Software Project Management

Dr. Michael F. Siok, PE, ESEP
University of Texas at Arlington (UTA)
Computer Science and Engineering

CSE 5324: Software Engineering: Analysis, Design, and Testing

Textbook: Object-Oriented Software Engineering: An Agile Unified Methodology, by David C. Kung, ISBN 978-0-07-3376257

Key Takeaway Points

- Software Project Management is concerned with the management aspects of software engineering projects
- Software Project Management s focused on effort estimation, planning and scheduling, software process, and risk management
- For our class, we will look at Effort Estimation Only with CoCoMo II

Why Project Management for Software?

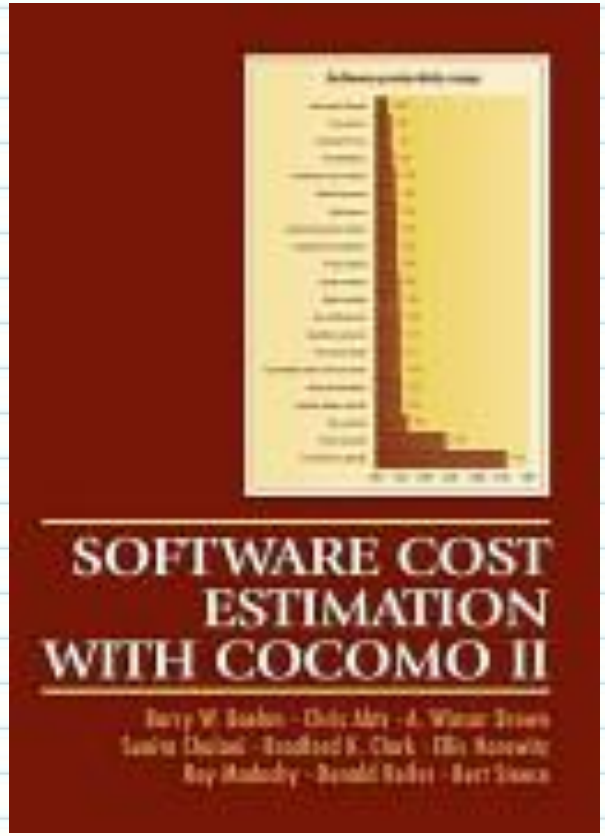
- Software projects tend to be large intellectual products
- Software is intangible; difficult to accurately and directly measure it
- Effective and frequent communication is essential
  - Incurs high overhead cost
- Risks need to be explicitly stated and managed for software

- Software Project Management is concerned with the management aspects of software engineering to:
  - Increase software productivity
  - Increase product and process quality
  - Reduce software time to market

COCOMO 2 = COCOMO 95

- Barry Boehm et al. 1995 article:
  - "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0"
- Barry Boehm et al. 2000 book:
  - "Software Cost Estimation with Cocomo II"
- Adapts COCOMO to more recent software lifecycle practices

- Free interactive cost estimation tool:
  - http://csse.usc.edu/tools



COCOMO 2

- Three increasingly detailed cost models
  - *Application composition model*
    - Deals with prototyping
    - Based on Object-points (object = screen, report, ..)
    - Following slides, for details see 1995 paper Figure 3
  - *Early Design Model*
    - Deals with architectural design stage
    - [+] Uses Function Points or Direct KSLOC estimates
  - *Post-architecture model*
    - Deals with actual development stage
    - Similar to COCOMO 85; consider it an update to the original COCOMO

Application Composition Model

1. Estimate number of screens, reports, 3GL components
2. Determine complexity of each screen, report, 3GL
3. Determine complexity weights for each screen, report, and 3GL component

and source of data tables

# of views	Count # of Views Contained	< 4 total: < 2 on server < 3 on client	< 8 total: 2-3 on server 3-5 on client	≥ 8 total: > 3 on server > 5 on client
	< 3	simple	simple	medium
	3 – 7	simple	medium	difficult
	> 7	medium	difficult	difficult

Count # of Sections Contained	< 4 total: < 2 on server < 3 on client	< 8 total: 2-3 on server 3-5 on client	≥ 8 total: > 3 on server > 5 on client
< 2	simple	simple	medium
2-3	simple	medium	difficult
> 3	medium	difficult	difficult

Complexity Weights

Object type	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL component			10

Figure 23.2 For Screens

Determine Complexity of Objects

4. Add the weighted counts of screens, reports, and 3GL components to produce 1 number called Object point count
5. Estimate % reuse and compute new object point count: NOP = OP x (100 – Reuse)/100
6. Determine object point productivity from table below

Developer experience and capability	ICASE maturity and capability	PROD
very low	very low	4
low	Low	7
nominal	Nominal	13
high	High	25
very high	very high	50

7. Compute person month effort: Effort in PM = NOP/PROD

COCOMO 2

Application Composition Model
Early Design Model
Post Architecture Model

- Three increasingly detailed cost models
  - *Application composition model*
    - Deals with prototyping
    - Based on Object-points (object = screen, report, ..)
    - Following slides, for details see 1995 paper Figure 3
  - *Early Design Model*
    - Deals with architectural design stage
    - [+] Uses Function Points or Direct KSLOC estimates
  - *Post-Architecture model*
    - Deals with actual development stage
    - Similar to COCOMO 85; consider it an update to the original COCOMO

Application Composition Model
Early Design Model
Post Architecture Model

Early Design Model

- Can estimate KSLOCs directly or use Function Points
- For Function Points:
  - Count Unadjusted Function Points (UFP)
    - Follow Function Point Analysis rules (See page 585 and Figure 23.3 of textbook)
  - Convert UFP to SLOC
    - Use conversion table based on language
    - KSLOC = (UFP x (SLOC/UFP))/1000
  - Example table from
    - Jones 1991 book: "*Applied Software Measurement*"
    - Adapt table to your environment (i.e., company you work for may use different SLOC conversions by language)

Language	SLOC / UFP
Ada	71
Assembly	320
Assembly (Macro)	213
ANSI/Quick/Turbo Basic	64
C	128
C++	29
ANSI Cobol 85	91
Fortran 77	105
Lisp	64
Modula 2	80
Pascal	91
Prolog	64
Report Generator	80
Spreadsheet	6

Considers Seven Cost Drivers and Five Scale Factors

Application Composition Model
Early Design Model
Post Architecture Model

- Estimated Effort (in person months) = $(SLOC)^b$ * (Product of Cost Drivers)
- Seven (7) cost drivers
  - Cost Drivers for Early Design Model
    - RCPX – Product Reliability and Complexity
    - RUSE – Required Reuse (same as Post Architecture model)
    - PDIF – Platform Difficulty
    - PERS – Personnel Capability
    - PREX – Personnel Experience
    - FCIL -- Facilities
    - SCHED – Schedule (same as Post Architecture model)
  - Each Cost Driver is measured on a 6 point scale (NOM = 1.0)
    - VLO, LO, NOM, HI, VHI, XHI (See Table 23.5)
  - b is computed based on the sum of 5 scale factors: $b = 0.91 + 0.01 \times (SF_1 + \ldots + SF_5)$
    - See Table 23.4 for scale factor names and descriptions

COCOMO 2

Application Composition Model
Early Design Model
Post Architecture Model

- Three increasingly detailed cost models
  - *Application composition model*
    - Deals with prototyping
    - Based on Object-points (object = screen, report, ..)
    - Following slides, for details see 1995 paper Figure 3
  - *Early Design Model*
    - Deals with architectural design stage
    - [+] Uses Function Points or Direct software SLOC estimate
  - *Post-Architecture model*
    - Deals with actual development stage
    - Similar to COCOMO 85; consider it an update to the original COCOMO

Post-Architecture Model

Application Composition Model
Early Design Model
Post Architecture Model

- Most detailed model and used when software architecture has already been developed
  - **a (kSLOC)^b (product of cost drivers)**
- Constant a
  - "provisionally set to" 2.5
- Scale factors: 5
  - *b* can range from 1.01 to 1.26
- Cost drivers: 17
  - Product ranges from 0.0475 to 151.8 (nominal is 1.0)
  - May be organized by categories: Product factors, Platform Factors, Personnel Factors, and Project Factors
- Differences from (original) COCOMO 81
  - Cost drivers changed (from 7 to 17)
  - *b* can range from 1.01 to 1.26 vs. 3 COCOMO modes

17 Cost Drivers: "*Product Factors*"

Application Composition Model
Early Design Model
Post Architecture Model

- Reliability Required (i.e., Effect of software failure)
  - From "*slight inconvenience*" to "*risk to human life*"
- Database Size
  - Larger database → Harder to generate test data
- Product Complexity
  - Based on product characterization (i.e., Control, computational, device-dependent, data management, or UI management)
- Required Reusability (i.e., designs, documents, code, test components)
  - From "*Will be reused by nobody*" to "*used across multiple product lines*"
- Documentation needs
  - From "*Many lifecycle needs uncovered*" to "*very excessive for life-cycle needs*"

	Very Low	Low	Nominal	High	Very High	Extra High
Reliability required	0.75	0.88	1.00	1.15	1.39	
Database size		0.93	1.00	1.09	1.19	
Product complexity	0.75	0.88	1.00	1.15	1.30	1.66
Required reusability		0.91	1.00	1.14	1.29	1.49
Documentation needs	0.89	0.95	1.00	1.06	1.13	

Cost Drivers: "*Platform Factors*"

Application Composition Model
Early Design Model
Post Architecture Model

- Execution time & main storage constraints
  - Can use from ≥ 50% to 5% of available resources
- Platform volatility: Underlying software + hardware
  - Major change every 12 months to 2 weeks
  - Minor change every 1 month to 2 days

	Very Low	Low	Nominal	High	Very High	Extra High
Execution-time constraints			1.00	1.11	1.31	1.67
Main storage requirements			1.00	1.06	1.21	1.57
Platform volatility		0.87	1.00	1.15	1.30	

Application Composition Model
Early Design Model
Post Architecture Model

Cost Drivers: "*Personnel Factors*"

- Analyst capability
  - Analysis and design ability, efficiency, thoroughness, and the ability to communicate and cooperate
  - From 15th to 90th percentile

	Very Low	Low	Nominal	High	Very High	Extra High
Analyst capability	1.50	1.22	1.00	0.83	0.67	
Programmer capability	1.37	1.16	1.00	0.87	0.74	
Application experience	1.22	1.10	1.00	0.89	0.81	
Platform experience	1.24	1.10	1.00	0.92	0.84	
Language and tool experience	1.25	1.12	1.00	0.88	0.81	
Personnel continuity	1.24	1.10	1.00	0.92	0.84	

Cost Drivers: "*Personnel Factors*" *(Cont'd)*

Application Composition Model
Early Design Model
Post Architecture Model

- Programmer capability
  - Capability of the programmers as a team
  - Ability, efficiency, thoroughness, and the ability to communicate and cooperate
- Application / Platform / Language / Tool experience
  - Team's experience with this type of app / platf / lang / tool
  - From ≤ 2 months to 6 years
- Personnel continuity
  - Personnel turnover from 48% to 3% per year

Cost Drivers: "*Project Factors*"

→ Application Composition Model
Early Design Model
Post Architecture Model

- Tools from only edit-code-debug to mature well-integrated lifecycle tools
- Multi-site: Communication tools
  - From some phone and mail to interactive multimedia
- Schedule stretch-out compared to nominal schedule
  - From 75% (a compression) to 160%

	Very Low	Low	Nominal	High	Very High	Extra High
Use of software tools	1.24	1.12	1.00	0.86	0.72	
Multi-site development	1.25	1.10	1.00	0.92	0.84	0.78
Required development schedule	1.29	1.10	1.00			

Five (5) Scale Factors

Application Composition Model
Early Design Model
Post Architecture Model

- $a\ (kSLOC)^b\ (\prod cost\ drivers_i)$
- Elaborate scaling model for exponent *b*
  - $b = 1.01 + 0.01 \sum(W_i)$
- Five scale factors W_i (next slides)
- Rate each of the scale factors (choose 1):
  - very low (5)
  - low (4)
  - nominal (3)
  - high (2)
  - very high (1)
  - extra high (0)

Five Scale Factors (cont'd)

Application Composition Model
Early Design Model
Post Architecture Model

- Precedentedness
  - How novel is project to organization
  - There is an organizational understanding of product objectives
  - Experience in working with related software systems
  - Concurrent development of associated new hardware and operational procedures
  - Need innovative architectures, algorithms

- Development flexibility
  - Need for software conformance with pre-established requirements
  - Need for software conformance with external interface specifications
  - Premium on early completion

Five Scale Factors (cont'd)

Application Composition Model
Early Design Model
Post Architecture Model

- Architecture / risk resolution (COCOMOII Model Manual, Overall Model Definition, Table I-3)
  - Risk Management Plan identifies all critical risk items, establishes milestones for resolving them by PDR
  - Schedule, budget, and internal milestones through PDR compatible with Risk Management Plan
  - Percent of development schedule devoted to establishing architecture, given general product objectives
  - Percent of required top software architects available
  - Tool support available for resolving risk items, developing and verifying architectural specs
  - Level of uncertainty in key architecture drivers: mission, UI, COTS, hardware, technology, performance
  - Number and criticality of risk items

Five Scale Factors (cont'd)

Application Composition Model
Early Design Model
Post Architecture Model

- Team Cohesion (COCOMOII Model Manual, Overall Model Definition, Table I-4)
  - Consistency of stakeholder objectives and cultures
  - Ability, willingness of stakeholders to accommodate other stakeholders' objectives
  - Experience of stakeholders in operating as a team
  - Stakeholder teambuilding to achieve shared vision and commitments

- Process Maturity
  - Maturity of project organization
    - Per CMM (HVV section 6.6)

Counting Code we Reuse

- COCOMO 2 has lower cost estimate if we can reuse
  - Identify fractions of the reused system that need Design Modification (DM), Code Modification (CM), and Integration Modification (IM) to fit into the estimate
    - Design, Code, Integration Phases of project take 40%, 30%, and 30% on average effort in COCOMO II
  - Adjustment factor AAF := 0.4 DM + 0.3 CM + 0.3 IM
  - Adjusted KLOC for reuse SW, AKLOC := kLOC * AAF/100

- COCOMO 2 also captures 2 more factors for Reuse (next slide)
  - Quality of reused code, its "*Software Understanding*", SU
  - Effort needed to test applicability of reused code: "*Assessment and Assimilation*", AA
  - Equivalent kLOC for reuse SW, EKLOC := kLOC * (AAF + SU + AA)/100

Reuse Model

- Software Understanding, SU (COCOMOII Model Manual, 'A Reuse Model', Table II-5)
  - Structure: From "low cohesion + high coupling" to "strong modularity + information hiding"
  - Application clarity: Match between program and application worldviews
  - Self-descriptiveness: From "obscure code + missing/ obsolete documentation" to "documentation up-to-date + well-organized with design rationale"
  - From 50% to 10%

- Assessment and Assimilation AA (COCOMOII Model Manual, 'A Reuse Model', Table II-6)
  - From no effort (0%) to extensive module test & evaluation, documentation (8%)

Demonstration