

Software Cost Estimation: Open Problems and Research Opportunities

Srinivasa Gopal
Ramanujan Society for Academic Research and Promotion of Science

About the Author



Srinivasa Gopal

Co-Founder – Ramanujan Society for Academic Research and Promotion of Science (https://ramanujan-society-academic-research-science.com/index.html).

Education :

B.Tech in Mech Engg , Indian Institute of Technology, Madras(IITM) 1990

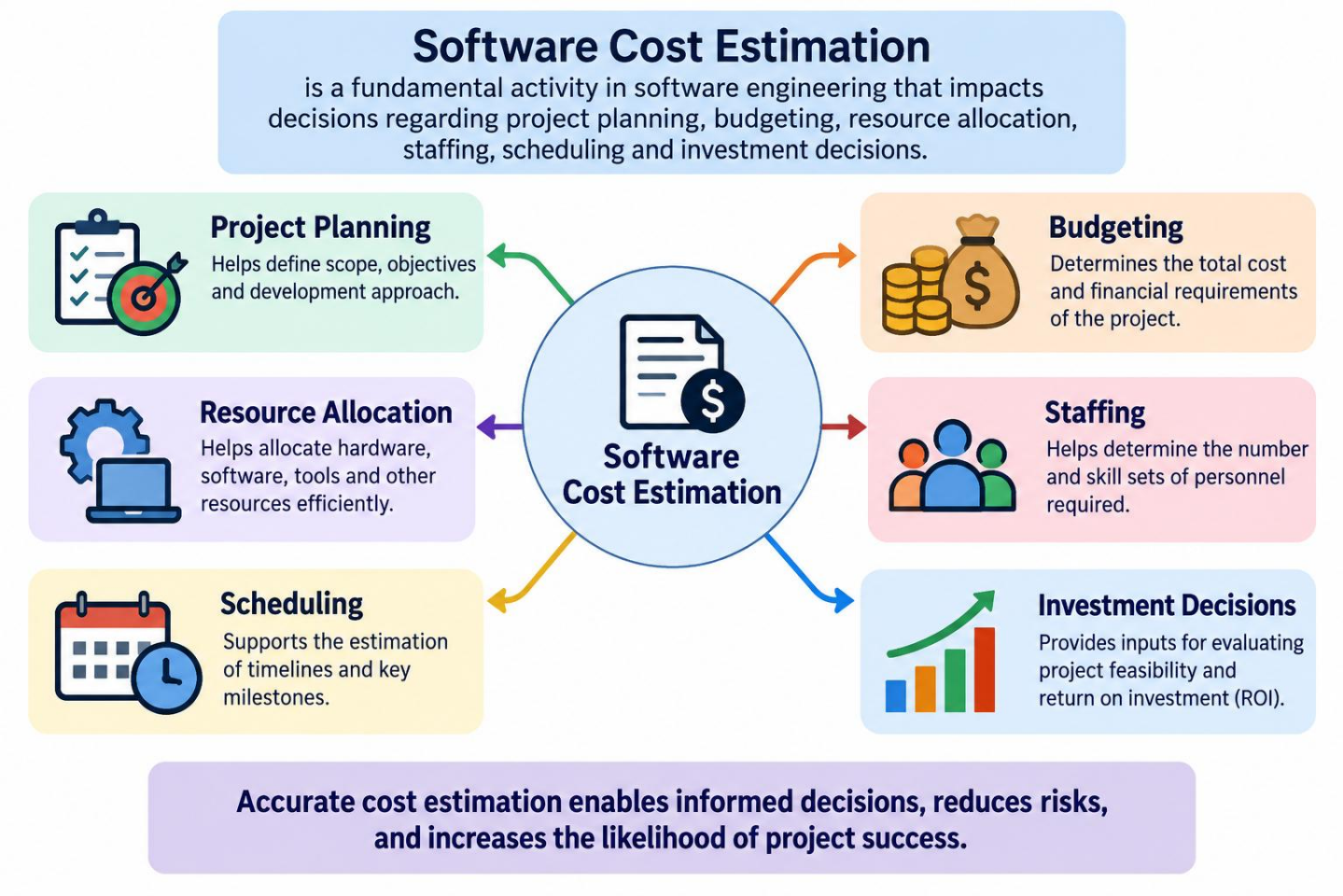
MS by Research in Industrial Systems Engineering, University of Regina, Saskatchewan, Canada – 1992

MS by Research in Information Technology, International Institute of Information Technology, Bengaluru, 2013

Awards : Governor General of Canada Gold Medal - 1992

Work Experience : Mainly in IT and Quality Assurance in multinational companies such as Infosys, Emirates Airlines, Unisys Corp, Land Mark IT, MasTech, GAVS Info Services, Munify Data Tech Pvt Limited. Guest Faculty for SE at IIT-Madras.

What Is Software Cost Estimation and Why Is it Important ?



How Much Will Software Cost?

The answer depends on what is being built—not just on the word “software.”

Like asking “How much does a car or a property cost?” The price depends on the model, features, quality and requirements.

Example: building a login / registration feature

1 Basic login

Initial scope is limited.
Budget estimate: $x

→

2 Add OAuth

Customer later requests a more convenient sign-in option.
Illustrative estimate: $2x

→

3 Prevent spam

OTP or CAPTCHA is added to deter fake registrations.
Illustrative estimate: $4x

The $x → $2x → $4x figures are illustrative: each newly discovered requirement changes the scope, effort and cost.

Key point: an estimate is only as reliable as the requirements and assumptions on which it is based.

The Cone of Uncertainty

Early estimates have a wider range; they become more reliable as the project becomes better understood.

Uncertainty narrows as evidence grows

Project start

Broad assumptions

Requirements

Scope becomes clearer

Design & build

Decisions and evidence accumulate

Testing / release

Estimate is better informed



Conceptual illustration—not a calibrated numerical scale.

Core principle

Variability in the estimate depends on variability in the project.

Reduce scope changes, clarify requirements and surface risks early.

Less project uncertainty → more dependable estimates

Better estimation is not just about better formulas—it also requires reducing uncertainty in the work to be done.

When Software Cost Escalation Ends in Cancellation

Selected cases • Reported historical costs • Click the source links to verify

01 FBI Virtual Case File (US)

~US$170m • abandoned (2005)

Requirements/design weaknesses and inadequate project oversight contributed to termination before delivery.

Estimation lesson: stabilise requirements; review design maturity and governance early.

US DOJ Inspector General

02 BBC Digital Media Initiative (UK)

£98.4m net cost • cancelled (2013)

The programme was cancelled after substantial expenditure; business requirements and oversight were criticised.

Estimation lesson: track delivery evidence and set explicit stop/re-plan decision points.

UK National Audit Office

03 TAURUS — London Stock Exchange (UK)

~£75m • abandoned (1993)

The securities-settlement system was abandoned after growing complexity and competing stakeholder requirements.

Estimation lesson: model scope changes explicitly; consider phased delivery.

Why Projects Fail — case study

04 ASX CHESS replacement (Australia)

A$245–255m project costs written down (2022)

The original replacement solution was paused after major delivery/design problems; the broader replacement goal continued.

Estimation lesson: revise forecasts when design risk and unresolved issues change materially.

ASIC / Reserve Bank statement

Estimating the Recruitment Application

Numeric examples using different approaches; all assumptions are visible and must be tested against project evidence.

ORIGINAL PLAN | 280 hours **ACTUAL EFFORT | 360 hours** **VARIANCE | +80 hours (+28.6%)**

1 Work breakdown estimate

Requirements 40 h
Design 56 h
Coding 120 h
Testing 64 h

TOTAL = 280 hours
Matches the original baseline.

2 Three-point estimate

Optimistic = 280 h
Most likely = 320 h
Pessimistic = 400 h

PERT = (280 + 4×320 + 400) ÷ 6
= 326.7 ≈ 327 hours

3 Basic COCOMO

Assume 0.9 thousand lines of code and organic mode.

Effort = 2.4 × 0.9^1.05
= 2.15 person-months
× 160 h = 344 hours

4 ISBSG analogy

Assume target size = 53.2 function points.
Sample peer rate = 6.52 effort-hours per point.

53.2 × 6.52 = 347 hours
Low confidence: one new-development peer in this sample.

ESTIMATES COMPARED WITH THE 360-HOUR ACTUAL

Work breakdown	Three-point	Basic COCOMO	ISBSG analogy	Actual effort
280 h	327 h	344 h	347 h	360 h

Calibrating Effort with Function Points and ISBSG

Worked example for the recruitment application; size counts and the risk allowance are illustrative.

COUNT THE APPLICATION'S FUNCTIONAL SCOPE

Function type	Count	Weight	Points
External inputs	4	4	16
External outputs	3	5	15
External inquiries	2	4	8
Internal logical files	1	10	10
External interface files	1	7	7

Unadjusted total = 56 points × 0.95 adjustment factor = 53.2 adjusted function points

CALIBRATE PRODUCTIVITY FROM THE SAMPLE

ISBSG sample project 52982 • Business application, new development
422 adjusted function points; 2,752 normalised work-effort units

2,752 ÷ 422 = 6.52 effort units per function point

53.2 × 6.52 = 347 effort units
Add an illustrative 8% risk allowance: 375 units

Assuming the release's normalised effort unit is an hour: base estimate ≈ 347 hours; risk-adjusted estimate ≈ 375 hours.

HOW TO CALIBRATE

1. Filter peers by development type, business domain, platform, language and data quality. 2. Calculate effort per adjusted function point across several comparable projects; prefer a stable median, not one record. 3. Apply that rate to the target size, then validate on completed local projects and widen the range when evidence is weak.

Improve Historical Data Accuracy Through Automated Evidence

Replace manual observation recording with automated collection and consistent measurement from many project sources.

1 HETEROGENEOUS EVIDENCE SOURCES

- Email and meeting messages
- WhatsApp conversations
- Social media posts
- Ticketing systems
- Informal notes and documents
- Code repositories and reviews
- Time records and work logs
- Testing and defect reports

2 AUTOMATED COLLECTION AND ANALYSIS

A COLLECT, LINK AND DEDUPLICATE

Gather records automatically; connect each item to its project, module, task and date. Retain the original source.

B EXTRACT EVIDENCE INTO MEASUREMENTS

Algorithms or Agentic AI identify evidence of size, effort, defects, rework and scope changes; attach confidence and source links.

C STANDARDISE AND VALIDATE

Use common definitions and units; detect gaps and contradictions. People review uncertain cases rather than re-enter every record.

3 MEASURED HISTORICAL DATA

PROJECT / MODULE SIZE

Functional scope or code size

EFFORT

Hours by project, module or task

DEFECTS

Confirmed defects and severity

REWORK AND SCOPE CHANGE

Reopened work and added requirements

MANUAL RECORDING — RISK OF BIAS AND ERROR

People scan messages and notes, then record observations by hand. Records can be missed, misremembered, inconsistently interpreted or measured in different units.

AUTOMATED MEASUREMENT — MORE CONSISTENT EVIDENCE

Collect repeatedly, apply the same rules, preserve source links and check data quality. Review exceptions and audit samples to detect systematic errors.

RESULT: a more complete, consistent and traceable historical baseline for more reliable software estimation.

Recruitment Application: Effort Variance and Possible Causes

Observed: +80 h. The causes below are possibilities to check against project records—not proven facts.

1 REQUIREMENTS

FR-01 Create job openings
FR-02 Record candidates and applications
FR-03 Schedule interviews and scorecards
FR-04 Route offers for approval
NFR-01 Restrict access by user role

2 PLANNED → ACTUAL

Requirements	**40 → 52 h (+12)**
Design	**56 → 64 h (+8)**
Coding	**120 → 158 h (+38)**
Testing	**64 → 86 h (+22)**

TOTAL: 280 → 360 h | +80 h (+28.6%)

WHAT THE VARIANCE MEANS

Coding has the largest difference (+38 h), but that does not tell us why. Check whether extra work came from late requirement changes, overlooked cases, or difficult-to-change code.

3 ISHIKAWA — POSSIBLE ROOT CAUSES



Confirm the cause before attributing the +80 h to it or changing future estimates.

The Fallacy: Correlation Is Not Cause

A variance tells us what happened. It does not, by itself, explain why it happened.

OBSERVED FACT

Coding was +38 h; total project was +80 h.

FALLACY (WRONG INFERENCE)

"Add 38 h to every future coding estimate."

CORRECT INFERENCE

"Find what caused the extra work; adjust only for that cause."

TWO EXAMPLES

EXAMPLE 1 — LATE SCOPE CHANGE

A new interview-report feature is added after coding starts. Developers change code and rerun tests, adding 12 h.

Wrong: add 12 h to every future coding estimate.
Right: estimate the report when it is in scope; use change control for late additions.

EXAMPLE 2 — FRAGILE, HARD-TO-CHANGE CODE

Changing candidate details unexpectedly breaks interview scheduling because both features share tightly coupled code. Developers spend 20 h fixing regressions.

Wrong: add 20 h to all coding estimates.
Right: if this coupling recurs, allow for refactoring—or invest in modular design and retest the effect.

Correct the estimate by tracing hours to a verified cause. A one-off scope change needs a scope/change adjustment; recurring code complexity may need a recurring allowance or a code-quality improvement.

Correlation Is Not Causation: Eight Estimation Examples

Use the correlation to ask a question—not to assume a causal lever. Examples and numbers are illustrative.

1 LOC → EFFORT

Observed: larger codebases often require more effort.
Wrong: 10× LOC means 10× effort.
Causal check: complexity, language, reuse and boilerplate differ.

2 STORY POINTS → VALUE

Observed: high velocity may accompany successful releases.
Wrong: inflate points to deliver more value.
Causal check: points are relative effort; value depends on outcomes.

3 TEAM SIZE → SPEED

Observed: larger teams may deliver more overall.
Wrong: 6 people make a 6-month job take 2 months.
Causal check: onboarding, dependencies and coordination constrain parallelism.

4 TEST COVERAGE → ZERO BUGS

Observed: strong testing may accompany fewer escaped defects.
Wrong: 95% coverage means zero fix effort.
Causal check: assertion quality, edge cases and integration risk matter.

5 PAST VELOCITY → FUTURE

Observed: recent velocity can predict similar sprints.
Wrong: 40 points guarantees 40 next sprint.
Causal check: legacy complexity, domain and team conditions may change.

6 LOC → DEFECTS

Observed: bigger systems often have more defects in total.
Wrong: reducing LOC alone reduces defects.
Causal check: use defect density and examine complexity, risk and change rate.

7 LOW WIP → FEWER DEFECTS

Observed: low-WIP teams may have fewer defects.
Wrong: halve WIP and defects fall 75%.
Causal check: discipline, context switching and testing may explain both.

8 MORE REVIEWS / MEETINGS → OVERRUN

Observed: review-heavy or meeting-heavy projects may show more defects or overruns.
Wrong: reviews or meetings caused the problem.
Causal check: harder work and unstable scope may trigger both.

Rule: check confounders, reverse causality and denominators; test interventions and validate on unseen projects before changing estimates.

Observation vs. Experimentation

Observation identifies patterns in project history; controlled experiments test whether a change actually improves outcomes.

A OBSERVATION-BASED ESTIMATION

Learn from completed projects

HISTORY
Effort logs, change records

PATTERNS
Where effort / rework rose

ESTIMATE
Similar future work

Example: the recruitment project overran by 80 h; coding was +38 h. This shows where to investigate, not what caused it.

Limitation: historical correlation alone cannot show whether scope changes, complex code, team capability or another factor caused the overrun.

B EXPERIMENTATION-BASED ESTIMATION

Test a targeted change on comparable tasks

HYPOTHESIS
Suspected cause

TRIAL
Try one corrective action

MEASURE
Effort, rework, defects

Example: agree acceptance criteria before coding, then compare late changes and rework with similar tasks using the usual process.

Experiments strengthen causal claims when tasks are comparable, team capability is considered, and other differences are controlled.

Observe to locate the variance. Experiment to test the cause. Update estimates only when the effect is repeatable.

Using Experiments (Randomized Controlled Trials) to Improve Software Cost Estimations

Combine historical data with experimental results to identify causal factors and build more accurate estimation models.

1 SELECT & RANDOMIZE

Choose comparable tasks / teams. Randomly assign them to control (current practice) or treatment (proposed fix).

2 COLLECT RESULTS

Use the same measures in both groups: effort, schedule, defects and rework.

3 ESTIMATE EFFECT

Compare group averages; check task size, complexity and team capability.

4 COMBINE EVIDENCE

Combine trial results with historical project data; record limits and uncertainty.

5 UPDATE MODEL

Use repeatable effects to improve estimates for similar future projects.

ILLUSTRATIVE EXAMPLE: DOES A REQUIREMENTS CHECKLIST REDUCE REWORK?

CONTROL GROUP: similar tasks use the usual process. Record late requirement changes, rework hours and total effort.

TREATMENT GROUP: similar tasks use agreed acceptance criteria and edge-case reviews. Record the same measures.

If the treatment group repeatedly has less rework and lower total effort—and other factors are comparable—the evidence supports using the measured reduction in future estimates.

Macro and Micro Parameters: Why Effort Varies

Project-level characteristics explain part of the effort; day-to-day events explain additional variation

MACRO PARAMETERS — WHAT THE PROJECT IS

- Software size, complexity and functional scope
- Developer skills, team size and domain experience
- Technology stack, architecture and requirements volatility
- Reusable components available and fit for purpose

MICRO PARAMETERS — CONDITIONS THAT CHANGE EFFORT

- KBase / reusable components available, reusable and understood
- Server crash, server unresponsive, unavailable backups
- Power cuts, workstation failure and computer maintenance
- Team reluctance to use tools; learning and setup time
- HR policies, employee wellness, absence and turnover
- Remuneration, retention and incentives affecting continuity

MICRO EVENTS: SAVED EFFORT vs LOST EFFORT

SAVED: reuse a proven component; documented KBase fix; early clarification; automated tools adopted well

LOST: server outage / crash; missing backup; power cut; maintenance downtime; tool resistance; handover or staff absence

Risk controls: test backup restoration; commit and push code regularly; monitor server availability; plan maintenance; provide tool training and protect employee wellbeing. Track actual effort separately from waiting time.

Micro-Events That Macro Estimates Miss

Project size, technology and team skill describe the work; small day-to-day events explain effort lost or added.

CHANGE REQUESTS/VOLATILITY

– Impact on Cost Prediction Accuracy:

Track changes, analyse their impact, use historical data, and continuously update cost estimates as requirements evolve.

SERVER UNRESPONSIVE

3 incidents x 1 h

+3 h

Waiting is calendar delay; count only active diagnosis/recovery effort

SERVER CRASH / BACKUP GAP

1 recovery x 4 h

+4 h

Restore work, reconstruct lost changes; regular remote commits reduce exposure

REUSABLE COMPONENTS

Component missing / unsuitable

+6 h

Rebuild or adapt functionality that could have been reused; verify fit and integration effort

POWER CUT / MAINTENANCE

2 incidents x 1.5 h

+3 h

Lost productive time or recovery; distinguish downtime from effort

TOOL RELUCTANCE / LEARNING

4 x 0.75 h

+3 h

Setup, training or manual workarounds when tools are not adopted

HR / WELLNESS / ABSENCE

1 handoff x 2 h

+2 h

Handover and continuity effects; do not treat personal wellbeing as a fault

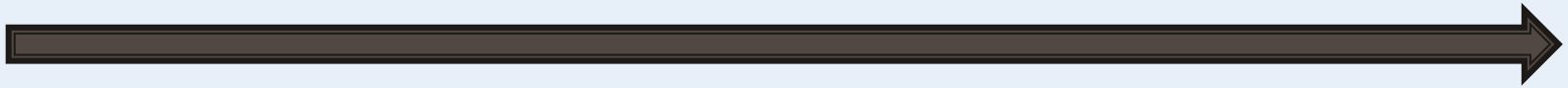
REMUNERATION / RETENTION

1 replacement x 5 h

+5 h

Illustrative onboarding and knowledge-transfer effort after turnover

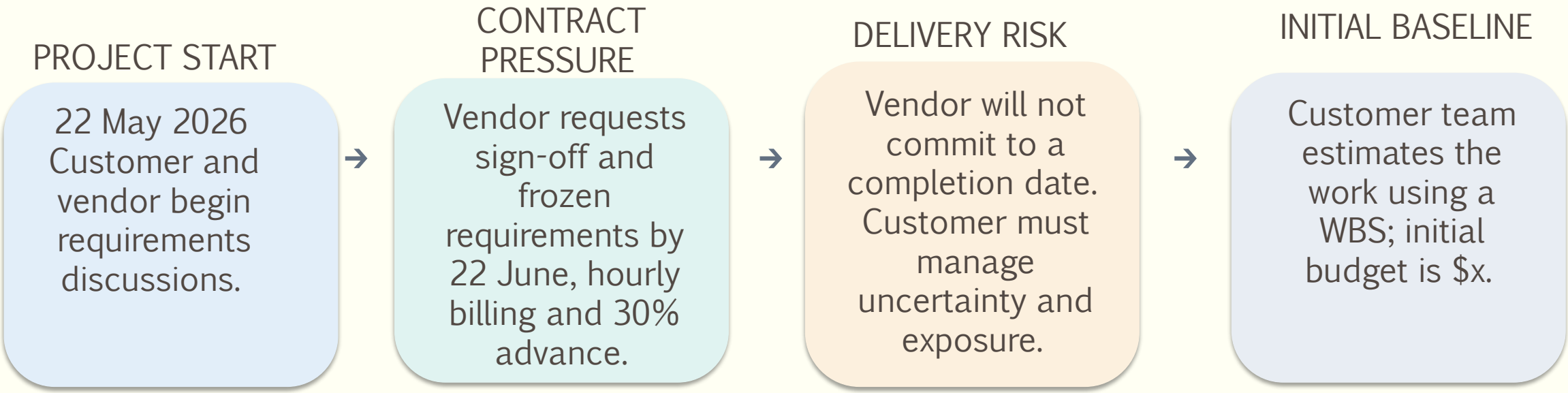
ILLUSTRATIVE EXAMPLES



Not additive

Predictor–Corrector Estimation: A Realistic Project Scenario

Can early evidence and corrective action help the customer meet the planned date and budget?



Predictor–corrector: compare actual progress with the baseline, explain deviations, revise the forecast and act early.

Illustrative scenario and values; $x represents the customer's original budget.

Key question: can the team control scope, productivity and remaining effort before small problems become major cost overruns?

The estimate is a forecast—not a guarantee. Update it as real evidence emerges.

Scenario 1: A WBS estimate is not corrected as project evidence emerges

1. INITIAL WBS ESTIMATE

Customer team estimates scope and effort from the available requirements. Planned budget: $x.

2. MINIMUM BASELINE

Stable requirements, a minimum design baseline and coding standards are established.

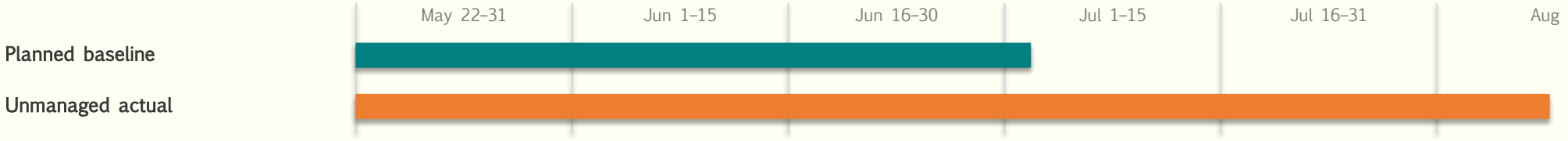
3. RISKS EMERGE

Actual vendor productivity and training needs are not visible early enough; design and delivery risks accumulate.

4. COST ESCALATES

Effort is consumed by slower delivery, rework and late corrections. Illustrative cost rises to $4x.

ILLUSTRATIVE GANTT · planned work vs. unmanaged slippage



Without reviews, risk and rework can extend delivery well beyond the baseline.

Scenario 2: Predictor–Corrector Reviews — 10 June, 1 July and 20 July 2026

Incident

The customer asks the vendor to begin developing selected features before requirements are fully frozen.

Evidence found

At the 10 June review, actual developer productivity and the training required for the chosen coding platform are not yet visible or adequately demonstrated.

Correction

The customer flags the risk of an upward effort spike, requests alternate suitably skilled resources and updates size, productivity and remaining-effort assumptions.

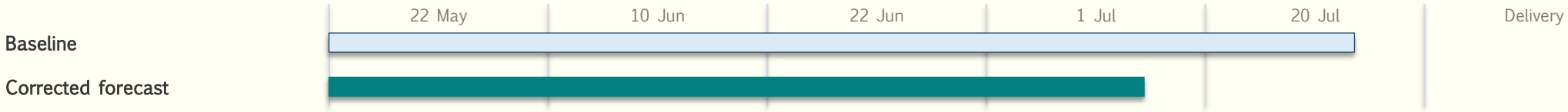
Predictor–corrector action: record actual effort and progress, compare them with the WBS baseline, consult historical deviations, then revise the forecast and intervene early.

5. CORRECTOR STEP — 20 JULY 2026: UI STYLING INTERVENTION

The customer team finds that the styling is still inadequate. A short-term intervention brings in UI specialists to provide styling specifications—not to undertake the full implementation. The existing team can then apply the specifications.

Without this intervention, an additional two months of developer effort could have been needed. With hourly billing, that delay could have driven substantial further cost escalation.

ILLUSTRATIVE GANTT · review points trigger forecast updates and corrective action



Beyond Accuracy: What Should Future Models Achieve?

01

Explainable

Show why an estimate changes and which project factors contribute to the prediction.

02

Adaptive

Update estimates as requirements, productivity, defects, rework and team conditions evolve.

03

Uncertainty-aware

Represent the range and confidence of an estimate—not just a single number.

Research objective: develop estimation models that explain their predictions, respond to new evidence and communicate uncertainty.

Success means estimates that are more useful for decisions—not merely more accurate on average.

AI-Assisted Coding: What Changes—and What Must Be Tested?

Use historical deviations and measured progress to revise the plan

1 Software size

If manual coding produces N LOC, will AI produce more or fewer LOC for the same requirements and functionality?

Measure LOC and functional equivalence; LOC alone is not a measure of productivity.

2 Effort distribution

Hypothesis: AI may reduce effort to implement a fixed scope (e.g., N LOC or a fixed number of FPs).

Measure coding, code review, testing, rework and total effort separately.

3 Defect density

Does AI-assisted code have fewer defects per KLOC or per function point?

Results may depend on developer/team capability, task type, prompts, tools and review practices.

Experimental design

Randomly assign comparable programming tasks to manual and AI-assisted groups; balance or stratify by team capability. Keep requirements, acceptance tests, language, tools and time limits comparable. Replicate across tasks and teams, then compare outcomes with confidence intervals and statistical tests.

Record for each task

- LOC and functional scope
- Coding / review / test hours
- Defects found, severity and rework

Compare

- Effort per FP or equivalent task
- Defects per KLOC and per FP
- Total effort and delivered quality

Statistical conclusion

Estimate effect sizes and 95% confidence intervals; test whether differences are statistically meaningful and whether they vary by team capability.

Research objective: determine where AI saves effort, whether quality changes, and for which teams and task types.

Using AI Across the Software Estimation Lifecycle

Use AI to reduce uncertainty before development, improve reviews during delivery, and learn from project evidence.

1 Clarify requirements

BEFORE ESTIMATING

Read the requirements document

- Flag ambiguity, conflicts and gaps
- Check acceptance criteria and quality needs
- Prioritise questions by likely effort impact

RESULT: agreed requirements baseline

2 Build the estimate

SIZE + HISTORICAL EVIDENCE

- Propose a work breakdown and functional size
- Retrieve comparable completed projects
- Compare function-point productivity, COCOMO and data-driven models
- Show a range, assumptions and risks

RESULT: explainable effort range

3 Review code and tests

DURING DEVELOPMENT

- Summarise code changes quickly
- Flag potential defects and security or maintainability risks
- Suggest tests and affected modules
- Track review, testing and rework separately

RESULT: faster reviews and clearer impact

THE KEY PRINCIPLE

Use AI to surface uncertainty and evidence—not to hide assumptions. Preserve source links, validate size counts and AI findings, and keep human approval for baselines and commitments.

AI supports the estimator; the quality of the estimate still depends on reliable data, clear scope and validation.

AI-Powered Predictor–Corrector: From Deviation to Action

The goal is not just to detect an overrun—it is to investigate evidence, act early and improve the next forecast.

PLANNED CODING	ACTUAL CODING	DEVIATION	AI INVESTIGATES WHY
120 hours	158 hours	+38 hours	Evidence before conclusions

1 Detect

Compare planned and actual

Track effort and completed work. Recalculate remaining effort and forecast as evidence changes.

2 Investigate

Link the evidence

Check change requests, email or WhatsApp discussions, tickets, code revisions, time records and tests.

3 Test explanations

Rank possible root causes

Late scope change? Hidden complexity? Defects or rework? Show supporting records and uncertainty; do not assume causation.

4 Correct and learn

Act, then verify

Clarify scope, target the blocker, improve design or tests, or adjust resources. Reforecast and record whether the action worked.

HUMAN + AI CONTROL LOOP

AI links each suggested cause to evidence and recommends options. The project manager approves scope, budget and delivery changes. Verify outcomes and feed confirmed measurements back into future estimates.

Research test: measure forecast error, time to diagnose, confirmed causes, false alarms and whether corrective actions reduce rework.

From a Static Estimate of X to 4X — and Back Toward X

How unrefined assumptions allow cost to compound—and how evidence-led interventions can control the forecast.

CUSTOMER GIVES VENDOR STATIC ESTIMATE = X

PATH A | NO CALIBRATION OR EARLY CORRECTION

X — **Unrefined static estimate**
Ambiguous scope; hidden assumptions

2X — **Baseline misses delivery conditions**
Optimism and weak comparables

3X — **Untracked effort accumulates**
Rework, integration and micro-events

4X — **Actual cost escalates**
Late discovery; limited time to intervene

UNCONTROLLED OUTCOME 4X COST
Illustrative escalation from the original X

PATH B | EVIDENCE-LED INTERVENTIONS

1. **Calibrate the estimation model**
   Use comparable internal / ISBSG projects
2. **Analyse causes, not correlations**
   Verify why effort differs from the estimate
3. **Track micro-parameters**
   Measure rework, wait time, build failures, handoffs
4. **Predictor–corrector reviews**
   Compare actuals, forecast remaining effort, act early
5. **Automate evidence with AI agents**
   Link messages, tickets, code, tests and time records

CONTROLLED TARGET ≈ X COST
Keep actual effort close to the calibrated baseline

Illustrative scenario, not a guaranteed 4X-to-X reduction. The result depends on scope control, data quality, timely intervention and measured validation.

Thank You