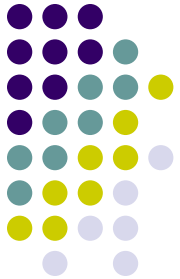
Professor Hausi A. Müller PhD PEng FCAE
Department of Computer Science
Faculty of Engineering
University of Victoria

www.engr.uvic.ca/~seng321/
courses1.csc.uvic.ca/courses/201/spring/seng/321

SENG 321 Calendar

Quiz 3: Course Experience	Wed, Mar 23	In class	2% of course
Deliverable C3	Thu, Mar 24	C3 feedback on S3a&S3b	10% of project
Easter break	Fri-Mon, Mar 25-28	Fri, Mar 25 no class	
Deliverable S4	Tue, Mar 29	S4 project demo (in ELL 167)	10% of project
Deliverable S4	Wed, Mar 30 & Fri, Apr 1	S4 project demo (in MAC D288)	10% of project
Deliverable C4	Fri, Apr 1	C4 feedback on S4 (in MAC D288)	5% of project
Last Day of Classes	Fri, Apr 1		
Final Exam	Sat, Apr 16	19:00-22:00 ECS 125	35%

Announcements



- Wed, March 23
  - Quiz 3
- Fri, March 25
  - Good Friday
  - No class
- Teaching evaluations
  - Until April 4

Final Exam
- Sat, April 16
- 19:00-22:00
- ECS 125

- Tue, March 29
  - In Elliot 167
    3:30-6:00 pm
    ←←←←←←←←←←
- Tue/Wed/Fri, March 29/30, April 1
  - In class and Tue lab demos
  - No labs on Thu
  - 3 presentations per hour
  - 15 mins per presentation
  - Evaluation form

CES—Course Experience Survey

- Your responses are important to me and future students taking the course; completing the CES is part of good university citizenship

- If you did not receive the invitation emails from UVic, simply visit: http://ces.uvic.ca/Blue. You'll be prompted to sign into UVic, then redirected to complete your CESs

- You can complete your CES until the last day of classes on April 4

Quiz 3—Course Experience

1. Describe in detail how selected requirements engineering topics discussed in this course topics could be beneficial for your career as an engineer or scientist. Note you are expected to answer this question even if you are aiming for a different career path than engineer or scientist.
2. What are selected requirements engineering learning outcomes and skills acquired in this course that you should consider including in your resume when applying for a job as an engineer or scientist. Note you are expected to answer this question even if you are aiming for a different career path than engineer or scientist.

Project Cost and Effort Estimation Techniques

- Seven traditional techniques for software cost estimation
  1. Algorithmic cost modeling
  2. Expert judgment
  3. Estimation by analogy
  4. Parkinson's law
  5. Pricing to win
  6. Top-down estimation
  7. Bottom-up estimation
- Some of these techniques are pathological (i.e., have problems built-in)!
  - ➔ Use more than one method

Comparing Techniques

Method	Strengths	Weaknesses
Algorithmic models	•Objective, repeatable, analyzable formula •Efficient, good for sensitivity analysis •Objectively calibrated to experience	•Subjective inputs •Assessment of exceptional circumstances •Calibrated to **past**, not future
Expert judgment	•Assessment of interactions, representativeness, exceptional circumstance	•No better than participants •Biases, **incomplete recall**
Analogy	•Based on representative experience	•How representativeness is the experience?
Parkinson's Law	•Correlates with some experience	•Reinforces poor practice
Price to win	•Often gets the contract	•Generally produces large cost overruns and losses
Top-down	•System level focus •Efficient	•Less detailed based •Less stable
Bottom-up	•More detailed basis •More stable •Fosters individual commitments	•May overlook system level costs •Requires more effort

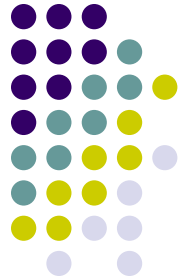
Combining Techniques

- Motivation
  - Each technique has advantages and disadvantages
- For large projects, several techniques should be used in parallel and results continuously compared.
  - If the results predict radically different costs …
    - … more information should be sought … and costing process repeated
- Process should continue until estimates converge
- Traditional cost models assume the existence of a firm set of requirements and a well-developed specification
  - Costing carried out with a solid SRS as a basis
  - Sometimes requirements and/or specification are changed to make sure that fixed costs are not exceeded

Algorithmic Models

- Systematic approach
- Not necessarily the most accurate
- Mathematical formula used to predict costs
- Based on estimates of
  - Project size
  - Number of programmers
  - Other process and product factors

Algorithmic Models



- Have exponential component
  - costs do not normally increase linearly with project size.
- As software size increases, extra costs incurred
  - communication overhead of larger teams
  - more complex configuration management
  - more difficult system integration

Highly Recommended Reading
Traditional Software Metrics
ftp://ftp.sei.cmu.edu/pub/education/cm12.pdf

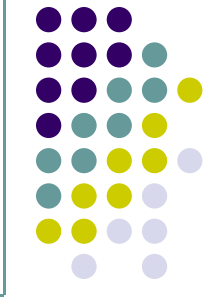
Algorithmic Models

- General form of equation: $E = a + b \bullet KLOC^{c}$
  - a = constant factor; local organizational practices, type of software
  - *b* = multiplier includes process, product, and development attributes
  - *KLOC*: measure of software size (lines of code)
  - *c* usually between 0.9 and 1.5
- Halstead [1977]: $E = a + 0.7 \bullet KLOC^{1.50}$
- Boehm [1981]: $E = a + 2.4 \bullet KLOC^{1.05}$ TRW
- Walston-Felix [1977]: $E = a + 5.2 \bullet KLOC^{0.91}$ IBM
- RADC [1977]: $E = a + 4.86 \bullet KLOC^{0.976}$ Rome Air Develop. Center
- Doty [1984]: $E = a + 5.28 \bullet KLOC^{1.047}$ Doty Associates
- JPL [1981]: $E = a + 2.43 \bullet KLOC^{0.962}$ Jet Propulsion Lab

Analysis of Algorithmic Models

- All models suffer from same basic difficulties
- Difficult to estimate size at an early project stage
  - Usually only specification available at this time
  - Function point and object point estimates are easier to produce than code size … may also be inaccurate
- Estimates of *b and c are highly subjective*
  - Depend on background and experience and vary from
    - One person to another
    - One model to another
    - One company to another
    - One domain to another

Estimating Size



- Most commonly used metric: lines of source code (LOC)
  - Measure of finished system … but of course, we don't have the system yet
- Size estimation therefore involves estimation by
  - Analogy with other projects
  - Estimation by ranking sizes of system components, using known reference components to estimate size; measure previously developed system to estimate the model parameters a, b, c
  - Application of engineering judgement
- Code size estimates uncertain because of dependencies
  - Hardware choices
  - Software choices
  - Commercial DBMS choices
  - Middleware choices

Functions Points
A better measure than KLOC

- Related to functionality of software rather than size
- Function points computed by counting the following characteristics
  - External inputs and outputs
  - User interactions
  - External interfaces
  - Files or databases used by system
- Each characteristics is individually assessed for complexity
- Each characteristics is given weight for complexity
  - 3 for simple external inputs
  - 15 for complex internal files

Counting Functions Points

- Count function points
  - Multiply each raw count by estimated weight, then sum all values
- Next multiply with project complexity factors
  - Degree of distributed processing, amount of code reuse, performance
- Function point analysis can be combined with LOC estimation techniques
  - Function points used to estimate final code size
- Uses historical data
  - AVC: average number of lines of code required to implement one function point
  - *Code Size = AVC * Number of function points*
- Advantage
  - Easier to estimate points than LOC early in the development process
  - Can be reaadily done with a completed SRS

COnstructive COst MOdel (COCOMO) Barry Boehm

- COCOMO is an algorithmic Software Cost Estimation Model
- The model uses a basic regression formula, with parameters that are derived from historical project data and current project characteristics

COnstructive COst MOdel (COCOMO) Barry Boehm

- One of first empirically-based models for effort and cost estimation
  - Considers wide variety of factors
  - Provides values for *b*, *c* and project size (KLOC or proxy) of effort equation
- Projects fall into three categories
  - Organic: small team, known environment
  - Semidetached: intermediate category—mix of experience, may be large project but not excessively so
  - Embedded: inflexible and constraining environment

COnstructive COst MOdel (COCOMO) Barry Boehm

Project Type	Size	Innovation	Deadline / Constraints	Development Environment
Organic	Small	Little	Not tight	Stable
Embedded	Large	Greater	Tight	Complex hardware Custom interfaces
Semi-detached	Medium	Medium	Medium	Medium

COnstructive COst MOdel (COCOMO) Barry Boehm

Type	Organic	Semi-detached	Embedded
b	2.4	3.0	3.6
c	1.05	1.12	1.20

- Mode and effort formulas
  - Organic: $E = 2.4\ size^{1.05}$
  - Semidetached: $E = 3.0\ size^{1.12}$
  - Embedded: $E = 3.6\ size^{1.20}$

$$E = a + b \bullet KLOC^{c}$$

- Examples: size = 200 KLOC
  - Organic: $E = 2.4\ (200^{1.05}) = 626$ staff-months = 52 staff-years
  - Semi-detached: $E = 3.0 * (200^{1.12}) = 1133$ staff-months
  - Embedded: $E = 3.6 * (200^{1.20}) = 2077$ staff-months

COCOMO Intermediate Model

- Intermediate model uses size plus 15 other cost drivers
  1. Software reliability
  2. Size of application database
  3. Complexity
  4. Analyst capability
  5. Software engineering capability
  6. Applications experience
  7. Virtual machine experience
  8. Programming language expertise
  9. Performance requirements
  10. Memory constraints
  11. Volatility of virtual machine
  12. Environment
  13. Turnaround time
  14. Use of software tools
  15. Application of software engineering methods

COCOMO Intermediate Model

- Mode + effort formulas
  - *d*: additional cost drivers parameter
  - Organic: $E = 2.4 * (size^{1.05}) * d$
  - Semidetached: $E = 3.0 * (size^{1.12}) * d$
  - Embedded: $E = 3.6 * (size^{1.20}) * d$
- Examples: *size* = 200 KLOC
  - Cost drivers
    - 0.88 Low reliability
    - 1.15 high product complexity
    - 1.13 low application experience
    - 0.95 high programming language experience
    - $d = 0.88 * 1.15 * 1.13 * 0.95 = 1.086$
  - Organic: $E = 2.4\ (200^{1.05}) * 1.086 = 906$ staff-months
  - Semi-detached: $E = 3.0 * (200^{1.12}) * 1.086 = 1231$ staff-months
  - Embedded: $E = 3.6 * (200^{1.20}) * 1.086 = 2256$ staff-months

$$E = a + b \bullet (size^{c})d$$

Summary—Project Cost and Effort Estimation Techniques

- Variety of cost estimation methods algorithmic estimation models based on real data and experience (e.g., IBM, TRW, JPL)
  - Challenge: how to translate experience to local situation
  - Some success with well-specified problems/domains
- Depend on solid requirements at estimation time
  - For many situations this is impossible
- Algorithmic models are from late Seventies & early Eighties
  - Today we have very powerful programming environments and reuse strategies and generate a lot of code—UI, middleware, libraries
- Recommendation
  - Combine techniques; one technique alone never suffices

SENG 321 Résumé Entries

- Requirements engineering
  - Requirements process
    - Elicitation, Analysis, Specification, Validation
  - Methods, techniques, and tools
    - Use case modeling techniques
    - Domain analysis and modeling
    - Review techniques
      - Walkthroughs
      - (Formal) inspection and validation
      - Inspection meetings
      - Inspection checklists
    - CRUD (Create, Read, Update, Delete) Matrix

SENG 321 Résumé Entries

- Requirements engineering
  - Methods, techniques, and tools
    - Working with UML
      - UML 2.5 (14 diagrams)
      - Structural and behavioural diagrams
      - Use case, class, interaction, sequence, state, collaboration, activity diagrams (tutorial)
      - Use of UML tools (tutorial)

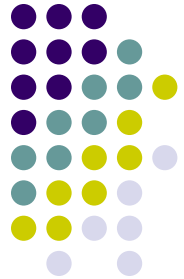
SENG 321 Résumé Entries

- Software life cycle models
  - Waterfall and Spiral models
- Requirements analyst
  - Interface between customers and developers
  - Requirements specification documentation skills
  - IEEE Std 830-1998 Requirements Standard and Specification Template
  - Documentation skills
    - Visio (tutorial)
    - Project (tutorial)
- Project Cost and Effort Estimation Techniques
  - COCOMO model (Barry Boehm)

SENG 321 Résumé Entries

- Codes of Ethics
  - APEGBC
  - ACM Software Engineering
- Communication and management skills
  - Presentation skills
  - Teamwork
  - Organization skills
  - Leadership skills
  - Management skills
  - Project management skills
  - Time management skills

Final Exam SENG 321 Format and Materials



- Format
  - 3 hours
  - Closed books, closed notes, no gadgets
  - The same format as the midterm
  - Mostly essay style questions
- Slides
  - 600+ slides posted on the course website
- Midterm
  - Similar format and questions
  - A couple of questions from midterm (e.g., major phases)

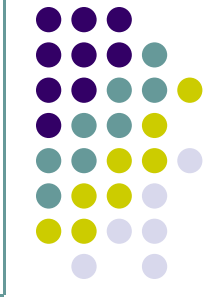
Final Review

- Use case modeling techniques
  - Use case scenarios
  - Process for identifying use cases
  - Use case template
  - Use case diagrams
  - Context diagrams
  - Use case mistakes and limitations
- Review techniques
  - Walkthroughs
  - (Formal) inspection and validation
  - Inspection meetings
  - Inspection checklists
- CRUD (Create, Read, Update, Delete) Matrix
  - Develop a CRUD matrix for a well-known scenario
    - For example, bank checking account

Final Review

- Codes of Ethics
  - APEGBC
  - ACM Software Engineering
- UML overview
  - Structural and behavioural diagrams
  - History of UML
  - What do you know about UML and its history?
  - Explain the uses of the 14 diagrams in UML 2.0
- Project cost and effort estimation techniques
  - Techniques
  - Parameters
  - Algorithmic models
  - Comparison of techniques
  - COCOMO model (Barry Boehm)
  - Contrast different project cost and effort estimation techniques

Course Discussion



- What caught your eye in this course?
- What have you learned in this course?
- How does this course prepare you for your work in industry?
- How was your group experience?
- What would you do differently?
- Was this course a character-building experience for you?
- What surprised you in this course?
- What did you learn about yourself?
- What did you learn about your team players?


Requirements Elicitation
Requirements Analysis
Requirements Specification
Requirements Validation
Requirements Techniques
The End