Measuring What Matters:
How CMMI-DEV Maturity Level 3 Turns Delivery Metrics into Proven Business ROI

By Sherif Salah Ibrahim, Value Prime · October 26, 2026

Abstract

Most software companies I visit can show me a dashboard; far fewer can tell me what those numbers actually bought them. In this article I walk through three release cycles of an employee portal built by one of our clients, a regional enterprise working at CMMI-DEV Maturity Level 3 (V3.0). The portal pulls HR and leave data from Odoo ERP and sits alongside Microsoft 365, so it has to be quick, secure and correct. I describe the problems the team hit, what they changed, and how the Managing Performance and Measurement (MPM) practices turned each fix into evidence that leadership can act on. Some metrics improved a lot, a few didn't, and the financial indicators haven't been switched on yet, which I flag where it matters. I close with four recommendations for executives.

Keywords: CMMI-DEV, CMMI Maturity Level 3, CMMI-DEV V3.0, measurement and performance management, MPM, realized ROI, software delivery metrics, CI/CD pipeline reliability, SonarQube quality gate, DevOps ROI, CMMI business value

www.value-prime.com

Lots of Numbers, Not Much Proof

If you ask a delivery manager how the last release went, nine times out of ten you get a feeling (“it was OK, a bit rushed at the end”) rather than an answer. Ask a CTO what last year's process improvement budget actually returned and the room goes quiet. The data is there. Jira has it, the pipeline has it, SonarQube has it. What's missing is a link from a number to a decision, and from the decision to a result.

That gap is exactly what MPM in CMMI-DEV was written for. At Level 3 the rule is simple, though surprisingly hard to fake. A metric exists only because some business objective needs it. Miss a target and you write down why (the real why, not the polite one) and what you'll do about it. Then, next cycle, somebody goes back and checks whether it worked. Do that release after release and the meetings change. People stop defending their opinions and start arguing over the same evidence, which is a much better argument to have, and ROI becomes something you can put in front of the board rather than a promise.

The Setting: An Employee Portal Where Mistakes Show

The client here is building a “digital workplace.” In plain words that's one portal for the things employees otherwise chase across five systems: HR self-service, the staff directory, the org chart, process documents, a calendar and a few short courses. Under it sits React, ASP.NET Core on .NET 9 and SQL Server, all packed in Docker containers on the client's own servers, with the usual four stops (dev, test, UAT, prod). The team plans in Jira, and builds and deploys from Azure DevOps.

There are two design choices I'd point to, because everything later depends on them. First, Odoo ERP stays the master for HR and leave data and only the back end is allowed to talk to it (the browser never does, which security people like). Second, code can't merge until it survives an automated gate (SonarQube, unit tests with coverage, integration tests against an Odoo sandbox), and if something still goes wrong in production a health check quietly rolls the release back, no 2 a.m. phone call needed. Why so strict? Because the portal carries employee records and leave balances, and the business wants pages under three seconds with roughly a hundred people on it at the same time. Get that wrong and HR hears about it before IT does.

Where CMMI-DEV Level 3 Comes In

Most teams measure something. What Level 3 adds is that everybody measures the same things, the same way, on the same rhythm, so numbers from two teams can actually be compared. Each metric is pinned to a goal an executive would recognize; “pipeline success rate” isn't left as an engineering stat, it sits under “improve operational efficiency

and quality of execution." And the team is expected to be honest about causes. One entry I liked a lot just said the team hadn't logged its hours in Jira. Not glamorous! But it's the reason the next number could be trusted.

Level 3 also pushes you to tie each metric to a real control point in how the software is built and shipped, which is what lets a CEO see why a percentage deserves attention. For the main ones it looks like this:

Metric	Where it's controlled	Why the business cares
Pipeline success rate	Automated quality gate before every merge	Defects are stopped before they reach any test environment
Unit test coverage	Coverage check inside the same gate	A safety net under code that handles HR and leave data
Cycle and lead time	The Odoo integration and its data quality	Unclear Odoo data was the biggest single source of delay
Reopened bugs	The path from development to testing to UAT	Every reopened bug means another test round and deployment
Team utilization	Time logged in Jira	Capacity data the PMO can plan and bill against

Three Release Cycles: Problem, Fix, Result

CI/CD pipeline success rate (target: 90% or higher)

- **Starting point:** 83.9% of runs passed in R1.2, close enough to the 90% target that nobody worried.
- **The problem:** then R1.3 came in at 29.0%. Back-end builds were failing the gate over and over, on coverage, duplicated code and security hotspots.
- **The fix:** the developers worked through the Sonar findings and added their own checks before pushing, so problems got caught on the laptop and not in the pipeline.
- **Result:** R1.4 recovered to 63.0%, a 34-point jump in one cycle. Good, but still below R1.2 and well short of 90%.
- **Business value:** every one of those failed runs was a defect that never reached a tester, let alone an employee. What it cost was rework hours, so the aim now is to send fewer broken builds to the gate (not to loosen the gate).

Unit test coverage (target: 80% or higher)

- **Starting point:** 75.7% in R1.2; the back end was fine, the front-end tests were thin.
- **The fix:** more front-end tests and a minimum-coverage rule built into the pipeline.
- **Result:** 85.0% in R1.3 and 83.7% in R1.4, so above the 80% line two cycles running.
- **Business value:** this is the safety net under the leave and HR calculations. One gap is left: the front end still doesn't have a hard gate like the back end does.

Cycle time and lead time (targets: 3 and 5 days)

- **Starting point:** a change took 7.25 days of cycle time (7.5 days lead time) in R1.2, against targets of 3 and 5.
- **Root cause:** data coming back from Odoo kept surprising the developers, and every surprise became a change request in the middle of a sprint.
- **The fix:** sit down with the Odoo team and validate the data before development starts, rather than discovering it halfway.
- **Result:** cycle time dropped to 4.5 days in R1.3 (roughly 38% quicker). Lead time hardly moved, 7.5 to 7.0, which tells me there's a queue somewhere upstream that still needs its own root-cause look.
- **Business value:** close to three working days saved on every change. The Odoo data-quality question has been formally open since July, though, and until somebody closes it the team will keep paying in days.

Team utilization (target: 80% or higher)

- **Starting point:** 26.1% in R1.2. People were working; they just weren't logging hours in Jira.
- **The fix:** plain follow-up from the team lead so actual time gets recorded.
- **Result:** 94.5% in R1.3. Nobody worked a minute more, the PMO simply started getting capacity data it can plan and invoice from. (R1.4 wasn't measured.)

www.value-prime.com

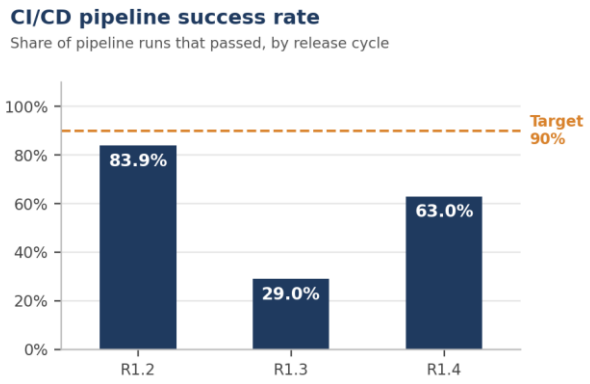

Figure 1. Pipeline success rate by release cycle.

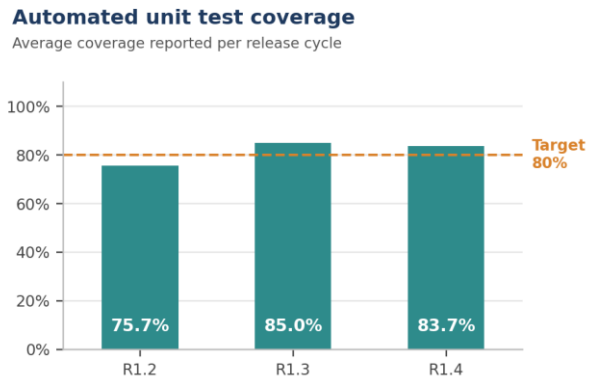

Figure 2. Unit test coverage by release cycle.

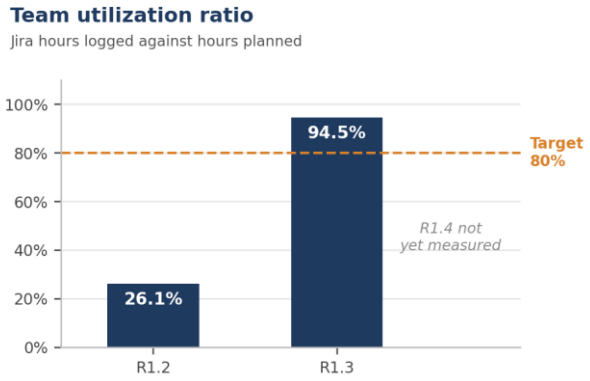

Figure 3. Team utilization, before and after the time-logging fix.

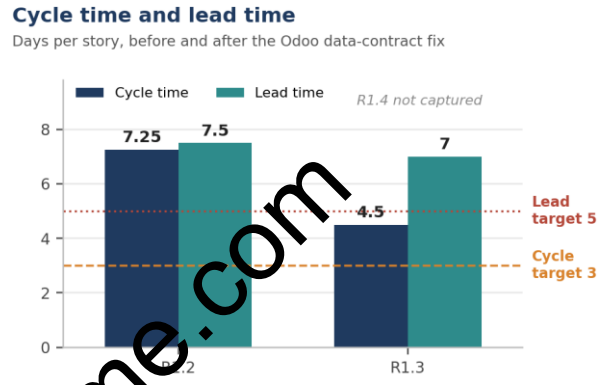

Figure 4. Cycle and lead time, before and after the Odoo data fix.

Four metrics that did not improve

- **Reopened bugs:** 20%, 14% and then 30%, against a 2% target. The cause was the same every single time: fixes got verified on the development environment instead of the QC one.
- **Velocity variance:** 51.6% in R1.2 and 89.2% in R1.3 versus a 15% ceiling, mostly because stories reached testers late in the sprint.
- **Weighted defect density:** went from 1.5% to 23.5% in R1.3 (limit 10%), the sprint where a big pile of carried-over stories all landed at once.
- **Bug detection rate:** 0.11, 0.10, 0.18 bugs per test case, below the 0.25 target, which is what you'd expect when stories hit QC before the happy path even works.

I left these in deliberately. If a report only ever shows wins, I stop believing it, and I suspect your board does too.

Where the ROI Actually Shows Up

A word of caution first. Not one of these numbers is in dollars or riyals. On-time delivery, budget adherence and customer satisfaction, which are the indicators that would complete a formal ROI case, are still empty for this client, and the requirements metrics show no activity after R1.2 (I'm not counting that as an improvement). That's where this ML3 journey is today, and it's better to say so.

Even so, the return is real and you can already see it, mainly because MPM forces it to be written down. Each change now moves almost three days faster, and if you multiply that by your blended daily team cost you get an actual figure for your own context. Defects are stopped at the pull request, which is the cheapest place there is to fix them, instead of turning up in UAT or in front of employees. The PMO can finally plan and bill from real hours. Automatic rollback protects production too, but since no rollbacks have been counted yet I'd still call that one a promise, not a result.

If I had to put one sentence in front of a CEO it would be this: without the MPM loop none of these gains could be proven, they'd just be stories people tell in steering committees. With the loop, every gain has a baseline, a cause, an action and a measured result, and that is how money spent on process becomes realized ROI.

www.value-prime.com

What I'd Tell Leadership

For a leadership team deciding how far to take this, I'd put my energy into four recommendations before touching any dashboard:

1. **Look back before you look forward.** Check whether last cycle's fixes worked before setting new targets. Reopened bugs show why this matter, since that fix still hasn't held.
2. **Close the Odoo data contract.** Agree the interface in writing and back it with contract tests, so data surprises stop landing in the middle of sprints.
3. **Give the front end the same gate, and let the security scan say no.** Front-end failures caused a lot of the early pain, and right now the container scan reports vulnerabilities without ever blocking a release.
4. **Measure what the business was promised.** Start tracking page speed, Odoo response time and concurrent users, then switch on on-time delivery, budget adherence and customer satisfaction. That is the point where ROI shows up with a currency sign.

None of this requires buying anything. Jira, Azure DevOps and SonarQube are already spitting out every number I've quoted here. What's missing in most companies I see is the habit itself: ask why, write the answer down, look again next cycle.

References

1. ISACA / CMMI Institute. *CMMI V3.0 Model* — Managing Performance and Measurement (MPM) practice area. isaca.org/cmmi.
2. Basili, V. R., Caldiera, G., & Rombach, H. D. (1994). "The Goal Question Metric Approach." *Encyclopedia of Software Engineering*. Wiley.
3. Kaplan, R. S., & Norton, D. P. (1996). *The Balanced Scorecard: Translating Strategy into Action*. Harvard Business School Press.
4. SonarSource. "Quality Gates." *SonarQube Documentation*. docs.sonarsource.com.
5. Atlassian. "Agile Metrics: Velocity, Cycle Time, and Lead Time." *Jira Software Documentation*. atlassian.com.
6. ValuePrime client engagement records, 2026 (client anonymized).

www.value-prime.com

About the Author

Sherif Salah, MBA
Co-founder, ValuePrime

Performance excellence expert with 24+ years leading enterprise governance and operational-excellence engagements across Egypt and the GCC. CMMI Maturity consultant (ML3-ML5), COBIT 2019 implementer & assessor, and TOGAF-certified Enterprise Architect. Sherif has guided software, fintech, government, and healthcare organizations from reactive delivery to predictable, data-driven performance.

Contact

- Email: sherif.salah@value-prime.com
- Website: www.value-prime.com
- LinkedIn: www.linkedin.com/in/sherif-salah-073009b