Defect Inflow Measurement and Forecasting System

Juliette Waltregny

October 17, 2025

Stakeholder

Project Manager (Shola)
Information need: to predict how many resources (engineers and QA) will be required next week based on current defect inflow and resolution rates.

1 Introduction

The aim of this project was to design and implement a data-driven measurement system that monitors *defect inflow* and *outflow*, evaluates software quality health, and forecasts the resources required for upcoming development cycles.

The design is informed by an in-depth interview with the project manager, Shola, who emphasized that effective defect monitoring and prediction should support decision-making in Agile sprint planning and help detect bottlenecks early.

As Shola explained:

> "The number of defects is very important—you want to know how many defects are coming in—and the severity is critical, because it helps with prioritization."

He further noted the connection between defect data and predictive planning:

> "If you have all these details, then you can predict how you would manage resources in advance, how to allocate them within your team, who is allocated to what, and what to prioritize first in order to give value to the customer."

Shola also described the operational importance of these indicators for project management:

> "If the amount of incoming is more than the outgoing that you've resolved, then that's a problem. And if the backlog list is growing crazily, then you know there's a problem."

These insights directly shaped the metrics, indicators, and forecasting model used in the system.

2 Measurement Design

2.1 Metrics

From the courses insights, three categories of measurements are important to identify:

1. **Base measures**, capturing raw quantities of defects and their characteristics.
2. **Derived measures**, computing relationships such as inflow versus outflow and backlog dynamics.
3. **Indicators**, assessing project health based on severity, resolution time, and accumulated backlog.

The stakeholder gave some examples of important measures to monitor:

> "The severity, resolution time, and also the amount of defects in the backlog will tell you how bad or how good that particular project is going."

2.2 Base Measures

The following base measures were implemented:

- `defects_inflow_total` – number of new defects opened that week.
- `defects_outflow_total` – number of defects resolved that week.
- `severity_critical_in`, `severity_high_in`, `severity_medium_in`, `severity_low_in` – weekly severity distribution.
- `avg_resolution_time_hours` – average time taken to resolve a defect.
- `backlog_total` – cumulative number of unresolved defects.

These measures reflect Shola's emphasis on monitoring not just quantities, but their *context and severity*, as he stated:

> "Depending on the severity, that helps in prioritization. It allows you to know where the bottleneck is, especially if it's an important aspect of your development model."

2.3 Derived Measures

Derived measures provide a deeper understanding of project performance over time:

$$\texttt{net\_flow} = \texttt{defects\_inflow\_total} - \texttt{defects\_outflow\_total}$$

$$\texttt{severity\_weighted\_inflow} = \sum_i w_i \times \texttt{severity\_i\_in}$$

$$\texttt{severe\_inflow} = \texttt{critical} + \texttt{high}$$

$$\texttt{mttr\_hours} = \texttt{avg\_resolution\_time\_hours}$$

These variables were selected based on Shola's observation that trends in these measures reveal potential bottlenecks and quality issues:

> "If you're having similarity, then there's a problem with the quality of code... those indicators allow you to know what the root cause is."

2.4 Indicators and Thresholds

Indicators interpret the derived measures into actionable signals. Shola emphasized that sustained imbalance between inflow and outflow, or rapid backlog growth, are key indicators of trouble:

> "When continuously the amount of defects exceeds the amount of resolved defects, then there's a problem."

Based on his practical experience, the following health thresholds were implemented in the configuration:

- `healthy_max_per_deployment` = 10
  This threshold defines what is considered an acceptable number of defects detected after a deployment. According to Shola, "If we are having between 7 to 10 defects per deployment, we can say we are still within a healthy threshold. But if it's 20 or more, then we have to start paying attention to technical debt." This value thus represents the upper boundary of what can be tolerated without triggering a red health status.
- `inflow_gt_outflow_consecutive_weeks` = 2
  This setting specifies how many consecutive weeks the inflow (new defects) can exceed the outflow (resolved defects) before the system flags an `inflow>outflow` warning. Shola emphasized that "when continuously the amount of defects exceeds the amount of resolved defects, then there's a problem." A two-week tolerance allows the team to account for normal fluctuations while still catching sustained imbalance early.
- `backlog_healthy_max` = 15
  This defines the maximum acceptable number of unresolved defects in the backlog before the system indicates that quality or capacity may be degrading. As Shola noted, "If the backlog list is growing crazily, then you know there's a problem." Setting the threshold to 15 ensures that minor fluctuations are not overreacted to, but significant accumulation triggers a red indicator.
- `critical_severe_min` = 5
  This parameter defines the minimum number of high or critical severity defects required within a short time window to raise a `severe_spike` alert. Shola linked this directly to prioritization and sprint adjustments: "When the severity level is high and it's an important module, we know we need to change the sprint goal and reallocate resources to address it immediately." Reaching this threshold indicates that the system may be facing stability risks that warrant urgent intervention.

These values reflect empirical guidelines provided by Shola:

> "If we are having between 7 to 10 defects per deployment, we can say we are still within a healthy threshold. But if it's 20 or more, then we have to start paying attention to technical debt."

He also specified that severity thresholds drive resource planning decisions:

> "When the severity level is high and it's an important module, we know we need to change the sprint goal and reallocate resources to address it immediately."

3 Forecasting Model

The system predicts upcoming defect inflow and outflow using two forecasting approaches:

- **Exponential Weighted Moving Average (EWMA)** for short-term (1–3 week) smoothing.
- **Linear Regression** for long-term trend projection.

The hybrid design ensures responsiveness in the short term and stability in longer-term forecasting. Shola described this predictive perspective as essential:

> "With data from similar projects, we can predict what to expect—it helps us plan human resources, cost, and the type of skills needed in advance."

4 Resource Planning

The resource model estimates total engineering and QA hours using severity-weighted forecasts:

$$H_{total} = \sum_i (\text{hours\_per\_defect}_i \times \text{predicted\_defects}_i)$$

$$E_{required} = \lceil \frac{H_{total}}{\text{capacity}} \rceil$$

$$H_{QA} = H_{total} \times \text{qa\_share}$$

With capacity being the number of hours that each engineer can work on the defects and qa_share being the percentage of work that should be allocated to quality assurance, which is a set constant.

This directly addresses Shola's need for predictive workforce management:

> "Because if you're having similarity, then there's a problem with the quality of code. But anyway, these days, quality check out there before everything is pushed to a live environment. And then it allows for quality assurance to know, okay, what amount of resources should I allocate for quality assurance to improve this overall defect that we keep getting."

5 Results and Interpretation

Applied to data from the Microsoft PowerToys repository, the system produced the following forecasts:

- **Predicted inflow:** 97 defects
- **Predicted outflow:** 77 defects
- **Estimated total hours:** 388 h
- **Recommended engineers:** 13
- **QA effort:** 116.4 h

Persistent inflow greater than outflow was observed across multiple weeks, which aligns with Shola's interpretation of problematic conditions:

> "If the amount of defect inflow is higher, then it's a problem, definitely."

This imbalance indicates that new defects are being created faster than they are being resolved, leading to a progressive increase in backlog. Several underlying causes may explain this trend:

1. **High user activity and reporting volume.**
   PowerToys is an open-source project with a large user community. The number of users generating bug reports is high, and many of these issues are minor or duplicates.

2. **Limited developer capacity.**
   The project relies on a relatively small team of maintainers compared to the number of active users. Weekly outflow is therefore constrained by available developer hours. The resource planning model captures this by recommending 13 engineers for the next cycle, indicating that the current team may be under-resourced.

3. **Backlog accumulation from prior sprints.**
   Sustained inflow exceeding outflow over several weeks suggests unresolved technical debt. Once backlog levels rise beyond the healthy threshold (15 unresolved defects), new inflow compounds the existing issue, reducing available time for new work.

From a management perspective, the observed **inflow** $>$ **outflow** trend reflects a predictable maintenance bottleneck rather than a system malfunction.

In summary, the PowerToys case demonstrates how the defect inflow measurement system effectively captures capacity constraints, backlog growth, and release-related fluctuations. The resulting indicators and forecasts provide actionable insights for maintaining project stability.

6 Conclusion

This measurement system operationalizes Shola's perspective into a practical predictive model. It supports proactive decision-making by combining:

- **Base measures** for defect monitoring,
- **Derived measures** for identifying quality trends, and
- **Indicators** for health assessment and prioritization.

Through data-driven forecasting and configurable thresholds, it helps managers like Shola allocate resources, define sprint goals, and reduce production risks.