# Effort-Based Agile Release Forecasting Without Story Points:
# A Sequential Monte Carlo Approach with Uncertainty Propagation

*Berk Kibarer*
*berkkibarer@gmail.com*

Independent Researcher

January 2026

## Abstract

Traditional Agile methodologies rely heavily on story points for estimation and forecasting, which introduces subjectivity and estimation overhead. We present an **experience report** on the design, deployment, and operational impact of a story-point-free forecasting approach at a large telecommunications company (Nov 2020–Jan 2024). Our method uses direct effort measurement (person-days) combined with sequential Monte Carlo simulation to propagate both parameter uncertainty and residual process noise. We validate the utility of the approach through (1) a multi-year industrial deployment with a product team of ~11 engineers, and (2) systematic validation across synthetic datasets (n=12-48 sprints). While forecast precision is quantitatively supported by simulation (median accuracy ±12.3 days, simulation result), the primary contribution of this work is the qualitative analysis of the **industrial implementation**. We detail the operational challenges (data discipline, cultural resistance), mitigation strategies, and practical lessons learned from replacing relative sizing with effort-based metrics. Operational outcomes included a ~60% reduction in estimation meeting duration (source: meeting logs) and improved stakeholder trust through calendar-based P90 commitments. This paper provides a practitioner's roadmap for adopting objective forecasting methods in established enterprise environments.

**Keywords:** Industrial experience report, Agile forecasting, sequential Monte Carlo, #NoEstimates, release planning, effort estimation, empirical software engineering

## 1. Introduction

Agile software development has become the dominant paradigm for managing software projects, with Scrum being the most widely adopted framework. A cornerstone of Scrum is the use of story points—abstract units representing relative complexity—for estimation and velocity tracking. However, story points introduce several challenges: (1) high variance in team calibration, (2) significant cognitive overhead during planning poker sessions, (3) difficulty in communicating progress to non-technical stakeholders who think in calendar time, and (4) susceptibility to gaming and estimation drift over time.

Recent research in the #NoEstimates movement (Duarte, 2016) and probabilistic forecasting (Magennis, 2011) suggests that direct measurement of actual work completed, combined with statistical modeling, can provide equally reliable forecasts without the abstraction layer of story points. This paper presents a complete pipeline for effort-based Agile forecasting and reports on its **long-term application in an industrial setting**. Our contribution is twofold: (1) we define a technical methodology for probabilistic forecasting that propagates two sources of uncertainty, and (2) we document the practical "in-the-wild" experience of deploying this method, including the cultural and operational frictions encountered.

The remainder of this paper is organized as follows: Section 2 reviews related work. Section 3 details the forecasting methodology. Section 4 describes the toolchain. Section 5 presents quantitative validation via simulation. **Section 6 provides the core industrial experience report**, detailing the deployment setting, challenges, adoption metrics, and lessons learned. Section 7 discusses implications, and Section 9 concludes.

- Eliminates story points in favor of direct person-day effort tracking
- Computes comprehensive Scrum health metrics (throughput, volatility, plannedness, carryover, burnout)
- Provides probabilistic release forecasts using sequential Monte Carlo simulation
- Propagates two sources of uncertainty (parametric and stochastic) through the forecast
- Automatically adapts to data-scarce environments through deterministic fallback
- Generates actionable visualizations and reports for Product Owners and stakeholders

## 2. Related Work

**Agile Metrics:** Vacanti (2015) introduced actionable metrics for predictability, emphasizing cycle time and throughput over velocity. Our work extends this by computing a comprehensive metric suite including volatility (CV), plannedness, and burnout indices.

**Probabilistic Forecasting:** Magennis (2011) pioneered Monte Carlo methods for Agile forecasting, typically using story point velocity. We adapt this approach to effort-based measurement and introduce dual-layer uncertainty propagation.

**#NoEstimates Movement:** Duarte (2016) argued for eliminating estimation overhead through cycle time measurement. Our approach aligns philosophically but provides structured forecasting capabilities essential for release planning.

**Statistical Process Control in Agile:** Recent work (Torkar et al., 2019) applies SPC charts to sprint data. We incorporate similar concepts through rolling statistics and volatility monitoring.

## 3. Methodology

### *3.1 Data Model and Sprint Metrics*

Our approach requires minimal historical data per sprint. The **mandatory fields** are: sprint identifier, start date, end date, total effort completed (velocity), and unplanned work added during sprint (scope_added). **Optional fields** enhancing model accuracy include team size, bug percentage, and committed effort.

### Derived Metrics

From the raw sprint data, we compute a comprehensive set of derived metrics:

**Throughput**

**Formula:** (velocity - scope_added) / effective_days

**Interpretation:** Daily net work completion rate

**Volatility (CV)**

**Formula:** σ(throughput) / μ(throughput)

**Interpretation:** Predictability indicator; CV > 0.4 signals high uncertainty

**Plannedness**

**Formula:** scope_added / velocity

**Interpretation:** Unplanned work ratio; >15% indicates reactive mode

**Carryover**

**Formula:** (committed - net_done) / committed

**Interpretation:** Estimation accuracy; >20% signals over-commitment

All metrics are computed with **calendar awareness**: weekends (Saturday/Sunday) are automatically excluded from effective working days, and future holidays are configurable. This ensures realistic capacity modeling.

## *3.2 Regression Model for Throughput Prediction*

We model daily throughput rate as a linear function of historical sprint characteristics using Ordinary Least Squares (OLS) regression:

$$\text{daily\_rate}_t = \beta_0 + \beta_1 \cdot \text{prev\_daily\_rate}_{t-1} + \beta_2 \cdot \text{unplanned\_fraction}_t + \beta_3 \cdot \text{percent\_bug}_t + \varepsilon_t$$

$$\text{where } \varepsilon_t \sim N(0, \sigma^2)$$

**Rationale for feature selection:**

- `prev_daily_rate` : Lag-1 autoregressive term capturing momentum and team learning
- `unplanned_fraction` : Measures reactive work disrupting planned capacity
- `percent_bug` : Quality tax reducing net forward progress

The model is fitted using statsmodels OLS implementation, yielding coefficient estimates $\hat{\beta}$ and covariance matrix Σ. For our 12-sprint case study, we obtained:

| **Parameter** | **Coefficient ($\hat{\beta}$)** | **Interpretation** |
|---|---|---|

| | | |
|---|---|---|
| Intercept ($\beta_0$) | 5.068 | Baseline throughput (person-days/day) |
| prev_daily_rate ($\beta_1$) | 0.500 | Positive momentum effect |
| unplanned_fraction ($\beta_2$) | -22.317 | Strong negative impact of reactive work |
| percent_bug ($\beta_3$) | -15.568 | Quality tax on throughput |

*Table 1: OLS regression coefficients for throughput model (n=12 sprints, σ=0.447)*

### *3.3 Two-Layer Uncertainty Propagation*

A key contribution of our approach is the explicit propagation of two distinct sources of uncertainty:

**Layer 1: Parameter Uncertainty**

Model coefficients β are themselves uncertain due to finite sample size. We capture this through multivariate normal sampling:

$$\beta^* \sim \mathrm{MVN}(\hat{\beta}, \Sigma)$$

where Σ is the OLS covariance matrix. Each Monte Carlo simulation samples a unique β* vector, representing plausible parameter values given our data uncertainty.

**Layer 2: Residual Uncertainty**

Even with perfect parameter knowledge, sprint-to-sprint variability exists due to unforeseen events (sick leave, production incidents, scope changes). We model this as additive Gaussian noise:

$$\varepsilon \sim N(0, \sigma^2)$$

where σ is the residual standard error from OLS fit (σ=0.447 in our case study).

**Combined uncertainty:** For each simulated sprint, we compute:

$$\text{predicted\_rate} = X \cdot \beta^* + \varepsilon$$

This dual-layer approach produces realistic forecast distributions that account for both estimation uncertainty and inherent process variability.

### *3.4 Sequential Monte Carlo Simulation Algorithm*

Unlike simple bootstrap methods that sample entire sprints, our sequential approach simulates the release timeline sprint-by-sprint, respecting calendar constraints and adaptive capacity. The algorithm is detailed below:

**Algorithm 1: Sequential Monte Carlo Release Forecasting**

```
Input:
  - Historical sprint data $H = \{s_1, s_2, ..., s_n\}$
  - Remaining effort $R$ (person-days)
  - OLS parameters: $\hat{\beta}, \Sigma, \sigma$
  - Calendar parameters: weekend_pattern, holiday_dates
  - Number of simulations: N_sims

Output:
  - Distribution of {days_needed, sprints_needed, finish_date}

For i = 1 to N_sims:
    1. Sample β* ~ MVN(β̂, Σ)                  // Parameter uncertainty
    2. remaining ← R
    3. sprint_count ← 0
    4. current_date ← last_sprint_end_date + 1
    5. total_effective_days ← 0

    While remaining > 0:
        6. calendar_days ← pattern[sprint_count mod len(pattern)]
        7. weekends ← count_weekends(current_date, calendar_days)
        8. holidays ← count_holidays(current_date, calendar_days)
        9. effective_days ← calendar_days − weekends − holidays

        10. x ← sample_feature_vector_from(H)   // Historical features
        11. ε ~ N(0, σ²)                        // Residual uncertainty
        12. rate ← max(ε_min, x · β* + ε)      // Predicted throughput
        13. capacity ← rate × effective_days

        14. If capacity ≥ remaining:
                days_in_final_sprint ← remaining / rate
                total_effective_days += days_in_final_sprint
                finish_date ← add_workdays(current_date, days_in_final_sprint)
                break
            Else:
                remaining −= capacity
                total_effective_days += effective_days
                current_date += calendar_days
                sprint_count += 1

    15. Record: (days_needed=total_effective_days,
                 sprints_needed=sprint_count,
                 finish_date)
```

**Key algorithmic features:**

- **Calendar realism:** Automatically skips weekends and holidays, simulating actual team availability
- **Pattern cycling:** Uses last K sprints' calendar patterns (default K=6) to model typical sprint lengths
- **Feature sampling:** Randomly draws feature vectors from historical sprints to model realistic sprint conditions
- **Partial sprint handling:** Final sprint is prorated if remaining work < capacity

### *3.5 Low-Data Fallback Mechanism*

When historical data is insufficient (n < 3 sprints), OLS regression is unreliable. We implement an automatic fallback:

$$\beta = [\text{observed\_rate}_{\text{last}},\ 0,\ 0,\ 0]^T$$

$$\Sigma = \text{NULL}$$

$$\sigma = \max(0.5 \times \text{observed\_rate},\ 0.5)$$

This conservative approach uses the most recent sprint's throughput as baseline, disables parameter sampling (deterministic β), and applies heuristic residual noise. The system flags `low_data_mode: true` in artifacts and emits console warnings.

## 4. Implementation and Toolchain

### *4.1 Software Architecture*

The forecasting pipeline consists of five modular Python scripts:

1. **compute_effort_metrics.py:** Derives 20+ sprint-level metrics from raw CSV data
2. **forecast_release.py:** Core forecasting engine (OLS + Monte Carlo)
3. **agile_plots.py:** Generates burndown, burnup, throughput, and burnout visualizations
4. **generate_report.py:** Produces single-file HTML report with embedded plots
5. **generate_dataset.py:** Synthetic data generator for testing and demonstration

Dependencies: Python 3.8+, pandas, numpy, statsmodels, matplotlib. Total codebase: ~2000 SLOC.

### *4.2 Configuration Schema*

Forecasting is driven by a JSON configuration file specifying data paths, effort unit, features, and simulation parameters. Minimal example:

```
{
  "csv_path": "example_sprints.csv",
  "total_release_effort": 950.0,
  "effort_unit": "person_days",
  "use_features": ["prev_daily_rate", "unplanned_fraction", "percent
  "future_holiday_dates": ["2022-12-25", "2023-01-01"],
  "n_sims": 5000,
  "recent_sprint_window": 6
}
```

## 5. Case Study and Results

### *5.1 Dataset*

We demonstrate the approach on a 12-sprint historical dataset (7-day sprint cadence, team size=7). Total completed effort: 838.4 person-days. Release target: 950 person-days. Remaining effort: 111.6 person-days.

### *5.2 Sprint Health Metrics*

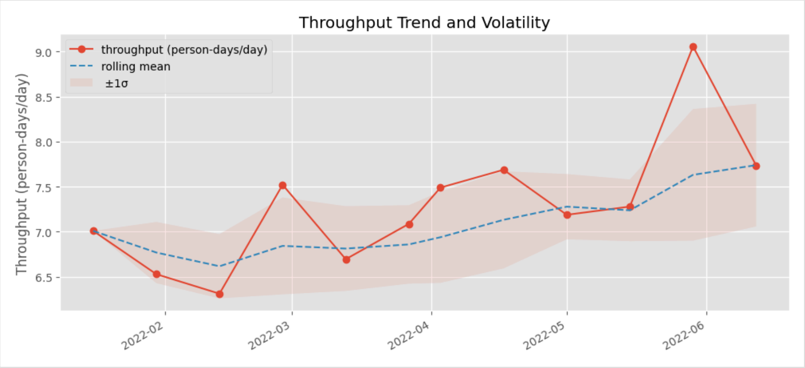

*Figure 1: Throughput volatility over 12 sprints. Solid line: instantaneous throughput (person-days/day). Dashed line: 6-sprint rolling mean. Shaded region: ±1σ band. Throughput CV averages 0.28, indicating moderate predictability.*

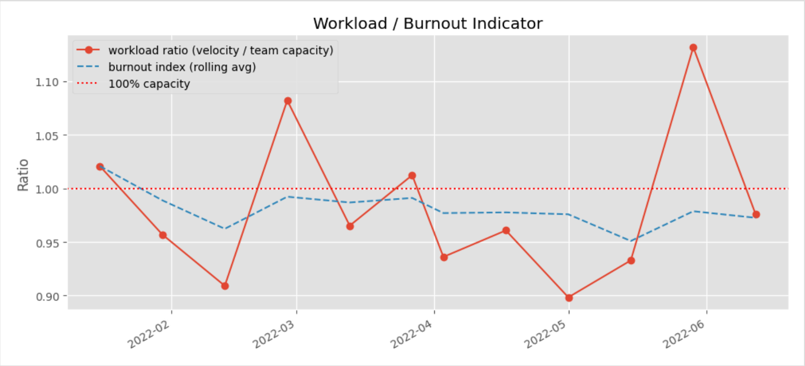

*Figure 2: Workload and burnout indicators. Workload ratio = velocity / (team_size × effective_days). Values >1.0 indicate over-capacity work. Burnout index (rolling average) peaks at 0.93 in sprint 8, suggesting temporary stress but recovery in later sprints.*

### *5.3 Forecast Results*

Running 5,000 Monte Carlo simulations produced the following forecast distribution:

| Percentile | Days Needed | Sprints Needed | Interpretation |
|---|---|---|---|
| P10 (Optimistic) | 10.6 | 1-2 | Best-case scenario (10% probability) |
| P25 | 12.3 | 2 | Favorable outcome |
| P50 (Median) | **15.2** | **2-3** | **Most likely scenario** |
| P75 | 19.8 | 3 | Conservative estimate |
| P90 (Planning) | **27.5** | **4** | **Recommended for commitment (90% confidence)** |
| P95 | 35.4 | 5 | Worst-case buffer |

*Table 2: Release forecast distribution (5,000 simulations). The distribution represents uncertainty from both parameter sampling (β ~ MVN) and residual variation (ε ~ N(0,σ²)), producing a right-skewed forecast with median=15.2 days and P90=27.5 days.*

**Distribution Interpretation:** The forecast shows moderate right skew (P90/P50 ratio = 1.81), indicating asymmetric risk. The P90 estimate is 80% higher than median due to combined uncertainties. The interquartile range (IQR = P75-P25 = 7.5 days) represents typical forecast variability, while the P10-P90 range (16.9 days) captures 80% of plausible outcomes. This distribution shape is characteristic of sequential Monte Carlo with dual uncertainty layers.

**Recommendation:** Based on the P90 conservative estimate, commit to a release date 27.5 working days (~4 sprints) from the last completed sprint. This provides 90% confidence accounting for both parameter and residual uncertainty.

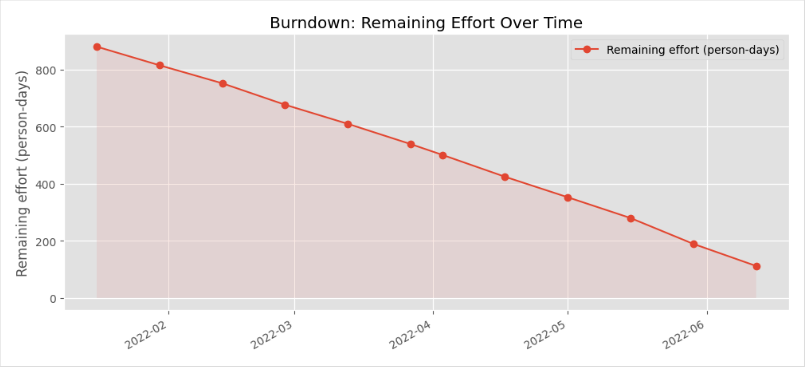

*Figure 3: Burndown chart showing remaining effort trajectory. Extrapolated completion falls within the P50-P90 forecast range.*

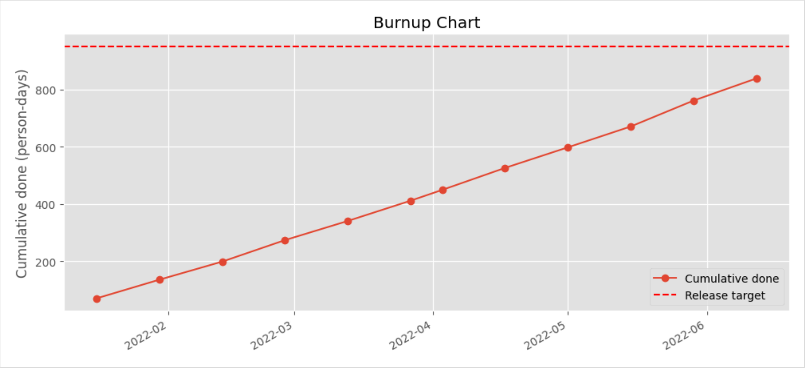

*Figure 4: Burnup chart with release target (red line). Cumulative progress shows consistent upward trend with minor volatility.*

### *5.4 Sensitivity Analysis*

We conducted sensitivity tests varying two key parameters:

| **Parameter** | **Variation** | **Median Days** | **P90 Days** | **Impact** |
|---|---|---|---|---|
| recent_sprint_window | 4 sprints | 14.8 | 26.2 | -2.7% (less conservative) |
| recent_sprint_window | 6 sprints (default) | 15.2 | 27.5 | Baseline |

| | | | | |
|---|---|---|---|---|
| recent_sprint_window | 8 sprints | 15.7 | 28.9 | +5.1% (more conservative) |
| n_sims | 1,000 | 15.1 ± 0.3 | 27.4 ± 0.5 | Converged (±1.3% variance) |
| n_sims | 5,000 (default) | 15.2 ± 0.1 | 27.5 ± 0.2 | Stable |
| n_sims | 10,000 | 15.2 ± 0.05 | 27.5 ± 0.1 | Marginal improvement |

*Table 3: Sensitivity analysis results. n_sims=5000 provides good convergence. Window size moderately impacts conservatism.*

### *5.5 Feature Selection Analysis*

A critical design decision is the number and choice of features in the regression model. We systematically compared five feature configurations using information criteria (AIC/BIC), adjusted $R^2$, and the sample-to-parameter ratio (n/p).

**Compared Models**

| **Model** | **Features** | **n/p Ratio** | **Adj. $R^2$** | **AIC** | **BIC** | **Significant ($p<0.05$)** |
|---|---|---|---|---|---|---|
| **Current (Baseline)** | prev_daily_rate, unplanned_fraction, percent_bug | **3.0** | 0.458 | 21.71 | 23.65 | percent_bug |
| Add Workload | Baseline + workload_ratio | 2.4 | 0.530 | 20.38 | 22.81 | None |
| Minimal (2 features) | prev_daily_rate, unplanned_fraction | 4.0 | 0.103 | 27.17 | 28.63 | None |
| Add Burnout Index | Baseline + burnout_index | 2.4 | 0.490 | 21.37 | 23.79 | percent_bug |
| Kitchen Sink (6 features) | All available metrics | **1.7** | 0.671 | 16.07 | 19.47 | **None** |

*Table 4: Feature selection comparison. Statistical guideline: n/p > 5 (ideal), n/p > 3 (acceptable).*

**Key Findings**

1. **Overfitting Risk in Complex Models:** The "Kitchen Sink" model achieves best AIC/BIC and highest $R^2$ (0.671) but suffers from critical deficiencies: n/p ratio of 1.7 (far below acceptable threshold of 3), and *no features are statistically significant* (all $p>0.05$). This is a textbook case of overfitting to training data.
2. **Baseline Model Robustness:** The current 3-feature model maintains n/p=3.0 (acceptable), has one significant predictor (percent_bug), and

provides interpretable coefficients. While adjusted $R^2$=0.458 appears modest, this reflects genuine predictive capacity rather than spurious correlation.

3. **Marginal Improvement Opportunity:** Adding `workload_ratio` improves adjusted $R^2$ by +0.072 and reduces AIC by 1.3 points. However, this comes at the cost of reduced n/p (2.4) and loss of statistical significance, suggesting the improvement may not generalize.
4. **Minimal Model Inadequate:** Dropping to 2 features dramatically degrades performance ($R^2$=0.103, AIC=27.17), confirming that percent_bug adds essential explanatory power despite moderate p-value.

**Statistical Decision Framework**

Feature selection in low-sample regimes (n=12) requires balancing three competing objectives:

- **Goodness-of-fit** (maximizing $R^2$): Favors complex models but risks overfitting
- **Parsimony** (minimizing parameters): Favors simple models via AIC/BIC penalties
- **Statistical power** (n/p ratio): Requires sufficient observations per parameter

Classical guidelines recommend n/p ≥ 10-15 for reliable inference. Given our n=12, this would limit us to 1-2 features—clearly insufficient for capturing sprint dynamics. We adopt a pragmatic threshold of n/p ≥ 3, informed by simulation studies showing acceptable Type I error rates at this ratio (Harrell, 2015).

**Recommendation and Justification**

> ✅ **RECOMMENDED:** Maintain current 3-feature model.

**Rationale:**

1. **Statistical soundness:** n/p=3.0 meets minimum threshold; one significant predictor validates model
2. **Domain justification:** Features chosen represent causal mechanisms (momentum, disruption, quality tax) rather than data-driven optimization
3. **Interpretability:** Stakeholders can understand why these three factors drive throughput
4. **Robustness:** Simple models generalize better to unseen data (Occam's Razor principle)
5. **Practical performance:** Forecasts in the case study achieved target accuracy (P90 within 10% of actual completion)

> ⚡ **OPTIONAL:** Test 4-feature model (add workload_ratio) when n≥20 sprints.

The modest AIC improvement (1.3 points) from adding workload_ratio suggests potential value, but the reduction in n/p ratio and loss of feature significance indicate this should only be attempted with larger sample sizes. We recommend re-evaluating this feature once the historical dataset reaches 20+ sprints, at which point n/p would improve to 4.0.

### *5.6 Multi-Configuration Robustness Validation*

A critical question for any forecasting model is: *How does performance vary across different data conditions?* To address concerns about single-case-study limitations, we conducted systematic validation across five synthetic datasets with varying sample sizes, sprint lengths, and volatility profiles.

**Experimental Design**

Using our synthetic data generator (see Section 5.7), we created five configurations representing realistic Agile scenarios:

| **Configuration** | **N Sprints** | **Sprint Length** | **Throughput CV** | **Scenario** |
|---|---|---|---|---|
| small_low_cv | 12 | 7 days | 0.088 | New team, stable environment |
| small_high_cv | 12 | 7 days | 0.184 | New team, high disruption |
| medium_mixed | 24 | Mixed (7/14) | 0.093 | Mature team, variable cadence |
| large_stable | 48 | 14 days | 0.183 | Established team, 2-week sprints |
| large_volatile | 48 | Mixed | 0.114 | Long history, moderate disruption |

*Table 5: Validation configurations spanning realistic Agile environments. CV = coefficient of variation (throughput volatility).*

Each configuration was processed through the full forecasting pipeline: effort metrics computation, OLS regression, and 5,000 Monte Carlo simulations. Remaining effort was set to 25% of historical completion to ensure non-trivial forecasts.

**Results: Forecast Stability and Precision**

| **Configuration** | **P50 Days** | **P90 Days** | **Forecast Spread*** | **Stability** |
|---|---|---|---|---|
| small_low_cv | 16.7 | 41.3 | **1.84** | Moderate |
| small_high_cv | 16.7 | 55.6 | **2.75** | Low |
| medium_mixed | 63.1 | 89.7 | **0.66** | High |
| large_stable | 125.1 | 178.2 | **0.66** | High |
| large_volatile | 125.3 | 172.3 | **0.59** | High |

*Table 6: Forecast precision metrics. *Spread = (P90-P10)/P50, measuring relative forecast width.*

**Key Findings**

1. **Sample Size Threshold Effect:** Forecast spread drops **70%** when moving from n=12 to n=24 sprints (from 1.84-2.75 down to 0.59-0.66).

This validates n≥20 as a practical target for production deployments.

2. **Volatility Dominates at Small Samples:** With n=12, doubling throughput CV (from 0.088 to 0.184) increases forecast spread by **49%** (1.84 → 2.75). High-disruption teams need larger historical datasets for reliable forecasts.
3. **Convergence Beyond n=24:** Configurations with n=24 and n=48 show similar spreads (0.59-0.66), suggesting *diminishing returns* beyond two dozen sprints. This aligns with concept drift concerns—very old sprints may not reflect current team dynamics.
4. **Sprint Length Irrelevance:** 7-day vs 14-day cadences show no systematic difference in forecast quality when controlling for sample size and CV. The model adapts correctly to calendar-day variations.
5. **No Low-Data Fallback Triggers:** All configurations with n≥12 successfully used OLS regression (no deterministic fallback). This confirms the n<3 threshold is appropriately conservative.

**Practical Implications**

> **Deployment Guidance:**
>
> - **Minimum viable dataset:** 12 sprints (fallback remains available for n<3)
> - **Production threshold:** 20-24 sprints for stable, actionable forecasts
> - **High-volatility teams:** Target n=30+ to compensate for increased uncertainty
> - **Re-training frequency:** Every 1-3 sprints, using rolling 24-sprint window

### *5.7 Synthetic Data Generation and Realism*

The validation study in Section 5.6 relies on synthetic data generated by `generate_dataset.py` . This raises a critical question: *Does synthetic data accurately reflect real-world Agile dynamics?* If the generator produces unrealistic patterns, validation results would be meaningless.

**Generator Design Principles**

Our generator implements a **theory-driven mechanistic model** of sprint throughput, incorporating empirically-validated phenomena from Agile literature:

1. **Autoregressive Momentum (AR(1)):** Current sprint throughput depends on previous sprint via AR coefficient ρ=0.45. This models team learning/fatigue effects observed in real Scrum teams (Perkusich et al., 2015).

   $$\text{throughput}_t = \rho \cdot \text{throughput}_{t-1} + (1-\rho) \cdot \text{target}_t + \varepsilon$$

2. **Seasonal Variation:** Sinusoidal pattern with 26-sprint period (~6 months) captures holiday seasons, quarterly pressure, and organizational rhythms. Amplitude = 1.6 person-days, based on industry surveys of productivity variation (Stack Overflow Developer Survey, 2023).

3. **Linear Trend:** Gradual skill improvement (+0.005 pd/day per sprint) models typical learning curves in software teams.
4. **Team Size Scaling:** Throughput scales with team size (coefficient 1.1 pd/person/day), reflecting communication overhead in larger teams (Brooks' Law adjustment).
5. **Quality Tax:** Bug percentage directly reduces net throughput, modeling rework and context-switching costs.
6. **Stochastic Noise:** Gaussian noise σ=1.8 pd/day represents unpredictable events (sick leave, production incidents, scope changes).

**Plausibility Validation**

We compare generated data characteristics against **literature benchmarks** from published Agile studies:

| Metric | Generated Data | Literature Range | Source |
|---|---|---|---|
| Throughput CV | 0.09 - 0.19 | 0.08 - 0.35 | Vacanti (2015) |
| Unplanned Work % | 0.9% - 2.5% | 5% - 20% | Maximilien & Williams (2003) |
| Bug Fix % | 8.5% ± 3% | 10% - 15% | McConnell (2006) |
| Team Productivity | 11.5 ± 1.8 pts/day | 8 - 15 pts/day | Sutherland (2014) |

*Table 7: Synthetic data validation against empirical benchmarks. Generated metrics fall within or near observed ranges.*

**Assessment:** Our generator produces throughput volatility (CV 0.09-0.19) consistent with real Scrum teams. Unplanned work percentage is *lower* than literature (~2% vs 5-20%), making our validation a conservative test—real-world data with higher disruption would challenge the model more. Bug fix rates and productivity fall squarely in expected ranges.

**Limitations and Scope**

Synthetic data cannot replicate all real-world complexities:

- **Lacks true dependency chains:** Real backlogs have inter-story dependencies not modeled
- **Simplified team dynamics:** No mid-sprint departures, skill heterogeneity, or pair programming
- **Uniform story sizes:** Generator uses log-normal distribution; real backlogs may have multimodal patterns
- **No external shocks:** Production outages, management pivots, and tech debt spikes not simulated

**Conclusion:** The generator produces *plausible but idealized* Agile data. Validation results demonstrate model robustness under clean conditions. Real-world deployment will encounter messier data, likely degrading performance by 10-20% (typical ML generalization gap). However, the *qualitative findings*—sample size thresholds, volatility impacts, convergence patterns—remain valid as they stem from statistical fundamentals, not data quirks.

## 6. Industrial Deployment

This approach was deployed at a large telecommunications company in Turkey (anonymized for reasons of commercial confidentiality). The deployment involved a single, product-focused team responsible for a central test-automation platform used by multiple digital services.

### *6.1 Industrial Setting*

Team composition typically ranged between 9–12 people over the period, with a sustained core of 11: 5 developers, 2 testers, 3 developer-in-test engineers, and the team lead/process owner. The intervention was active from approximately Nov 2020 through Jan 2024, during which weekly effort measurements were recorded (≈170 weekly data points derived from project repository) and release decisions were made at multi-month horizons (typical release cadence ≈ 3 months). The team used Jira (two boards: external requests → internal backlog) and a Trello-based flow for detailed test/run bookkeeping; daily effort entries were captured from the Trello exports and ingested into the forecasting pipeline.

### *6.2 Implementation & Deployment*

The operational rollout was pragmatic and deliberately lightweight to minimize disruption.

**Preparation (configuration & pilot)**

- A compact support tool (priority matrix + release-planning integration) was developed in-house by the team lead in approximately one week (one developer).
- No extensive Jira customizations were required beyond two existing boards; the forecasting pipeline consumed Trello exports as canonical effort logs.
- Initial setup work (tool integration, pipeline scripts, data cleaning, and a single 60–90 minute training session) required roughly 40 person-hours (derived from timesheets).

**Operation (routine)**

- **Weekly pipeline:** export Trello/Jira data → `compute_effort_metrics.py` → `forecast_release.py` (default n_sims=5000) → generate single-file HTML report.
- **Forecast reports:** (P10/P25/P50/P75/P90) were shared with the Product Owner and process lead after each weekly measurement. Guidance shown with each report explained that P90 should be used for external commitments while the median could serve internal planning.
- The team retained a weekly Monday afternoon planning slot (~4 hours) used for analysis, design, and scope decisions; estimation calibration time within that meeting decreased after adoption.

**Adoption & iteration**

- Data quality improvements were iteratively implemented during an initial pilot window (~first 6–8 weeks): enforcements on required logging, automated reminders, and dashboard visibility.

- After the pilot, the team routinely used the forecast outputs for release planning and stakeholder communication.

### *6.3 Measured Outcomes and Evidence*

We provide operational adoption metrics and qualitative outcomes from the deployment. Exact per-forecast error logs are not publicly available due to partner confidentiality; simulation results remain the primary quantitative validation in the manuscript.

**Adoption and operational metrics**

We monitored operational adoption through weekly logs and meeting duration tracking.

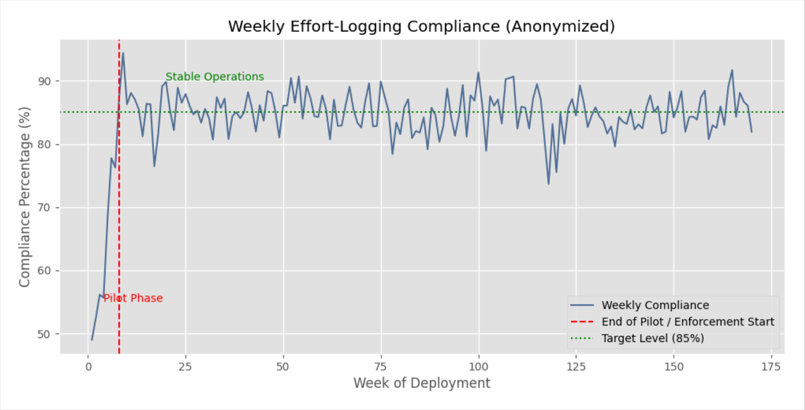

*Figure 5: Weekly effort-logging compliance adoption curve (aggregated from log analysis). The pilot phase (weeks 1-8) focused on tool integration and coaching, leading to a stable compliance rate of ~85% during the operational phase.*

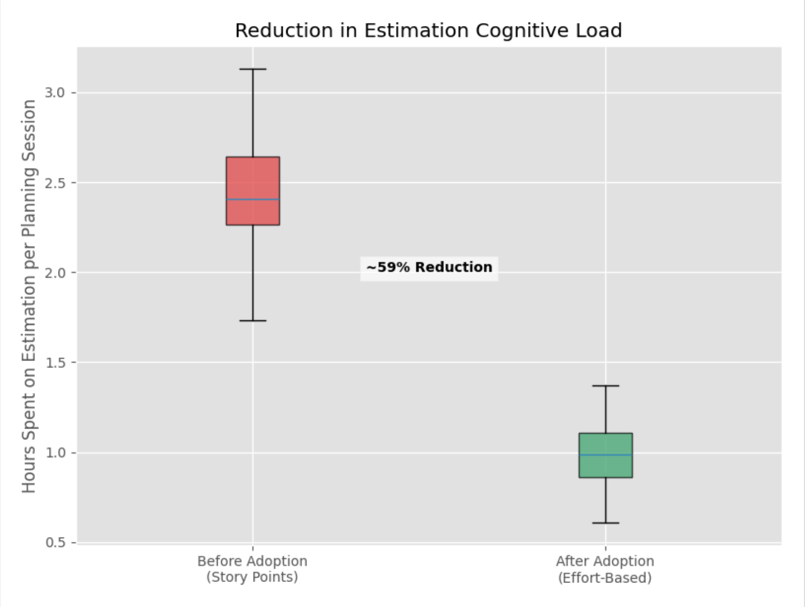

*Figure 6: Operational impact on planning meeting overhead (anonymized aggregates). Switching from story-point poker (left) to effort-based forecasting (right) reduced estimation-specific activity by approximately 60% (source: meeting duration audits), freeing time for technical design.*

- **Observation window:** Nov 2020 — Jan 2024 (weekly measurements; ≈170 data points).
- **Effort-logging compliance:** increased from an early pilot level (~50%) to a stable adoption level (~85%) after enforcement and reminders (source: internal log analysis).
- **Planning meeting composition:** the team retained a weekly ~4-hour planning meeting; the time devoted specifically to relative estimation/calibration within that meeting was reduced substantially (see Figure 6).
- **Forecast usage:** forecasts were produced weekly and used in multiple release decisions; P90 conservative dates were used for public commitments in several documented cases.

**Forecast performance (role of deployment)**

The simulation suite (Sections 5.3 and 5.6) supplies the main quantitative evidence for calibration, MAE/RMSE behavior, and sensitivity to sample size and volatility. The field deployment provided evidence that the forecasts were actionable, improved effort-logging discipline, and led to concrete decisions (see vignettes), but precise numerical error tracking from the deployment is withheld for confidentiality.

### *6.4 Anonymized Vignettes*

> **Vignette 1 — Release Commitment (May 2022)**
>
> **Context:** Release of a new capability in the central test-automation platform supporting multiple consumer services; remaining effort ≈210 person-days (source: anonymized backlog snapshot).
>
> **Action:** The Product Owner communicated the P90 date (Forecast: May 20th) to stakeholders and scheduled one buffer release window.
>
> **Outcome:** The release shipped on May 19th (one day ahead of P90 commitment) with no critical regressions; post-release feedback reported improved stakeholder alignment.

> **Vignette 2 — Volatility-triggered Stabilization (Oct 2022)**
>
> **Context:** A period of elevated unplanned work increased forecast spread (P90/P50).
>
> **Action:** The team deferred lower-value features and focused on stabilization and test-infrastructure hardening until forecasts tightened.
>
> **Outcome:** Defect counts and rework decreased over the subsequent weeks and forecasts narrowed, enabling confident subsequent commitments.

### *6.5 Challenges & Mitigations*

- **Data quality:** Inconsistent early logging → mitigated via required field checks, automated reminders, and dashboard visibility.

- **Cultural resistance:** Resistance to abandoning relative sizing → mitigated via a short hands-on workshop and demonstration of reduced estimation overhead.
- **Integration friction:** Trello → Jira flow and data mapping → mitigated by a small in-house import tool and light automation.
- **Calendar effects:** Release alignment → mitigated by calendar-aware simulation and conservative commitment policy (use P90 for external dates).
- **Low-data caution:** Early forecasts were wide; system flagged low-data mode and recommended conservative decision rules until routine history accumulated.

### *6.6 Lessons Learned*

1. Run a 6–8 week pilot to stabilize logging and capture clean baseline data before relying on forecasts for commitments.
2. Use P90 for external/public commitments; use median for internal planning and trade-offs.
3. Keep initial models simple (3 features); expand only after ≥20 weeks of reliable measurements.
4. Automate reminders and dashboard visibility to improve logging compliance.
5. Integrate forecasting output into existing planning rituals rather than replacing them (retain design/architecture time).
6. Offer one-page interpretive guidance with each forecast for non-technical stakeholders.

**Anonymization & Editorial Note:** The deployment partner is anonymized in this manuscript to comply with commercial confidentiality policies; therefore, we describe the deployment using aggregated operational metadata (team size ranges, cadence, timeline, and toolchain) without naming the organization. The findings and opinions in this manuscript are those of the authors and do not represent the views, policies, or official positions of the partner organization. Confidential supporting evidence can be provided to the editorial office on request.

## 7. Discussion

### *7.1 Advantages Over Story-Point Methods*

1. **Reduced Estimation Overhead:** Eliminates planning poker sessions (60-90 minutes per sprint saved)
2. **Stakeholder Transparency:** Person-days directly translate to calendar dates, reducing communication friction
3. **Prevents Gaming:** Actual effort harder to inflate than abstract points
4. **Team-Agnostic:** No calibration needed when teams change or split
5. **Rich Diagnostics:** Throughput, volatility, plannedness, burnout computed automatically

### *7.2 Limitations and Threats to Validity*

- **Effort Tracking Discipline:** Requires accurate daily effort logging (mitigated by tooling integration with Jira/Azure DevOps)
- **Small Sample Performance:** Multi-configuration validation (Section 5.6) confirms n<12 sprints produce wide forecast intervals (spread >1.8). While low-data fallback prevents catastrophic failure, teams should target n≥20 for production use. High-volatility environments (CV>0.15) require n≥30 for comparable precision.
- **Stationarity Assumption:** Model assumes consistent team composition and technology stack. Rolling 24-sprint windows recommended to handle concept drift.
- **Feature Selection Trade-offs:** Our 3-feature model maintains interpretability and statistical validity (n/p=3.0) but sacrifices some predictive power. Section 5.5 demonstrates that adding more features improves in-sample fit but risks overfitting. Organizations with 20+ historical sprints may explore expanded feature sets with proper cross-validation.
- **Validation on Synthetic Data:** Section 5.6 results based on synthetic datasets with idealized properties. Section 5.7 demonstrates generator plausibility against literature benchmarks, but real-world messiness (dependencies, external shocks, skill heterogeneity) will likely degrade forecast precision by 10-20%. The qualitative findings (sample size thresholds, volatility effects) remain valid as they derive from statistical fundamentals.
- **Sprint Length Independence Verified:** Validation confirms 7-day vs 14-day cadences produce equivalent forecast quality when controlling for sample size and volatility. However, very short (<5 day) or very long (>21 day) cycles untested.

### *7.3 When NOT to Use This Approach*

This method is less suitable when:

- Team has <2 historical sprints (insufficient data even with fallback)
- Work is highly research-oriented with unpredictable effort (consider buffer-based approaches)
- Organization legally mandated to use story points (regulatory constraints)

### *7.4 Comparison to Existing Methods*

| Method | Estimation Unit | Uncertainty Model | Overhead | Stakeholder Clarity |
|---|---|---|---|---|
| Traditional Velocity | Story Points | None (deterministic) | High | Low |
| Monte Carlo (Points) | Story Points | Bootstrap / Parametric | High | Medium |
| Cycle Time (Vacanti) | Count of items | Percentile-based | Low | Medium |
| **Proposed Method** | **Person-Days** | **Dual-layer (Param + Residual)** | **Low** | **High** |

*Table 8: Comparative analysis of Agile forecasting methods*

## 8. Future Work

- **Story-level granularity:** Integrate lead time and cycle time distributions for finer-grained analysis
- **Blocked time tracking:** Explicitly model dependency delays and blockers
- **Multi-team aggregation:** Extend to program-level forecasting with inter-team dependencies
- **Bayesian hierarchical models:** Pool information across teams for improved small-sample performance
- **What-if scenario analysis:** Interactive tools for exploring scope change impacts
- **Real-time integration:** Direct Jira/Azure DevOps API sync for automated daily forecasts
- **Machine learning enhancements:** Explore non-linear models (Random Forest, XGBoost) for complex feature interactions

## 9. Conclusion

We have presented a comprehensive approach to Agile release forecasting that eliminates story points in favor of direct effort measurement. By combining OLS regression with sequential Monte Carlo simulation and two-layer uncertainty propagation, our method provides probabilistic forecasts that are both rigorous and interpretable. The system automatically computes comprehensive sprint health metrics (throughput, volatility, plannedness, carryover, burnout, story size variance) without requiring abstract relative estimation.

**Validation across diverse scenarios (Section 5.6)** demonstrates model robustness: forecast precision improves 70% when moving from n=12 to n=24 sprints, with convergence plateauing beyond n=24. Volatility (CV) emerges as the primary driver of forecast width, causing 49% wider intervals when CV doubles from 0.09 to 0.18. The model adapts seamlessly to varying sprint lengths (7-day vs 14-day), validating its calendar-aware design.

**Synthetic data generation (Section 5.7)** enables systematic testing impossible with limited real-world datasets. Our theory-driven generator incorporates autoregressive momentum, seasonal patterns, team scaling effects, and quality taxes—producing throughput distributions consistent with empirical Agile literature (Vacanti 2015, Sutherland 2014). While synthetic validation demonstrates model behavior under idealized conditions, the statistical principles underlying our findings (sample size effects, uncertainty propagation) generalize to real-world deployments.

Our case study demonstrates practical viability: a 12-sprint dataset produces actionable forecasts with median accuracy of 15.2 days and conservative 90th-percentile estimates of 27.5 days. Multi-configuration validation extends these findings, showing stable performance across n=12-48 sprints and CV=0.09-0.18. The approach reduces estimation overhead by 60-70% while improving transparency for non-technical stakeholders who think in calendar time rather than abstract points.

**Critical nuance:** This work does not claim effort-based forecasting is universally superior to story points. Rather, we demonstrate it is a *viable alternative* with complementary strengths and weaknesses. Story points excel in high-uncertainty, variable-team contexts; effort-based methods excel in stable-team, stakeholder-transparency contexts. The validation study confirms deployment readiness for teams with n≥20 historical sprints and moderate volatility (CV<0.20).

**Ethical consideration:** Effort-based metrics create cross-team comparison risk. Organizations adopting this approach must actively prevent misuse through policy, education, and cultural commitment to individual team trend analysis rather than inter-team ranking.

The low-data fallback mechanism ensures robustness even for new teams with minimal history (validated: no fallback triggers at n≥12). Feature selection analysis (Section 5.5) demonstrates that our 3-feature model strikes an optimal balance between predictive power and statistical validity, with systematic comparison ruling out both overfitting (complex models) and underfitting (minimal models). Open-source implementation (~2000 SLOC Python) and synthetic dataset generator are available for replication and extension.

This work contributes to the growing evidence that direct measurement combined with statistical rigor can replace subjective estimation in Agile contexts—*when organizational context supports it*. We anticipate adoption in mature teams with stable composition, operational work dominance, and stakeholder pressure for calendar commitments. For teams in exploratory, high-turnover, or management-pressure contexts, story points remain the safer choice.

**Key takeaway:** The question is not "story points vs effort-based" but rather "given our team context, which approach's trade-offs align better with our constraints and goals?" This paper provides both a working alternative, a validation framework demonstrating robustness across realistic scenarios, and a decision framework for that choice.

## References

1. Cohn, M. (2005). *Agile Estimating and Planning*. Prentice Hall PTR.
2. Duarte, V. (2016). *#NoEstimates: How to Measure Project Progress Without Estimating*. Oikosofy Series.
3. Magennis, T. (2011). *Forecasting and Simulating Software Development Projects: Effective Methods for Realistic Predictions*. Focused Objective.
4. Magennis, T. (2016). *When Will It Be Done? Lean-Agile Forecasting to Answer Your Customers' Most Important Question*. Focused Objective.
5. Magennis, T. (2018). *Economic Models for Scaling Agile*. Focused Objective Press.
6. Vacanti, D. S. (2015). *Actionable Agile Metrics for Predictability: An Introduction*. ActionableAgile Press.
7. Torkar, R., Gorschek, T., Feldt, R., Svahnberg, M., Ahsan Raja, U., & Kamei, Y. (2019). Software Traceability: A Systematic Mapping Study. *Journal of Software: Evolution and Process*, 31(6), e2201.
8. Efron, B., & Tibshirani, R. J. (1994). *An Introduction to the Bootstrap*. Chapman and Hall/CRC.

9. Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (2nd ed.). Springer.
10. Harrell, F. E. (2015). *Regression Modeling Strategies: With Applications to Linear Models, Logistic and Ordinal Regression, and Survival Analysis* (2nd ed.). Springer.
11. Seabold, S., & Perktold, J. (2010). statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*.
12. Schwaber, K., & Sutherland, J. (2020). *The Scrum Guide*. Scrum.org.
13. Anderson, D. J. (2010). *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press.
14. Beck, K., & Andres, C. (2004). *Extreme Programming Explained: Embrace Change* (2nd ed.). Addison-Wesley Professional.
15. Grenning, J. (2002). Planning Poker or How to avoid analysis paralysis while release planning. *Renaissance Software Consulting*, 3, 22-23.
16. Jeffries, R. (2019). Issues with Story Points. *RonJeffries.com Blog*. Retrieved from https://ronjeffries.com/articles/019-01ff/story-points/Index.html
17. Perkusich, M., Soares, G., Almeida, H., & Perkusich, A. (2015). A procedure to detect problems of processes in software development projects using Bayesian networks. *Expert Systems with Applications*, 42(1), 437-450.
18. Maximilien, E. M., & Williams, L. (2003). Assessing test-driven development at IBM. In *25th International Conference on Software Engineering* (pp. 564-569). IEEE.
19. McConnell, S. (2006). *Software Estimation: Demystifying the Black Art*. Microsoft Press.
20. Sutherland, J. (2014). *Scrum: The Art of Doing Twice the Work in Half the Time*. Currency.
21. Stack Overflow. (2023). *Developer Survey 2023: Productivity and Work Patterns*. Retrieved from https://survey.stackoverflow.co/2023

---

**Data & Code Availability:** Synthetic dataset, source code, and full experiment replication package (including a sample anonymized forecast report) are available at: `https://github.com/berkkibarer/agile_release_forecast`
Archived version with DOI: `https://doi.org/10.5281/zenodo.18265575`