ECE450 – Software Engineering II

Today: **Estimation**

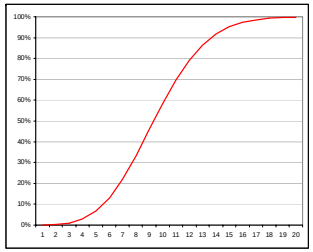
"It's tough to make predictions,
especially about the future"
-Yogi Berra

Software Estimation

- What is an estimate?
  - A *prediction* regarding the effort required to complete a project
  - Might take one of several forms:
    - *Person-months:* Project X will need 26 person-months to complete
    - *Dollars:* Project X will cost $2 million
    - *Time:* Project X will be finished in one year
    - *Features:* Given the time and money we have, we will deliver features *a,b,...,g* in this release of project X
  - All of the above can also be given as *intervals*
    - E.g., Project X will cost between $1.8 and 2.5 million

Software Estimation

- What is an estimate? *(cont.)*
  - An essential but overlooked characteristic of estimates is that they have a *probability* of being true
  - This is a source of conflict:
    - Academics often mean "estimate = 50% likelihood"
    - Managers often mean "estimate = 80% likelihood"
    - Developers often mean "estimate = most optimistic outcome" (about 10% likelihood!)



Software Estimation Woes

- Estimation woes 1 – Estimates as wishful thinking
  - "When will my car be ready?"
  - "By tomorrow afternoon. Thursday morning for sure."
    - *Meaning:* In **theory**, I could fix your car by tomorrow afternoon. This implies:
      - ...that I can find supplies
      - ...that no urgent jobs come up
      - ...that I diagnosed your car's problem correctly
      - ...that I don't get lazy or sick tomorrow, and arrive here on time
      - ...that none of my tools breaks down
      - ...that there won't be weather problems, nor electrical blackouts
    - "And since I can't guarantee all of that, I'm giving myself a half-day buffer. Should be enough."
  - *Real meaning:* It'll be ready in two months.

Software Estimation Woes

- Estimation woes 2 – Estimates as guessing games
  - "Scotty, what's the problem with the warp drive?"
  - "It's broken, captain."
  - "How long will it take to fix it?"
  - "Seven hours. Maybe eight."
  - "Seven hours?! You got fifteen minutes."
  - "Yes sir."
    - *Meaning:* I didn't really want an estimate. I wanted you to *guess* the answer I was thinking of, you fool.
      - I assume your estimate was a bargaining chip. Maybe you're lazy and wanted to buy some procrastination time.
      - I can make you bend reality if I pressure you hard enough.
  - *Real meaning:* It'll be fixed in eight hours. Maybe twenty.

Software Estimation Woes

- Estimation woes 3 – Estimates as negotiation tools
  - "Scotty, **now** what's the problem with the warp drive?"
  - "It's broken again, captain."
  - "How long will it take to fix it this time?"
  - "... ehem... Twelve days, captain. This one is hard."
  - "What?! That's insane! You got **ten hours**!"
  - "OK sir."
    - *Meaning:* I learned to play the game. Whatever I tell you you'll just cut it down irrationally. So I'll blow it up irrationally too.
      - NOW my estimate was a bargaining chip. I won't ever give a candid estimate anymore, thanks for the lesson.
  - *Real meaning:* It'll be fixed in eight hours, again. Maybe twenty.

Software Estimation Woes

- Estimation woes 4 – Self-fulfilling prophecies
  - "Hmm, what do you know, the warp drive thing seems simpler than I thought this time. This one could *actually* be fixed in under four hours!"
    - So that means I have six extra hours. Let's see if I can also fix that rattling sound that's been bugging me. And I'll bring the new guy to train him on how to fix the warp drive. And...
    - *Meaning:* The captain said I had ten hours, so I'll use ten hours.
    - Parkinson's law: Work expands to fill the time available.
      - The reason why almost no project ends *before* its estimated time
  - *Real meaning:* It'll be fixed in eight hours, maybe twenty.

Why is it hard?

- As we've seen, estimates (which are *predictions* with a certain degree of *probability*) are often treated as
  - Wishful thinking
  - Guessing games
  - Negotiation tools
  - Self-fulfilling prophecies
- Other problems:
  - *The Mythical Man-Month*
  - Just about everything can go wrong
  - Huge variability in individual and team performances
  - Radical design can't be estimated properly
  - Poorly stated requirements, moving goalposts
  - Really, software developers are romantics at heart!

Software Estimation Strategies

- There are dozens of techniques, but only a few strategies:
  - Model-based strategies
    - Fit software development into a mathematical model, use model's formulas to find estimate
  - Analogy-based strategies
    - We've done this before, it's reasonable to expect we'll perform similarly
  - Expert-based strategies
    - Estimation is too complex to model, so use all the tacit knowledge in experts' heads instead

Software Estimation Strategies

- Model-based techniques
  - Examples: COCOMO, SLIM, Checkpoint
  - Default academic idea of what estimation should be like
  - Key ideas:
    - Study the performance of previous projects around the world
    - Find the relevant variables that predict performance
    - (Essential variable is often a measure of *size)*
    - Summarize your findings in a mathematical model
  - Assumptions
    - Software development fits a mathematical model
    - ...and we can find the model's equations
    - Size and effort are strongly correlated
    - People are better at estimating size than effort *(proven wrong!)*
  - Results: Poor, although calibration is helpful

Software Estimation Strategies

- Model-based techniques *(cont)*
  - COCOMO
    - Effort = $a(\text{KLOC})^b$
      - (in person/months)
    - Development time = $c(\text{Effort})^d$
      - (in chronological months)
    - People required = Effort / Development time
    - *a, b, c,* and *d* depend on the characteristics of your project and personnel
      - Details in "Software Engineering Economics", by Boehm (1981)
    - Note reliance on kilo-lines of code
      - "The use of lines of code metrics for productivity and quality studies (should be) regarded as professional malpractice" –Capers Jones

Software Estimation Strategies

- Model-based techniques *(cont)*
  - COCOMO2 fixes the LOC problem by switching to *function points*
    - Function points are a *much* better technique to assess size than LOCs
    - Still requires skill to learn how to do it
    - Fundamentals: List number of instances of each of the following:
      - External inputs
      - External outputs
      - External inquiries
      - Internal logical files
      - External interface files
    - Each item should be classified as {high, medium, low} complexity
    - Adjust for your team's capabilities and project characteristics
    - The process will output a number of FPs, which substitutes KLOCs
    - Calibration is still essential!
      - Be careful with outliers

Software Estimation Strategies

- Analogy-based techniques
  - Key idea:
    - Look at our past performance to figure out our future performance
  - Assumptions:
    - We're doing something similar to what we've done before
    - Risks won't bite us, just as they haven't bitten us before
    - *Ceteris paribus*
  - Results: Much better than model-based techniques for known territory (normal design), poor otherwise (radical design)

Software Estimation Strategies

- Expert-based techniques
  - Examples: Work Breakdown Structure, Delphi
    - WBS: Partition, and estimate the pieces
    - Delphi: Gather a group of experts, have each submit an estimate, announce results, let them submit another estimate, keep the mean
  - Key idea:
    - Estimation is so complex, and it depends on so much tacit knowledge, that we won't attempt to model it – just leave it to the experts
  - Assumptions:
    - Humans are better at handling uncertainty than models or tools
  - Widespread use in industry
    - 62-85% use it as their primary estimation technique
    - (versus 10% for models)
  - Bad reputation in academia (often referred to as "mere guessing")
  - Results: Highly variable on the experts' **real** estimation expertise

Software Estimation Strategies

- Expert-based techniques *(cont)*
  - There are several problems with software estimation and human judgment:
    - Estimators do not distinguish between 50%, 90%, 99% confidence intervals
    - Managers prefer estimators that give narrow ranges, *even if they are wrong!*
    - Customer expectations play a role in the outcome of estimation processes
    - *Anchors* bias our responses ("will you be done in a week?")
    - Years of experience are not necessarily a good indicator of accuracy
  - "Everyone complains of his memory, but nobody complains of his judgment" – *La Rochefoucauld*
  - However, expert-based estimation has been shown to be, on average, at least as effective as model-based estimation

Suggestions

- **Use more than one method!**
  - If possible:
    - Use function points (or a similar metric)
    - Compare vs. past performance
    - Adjust if things seem off
- Shield yourself from *anchors*
  - Try *not* to know what your customer is expecting to hear
- Choose a project lifecycle that manages schedule risk
  - Incremental models
- Give estimates with wide margins, especially at the beginning
  - You can also use coarser units (e.g., quarters instead of months, months instead of weeks)
- At the end, analyze your estimation accuracy and adjust your techniques. This feedback loop is essential to get better at it!