Software Estimation

Pt I: Critical Estimation Concepts

Not hard, but not intuitively obvious.

Sophisticated research, move from ±10% → ±5%.

Typical Orgs can be 100% off.

Art vs. Science: Use rule of thumb, aim for 25% or less.

---

1. What is an estimate?

Different language between devs / manager / execs.

estimate / target / commitment.

A target is a statement of a desirable business outcome.

A commitment is a promise to deliver some quantity and quality of software, by a given date.

Estimate vs. plan,

↓

No preconceived notions or desires about how we want the outcome to look.

Plan - actively include our biases & priorities.

We plan specific means to reach a specific end.

estimates are the foundation for a plan

Planning considerations that depend on good estimates:

- Making detailed schedule
- Identifying a projects critical path
- Prioritising functionality for delivery
- Breaking a project into iterations

There's a limit to how well a project can go, there's no limit to how badly it can go.
An estimate for a project should have some concept of range, or probability.

Good Estimates are within 25% of result, 75% of the time.

±10% possible on well controlled projects.

Estimates & project control.
By measuring you change the estimate.
Start of project, do an estimate
But project changes as time passes things get added & removed, but if it delivers roughly the features to cost, then it "met it's estimate" regardless of changes.

Initial target & Initial estimate should be within 20% of each other, and we can make things work.

A good estimate provides a clear view of the project reality, to allow leadership to make good decisions about how to control the project to hit it's targets.

* Don't provide precise estimates like 90% unless you have a solid statistical basis for it.
* Avoid using artificially narrow ranges, and be sure the ranges accurately reflect your confidence level.
* If you feel pressure to make your estimates narrower, verify that this pressure is indeed coming from outside your team & yourself.

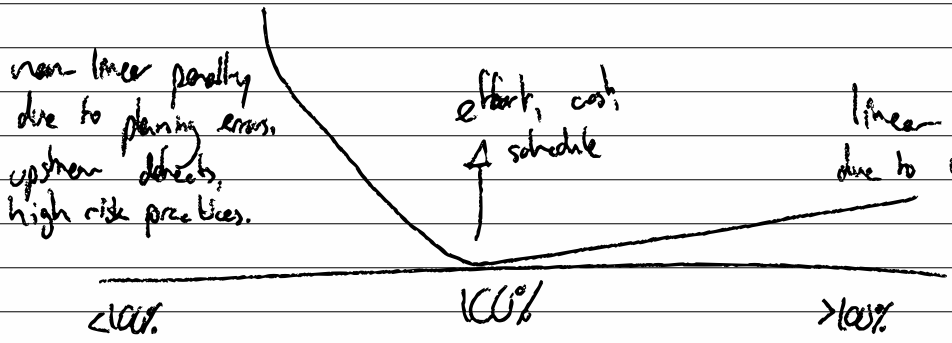
Overestimates vs. Underestimates.

Parkinson's Law - work will expand to available time.

Goldratt's "Student Syndrome" - given too much time, developers will procrastinate until late in the project, then rush to complete it, likely not on time.

Sometimes force urgency on team to get things done faster.

- Underestimating: Reduced effectiveness of project plans
- Statistically reduced chance of on-time completion.
- Poor technical foundation leads to worse than nominal results: may have to re-work requirements & design.
- Destructive late-project dynamics make the project worse than nominal: - more status meetings w/ upper mgmt to discuss how to get back on track.
- Frequent re-estimation late in project, to determine completion date.
- apologies to customers for lateness.
- Preparing interim releases.
- More discussions about which requirements absolutely must be added.
- Finally fixing problems arising from quick & dirty workarounds.



Don't intentionally underestimate.

Penalty for underestimation is more severe. Non-linear and unbounded!

Software project success is correlated w/ project size.

The error of estimating is vastly more due to underestimation.

Benefits of accurate estimates:

- improved status visibility / Compare plan with progress
- higher quality - accurate estimates help avoid schedule-related-stress.
  40% of all software errors are caused by stress.
- better co-ordination with non-software functions.
- better budgeting
- increased visibility for dev team.
- early risk information.

Don't allow a business target to be confused with the software estimate.
May instead change schedule / cost / functionality.

Projects will prioritise generic goals of schedule/cost/functionality + some specific goals.

Agile development focuses on flexibility, repeatability, robustness, sustainability and visibility.

CMM (Capability Maturity Model) (formality & optimisation) instead focuses on efficiency, improvability, predictability, repeatability & visibility.

Businesses prefer predictability of cost/schedule/functionality, over ability to change its mind.

Since estimation is so poor, should examine existing techniques. Do not use personal memory of similar projects, but this is the dominant strategy for 60-85% of all estimates!

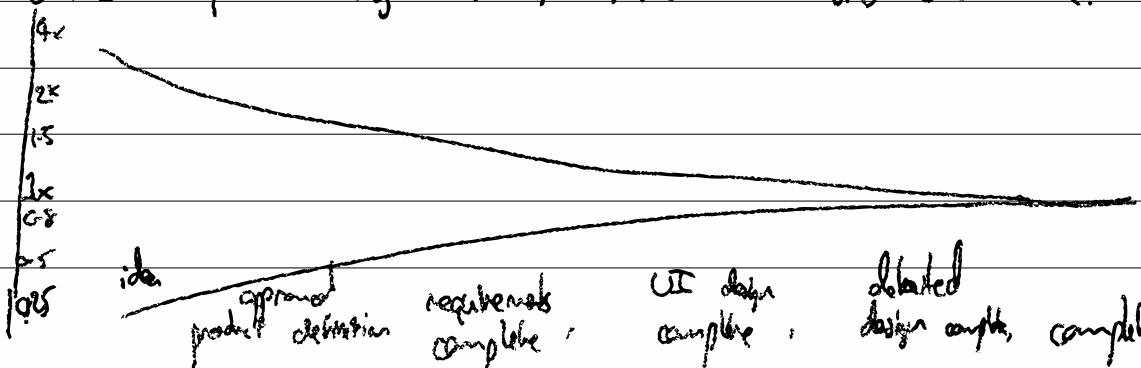
Where does Estimation Error Come from?

Error comes from 4 generic sources:
- Inaccurate info about the project being estimated.
- Inaccurate info about capabilities of the org doing the work.
- Too much chaos in the project to support accurate estimation.
- Inaccuracies arising from the estimation process itself.

There can be a factor of 10 difference in:
- design complexity in the same feature.
- different devs to code the same feature.
- number of defects in original implementation by different devs.
- time for different devs to correct & debug same problem.

These can all combine to much larger magnitude.

A constant process of resolving decisions gives very high uncertainty to begin with, which diminishes over time.



The more refined the definition, the more accurate the definition.

You can't beat this cone of best estimation, but you might be lucky.

Initial concept:	0.25x – 4x
Approved product definition:	0.5x – 2x
Requirements complete:	0.67x – 1.33x
UI design:	0.8x – 1.25x
Detailed design complete:	0.9x – 1.1x
Done.	1x

Time ↓

Cost / Effort

Cone only narrows as you make decisions that eliminate variability.

Common examples of project chaos:
- Requirements not well investigated
- Lack of end-user involvement in requirements validation
- Poor designs that lead to numerous errors in the code
- Poor coding practices give rise to extensive bug fixing
- Inexperienced personnel
- Incomplete or unskilled project planning.
- Abandoning planning under pressure.

Best way to address these issues is better project control. no better estimation will help a chaotic project.

Project managers often neglect to update cost/schedule estimates when requirements change.

You can't estimate a chaotic process - as a first step control the chaos.

Project control is a more powerful response than estimation responses. If you can't stabilise requirements, alternative development approaches like Scrum, XP, Agile ....

Omitted Activities

Most common source of estimation error, is forgetting to include necessary tasks in the project estimate.
Both at project level & individual developer level.
Devs accurately estimate their work, but omit other tasks, easily.
Omitted work in 3 categories:

- missing requirements
- missing development activities
- missing non-development activities.

Don't reduce developer estimates, they're probably overly optimistic anyway.

More control knobs for estimation is not better.

Off the cuff estimates are much less accurate than reviewed estimates Don't do it.

Less common sources of error:

- unfamiliar business
- unfamiliar technology

Diseconomies of Scale:

Large projects require more communication between more people. Increases exponentially as more people are added.

Don't assume that effort scales up linearly as project size does.

Different kinds of software are developed @ different speeds. Intranet sites vs. aviation code.

You can't accurately estimate a project if you don't know who will be doing the work. Individual perf. can vary by a factor of 10! Getting it wrong will give awful estimates.

High-level languages are faster than low-level.

C → C#/Java → Perl/Python → SQL.

1x 2.5x 6x 10x

Most Significant features for estimation are:

- Product Complexity
- Requirement Analyst Capability
- Programmer Capability

Product Complexity - Type of software you're building.

Requirements Analyst Capability - Competency here has single biggest personnel potential to reduce project estimate.

Part 2:

Fundamental Estimation Techniques

Estimating size refers to scope of technical work of a given feature set.

Project size: small: 5 or less technical staff.
Can use statistical methods because individual productivity variations drown out other factors.

Large: 25 people or more, 6-12 months or more.
Statistical methods to start with, when individuals contributing are not known, just: X web devs
Y DBAs
Z Architects
Top-down approach.

∴

bottom-up approach when you know the people involved & the work is much more well defined.

Medium: 5-25 people, 3-12 months.

2 development styles: Iterative & Sequential

What % of reqmts are defined up front,
vs. % defined after construction is underway.

Development stages:
- early: beginning → requirements mostly defined.
- middle: first 2-4 iterations of building (not each.)
- late: mid-construction to release.

Applicability of Techniques.

	Group Reviews	Calibration w/ Project Data
What's estimated:	Size, Effort, Schedule, Features	
Size of project:	- M L	S M L
Stage:	Early-Mid	Mid-Late
Accuracy Possible:	Med-High	High
Iterative/Sequential?	Both	Both

Count, Compute, Judge

Sometimes you have data you can actually count & use.
If you can't count the answer directly, you should compute the estimate from components.
Judgement alone should be the last resort.

Software projects produce numerous things you can count: - ex. marketing requirements
- features
- use cases
- stories

In the middle of the project, there are more specific things you can count: - eng. requirements, - web pages
- function points - reports
- change requests - database tables

Late in the project, can count even more specific things:
- code already written - tasks
- defects repaired - LOC / classes

decide what to count based on a few goals

Look for something you can count sooner rather than later, in development process.

Find something to count that's highly correlated w/ the size of the software you're estimating.
Must be relevant to the project!

Try to count something that will produce a statistically meaningful average. (20+ items).

Find something easy to count.

Collect historical data that allows you to compute an estimate from a count.

Expert judgment is the least accurate measure.

Historical data + computation is free of biases.
Don't tweak these estimates to conform to your professional judgement.

Organisational Influences:

- software complexity & time constraints
- committing to stable requirements.
- can we remove problem team members?
- team free to concentrate on the project? Or will there be interruptions from other work?
- Can we add team members as planned?
- Can we depend on project members staying, or do we have high turnover?

Using data avoids subjectivity & unfounded optimism.

It's difficult to change organisational productivity, so assume the next project will go the same as the last one - don't allow optimistic folks to suggest for any reason it'll somehow go better.

"Productivity is whatever the data says it is."

Data to collect:

- Size. (LOC / something else)
  Counted after software is released.
- Effort (staff months)
- Time (calendar months)
- Defects (classified by severity)

Useful even after 2-3 projects going forward
Start small, but start.

Issues relating to effort measure)

- Do you count time in hours / days / other?
- How many hours / day do you count? & or actual hours applied to project.
- Do you count unpaid overtime? holidays? training?
- Do you make allowances for company meetings?
- What kind of effort do you count?
  - Testing 1st level mgmt? Documentation? Requirements? Design? Research?
- How do you count time that's divided across multiple projects.
- How do you count time spent supporting sales calls / shows?
- How do you count fuzzy-front-end time? Firming up requirements,

Issues related to Calendar time measures:

- What signifies the project start? Initial discussion? / Fully staffed?
- When does the project end?

Issues relating to defect measures:

- Do you count change requests as defects? or only those that are ultimately classed as bugs?
- Do you count coding defects found before alpha / beta release?
- Do you count defects found after release? When the project is already considered "done"?

With historical data, start small.
Easiest to collect while you go, or on periodic basis.

Calibration. Trying to convert the data you collect into a model.

- our devs write X lines per staff month.
- a 3 person team can deliver X stories per calendar month.
- Our team is averaging X staff hours per use case to create the use case, or Y staff hours to deliver it.
- Testers create test cases at a rate of X hours per test case.
- Y lines of code per function point. - Defect correction avg X hours

These models are all linear though.
Use data from the actual project to estimate other tasks currently in the project.
Should aim to switch from org. average data to project specific data as soon as possible.

Individual Expert Judgement.

- Use of structured process
- Use of estimation checklist
- Estimating task effort in ranges.
- Comparing Task estimates to actuals.

By far most common, 5/6 estimators use it as primary technique.
Make most effective use of your expert. They know the tasks & stuck but are they being fully used? Using a good checklist & taking everything into consideration?

Intuitive 'judgement' vs. Structured judgement.

Estimates from people actually doing the work will be most accurate. (No surprise).
But still, higher ups will sometime try to override.

Split large tasks into smaller tasks before estimating. No task should be over 2 days effort. Larger tasks contain too many places where work can hide.

Often use ranges: Best & Worst Case.
These compared to single figure estimates, highlight the problem w/ optimism.

PERT - Program Evaluation & Review Technique
Use expert judgment for best case, most likely, worst case, and calculate:

$$\text{Expected Case} = \frac{\left[\text{Best Case} + \left(4^{x} \text{ Most likely Case}\right) + \text{Worst Case}\right]}{6}$$

But, estimates are often optimistic, so can adjust slightly. Until estimates are more accurate, try:

$$\text{Estimate} = \frac{\text{Best} + 3x \text{ Most likely} + 2x \text{ Worst}}{6}$$

Checklists for individual Estimates:

- ☐ Clearly defined what's being estimated?
- ☐ does estimate include all kinds of work needed for completion?
- ☐ does the estimate include all functionality areas?
- ☐ is estimate broken down into small enough chunks to expose hidden work?
- ☐ did you look @ documented facts from past work? (not memory)
- ☐ is it approved by the person who will do the work?
- ☐ is productivity assumed based on similar work?
- ☐ does it include best/worst/most likely cases?
- ☐ is worst case really the worst case? should it be worse?
- ☐ is the expected case computed appropriately?
- ☐ have assumptions been documented?
- ☐ has situation changed since estimate was prepared?

Then need to compare estimates to actuals.
Fill in actuals when work is considered done.
Calculate

$$\text{Magnitude of Relative Error} = \text{Abs}\left(\frac{\text{Actual} - \text{Estimate}}{\text{Actual}}\right)$$

For all tasks, check this, check if best < actual < worst.

As estimates improve, should see Avg. MRE ↓.
should see % actuals within range ↑.

When comparing, try to understand what went right, what went wrong, & adjust.
Essential to have a feedback loop & collect more data.

Decomposition & Recomposition.

- by Feature / Task
- by Work breakdown structure
- Computing best & worst cases from std. dev.

---

break it down, estimate each bit seperately, add.
Great idea, just avoid pitfalls.
Takes advantage of the law of large numbers.

Decomposition via activity-based Work Breakdown Structure.
Small - to - medium Projects.

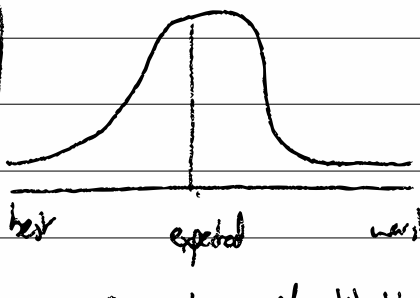
Create → Plan → Manage → Review → Rework / Defects.

$$\text{Std Dev} = \frac{\text{Sum (Worst Case Est. - Best Case Est.)}}{6}$$

For less than 10 tasks. Guess.
For more, should apply the formula to each task.
Compute variance, sum, sqrt.

$$\text{Std. Dev} = \sqrt{\sum \left(\frac{\text{worst} - \text{best}}{6}\right)^2}$$

variable based on accuracy.



use std. dev. to provide likelihood

Estimation by Analogy.

May have a prior project to compare to, can use its constituent components to get estimates for new project components.

Can then address uncertainty by expressing the estimate in uncertain terms (± 25%).

Ultimately, impact of uncertainty will flow through to business plans & commitments.

Can move commitments to act conservatively, but should not **bias** estimates

Proxy-based Estimates

- Fuzzy logic
- Standard Components
- Story Points
- T-shirt sizing

Can't measure what you really want?
Find something which acts as a proxy, and use historical data model to convert proxy value to rough desired value, and count, don't guess.

Useful for whole project / whole iteration estimates, but not for detailed task-by-task / feature-by-feature estimates.

halfway through

Fuzzy logic

Estimators can usually classify features into very small, small, med, large, v large.

Feature size	Avg. lines / feature	Nr. features	Proxy Estimate LOC
V. small	125	23	125 × 23
small	250	15	250 × 15
med	500	10	500 × 10
large	1000	30	1000 × 30
v. large	2000	8	2000 × 8
			≈ 60k

Works best w/ sizes calibrated from past historical data.

Important to all be on same page about how to categorize size of a feature.

Train estimators so they classify features correctly.

Document specific criteria for each size so they can be learned & sized appropriately.

Need at least 20 features to make use of this.

Can also use proxy for standard components:

eg Dynamic Web Page
Static Web page
DB tables
Reports
Business Rules

Same ideas, but can make use of best, likely, worst historical data

Doesn't rely on counting, but judgement, so can be flawed.

Story Points - a variation of fuzzy logic.
Normally assigned a numeric value instead of ordered category.
Powers of 2 or Fibonacci Sequence
1, 2, 4, 8, 16 1, 2, 3, 5, 8, 13

They're a unitless measure: doesn't translate into LOC, staff days, calender time. But all done at the same time using the same scale, free from bias.

Plan next iteration, where story points ≃ effort.
Can then calibrate to past iterations

Problem w/ numeric stories is the numbers imply a ratio, and unless very skillfully applied, will not be accurate.

Numeric story points may be no more valid than ordered categories v. small → v. large.

T-Shirt Sizing

Goal of software estimation is not pinpoint accuracy, but estimates accurate enough to support effective project control. T-shirt sizing is great for non-technical folk, who don't want staff hours, but an idea of cost of a feature. Classify feature sizes so you can give them out to sales/marketing. They understand value of a feature, so can prioritise requests accordingly.

Allows early project decisions.

Useful to combine cost w/ value to prioritise in a table.

Try to count & compute instead of using story points/t-shirt sizing if possible

Expert Judgement in Groups

Useful early on, estimating a large project, or large unknowns

- have each team member make their estimate individually, then compare.
- DON'T just avg. estimates & accept that.
  need to discuss differences in results.
- arrive at a consensus estimate that the whole group accepts
  If there's an impasse, can't go forward, must discuss.

Group reviewed estimates almost always more accurate.
Should have 3-5 experts w/ different backgrounds.

Wideband Delphi

Ineffective if not used properly, so follow the procedure.

1. Coordinator presents each estimator w/ spec & estimation form.
2. Prepare initial estimates individually.
3. Group meet, discuss estimation issues. If all agree, assign someone devil's advocate
4. Estimators give estimates anonymously to coordinator.
5. Coordinator prepares summary of estimates on iteration form.
6. meet to discuss variations in estimates.
7. Anonymously vote to accept estimate. Unanimous, or go to step 3.
8. Use values generated as expected case. Can also generate best & worst case.

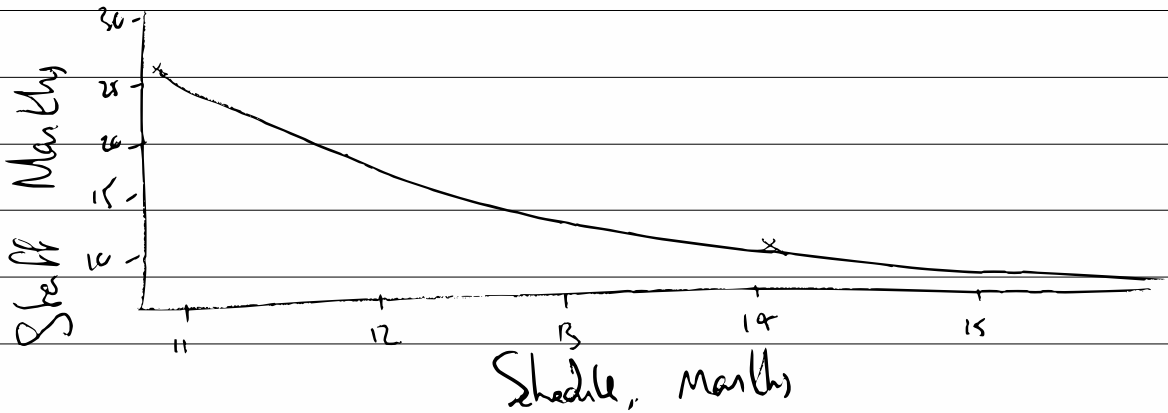
Wideband Delphi of group produces best initial estimates.

Not appropriate for detailed task estimates.
Expensive way to estimate, requires meetings.
Useful if you're working in a new business area, new tech, or new kind of software.
Useful for creating "order of magnitude" estimates at product definition or software concept time.

Tends to sharpen the definition of the scope of work.
Helps flush out estimation assumptions.

Wide cone of uncertainty, can be invaluable process.

Effort & time



Always present estimations to stakeholders as a range, which should narrow over time as the project progresses. Don't use point in time, as this will be subject to anchoring, even if the initial estimate isn't well handled.

Project is constantly being re-estimated, but can present re-estimates after each milestone/release.

Should feel confident in tightening your estimates.

Standardised Estimation Procedures.

Well defined process for creating estimates, which is adopted @ org level & provides guidance @ Individual project level.

- Protects against poor estimation practices, like off-the-cuff estimation & guessing.
- Protects against arbitrary estimation changes because a high-powered stakeholder doesn't like a specific result.
- Encourages consistency of estimation proces.
- Allows you to re-trace your steps in the event of a poor estimate.

Steps in a standardised procedure

- ☐ Emphasizes counting & computing over judgement.
- ☐ Calls for multiple estimation techniques & comparison of results.
- ☐ Communicates a plan to re-estimate @ predefined points in project.
- ☐ Defines how estimation approach changes over project.
- ☐ Contains a clear description of an estimate's inaccuracy.
- ☐ Defines when an estimate can be used as the basis for a project budget.
- ☐ Defines when an estimate can be used for int/ext. commitments
- ☐ Calls for archival of estimation data & review of process.

Must treat this as a Standard.

Should keep docs in a "Software Eng. Standards" page.
Fit it onto a stage-gate process.

Discovery → Scoping → Planning → Dev → Testing & Validation → Launch

Should co-ordinate Standard Estimation Procedure with SDLC.

Example Standardised Estimation Procedure for Sequential Project.

I. Exploratory Estimate (Approved Product Definition)

- Create one estimate from each of
  - bottom up work breakdown structure
  - standard components
  - estimation by analogy w/ similar projects
- Use Wideband-Delphi to converge to single point nominal value, N.
- Present estimates as a range, -50%, N, +150%
- Don't share N
- Not to be used for budget, only for moving to next stage.

II Budget Estimate (Product Design Complete)

Create new estimates using 2 approaches from step 1.

Work Breakdown structure & standard components.

Create function points estimate. Calibrate w/ historical data.

Iterate estimates until they're close together. Find avg. N.

Range will be say 0.8N – 1.25N.

Allocate N budget. 0.25N contingency budget.

III - Preliminary Commitment Estimate.

Detailed task list. Review by all.

ICs estimate effort required w/ Best / Worst / likely.

Calculate task nominals: Σ (Best + 4× likely + Worst) / 6.

Compare this with

Compute nominal estimate as $\frac{2\times \text{higher} + \text{lower}}{3}$

Compute estimate ranges as 1.0N – 1.1N.

Publish 1.1N externally as commitment. 1.0N internally.

IV - final commitment estimate (after several interim releases).

Re-estimate with calibrations using work done so far.

V- Project can be re-estimated at any time in response to Major changes in project assumptions.
change in requirements, staff availability, schedule targets.

VI Project Completion.
- Collect & Archive data on actual project results for future use.
- Review accuracy of estimates
  * Analyse root cause of any major events
  * Assess whether same accuracy could be produced w/ less effort.
  * Propose revisions to Standard Estimation Procedure.

Must have a period of reflection about estimates created, results achieved, accuracy of estimates.
Which techniques produced most accurate results.

Definitely look @ NASA,
nobody ever says "how long do you think this will take?" - All done by counting.

Part 3:

Specific Estimation Challenges

Special Issues in Estimating Size.

Measures of size:

- Features
- User stories
- Story points
- Requirements
- Use Cases
- Function Points
- Web Pages
- GUI components
- DB tables
- Interfaces
- Classes
- Functions / Subroutines
- **LOC** ← most common

LOC – easy to collect
- lots of historical data exists.
- effort for LOC is appear constant across programming languages
- measurements allow for comparisons across projects

---

* LOC/staff month does not take into account discrepancies of scale.
  error prone due to vastly different coding rules for different kinds of software.
* LOC can't be used to count or IC's assignments due to vast differences in productivity between programmers.
* different project code complexity may miscalibrate things.
* LOC difficult to estimate directly, must be estimated by proxy.

"LOC is terrible measure of size, but all the others are worse."

Special Issues in Estimating Effort.

Take estimates compute effort from a detailed task list.
early estimates are most accurate when computed from size estimates.

Biggest influences are - kind of software being built
- organisation productivity

Take an informal comparison to past projects by using schedules, size, effort, to compute productivity rates, size up effort required.

Can use Constrax tools.

Can use industry average LOC rates for your type of software! Very useful.

Combine estimates & take avg.

Weight data from your org more heavily than industry data.
Not all estimates are equal.

Special Issues in Estimating Schedule

Tasks on from effort.

Schedule in Months = $3 \times$ Staff Months$^{1/3}$

Coefficient will change w/ calibration.

Not modelled for small projects or take projects.

Assumes you're able to adjust team size to suit schedule.

Difficult to get an accurate schedule from effort using alt.

Best to use historical project data.

All researchers have concluded that shortening the nominal schedule will increase total development effort.

- Larger teams require more co-ordination & mgmt overhead.
- Larger teams → more communication paths, more chances to miscommunicate, meaning more errors, more correction.
  Shortest possible schedule produces most errors.
- Requires more work to be done in parallel. Some work will depend on unfinished / defective work. Adds re-work.

Cannot compress a nominal schedule by more than 25%.

Extending the schedule beyond Nominal schedule can reduce cost if you reduce team size.

Team size should ideally be 5-7.
Beyond that, diseconomies of scale begin to kick in, effort obviously increases as you add team members, but in 9-11 member, schedule also increased.
Beyond 15, schedule is flat, but effort dramatically increased.

Increased project size means higher percentage of time should be spent on architecture & system test.

	Small →	large
Arch	10%	20%
Build	70% →	45%
Test	20% →	35%

Roughly

Estimate Presentation Style

Essential to Document assumptions embodied in the estimate. Several categories.

- ☐ Which features are required, not required
- ☐ How elaborate certain features need to be.
- ☐ Availability of key resources
- ☐ Dependencies on 3rd party performance
- ☐ Major unknowns
- ☐ Major influences & sensitivities of the estimate
- ☐ How good the estimate is
- ☐ What the estimate can be used for.

Sets the expectation that the estimate is subject to variability.

Quantify the risks: +2 months if extra features.
-0.5 months if new dev tools works better than expected.

IEEE & ACM jointly decided:
"Software Developers have a professional responsibility to include uncertainty in their estimates."

Politics, Negotiation & Problem Solving.

Technical Staff are fairly good @ estimation, but poor at defending estimates.

3/4 technical staff introverts,

vs

1/3 of general population.

Executives often 10 years older, & more highly placed. Negotiation is part of their job.

Don't negotiate the estimate: it's founded in good practice, but can negotiate the commitment. It's problem solving not negotiation.

Estimate — Commitment — Target

Tech — (Joint) — Non-Tech

Estimation Sanity Check

- [ ] Standard procedure used?
- [ ] free from pressure that would bias result?
- [ ] if negotiated, was it purely inputs negotiated? not estimated prices itself?
- [ ] Expressed w/ precision matching its accuracy?
- [ ] Were multiple converging techniques used?
- [ ] productivity assumption realistic? Based on historical data?
- [ ] Is schedule realistic? at least $2 \times \sqrt[3]{\text{Staff Months}}$?
- [ ] Were workers involved in estimates?
- [ ] Reviewed by an expert estimator?
- [ ] Does it include non-zero allowance for risks on effort & schedule?
- [ ] Is it part of a series of estimates which will narrow as project progresses?
- [ ] Are all elements of project & work included?

10-12, highly accurate

7-9, maybe optimistic, but can provide project guidance

≤6 Shit. No meaningful use to project or anyone.