Risk-Based Computer System Validation Framework for Open-Source Manufacturing Software

Aldon P Smith

Published 15 May 2026

Executive Summary

Manufacturing software is changing quickly. Modern factories are increasingly built around connected equipment, historians, automation platforms, cloud services, analytics tools, AI-assisted workflows, and open-source software. At the same time, regulated manufacturing environments still require disciplined control over systems that affect product quality, patient safety, data integrity, and operational decision-making.

This creates a practical tension: open-source software moves through transparency, iteration, collaboration, and community contribution, while regulated manufacturing depends on documented intended use, controlled change, traceable requirements, testing evidence, access control, auditability, and quality oversight.

The Open Factory Initiative proposes this framework as a practical bridge between those two worlds.

The goal is not to make every open-source tool "validated" by default. That would be misleading. A software repository is not validated simply because it is documented, tested, or popular. Validation depends on the system's intended use, deployment context, configuration, data flows, users, procedures, risk controls, and maintained evidence within a specific operating environment.

Instead, this framework defines how open-source manufacturing software can be designed, governed, released, and adopted in a way that makes risk-based Computer System Validation (CSV) and Computer Software Assurance (CSA) more practical for regulated manufacturers.

In simple terms:

> Open-source software can be compatible with regulated manufacturing, but only when it is paired with clear intended use, risk-based assurance, controlled deployment, strong software governance, data integrity controls, and documented evidence.

This framework is especially relevant to the Open Factory Initiative's work on open-source factory intelligence, manufacturing data contextualization, historian integration, quality workflows, validation documentation, AI-assisted analysis, and operational decision support.

Important Note

This framework is not legal advice, regulatory advice, or a substitute for an organization's quality system, validation procedures, regulatory interpretation, or independent quality review. It is a

practical engineering and governance framework published by the Open Factory Initiative to help teams think clearly about risk-based assurance for open-source manufacturing software.

Each regulated organization remains responsible for determining applicable regulations, defining intended use, assessing risk, approving validation strategy, and maintaining objective evidence for its own deployment.

1. Why This Framework Exists

Open-source software has transformed modern technology. It powers cloud infrastructure, cybersecurity tools, machine learning frameworks, observability stacks, databases, developer platforms, and automation ecosystems. Manufacturing is no exception. Factories increasingly rely on open-source libraries, protocols, data tools, and integration frameworks, even when those components are embedded inside commercial platforms.

However, in regulated manufacturing, especially in life sciences, biotechnology, pharmaceuticals, medical devices, and other quality-critical industries, software adoption is not only a technical decision. It is also a quality decision.

A system may need validation or assurance when it is used to:

- Control or monitor a manufacturing process.
- Generate, modify, store, retrieve, transmit, or archive regulated records.
- Support quality decisions.
- Trigger deviations, investigations, CAPA, or batch-related actions.
- Provide data used for product release or process understanding.
- Manage recipes, equipment states, alarms, calibration, maintenance, or environmental conditions.
- Support audit trails, electronic signatures, or data review.
- Provide recommendations that influence human decisions in a cGMP environment.

Traditional CSV programs were often built around commercial vendor systems, fixed releases, formal requirements documents, scripted testing, and periodic upgrades. Open-source software introduces a different operating model:

- Source code may be public.
- Contributions may come from many people or organizations.
- Releases may occur frequently.
- Dependencies may change rapidly.
- Documentation may be community-maintained.
- Users may modify, configure, fork, or self-host the software.
- Responsibility is distributed between maintainers and adopters.

This does not make open-source software unsuitable for regulated manufacturing. It means the assurance model must be explicit.

The purpose of this framework is to help open-source projects become easier to evaluate, adopt, and control in regulated manufacturing environments.

2. Core Position

The central position of this framework is:

> Validation is not a property of a repository. Validation is the documented demonstration that a specific system, configured for a specific intended use, in a specific operating environment, can perform reliably and appropriately for that use.

Therefore, an open-source manufacturing project should not claim that the software itself is universally validated. Instead, it should provide the structure, documentation, release discipline, test evidence, security practices, and adoption templates that help regulated users perform their own risk-based validation or assurance activities.

For open-source manufacturing software, the project maintainers and adopting organizations have different responsibilities.

Open-source maintainers should provide:

- Clear intended-use boundaries.
- Architecture documentation.
- Release notes.
- Versioned documentation.
- Test strategy and automated test evidence where possible.
- Security practices.
- Dependency transparency.
- Contribution controls.
- Known limitations.
- Configuration guidance.
- Example risk assessments and validation templates.

Adopting organizations should own:

- Their site-specific intended use.
- GxP impact assessment.
- Data integrity assessment.
- Configuration and deployment control.
- User access and procedural controls.
- Validation or assurance plan.
- Testing evidence for their use case.
- Training.
- Change control.
- Periodic review.
- Final quality approval.

This separation is essential. Open-source maintainers can make validation easier, but they cannot validate every future deployment context for every organization.

3. Guiding Principles

3.1 Intended Use Drives the Assurance Strategy

The same software can have different risk levels depending on how it is used.

A dashboard that displays non-GxP engineering metrics may require minimal assurance. The same dashboard displaying data used for batch release decisions may require far more rigorous controls.

A historian connector used for troubleshooting may be low or moderate risk. That same connector feeding quality review, deviation triage, or AI-assisted batch analysis may become higher risk.

The first question should never be, “Is this software validated?”

The better question is:

> What is this software intended to do in this environment, and what could happen if it fails, produces incorrect information, loses data, or is used incorrectly?

3.2 Risk Determines Rigor

Risk-based validation does not mean less discipline. It means the level of rigor should match the level of risk.

Low-risk systems may be supported by configuration review, limited functional testing, and basic change control. High-risk systems may require detailed requirements, traceability, scripted testing, audit trail verification, security testing, data integrity review, disaster recovery testing, and quality approval.

The assurance burden should increase when software affects product quality, patient safety, regulated records, process control, or quality decisions.

3.3 Configuration Is Part of the System

In modern software, configuration often determines behavior as much as source code does.

For open-source manufacturing systems, configuration may include:

- Environment variables.
- Connector settings.
- Equipment mappings.
- Historian tags.
- Data models.
- Workflow rules.
- AI prompts.
- Model selection.
- Access control rules.
- Notification thresholds.
- Integration endpoints.
- Deployment manifests.
- Infrastructure-as-code files.

A validation strategy that ignores configuration is incomplete. Configuration should be versioned, reviewed, approved, and included in the system baseline.

3.4 Open Source Does Not Remove Supplier Assessment

Open-source projects may not have a traditional vendor. That does not eliminate the need for supplier or software assessment. It changes the assessment.

Instead of asking only whether a vendor has a quality system, adopters should also evaluate:

- Project governance.

- Maintainer credibility.
- Release discipline.
- Issue response patterns.
- Security practices.
- Documentation quality.
- Test coverage.
- Dependency management.
- License compatibility.
- Community health.
- Long-term maintainability.

The assessment should be proportionate to intended use and risk.

3.5 Evidence Should Be Continuous Where Possible

Traditional validation often relies heavily on static documents created at a point in time. Modern software assurance should preserve required documentation, but also take advantage of automation.

Examples of continuous evidence include:

- Automated test results.
- Build logs.
- Code review records.
- Dependency scans.
- Static analysis results.
- Software bill of materials generation.
- Release checksums.
- Signed release artifacts.
- Infrastructure deployment records.
- Configuration change history.
- Audit trail verification reports.

Continuous evidence does not replace quality approval, but it can make validation more accurate, repeatable, and maintainable.

3.6 Data Integrity Is Non-Negotiable

Manufacturing intelligence depends on trustworthy data. If the source data is incomplete, manipulated, miscontextualized, overwritten, or poorly governed, then the intelligence layer built on top of it cannot be trusted.

Any open-source manufacturing software that touches regulated data should consider:

- Attribution.
- Legibility.
- Contemporaneous recording.
- Original records or true copies.
- Accuracy.
- Completeness.
- Consistency.
- Availability.

- Auditability.
- Retention.
- Backup and recovery.
- Access control.
- Review by exception.

Data integrity should be evaluated from data creation through use, review, retention, and disposition.

3.7 Security Is Part of Validation

Cybersecurity and validation should not be treated as separate worlds. A system cannot be considered trustworthy for regulated manufacturing if it is easy to compromise, misconfigure, impersonate, or tamper with.

For open-source manufacturing systems, security considerations should include:

- Authentication.
- Authorization.
- Secrets management.
- Secure deployment defaults.
- Dependency scanning.
- Vulnerability disclosure.
- Patch management.
- Logging and monitoring.
- Least-privilege access.
- Network segmentation guidance.
- Supply-chain security.
- Release integrity.

Security risks can become quality risks when they affect manufacturing data, system availability, audit trails, configuration, recipes, or quality records.

3.8 Human Accountability Must Remain Clear

AI-assisted or automated decision-support systems should not blur accountability.

If software recommends an action, flags an anomaly, summarizes a deviation, or suggests a root cause, the system should clearly define:

- What the software is allowed to do.
- What the software is not allowed to do.
- Who reviews the output.
- Who approves any regulated decision.
- How the output is recorded.
- How errors are handled.
- How model or logic changes are controlled.

A system that supports human decision-making should make the human review step explicit.

4. Scope of This Framework

This framework applies to open-source or source-available software used in or around manufacturing environments, especially software that supports factory intelligence, automation data, quality workflows, or operational analytics.

Examples include:

- Manufacturing data platforms.
- Historian integration tools.
- Equipment data contextualization systems.
- MES/QMS/CMMS/ERP integration layers.
- Dashboards and reporting tools.
- Workflow engines.
- Validation documentation tools.
- AI-assisted deviation analysis.
- AI-assisted troubleshooting.
- Knowledge graphs for manufacturing context.
- Event correlation systems.
- Open-source connectors and agents.
- Software used to monitor process performance or quality indicators.

This framework is not intended to replace product-specific, site-specific, or regulation-specific validation requirements. It is also not intended to cover every category of software, such as embedded medical device software, product software, or safety instrumented control systems, although some principles may still be useful.

5. Risk Classification Model

The first practical step is to classify the system based on intended use and potential impact.

5.1 Risk Dimensions

The following dimensions should be considered:

1. **Product quality impact**
   Could the system influence product quality, batch disposition, process parameters, or quality decisions?
2. **Patient or end-user safety impact**
   Could a failure indirectly or directly affect patient safety, device safety, or product safety?
3. **Data integrity impact**
   Does the system create, modify, store, retrieve, transmit, archive, or display regulated records or quality-critical data?
4. **Process control impact**
   Does the system control, adjust, trigger, or stop manufacturing processes?
5. **Decision-making impact**
   Does the system provide recommendations, classifications, predictions, summaries, or alerts that humans may rely on?

6. **Auditability impact**
   Does the system need to preserve audit trails, electronic signatures, approvals, or traceable records?
7. **Cybersecurity impact**
   Could compromise of the system affect manufacturing operations, quality records, equipment, or sensitive information?
8. **Business continuity impact**
   Would system downtime disrupt manufacturing, quality review, investigations, or release activities?

5.2 Suggested Risk Levels

Risk Level	Description	Example Uses	Typical Assurance Approach
Low	The system is informational, non-GxP, or has no meaningful impact on quality decisions or regulated records.	Engineering dashboard for non-critical equipment trends; internal prototype; development sandbox.	Documented intended use, basic configuration review, limited functional testing, standard change control.
Moderate	The system supports decision-making, displays quality-relevant data, or processes data that may inform human review, but does not independently make regulated decisions or control the process.	Historian data contextualization for process monitoring; deviation triage dashboard; quality trend review support.	GxP assessment, risk assessment, requirements/control matrix, risk-based testing, access control review, data integrity review, change control.
High	The system directly affects product quality, process control, regulated records, batch disposition, electronic signatures, or automated quality decisions.	Recipe control; batch release workflow; automated quality hold/release; electronic GMP record system; closed-loop process adjustment.	Formal validation plan, detailed requirements, traceability, scripted and automated testing, audit trail verification, security testing, backup/recovery testing, quality approval, periodic review.

The same platform may contain components with different risk levels. For example, a manufacturing intelligence platform could have a low-risk public documentation site, a moderate-risk dashboard module, and a high-risk workflow module used for electronic quality approvals.

Risk classification should be performed at the system, module, workflow, and data-flow level when appropriate.

6. The Framework

This framework uses nine lifecycle steps.

Step 1: Define Intended Use

Every validation effort should begin with a clear intended-use statement.

The intended-use statement should answer:

- What problem does the system solve?
- Who uses it?
- What data does it use?
- What outputs does it produce?
- Are the outputs informational, advisory, required, or controlling?
- Does the system affect GxP decisions?
- Does it create or manage regulated records?
- Does it interact with equipment, historians, MES, QMS, CMMS, ERP, or LIMS systems?
- What decisions could a user make based on the system output?
- What is explicitly out of scope?

A good intended-use statement prevents validation scope creep and reduces ambiguity.

Step 2: Assess GxP and Data Integrity Impact

The next step is to determine whether the system has GxP impact, data integrity impact, or both.

A system may be GxP-impacting if it affects:

- Product quality.
- Manufacturing process control.
- Quality records.
- Batch records.
- Deviations or investigations.
- CAPA.
- Calibration or maintenance decisions.
- Environmental monitoring.
- Laboratory or production data.
- Material status.
- Equipment status.
- Product release.

A system may have data integrity impact if it handles records or data that must be trustworthy, accurate, complete, available, attributable, and reviewable.

This assessment should be documented and approved by appropriate quality and system owner roles.

Step 3: Perform Risk Assessment

Risk assessment should identify what can go wrong and what controls are needed.

Common failure modes include:

- Incorrect data mapping.
- Missing historian tags.
- Incorrect units of measure.
- Time synchronization errors.
- Data loss.
- Duplicate records.
- Uncontrolled configuration changes.
- Unauthorized access.
- Incorrect calculations.
- Misleading dashboard visualizations.
- AI-generated incorrect summaries.
- Unreviewed model changes.
- Broken integrations.
- Incomplete audit trails.
- Failure to retain required records.
- Inability to reconstruct historical decisions.
- Security compromise.

For each risk, define:

- Cause.
- Potential impact.
- Severity.
- Likelihood.
- Detectability.
- Existing controls.
- Additional controls needed.
- Verification method.
- Residual risk.
- Approver.

The goal is not to eliminate every possible risk. The goal is to understand risk, apply appropriate controls, and document why the residual risk is acceptable.

Step 4: Define the Assurance Strategy

The assurance strategy should be based on intended use and risk.

Possible assurance activities include:

- Code review.
- Configuration review.
- Requirements review.
- Automated unit testing.
- Integration testing.
- Scripted user acceptance testing.

- Unscripted exploratory testing.
- Challenge testing.
- Data migration testing.
- Audit trail verification.
- Access control testing.
- Security testing.
- Backup and restore testing.
- Disaster recovery testing.
- Performance testing.
- AI output evaluation.
- Human review workflow testing.
- Regression testing.

Testing should focus most heavily on functions and failure modes that create the greatest risk.

For low-risk use cases, lightweight evidence may be appropriate. For high-risk use cases, more formal evidence is needed.

Step 5: Establish the Controlled Baseline

A system cannot be meaningfully validated if the baseline is unclear.

The controlled baseline should include:

- Software version.
- Release artifact.
- Source code reference or commit hash.
- Deployment architecture.
- Infrastructure configuration.
- Application configuration.
- Connected systems.
- Data sources.
- User roles.
- Permission model.
- Critical settings.
- AI models or prompts, if applicable.
- Dependencies.
- Operating procedures.
- Known limitations.

For open-source software, the baseline should also identify whether the adopting organization is using an official release, a fork, a patched version, or a custom build.

Step 6: Verify the System

Verification should demonstrate that the system is fit for its intended use.

Evidence may include:

- Requirements verification.
- Risk-control verification.
- Functional tests.

- Integration tests.
- Data integrity tests.
- Configuration verification.
- Security verification.
- User role testing.
- Audit trail testing.
- Report accuracy testing.
- Error handling tests.
- Regression tests.
- AI output evaluation.

Testing does not always need to be scripted in the traditional sense. The level of scripting should match risk. However, evidence should be clear enough for an independent reviewer to understand what was tested, what passed, what failed, and how failures were resolved.

Step 7: Approve and Release

Before go-live, the system should be reviewed and approved based on its risk classification.

A release package may include:

- Intended-use statement.
- GxP assessment.
- Risk assessment.
- Validation or assurance plan.
- Requirements/control matrix.
- Test evidence.
- Open defects and risk acceptance.
- Configuration baseline.
- Security review.
- Training confirmation.
- SOP references.
- Release notes.
- Quality approval.

For open-source projects, maintainers can publish a release package that helps adopters evaluate the software. The adopter still needs to approve the site-specific release into its own environment.

Step 8: Maintain the System Under Change Control

Open-source software evolves continuously. Regulated deployments cannot blindly absorb every upstream change.

A controlled change process should define:

- How updates are evaluated.
- Who reviews release notes.
- How security patches are prioritized.
- How dependency updates are handled.
- What changes require regression testing.
- What changes require quality approval.

- How configuration changes are reviewed.
- How emergency fixes are documented.
- How forks or patches are maintained.

Change impact assessment should consider:

- Intended use.
- GxP impact.
- Data integrity impact.
- Security impact.
- User workflow impact.
- Integration impact.
- Validation evidence impact.

Step 9: Monitor, Review, and Retire

Validation is not finished at go-live. Systems should be monitored and periodically reviewed.

Periodic review may include:

- User access review.
- Audit trail review.
- Incident review.
- Deviation review.
- Backup/restore review.
- Security patch status.
- Dependency health.
- Open vulnerabilities.
- Configuration drift.
- Release currency.
- Training status.
- Known limitations.
- Continued fitness for intended use.

When a system is retired, the organization should define how records are retained, exported, archived, or migrated.

7. Open-Source Governance Controls

For open-source manufacturing software, project governance is part of trustworthiness.

A strong open-source project should define:

- Maintainer roles.
- Contribution process.
- Code review expectations.
- Branch protection rules.
- Release approval process.
- Security disclosure process.
- License policy.
- Dependency policy.
- Documentation standards.

- Testing expectations.
- Issue triage process.
- Roadmap transparency.
- Community code of conduct.
- Long-term sustainability model.

The goal is not to make community contribution difficult. The goal is to make changes understandable, reviewable, and traceable.

7.1 Recommended Repository Practices

Open-source manufacturing repositories should consider including:

- `README.md`
- `CONTRIBUTING.md`
- `CODE_OF_CONDUCT.md`
- `SECURITY.md`
- `LICENSE`
- `GOVERNANCE.md`
- `CHANGELOG.md`
- Architecture documentation.
- Release notes.
- Issue templates.
- Pull request templates.
- Threat model.
- Software bill of materials process.
- Dependency update policy.
- Validation support documentation.
- Example intended-use statements.
- Example risk assessments.
- Example test evidence.

7.2 Release Integrity

Regulated adopters need confidence that a deployed release corresponds to a known and reviewed version of the software.

Recommended practices include:

- Semantic versioning.
- Signed release artifacts.
- Checksums.
- Release notes.
- Tagged commits.
- Build provenance.
- Dependency inventory.
- Known vulnerability summary.
- Migration notes.
- Breaking-change notices.

For higher-risk use cases, adopting organizations may choose to build from source in their own controlled environment, mirror dependencies, or maintain an internally approved release branch.

8. Data Integrity Controls

Any software that handles manufacturing or quality data should be designed with data integrity in mind.

Important controls include:

- Unique user accounts.
- Role-based access control.
- Time synchronization.
- Audit trails for critical actions.
- Protection against unauthorized data changes.
- Backup and recovery.
- Record retention.
- Data export controls.
- Review workflows.
- Error handling.
- Data lineage.
- Source-system traceability.
- Versioned configuration.
- Controlled calculations.
- Report generation controls.

For manufacturing intelligence systems, data integrity also includes contextual integrity. A data point is not useful if the value is correct but the context is wrong.

Examples of contextual integrity issues include:

- Tag mapped to the wrong equipment.
- Incorrect engineering units.
- Time zone mismatch.
- Batch phase incorrectly associated with process data.
- Sensor data displayed without calibration status.
- Alarm data separated from relevant process conditions.
- AI analysis using incomplete or stale data.

Factory intelligence systems should treat context as a controlled asset.

9. AI and Machine Learning

AI and machine learning introduce additional assurance considerations because outputs may be probabilistic, context-dependent, or difficult to fully predict.

In regulated manufacturing, AI should be introduced carefully and with clear boundaries.

9.1 AI Intended Use Categories

Category	Description	Example	Assurance Considerations
Informational	AI summarizes or organizes information, but users do not rely on it for regulated decisions.	Summarizing maintenance notes for engineering review.	Basic review, clear limitations, user training.
Advisory	AI provides recommendations, but a qualified human reviews and decides.	Suggesting likely causes of a deviation.	Human-in-the-loop review, output evaluation, traceability, error handling.
Workflow-supporting	AI affects routing, prioritization, or task generation.	Prioritizing quality events based on risk signals.	Workflow verification, bias/error review, auditability, override controls.
Decision-impacting	AI output materially influences quality decisions.	Recommending batch disposition or quality hold.	High rigor, formal validation, quality approval, explainability, strong oversight.
Autonomous	AI directly controls or executes regulated actions without human approval.	Automatically changing process parameters based on model output.	Generally requires the highest level of scrutiny and may be inappropriate without extensive controls.

9.2 AI-Specific Controls

AI-enabled manufacturing systems should define:

- Approved use cases.
- Prohibited use cases.
- Human review requirements.
- Model version.
- Prompt or instruction version.
- Training or reference data sources.
- Evaluation datasets.
- Acceptance criteria.
- Known limitations.
- Monitoring strategy.
- Drift detection.
- Escalation process.
- Override process.
- Output retention rules.
- Auditability expectations.

Prompts, model settings, retrieval sources, embeddings, system instructions, and evaluation criteria

may all become controlled configuration items depending on intended use.

9.3 Practical Rule for AI in cGMP Contexts

A useful starting rule is:

> AI may assist regulated manufacturing decisions, but it should not obscure the evidence, replace accountable human review, or make quality decisions without appropriate controls.

AI output should be treated as a system-generated recommendation or analysis, not as unquestionable truth.

10. Example Application: Factory Intelligence Platform

The Factory Intelligence Platform concept is an open-source infrastructure layer for connecting manufacturing systems, contextualizing operational data, and supporting human decision-making across production, quality, maintenance, validation, and planning.

Potential modules may include:

- Equipment and asset model.
- Historian connector.
- Event model.
- Quality event context engine.
- Maintenance and reliability context.
- Process anomaly detection.
- AI-assisted troubleshooting.
- Validation documentation support.
- Open-source community governance.
- Dashboards and reports.

Different modules would have different risk classifications.

Module	Example Intended Use	Likely Risk Level
Public documentation site	Explains architecture and community contribution model.	Low
Engineering sandbox connector	Pulls non-GxP historian data for development testing.	Low
Manufacturing context model	Maps equipment, tags, processes, and events.	Moderate, depending on use
Quality trend dashboard	Displays data used in quality review.	Moderate to High
AI deviation assistant	Suggests possible causes or related events for human review.	Moderate to High
Electronic approval workflow	Captures regulated approvals or electronic signatures.	High

Closed-loop control agent	Changes process parameters automatically.	High or outside initial scope

For an initial open-source release, it may be wise to focus on lower-risk and moderate-risk modules first, with clear limitations and strong validation support artifacts.

A responsible MVP could include:

- Read-only data ingestion.
- Historian connectivity patterns.
- Equipment/context model.
- Human-reviewed dashboards.
- Documentation templates.
- Example risk assessments.
- Security and governance documentation.
- No autonomous quality decisions.
- No closed-loop process control.

This allows the project to build trust before expanding into higher-risk workflows.

11. Recommended Validation Support Package for Open-Source Projects

An open-source manufacturing project should consider publishing a validation support package with each major release.

The package may include:

1. **System Overview**
   Description of the system architecture, major components, data flows, integrations, and intended boundaries.
2. **Intended Use Guide**
   Example intended-use statements for common deployment patterns.
3. **Risk Assessment Template**
   A template adopters can use to evaluate site-specific GxP, data integrity, security, and operational risk.
4. **Requirements or Control Matrix**
   A list of functional, security, data integrity, and configuration controls that can be mapped to tests.
5. **Test Evidence Summary**
   Summary of automated tests, manual tests, and known test coverage for the released version.
6. **Configuration Guide**
   Explanation of critical configuration items and recommended controls.
7. **Security Guide**
   Authentication, authorization, secrets management, deployment hardening, vulnerability reporting, and patch guidance.

8. **Release Notes**
   Features, fixes, breaking changes, migration notes, known issues, and compatibility information.
9. **Dependency Inventory**
   Software bill of materials or dependency list to support supply-chain review.
10. **Adopter Validation Checklist**
    Checklist for regulated organizations to adapt the package to their own quality system.

This package does not validate the system for every user. It gives adopters a stronger starting point.

12. Template: Intended Use Statement

A simple intended-use statement may follow this structure:

> The system is intended to [perform function] for [user group] using [data sources] in order to support [business or quality process]. The system output is used for [informational/advisory/required/controlling] purposes. The system does/does not create, modify, store, retrieve, transmit, or archive regulated records. The system does/does not directly control manufacturing equipment or process parameters. Final responsibility for regulated decisions remains with [role/group].

Example:

> The Factory Intelligence Platform historian connector is intended to retrieve read-only process data from an approved manufacturing historian and associate that data with equipment and process context for engineering and quality review. The connector does not modify source historian records, does not control manufacturing equipment, and does not make product disposition decisions. Outputs may support human review of manufacturing events, but final quality decisions remain with authorized quality personnel.

13. Template: Risk Assessment Questions

The following questions can help structure assessment:

1. What is the system intended to do?
2. What is explicitly out of scope?
3. What data sources does the system use?
4. Does the system create or modify regulated records?
5. Does the system display data used for quality decisions?
6. Does the system control or influence equipment?
7. Could incorrect output affect product quality?
8. Could incorrect output affect patient or user safety?
9. Could downtime affect manufacturing or quality operations?
10. Could unauthorized access affect data integrity?
11. Are audit trails required?
12. Are electronic signatures required?
13. Are calculations or transformations performed?
14. Are AI or ML outputs involved?

15. What human review steps are required?
16. What procedural controls are needed?
17. What technical controls are needed?
18. What testing is necessary to demonstrate fitness for intended use?
19. What residual risks remain?
20. Who approves the risk assessment?

14. Template: Release Acceptance Checklist

Before approving a release for regulated use, an organization may review:

- Intended use documented.
- GxP impact assessed.
- Data integrity impact assessed.
- Risk assessment completed.
- Validation or assurance plan approved.
- Requirements or controls identified.
- Critical configuration documented.
- Test evidence completed and reviewed.
- Open issues assessed.
- Security review completed.
- User roles verified.
- Audit trails verified, if applicable.
- Backup and recovery verified, if applicable.
- Training completed.
- SOPs updated.
- Release notes reviewed.
- Dependency risks reviewed.
- Quality approval completed.

15. What This Means for the Open Factory Initiative Website and Project

For the Open Factory Initiative, this framework establishes a philosophy for building open-source manufacturing infrastructure that regulated companies can take seriously.

The project should not simply build software and hope regulated users figure out validation later. Instead, validation readiness should be part of the architecture from the beginning.

That means:

- Clear intended-use boundaries.
- Read-only integrations before higher-risk control functions.
- Human-in-the-loop workflows before autonomous decision-making.
- Strong data lineage and contextual integrity.
- Security-by-design.
- Release discipline.
- Documentation that quality teams can review.
- Templates that help adopters perform their own validation.
- Transparent governance.
- Practical examples grounded in real manufacturing environments.

Open-source manufacturing software does not need to copy old validation habits blindly. But it does need to respect why validation exists: to protect product quality, patient safety, data integrity, and trust.

16. Conclusion

Open-source software has an important role to play in the future of manufacturing. It can make factory intelligence more transparent, interoperable, explainable, and accessible. But for regulated industries, openness alone is not enough.

The future of open-source manufacturing software should be built around both innovation and assurance.

A risk-based framework allows teams to avoid two common mistakes:

1. Treating every software tool as if it requires the same level of validation.
2. Treating open-source tools as if validation does not apply.

The better path is balanced:

- Define intended use.
- Assess risk.
- Apply appropriate controls.
- Preserve data integrity.
- Secure the software supply chain.
- Maintain human accountability.
- Document objective evidence.
- Keep the system under control as it evolves.

That is the foundation for open-source software that can earn trust in real manufacturing environments.

The Open Factory Initiative is built on the belief that the next generation of manufacturing intelligence should be open, governed, secure, explainable, and usable by the people who understand manufacturing best.

This framework is a starting point for that work and a public reference for how the Open Factory Initiative intends to approach validation readiness, software assurance, security, and responsible adoption in regulated manufacturing environments.

Appendix A: Example Evidence Package Structure

A regulated adopter could organize evidence as follows:

1. System overview
2. Intended-use statement
3. GxP impact assessment
4. Data integrity assessment
5. Risk assessment
6. Validation or assurance plan
7. Requirements/control matrix
8. Architecture diagram
9. Configuration baseline

10. Security assessment
11. Test protocol or test summary
12. Test evidence
13. Defect summary
14. Training evidence
15. SOP references
16. Release notes
17. Dependency inventory
18. Approval record
19. Periodic review record
20. Retirement or archival plan, when applicable

Appendix B: Example Open-Source Maintainer Commitments

An open-source project intended for regulated manufacturing adoption may choose to publicly commit to:

- Maintaining versioned releases.
- Publishing release notes.
- Maintaining a security policy.
- Reviewing code before merge.
- Tracking issues transparently.
- Publishing architecture documentation.
- Documenting known limitations.
- Providing upgrade guidance.
- Maintaining validation support templates.
- Avoiding unsupported regulatory claims.
- Encouraging adopters to validate their own deployment.

Appendix C: Practical Boundary Statement for AI Features

For early-stage AI features in regulated manufacturing software, a practical boundary statement may be:

> AI-generated outputs are intended to support human review and investigation. They are not intended to independently make product quality decisions, replace approved procedures, modify manufacturing parameters, or approve regulated records. Users are responsible for reviewing source data, verifying conclusions, and following site-approved quality procedures.

This kind of boundary helps preserve accountability while allowing teams to explore AI-assisted workflows responsibly.

Appendix D: Suggested Future Work

This framework can be expanded into more detailed templates and examples, including:

- Open-source supplier assessment checklist.
- Historian connector validation template.
- Manufacturing data model qualification guide.
- AI-assisted deviation review assurance plan.

- Open-source release qualification checklist.
- Data integrity test examples.
- Cybersecurity threat model for factory intelligence systems.
- Example traceability matrix.
- Example CSA-style test plan.
- Example periodic review checklist.

These artifacts would help manufacturing teams move from general principles to practical adoption.

Notes

1. This article was drafted and reviewed by the Open Factory Initiative team. AI tools may have been used for editing, organization, or drafting assistance; final content reflects human review and responsibility.