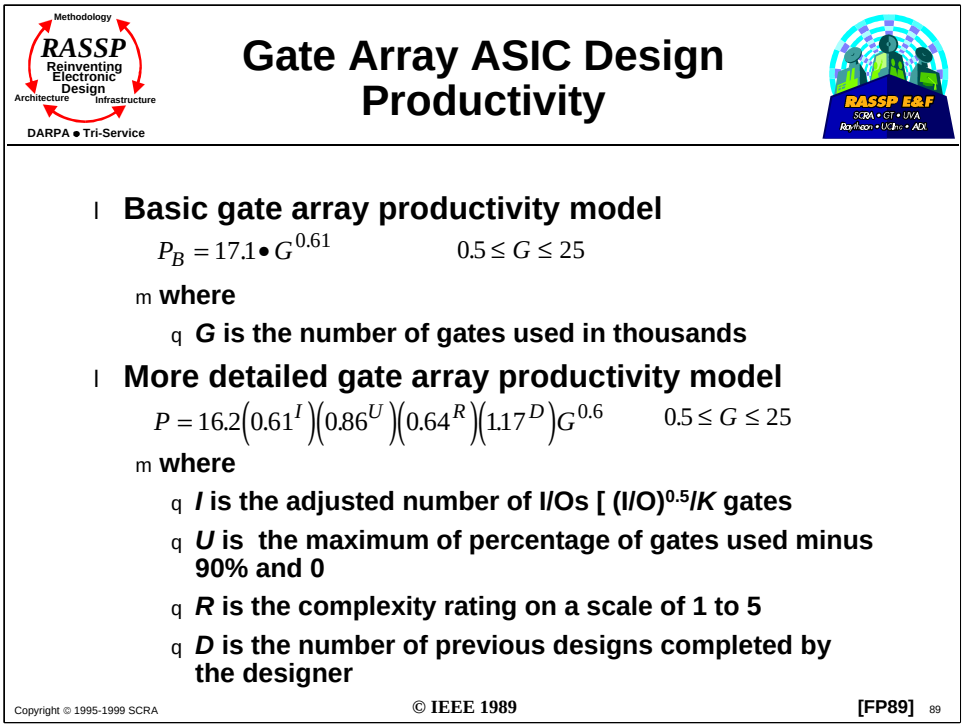
- Both productivity and requirements changes over time
  - the productivity will increase due to new tools, experience, reuse, etc.
  - the productivity may decrease due to higher performance, higher reliability
- The model adjusts productivity from year to year with parameters described above.
- For more information, see [FEY85].



- An equivalent transistor is a measure of the time required to design a transistor.
- The measure is based on the fact that various types of transistors require different amounts of design time.



- The parameter values above are estimates from data collected from both merchant market suppliers and in-house captives for N-MOS and C-MOS logic and memory design efforts between 1976 and 1983.
- The size of the designs ranged from 1000 to 300,000 transistors
- The relatively small number of parameters requires changes of parameter values where design requirements, designer experience, technology, or other circumstances differ from the sample average.
- Therefore, estimates of low and high values are also given.



- Productivity is measured in gates per person-week in the models.
- These models were derived from data collected on 70 designs from five major corporations during the period 1983-1988.
- The basic model indicates that productivity increases with the number of gates.
  - *This is fundamentally different from full-custom designs, where productivity decreases with the number of gates*
  - This increase in productivity with size is due to improvements in CAD tools and libraries to support gate array designs
- The basic model poorly correlates with actuals.
- The detailed more provides a much better fit to actual values. The correlation coefficient is 0.85.

Gate Array Design Cost

- **Gate array development effort (person-weeks)**

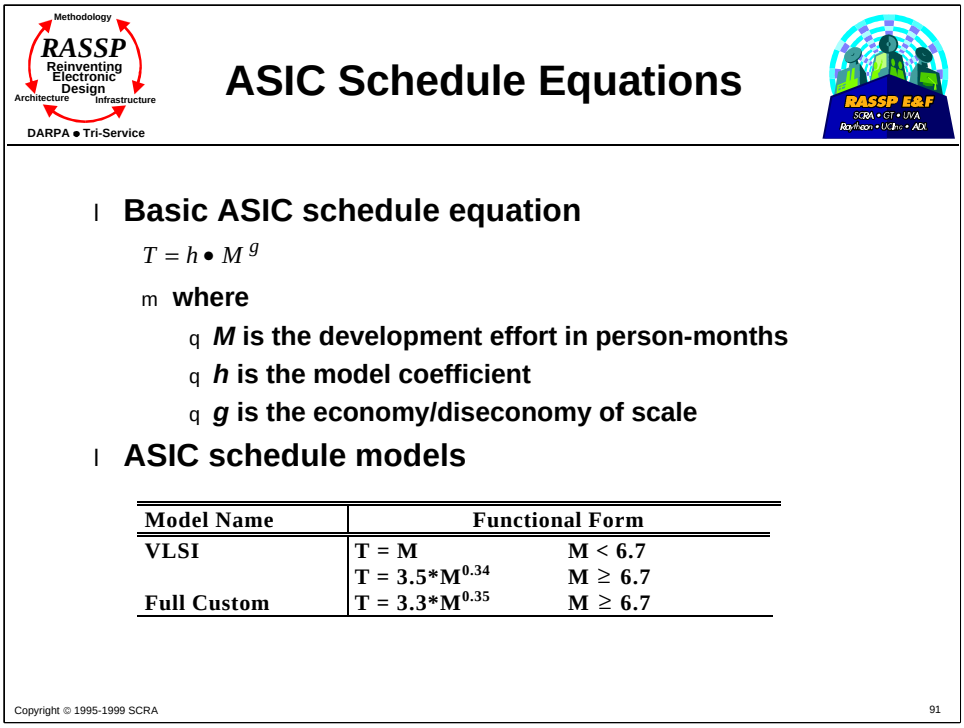
$$M = \frac{200 \bullet G}{P} \qquad 0.5 \le G \le 25$$

  - **where**
    - ***M*** **is the development effort in person-months**
    - ***P*** **can be the detailed or basic productivity values**
- **Example:**
  - **Using basic productivity model**

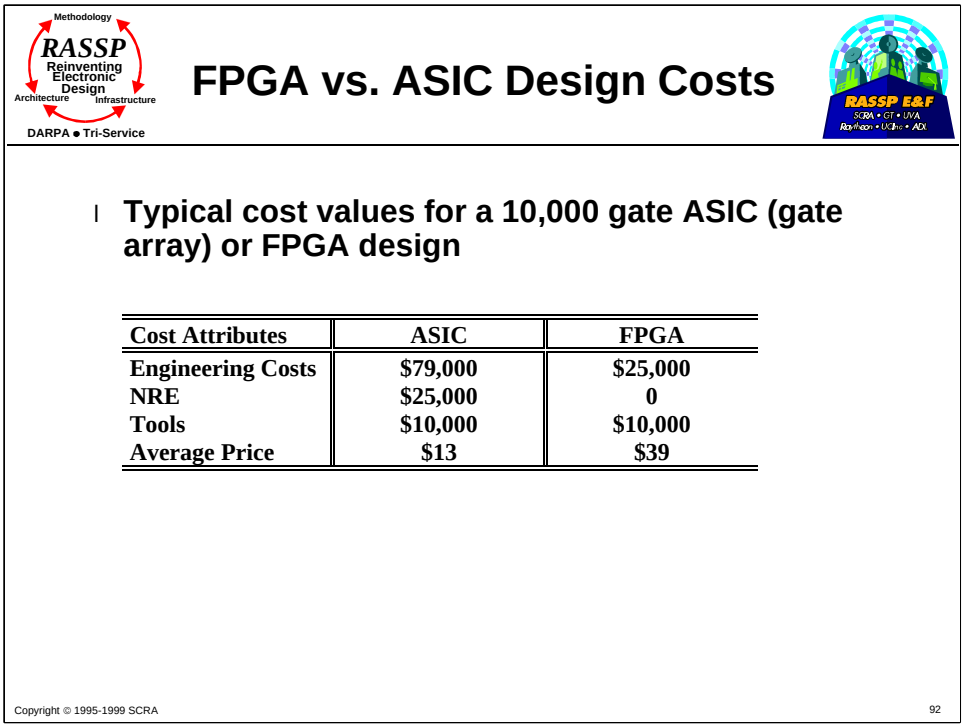
$$M = 11.7 \bullet G^{0.39}$$

    - **Development effort is proportional to an exponent of the number of gates**

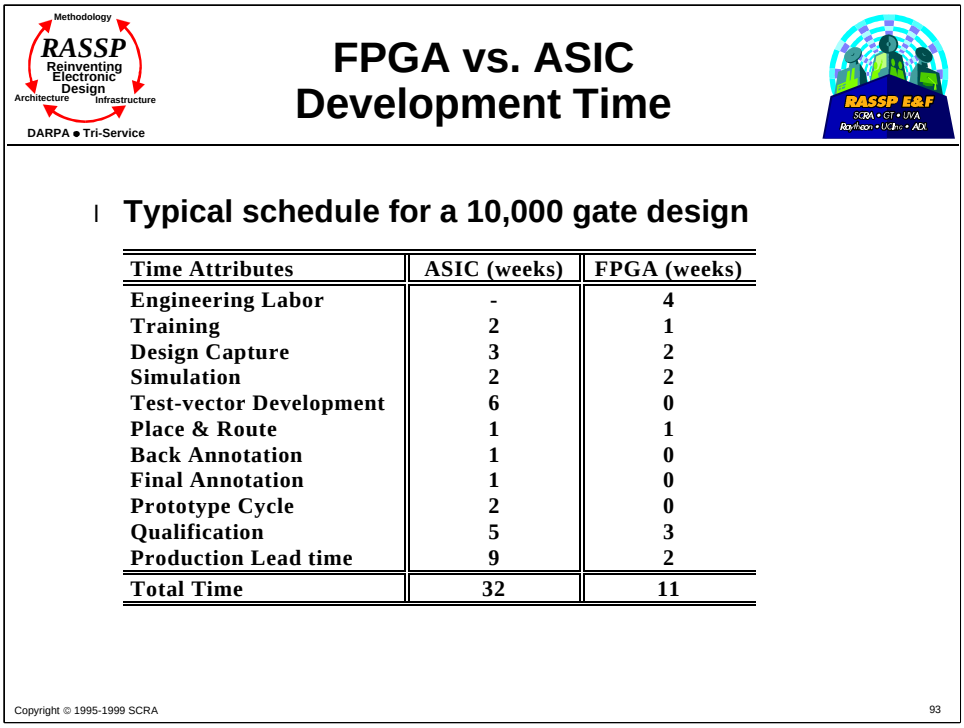
- This model was derived empirically from observation.



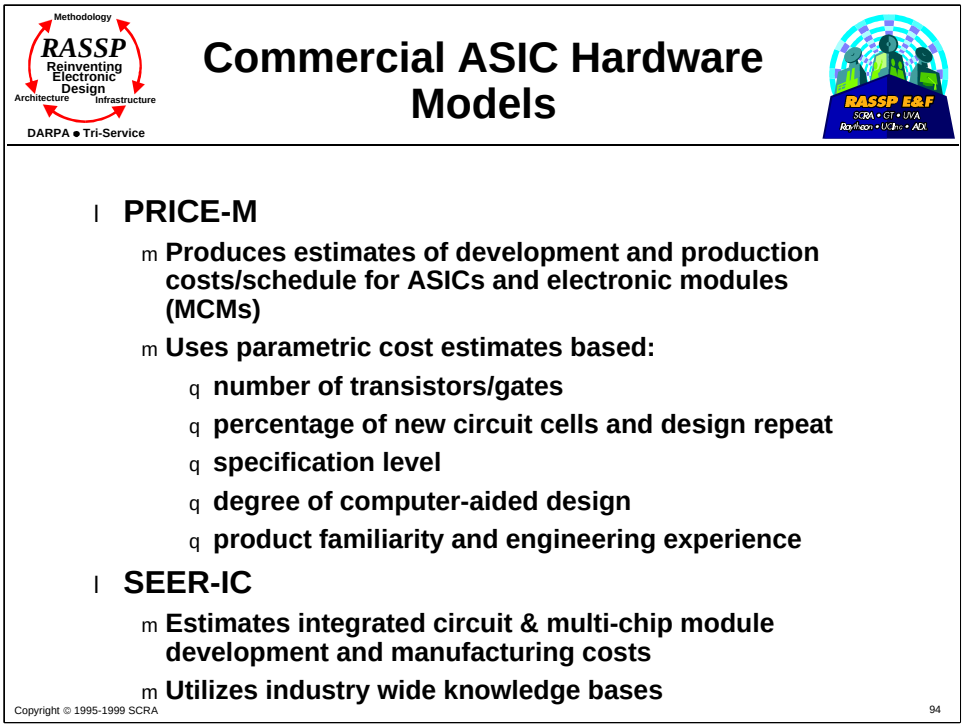
- These equations show that VLSI and software schedules have the same relationship to development effort.
- The model coefficients were chosen by using the least square method to fit the model to the data obtained from 81 designs
- The model assumes that VLSI designs with effort *M* less 6.7 person-months are produced by single person team
- These parametric equations have median error of 13 percent from project actuals
  - 75 percent of the errors were less than 29 percent
- VLSI refers to gate arrays, standard cells, and full-custom devices combined.
- If the VLSI system is partitioned into multiple independent subsystems, the schedule is a function of the largest subsystem's development effort.



- These number a based on “typical numbers” obtained from a world wide research conducted by Dataquest and Integrated Circuits Engineering.
- This model assumes a PQFP package type, volume of 18,000 units, and a product life of 36 months.
- Another ASIC cost factor is re-spin costs.
  - The potential for a re-spin expressed as a percentage of probability is typically 30%
  - The associated re-spin time is 17 weeks at $3000 per person per week
- For more information, see [LIU95].



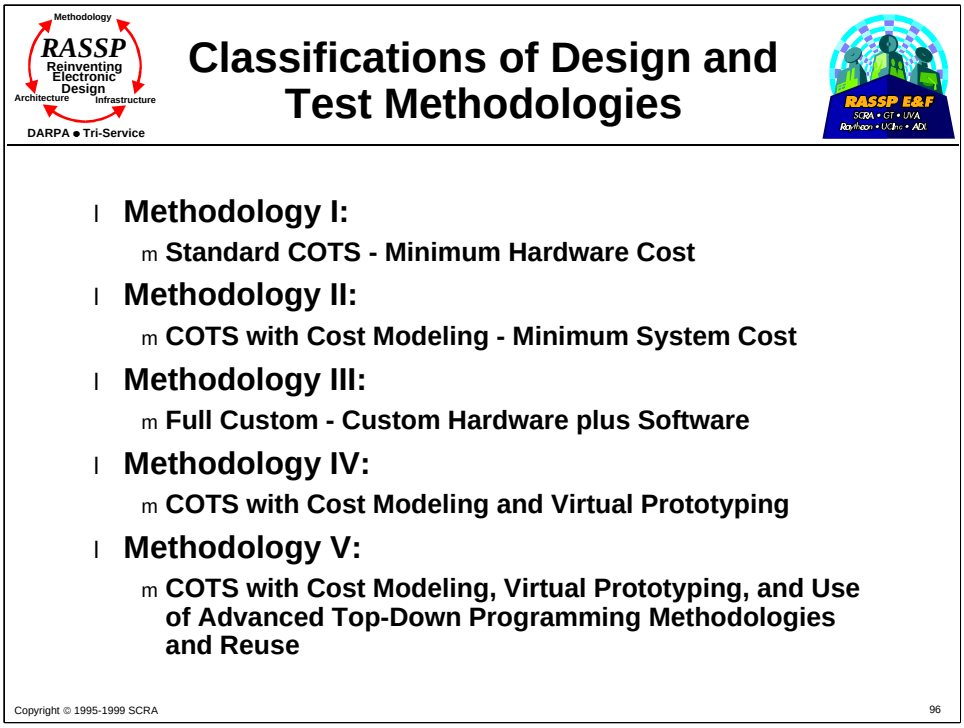
- This schedule was derived from Ben Romdhane, Madisetti, and Hines, “Quick Turnaround ASIC Design in VHDL”, Kluwer Academic Publishers, 1996 and represents values from 1995.



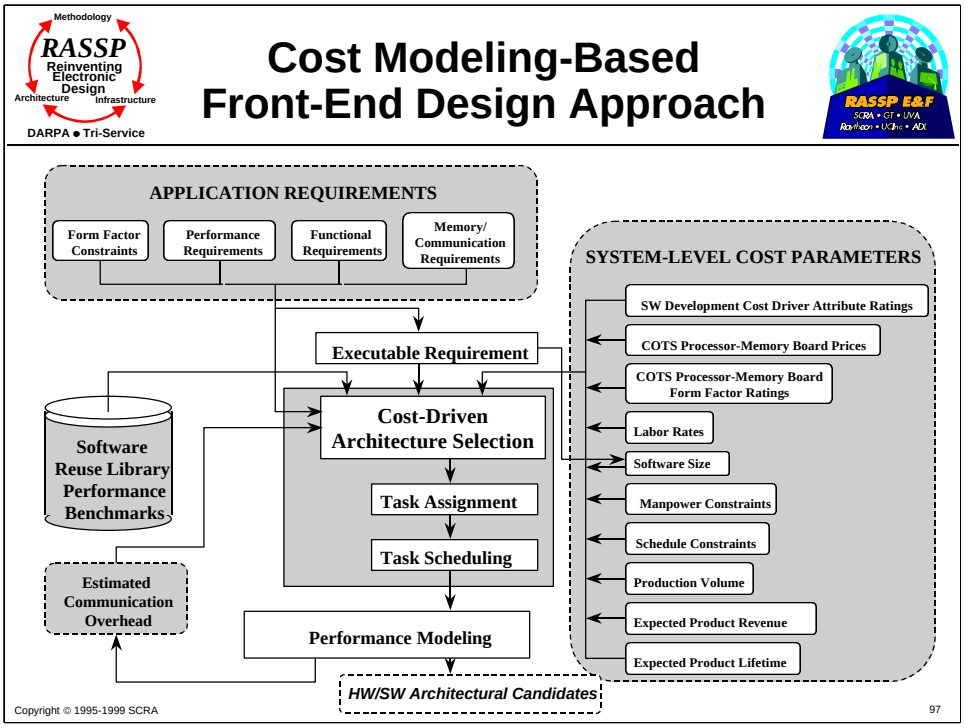
- These models are commercially available.

Module Outline

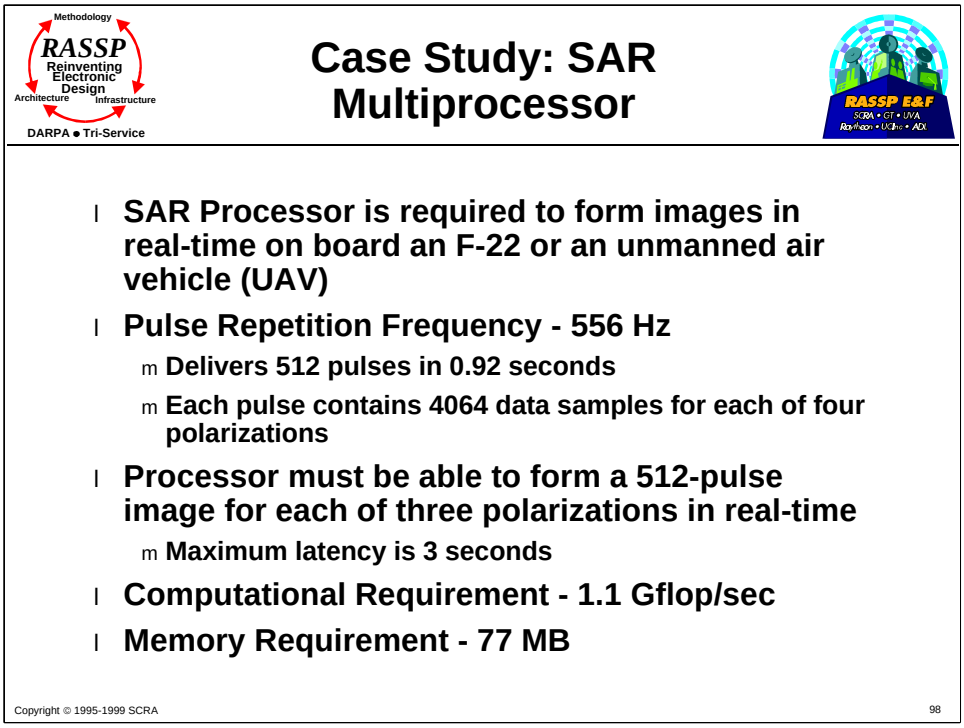
- Introduction to Cost Modeling-Based Embedded Systems Design
- Software Cost Estimation Process
- Parametric Software Cost Models
- Parametric Hardware Cost Models
- **Applications of Cost Modeling in the RASSP Design Process**
- Summary and Major Issues



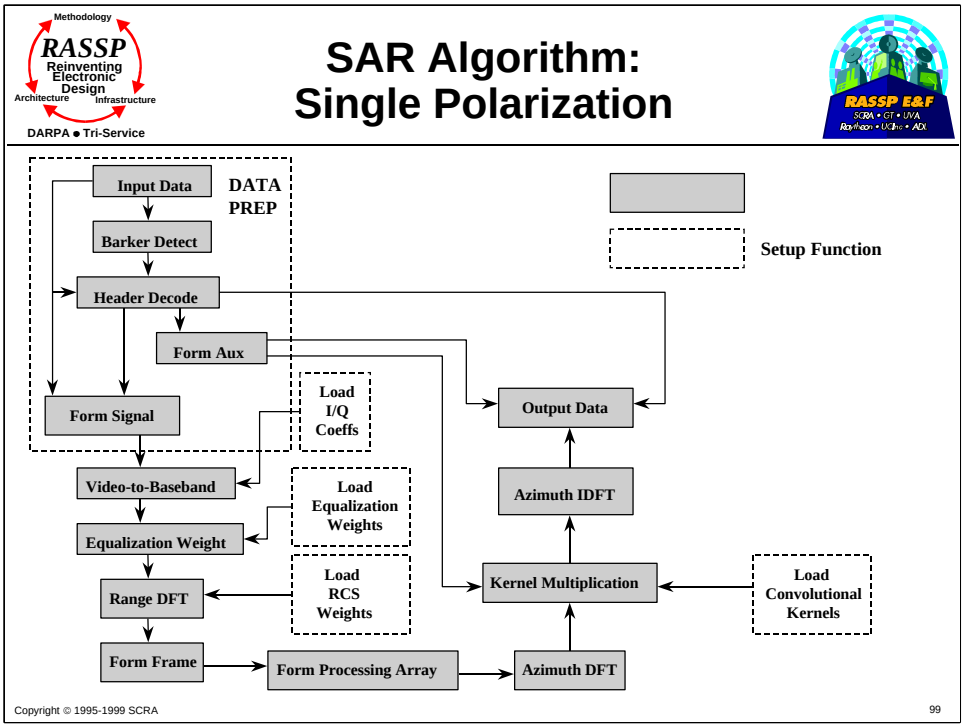
- Various system-level design and test methodologies can be classified into the following broad categories:
  - Methodology I: The most common approach within the industry tries to minimize hardware costs, and software is designed after hardware is fabricated. The latter task is complicated by errors in hardware and tight constraints on the hardware platform.
  - Methodology II: This approach is tries to minimize the sum of hardware and software costs.
  - Methodology III: Another common practice within the industry, especially for designs with severe form factor or application -specific constraints, is to develop custom hardware and software. This approach is usually schedule and cost intensive.
  - Methodology IV: Methodology II is improved upon using the new technology of virtual prototyping where hardware and software models are used to facilitate integration and test.
  - Methodology V: This methodology Is most advanced and has been proposed as part of DARPA's RASSP program and is expected to result in the best cost objective.
  - Methodologies I and III represent traditional COTS and custom design approaches.
- Methodology II is identical to the traditonal COTS approach except that it uses cost models to make HW/SW architectural trade-offs during conceptual design.
- Methodologies IV and V augment Methodology II with RASSP design practices.



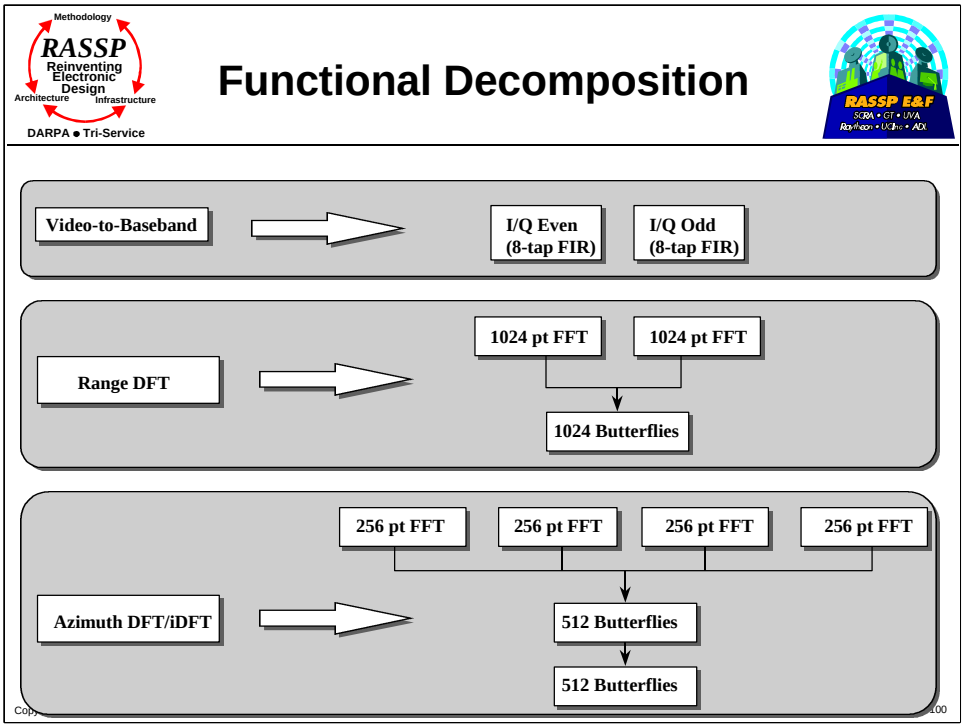
- The above cost modeling-based conceptual design approach is the core of Methodologies II, IV, and V (except Methodology II does not include the use of executable requirements).
- The process starts by translating written requirements into executable requirements, which are used to validate the original customer requirements and to remove ambiguities, thereby reducing requirements volatility.
- The level of detail required to develop the executable requirement facilitates the development of the software size estimate.
- Inputs to the cost-driven architecture selection process step include system-level cost parameters, application requirements, and performance statistics (benchmarked execution times of common DSP algorithms (FFT, FIR, etc.) on the various available programmable processors).
- The target architecture is composed of multiple programmable processors, DRAM, and I/O devices connected over a high performance interconnect network.
- During the cost-driven architecture selection phase, parametric cost models are used to make HW/SW architectural trade-offs in order to minimize total system costs.
- During this stage, the number and type of COTS programmable processor (i860, SHARC, etc.) boards, the memory capacity, and the packaging technology are chosen in a manner which minimizes total costs while satisfying form factor constraints.
- Task Assignment and Task Scheduling involve the mapping of the functional algorithm tasks to the processor architecture and the scheduling of each task on its assigned processor, respectively.
- The resulting architectural candidate is then simulated using performance models to verify that the architectural candidate meets performance requirements.
- The cost-driven architecture selection model is updated with the communication overhead estimates generated from performance modeling.



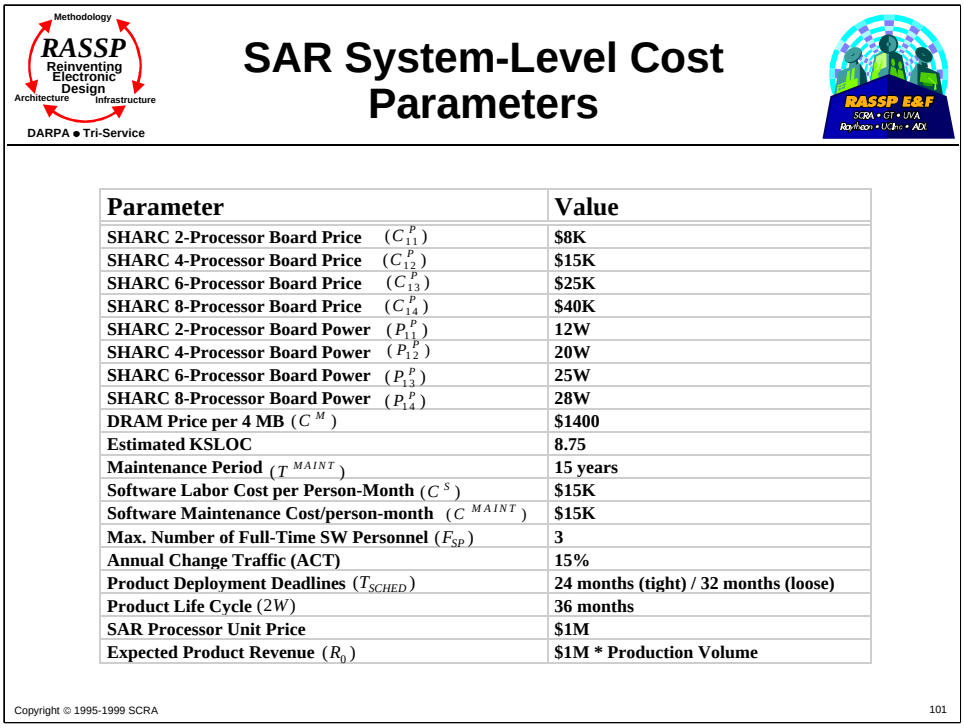
- Synthetic Aperture Radar (SAR) is an important tool for the collection of high-resolution, all-weather image data and has application to tactical military systems as well as civilian systems for remote sensing.
- In addition, SAR can be used to identify man-made objects on the ground or in the air.
- The SAR image processing application will serve as a benchmark for comparing implementations produced by the various design methodologies (Methodologies I-V)
- For more detailed information, see [ZUERN94].



- The above algorithm can be partitioned into six functional blocks:
  - Preamble detection and extraction of radar and auxiliary data from the input data stream
  - Video to baseband I/Q conversion
  - Range processing
  - Corner turn
  - Azimuth processing
  - Output data processing
- In order to implement the SAR algorithm in real-time by exploiting parallelism, we decomposed the data flow graph into three parallel data flow graphs.
- Each concurrent DFG performs the computation for a specific polarization of data.



- In order to meet real-time constraints, the computationally intensive tasks shown above are further decomposed to allow for the exploitation of parallelism inherent in the tasks.



- The standard system-level cost parameters which are used to perform cost/performance trade-offs for the SAR benchmark. These parameters are inputs to the cost-driven architecture selection model.
- These standard model parameters remain constant throughout all models representing the various methodologies unless explicitly stated otherwise.
- The cost of the 2-processor card is derived from the cost of two COTS processors, crossbar switches, printed circuit board area, etc. The 4-processor card consists of four COTS processors and all of the above.
- The 2-processor and 4-processor boards contain single chip packaging technology only. The 6-processor card is comprised of three COTS MCM-L modules, with each module consisting of two COTS processor chips. The 8-processor card consists of two COTS MCM-D multichip modules, with each module consisting of three COTS processor chips. Each board can contain up to 64 MB of DRAM.
- The DRAM price considers the the cost of the DRAM chip, the required printed circuit board area, associated interconnect, etc.
- The software size estimate has been adjusted for reuse of DSP software library elements.

SAR Software Cost Driver Ratings

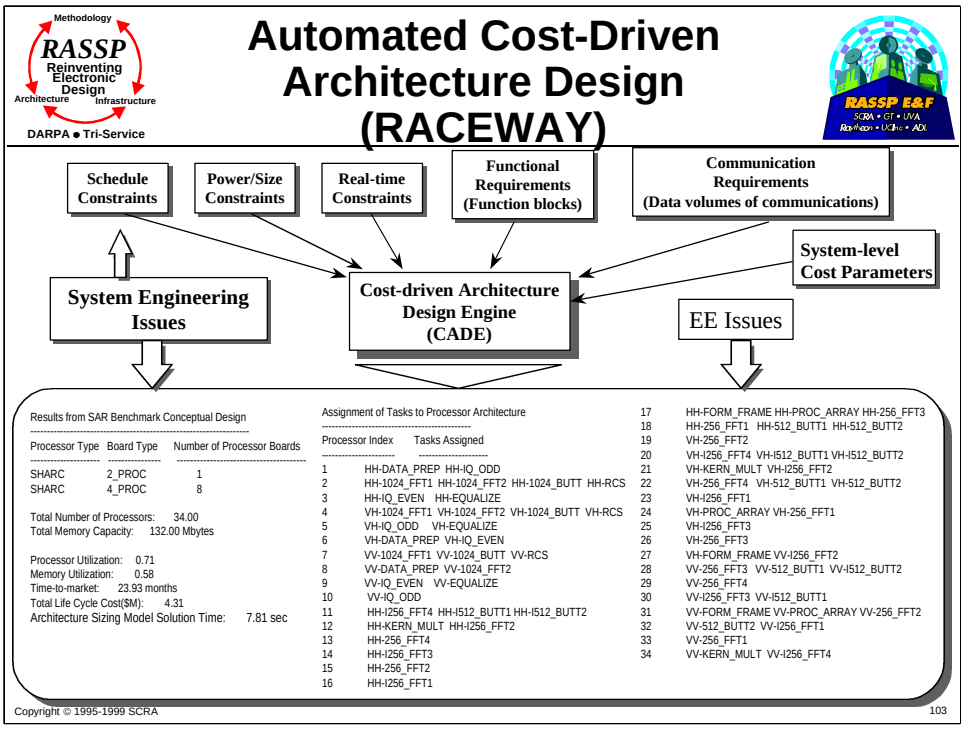
Cost Driver	Situation	Rating	Effort Multiplier
RELY	Local use of system. No serious recovery problems	Nominal	1.00
DATA	256 KBytes	Nominal	1.00
CPLX	real-time signal processing routines	Very high	1.30
VIRT	Based on COTS microprocessor hardware/software (one change every 3 months)	Nominal	1.00
TURN	Two-hour average turnaround time	Nominal	1.00
ACAP	Average senior analysts	Nominal	1.00
AEXP	Three years	Nominal	1.00
PCAP	Average senior programmers	Nominal	1.00
VEXP	Ten months	Nominal	1.00
LEXP	Eighteen months	Nominal	1.00
MODP	Some techniques in use over one year	Nominal	1.00
TOOL	At basic minicomputer tool level	Nominal	1.00
REQV	requirements have a very small amount of ambiguity	Nominal	1.00
REUSE	No reuse is required	Nominal	1.00
SECU	commercial product (unclassified)	Nominal	1.00
PLAT	Unmanned airborne	Nominal	1.40
	Effort adjustment factor (product of effort multipliers)		1.82

STEP 1 → STEP 2

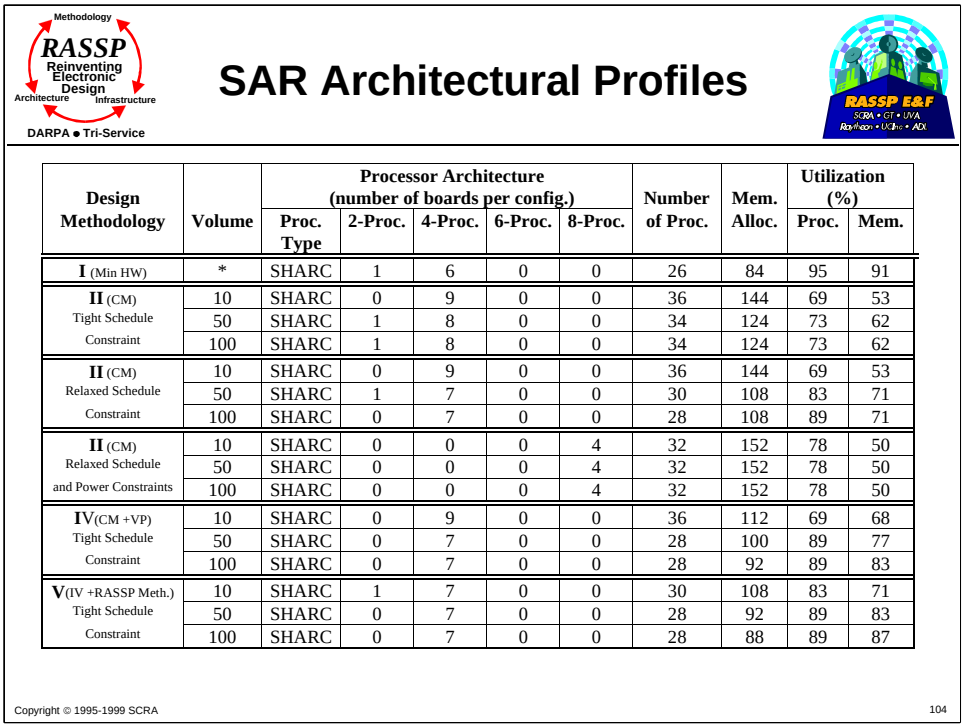
$$\left(\prod_{i=1}^{16} F_i\right)$$

102

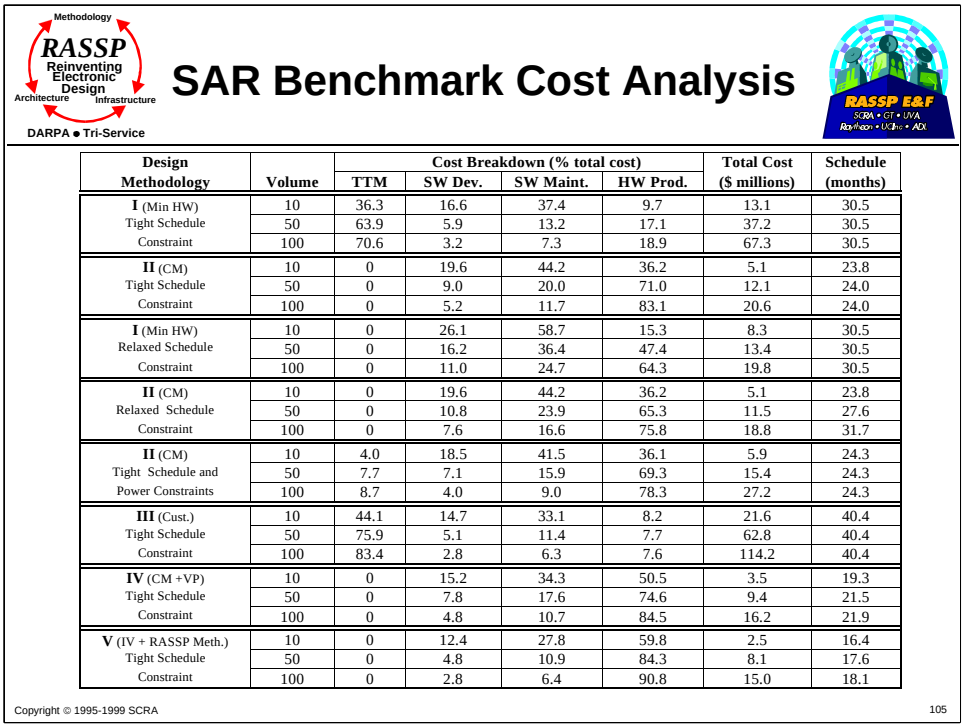
- Effort multiplier values used to perform the cost analysis for the various design methodologies (I-V).



- Example output file generated by Cost-Driven Architecture Design Engine (CADE) , being commercialized by VP Technologies (www.vptinc.com), which uses example architecture selection mathematical programming model, presented in the previous slides, to automatically generate optimum architectural implementations for the SAR application. The output includes:
- The number and type of processors
  - The amount of DRAM required
  - The time-to-market
  - Life cycle cost
  - HW Resource Utilization
  - Mapping of SAR algorithm tasks to processor elements

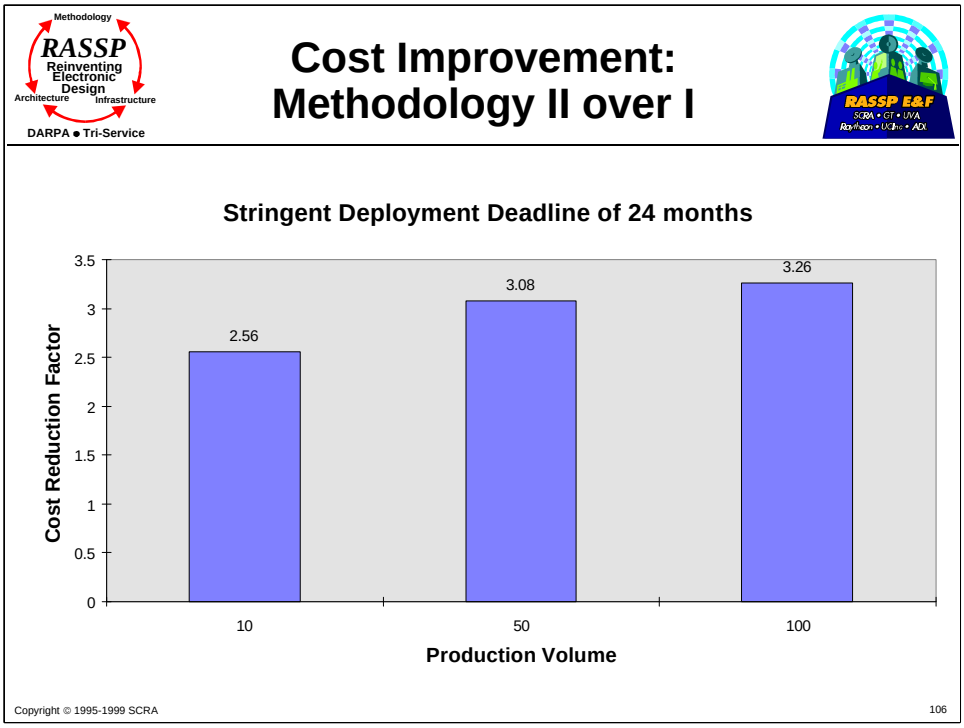


- SAR architectural profiles produced by the various system-level design and test methodologies over a range of production volumes.
- Due to the low cost of DRAM with respect to COTS processor cost, memory margins are extended more than execution time margins when attempting to reduce software development time and cost.
- However, as production volume increases, both processor and memory resources will be somewhat restricted to balance the hardware production costs against the time-to-market costs.



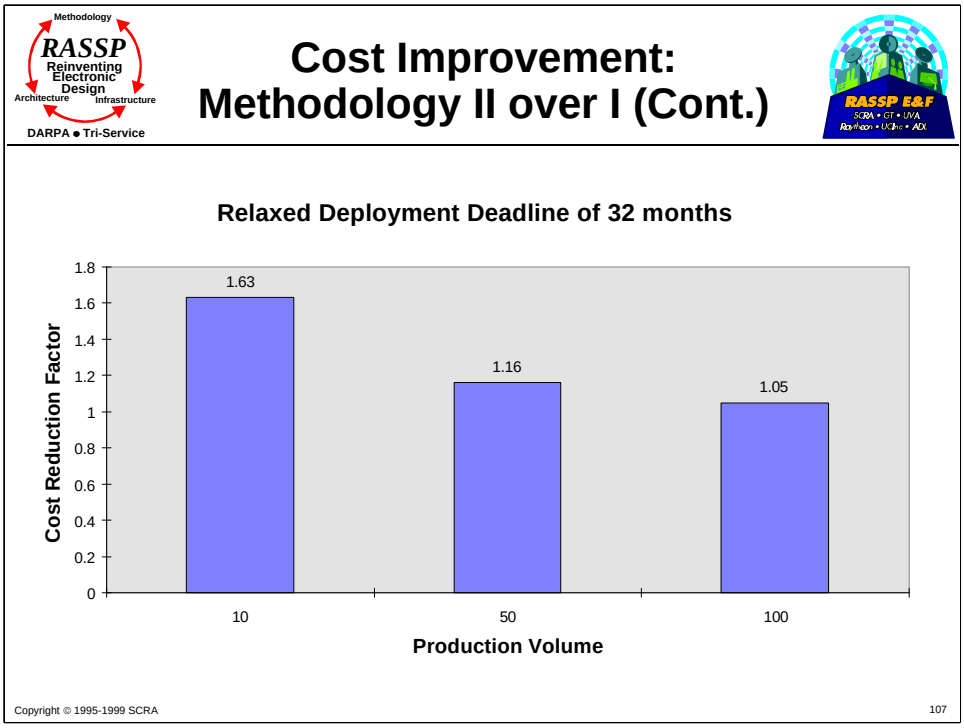
Design Methodology	Volume	Cost Breakdown (% total cost) TTM	SW Dev.	SW Maint.	HW Prod.	Total Cost ($ millions)	Schedule (months)
I (Min HW) Tight Schedule Constraint	10	36.3	16.6	37.4	9.7	13.1	30.5
	50	63.9	5.9	13.2	17.1	37.2	30.5
	100	70.6	3.2	7.3	18.9	67.3	30.5
II (CM) Tight Schedule Constraint	10	0	19.6	44.2	36.2	5.1	23.8
	50	0	9.0	20.0	71.0	12.1	24.0
	100	0	5.2	11.7	83.1	20.6	24.0
I (Min HW) Relaxed Schedule Constraint	10	0	26.1	58.7	15.3	8.3	30.5
	50	0	16.2	36.4	47.4	13.4	30.5
	100	0	11.0	24.7	64.3	19.8	30.5
II (CM) Relaxed Schedule Constraint	10	0	19.6	44.2	36.2	5.1	23.8
	50	0	10.8	23.9	65.3	11.5	27.6
	100	0	7.6	16.6	75.8	18.8	31.7
II (CM) Tight Schedule and Power Constraints	10	4.0	18.5	41.5	36.1	5.9	24.3
	50	7.7	7.1	15.9	69.3	15.4	24.3
	100	8.7	4.0	9.0	78.3	27.2	24.3
III (Cust.) Tight Schedule Constraint	10	44.1	14.7	33.1	8.2	21.6	40.4
	50	75.9	5.1	11.4	7.7	62.8	40.4
	100	83.4	2.8	6.3	7.6	114.2	40.4
IV (CM +VP) Tight Schedule Constraint	10	0	15.2	34.3	50.5	3.5	19.3
	50	0	7.8	17.6	74.6	9.4	21.5
	100	0	4.8	10.7	84.5	16.2	21.9
V (IV + RASSP Meth.) Tight Schedule Constraint	10	0	12.4	27.8	59.8	2.5	16.4
	50	0	4.8	10.9	84.3	8.1	17.6
	100	0	2.8	6.4	90.8	15.0	18.1

- The cost and schedule of SAR implementations generated by the proposed design and test methodologies.
- Methodology I: Under tight schedule constraints, software costs account for a large portion of the overall system costs for low production volumes. However, as production volume increases, time-to-market costs dominate the system costs due to the increased revenue losses resulting from being six months late to market. When the schedule is relaxed, the approach still suffers from high software costs at low production volumes. But as the volume increases, the minimum hardware cost approach approaches that of the minimum system cost.
- Methodologies II, IV, V: Under tight schedule constraints, the cost modeling-based approaches relax the hardware architecture to ensure that the time-to-market deadline is met as a first priority. As volume increase, the hardware resources are restricted to reduce hardware production costs while continuing to meet the time-to-market deadline.
- Methodology III: For low volume production runs and tight schedule constraints, software development costs and time-to-market costs dominate the system costs due to tight execution time and memory margins. However, due to the enormously long schedule delays, time-to-market costs will be dominant as the production volume increases.
- The following cost improvement graphs illustrates this analysis.



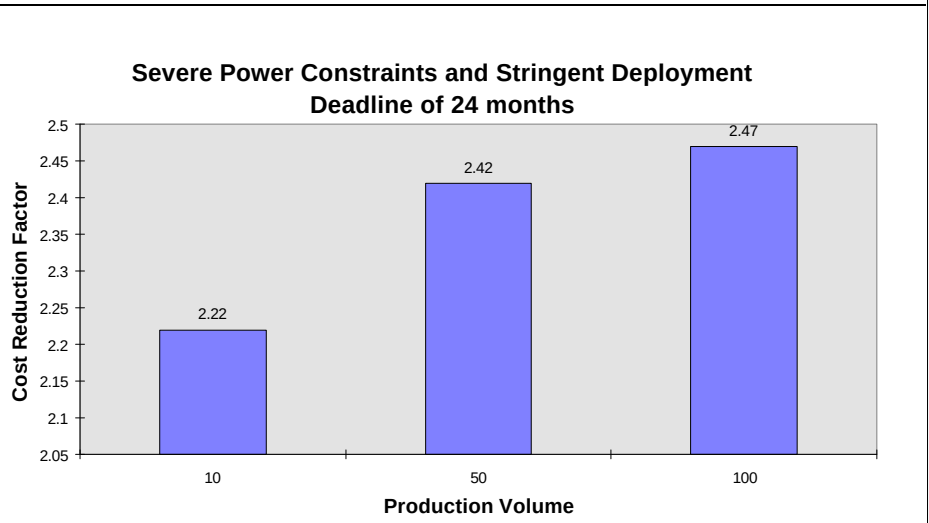
- Methodology I: Under tight schedule constraints, software costs account for a large portion of the overall system costs for low production volumes. However, as production volume increases, time-to-market costs dominate the system costs due to the increased revenue losses resulting from being six months late to market. When the schedule is relaxed, the approach still suffers from high software costs at low production volumes. But as the volume increases, the minimum hardware cost approach approaches that of the minimum system cost.
- Methodologies II: Under tight schedule constraints, the cost modeling-based approaches relax the hardware architecture to ensure that the time-to-market deadline is met as a first priority. As volume increase, the hardware resources are restricted to reduce hardware production costs while continuing to meet the time-to-market deadline.

- For more detailed information on the following slides, see [DEB97].



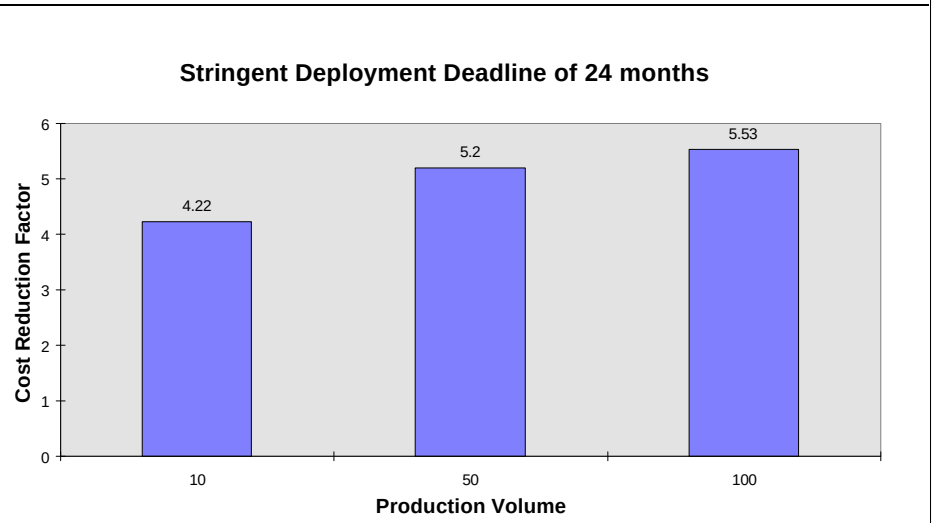
- Methodology I: Under tight schedule constraints, software costs account for a large portion of the overall system costs for low production volumes. However, as production volume increases, time-to-market costs dominate the system costs due to the increased revenue losses resulting from being six months late to market. When the schedule is relaxed, the approach still suffers from high software costs at low production volumes. But as the volume increases, the minimum hardware cost approach approaches that of the minimum system cost.
- Methodologies II: Under tight schedule constraints, the cost modeling-based approaches relax the hardware architecture to ensure that the time-to-market deadline is met as a first priority. As volume increase, the hardware resources are restricted to reduce hardware production costs while continuing to meet the time-to-market deadline.

Cost Improvement: Methodology II over I (Cont.)

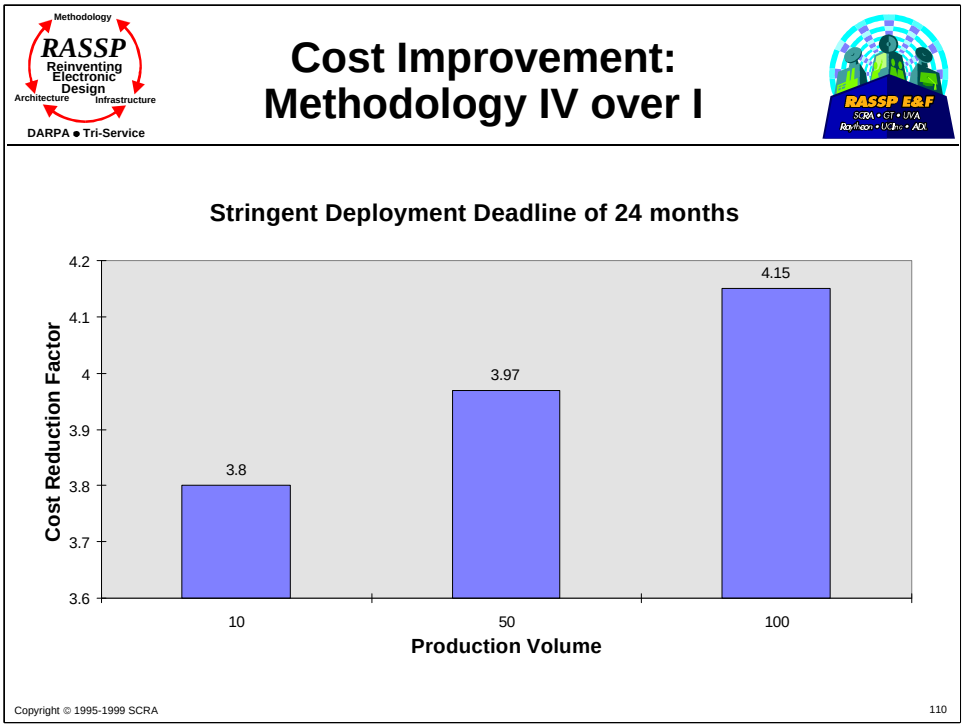


- The severe power constraint is 150 Watts.
- This case shows that denser MCM technologies can be effectively utilized to minimize schedule delays due to form factor constraints on the system implementation.

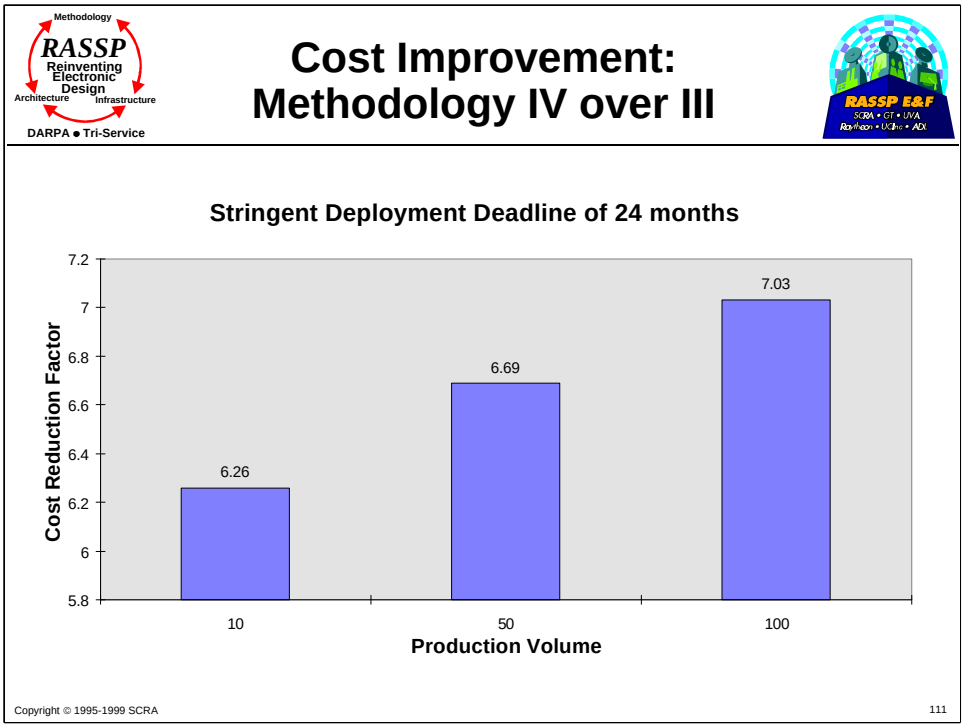
Cost Improvement: Methodology II over III



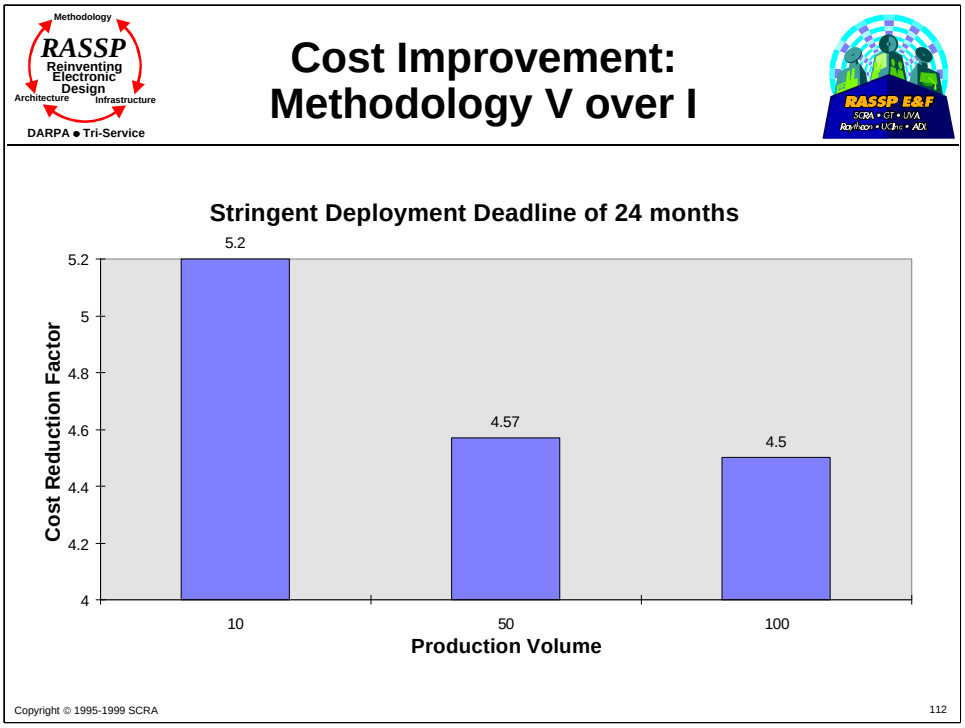
- An increased cost size of 12K SLOC is assumed for customization of software routines, as well as, an increase of 6 months in overall development time due to hardware custom board design time is assumed.
- There is a 40% reduction in overall board level cost due to this approach.



- The figure illustrates the large reduction in provided by Methodology IV over Methodology I. It is assume that virtual prototyping provides a 2X improvement in development productivity.

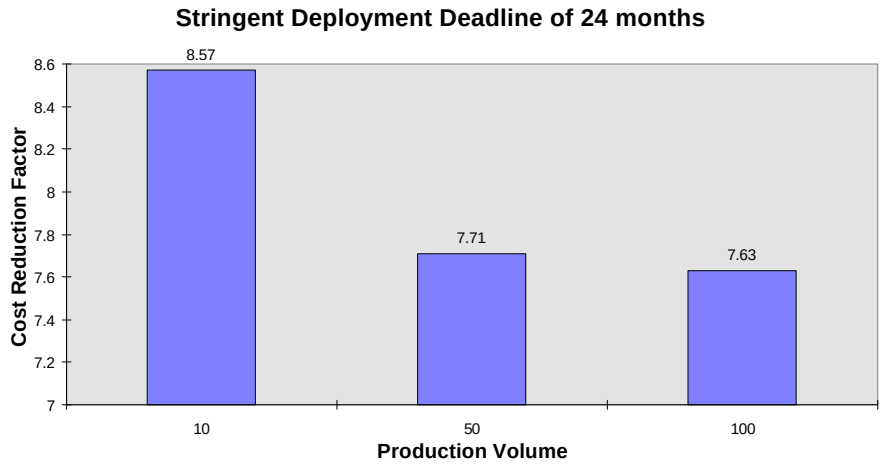


- Methodology IV: Under tight schedule constraints, the cost modeling-based approaches relax the hardware architecture to ensure that the time-to-market deadline is met as a first priority. As volume increase, the hardware resources are restricted to reduce hardware production costs while continuing to meet the time-to-market deadline.
- Methodology III: For low volume production runs and tight schedule constraints, software development costs and time-to-market costs dominate the system costs due to tight execution time and memory margins. However, due to the enormously long schedule delays, time-to-market costs will be dominant as the production volume increases.



- This graph shows the effect of the use of the RASSP methodology on life cycle cost.
- We assume there is an additional 1.96X improvement in development productivity due to the routine use of top-down design and programming methodologies and fully integrated software development tool environments.

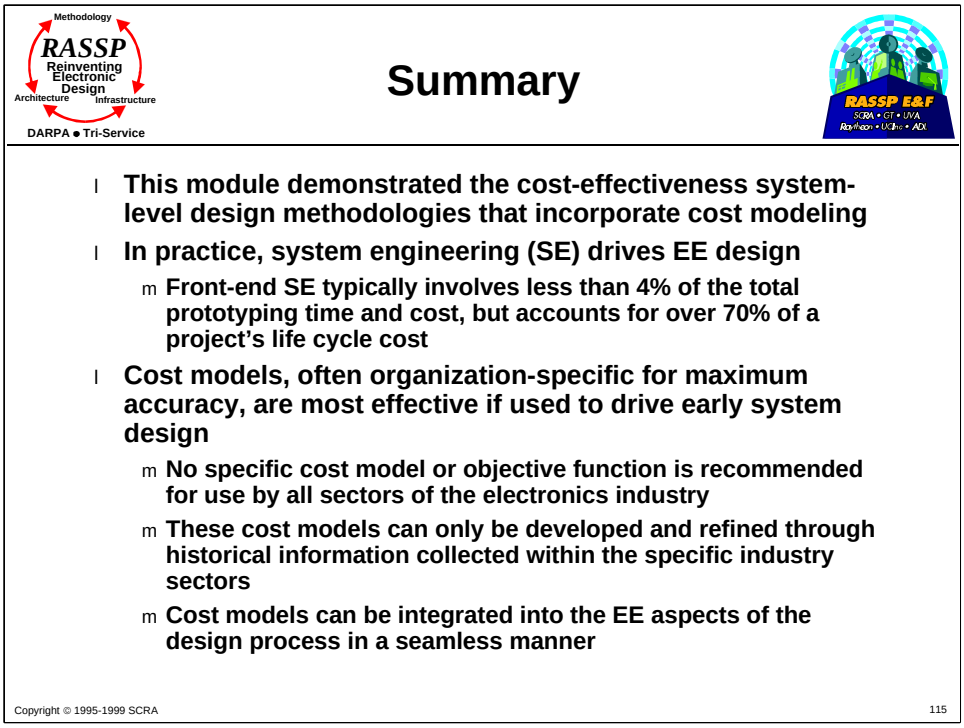
Cost Improvement: Methodology V over III



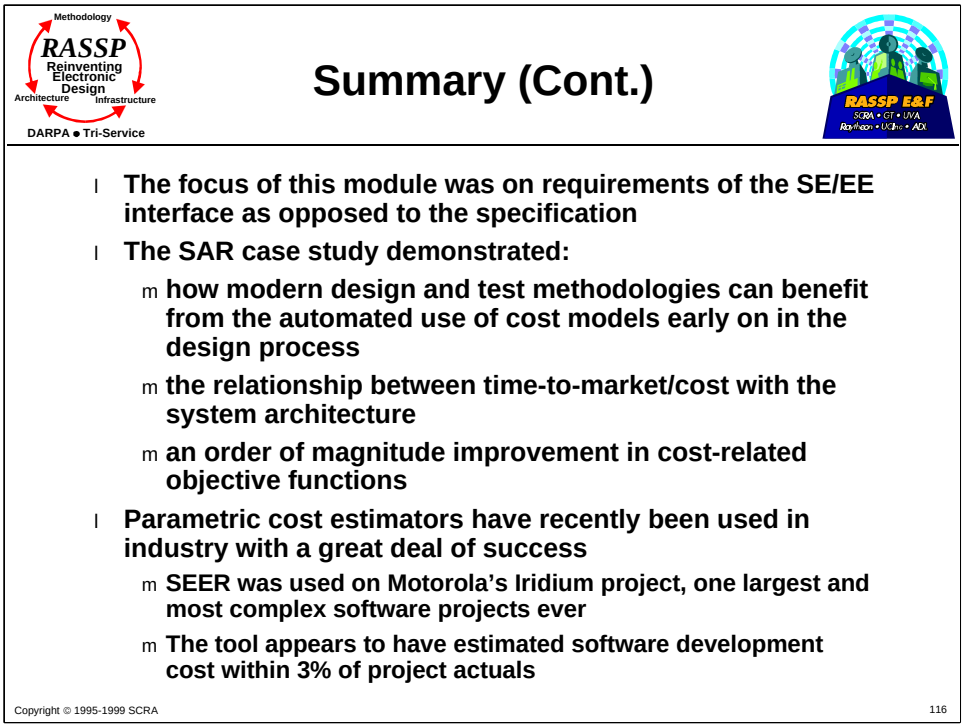
- Methodology V: Under tight schedule constraints, the cost modeling-based approaches relax the hardware architecture to ensure that the time-to-market deadline is met as a first priority. As volume increase, the hardware resources are restricted to reduce hardware production costs while continuing to meet the time-to-market deadline.
- Methodology III: For low volume production runs and tight schedule constraints, software development costs and time-to-market costs dominate the system costs due to tight execution time and memory margins. However, due to the enormously long schedule delays, time-to-market costs will be dominant as the production volume increases.

Module Outline

- Introduction to Cost Modeling-Based Embedded Systems Design
- Software Cost Estimation Process
- Parametric Software Cost Models
- Parametric Hardware Cost Models
- Applications of Cost Modeling to the RASSP Design Process
- **Summary and Major Issues**



- We wish to emphasize that it is not necessary to use a certain cost model, or prefer one over other. Organizations should choose their own cost models after careful study and use the methodology proposed in this module to develop design processes and options that are relevant to their organizational objectives.



- New tools corresponding to system-level design automation are expected to rely on cost modeling to synthesize hardware and software architectures (www.vptinc.com).

References

- **[AHRENS95] J. Ahrens, N. Prywes, "Transition to a Legacy- and Reuse-Based Software Life Cycle," *IEEE Computer*, pp. 27-36, Oct. 1995; © IEEE 1995**
- **[AF94] U.S. Air Force Analysis Agency, *REVIC Software Cost Estimating Model User's Manual Version 9.2*, Dec. 1992.**
- **[AND95] J. Anderson, "Projecting RASSP Benefits," *Proc. Second Annual RASSP Conf.*, ARPA, US Dept. of Defense, Arlington, VA 1995.**
- **[BOEHM81] B. Boehm, *Software Engineering Economics*, Prentice-Hall, Inc. Englewood Cliffs NJ, 1981.**
- **[BOEHM88] B. Boehm, "A Spiral Model of Software Development and Enhancement," *IEEE Computer*, pp. 61- 72, May 1988; © IEEE 1988**
- **[BOEHM95] B. Boehm, et al., "Cost Models for Future Software Processes: Cocomo 2.0," *Annals of Software Engineering*, 1995, Baltzer Science Publishers. Used with permission.**
- **[COCOMO] COCOMO manual, University of California, http://sunset.usc.edu/COCOMOII/suite.html**
- **[DEB97] J. DeBardelaben, V. Madisetti, A. Gadient, "The Economics of Design & Test of COTS-based Embedded Microelectronics Systems, " *IEEE Design & Test of Computers*, Fall 1997.**
- **[DOANE93] D. Doane, P. Franzon, *Multichip Module Technologies and Alternatives -The BASICS*, Van Nostrand Reinhold, 1993.**
- **[DOD95] Department of Defense, *Parametric Cost Estimating Handbook*, Fall 1995.**
- **[FERENS] Class notes from Dr. Daniel Ferens. Used with permission.**

References (Cont.)

- **[FEY85] C. Fey, "Custom LSI/VLSI Chip Design Productivity," *IEEE Journal of Solid-State Circuits*, Vol. sc-20, No. 2, April 1985, pp. 555-561.**
- **[FP89] C. Fey, D. Paraskevopoulos, "Studies in VLSI Technology Economics IV: Models for Gate Array Design Productivity," *IEEE Journal of Solid-State Circuits*, Vol. 24, No. 4, August 1989, pp. 1085-1091; © IEEE 1989**
- **[IEEE] All referenced IEEE material is used with permission.**
- **[LEVITT92] M. Levitt, "Economic and Productivity Considerations in ASIC Test and Design-for-Test," *Digest of Papers: Compcon '92*, pp. 440-445, Spring 1992.**
- **[LIU95] J. Liu, "Detailed Model Shows FPGAs' True Costs," *EDN*, pp. 153-158, May 11, 1995.**
- **[MAD95] V. Madisetti, T. Egolf, "Virtual Prototyping of Embedded Microcontroller-Based DSP Systems," *IEEE Micro*, Oct. 1995.**
- **[MAD96] V. Madisetti, "Rapid Digital System Prototyping: Current Practice, Future Challenges," *IEEE Design & Test of Computers*, Fall 1996, pp. 12-22; © IEEE 1996**
- **[MINK95] A. Minkiewicz, A. DeMarco, " The PRICE Software Model," PRICE Systems Internal Document, June 1995.**
- **[PF87] D. Paraskevopoulos, C. Fey, "Studies in LSI Technology Economics III: Design Schedules for Application-Specific Integrated Circuits," *IEEE Journal of Solid-State Circuits*, Vol. sc-22, No. 2, April 1987, pp. 223-229.**
- **[Richards97] Richards, M., Gadient, A., Frank, G., eds. *Rapid Prototyping of Application Specific Signal Processors*, Kluwer Academic Publishers, Norwell, MA, 1997**
- **[SEPO96] Software Engineering Process Office (SEPO), NCCOSC RDTE DIV, *Software Size, Cost, and Schedule Estimation Process Version 2.0*, March 1996.**
- **[ZUERN94] B. Zuerndorfer, G. Shaw, "SAR Processing for RASSP Application," *Proc. 1st Annual RASSP Conf.*, pp. 253-268, Arlington, VA, August 1994.**