QAQuest: A Gamified Plugin for Manual Testing

Joyce Almeida
jta@cin.ufpe.br
Federal University of Pernambuco
Recife, Brazil

Breno Miranda
bafm@cin.ufpe.br
Federal University of Pernambuco
Recife, Brazil

Abstract

QAQuest is a gamification plugin for manual testing workflows. Its main goal is to improve the quality of test artifacts by rewarding good testing practices, instead of rewarding only the amount of work produced. The plugin uses a weighted scoring model that evaluates signals such as Gherkin structure, description completeness, requirement traceability, and execution evidence. It also includes ten badges with five progression levels, a quality bonus based on portfolio coverage, and immediate feedback after test updates, always with personal metrics and no public ranking. QAQuest was designed so that its scoring logic, feedback flow, and progression model can be instantiated in different organizational contexts and adapted to different test management platforms. The resulting tool offers six dashboard views that combine operational indicators, quality dimensions, progression history, sprint context, and adoption metrics for managers. This work delivers a practical plugin ready for real team use and a foundation for evaluating whether quality focused incentives can sustain engagement over time without creating the distortions commonly seen in approaches centered only on volume. As preliminary empirical input, an exploratory one-month industrial pilot in an anonymized QA context (N=5) indicated favorable perceptions of utility and fairness for private progression, while highlighting contributor attribution in shared Test Plans as the main operational bottleneck. This paper focuses on design and implementation; empirical effectiveness is an explicit next step. The plugin is publicly available in the Atlassian Marketplace: QAQuest for Jira Cloud. The video demonstration is available on Figshare and YouTube: QAQuest - Figshare; QAQuest - YouTube.

Keywords

Gamification, Manual Software Testing, Quality Assurance, Test Design

1 Introduction

Software testing is a critical activity responsible for evaluating whether a system meets requirements before release [21]. In agile contexts, it encompasses both automated regression suites and manual evaluation (exploratory testing, acceptance testing, and user experience validation), activities that require human judgment and remain essential despite increasing automation [**?**]. Manual testing in agile environments presents compounding pressures: sprint velocity demands reduce time available for thorough artifact documentation; QA roles receive less recognition than development roles, creating motivation deficits [12]; and the investment required to produce well-structured test cases (with given-when-then notation, requirement links, and attached execution evidence) yields no immediate visible return. The consequence is pervasive: portfolios of test cases with auto-generated titles, missing descriptions, and no attached evidence, creating technical debt that undermines defect detection and test reuse over time [8].

Gamification is defined as the integration of game design elements into contexts that are not games to encourage specific behaviors [6]. Applied to software engineering, it employs mechanics such as points, badges, levels, and leaderboards to increase engagement in activities like code review, documentation, and testing [16, 20]. However, research identifies documented risks that naive implementations incur: (a) *novelty fade*, where engagement gains decay within weeks as the initial appeal of game elements wears off [11]; (b) *crowding out*, where extrinsic rewards reduce intrinsic motivation when they feel controlling rather than supportive of autonomy [18]; (c) *behavioral distortion*, where rewarding easily counted outputs (test count, lines of code) leads practitioners to optimize for the measured metric rather than the underlying goal [13]; and (d) *social tension*, where competitive leaderboards damage team cohesion and create status anxiety [14, 19]. Sustainable gamification must therefore ground design choices in motivation theory, target structural quality rather than volume, and avoid mechanisms that compare team members against each other.

Pedreira et al. mapped 57 studies on gamification in software engineering by 2015, finding that most applications reward participation volume rather than artifact quality, and that few demonstrate sustained behavioral change in realistic settings [16]. Applications have addressed code review participation [15], agile training [25], and software engineering education [17], but the bias toward counting discrete actions rather than evaluating structural quality creates perverse incentives across all these domains. Within testing specifically, Jesus et al. found that most gamification proposals for testing employ point accumulation, leaderboards, and badges based on volume [4]. Mäntylä and Smolander synthesized 18 empirical studies and identified a fundamental tension: gamification increased test count but not defect detection rate, suggesting testers produced more tests of lower individual value [12]. Fulcini et al.'s comprehensive tool review confirmed that current gamified testing systems rely predominantly on competitive mechanics and scoring based on volume, documenting risks to team climate, intrinsic motivation erosion, and novelty fade [8]. Straubinger and Fraser demonstrated that achievement badges without public competition can increase testing behavior in IDE contexts [22], but their work addressed automated testing in development environments, leaving manual testing artifact quality in agile QA teams unaddressed. Educational use is also plausible, but the current QAQuest implementation depends on Jira/Xray entities and sprint metadata; classroom adoption would therefore require adapters or a simplified data provider.

The research gap is clear: no current tool applies gamification to manual testing artifact quality (rewarding structural signals such as description completeness, requirement traceability, and execution evidence) while avoiding competitive mechanics and the risks they

entail. QAQuest addresses this gap through a system grounded in Self-Determination Theory [5, 18] and Flow Theory [2, 7], organized around four design principles:

Principle 1: Reward structure, not count. QAQuest scores Gherkin structure, description completeness, traceability, and evidence, attributes that testing research identifies as indicators of portfolio value sustained over time [8, 12]. This aligns the measured behavior with the organizational goal of building a maintainable test portfolio, rather than accumulating test count [13, 16].

Principle 2: Progress relative to one's own baseline. Rather than imposing fixed quotas, QAQuest tracks each practitioner's evolution against their own portfolio history. A tester who improves description coverage from 40% to 55% receives recognition equal to one who improves from 80% to 85%, respecting autonomy and contextual variation [1, 18].

Principle 3: Personal progression without public ranking. All feedback is private; no leaderboard exposes comparative scores. Self-Determination Theory evidence shows that competitive visibility undermines intrinsic motivation and team collaboration [18, 19]; its absence removes status threat and preserves team climate [14].

Principle 4: Immediate feedback within the workflow. When a tester creates or updates a test, QAQuest computes the score within seconds and shows a contextual notification in the work interface. Real time, action-specific feedback maintains the challenge-skill balance that Flow Theory associates with sustained engagement [2, 9, 10].

2 Tool Description and Functionality

QAQuest is a gamification plugin for test management systems. It captures test artifact creation and update events, computes quality scores, and presents results through an interactive dashboard embedded in the team's work environment. Target users include QA leads, test engineers, and agile teams seeking to improve test artifact quality without introducing friction or team conflict. The same core logic can be instantiated in different organizational contexts and connected to different test management platforms.

End-to-End User Workflow. QAQuest is used in a continuous cycle embedded in day-to-day work. First, a tester creates or updates a *Test* or *Test Execution* issue in Jira/Xray. Second, webhook handlers process the event, recompute portfolio metrics from current artifacts, and enqueue a contextual reward message when a quality signal is present. Third, the tester receives near-real-time feedback through a toast notification and sees updated points, level, quality coverage, and badge progress in the dashboard. Fourth, QA leads inspect aggregated usage telemetry to monitor adoption. Progression does not reset at sprint boundaries by default. After reaching maximum badge level, the tester keeps the badge at level 5 while metric values continue to be shown; because scoring is recomputed from the current portfolio state, quality regressions can reduce coverage percentages and consequently reduce current level/badge status in percentage-based badges.

Preliminary Pilot Signals. Before Marketplace publication, QAQuest was exercised in an exploratory one-month industrial pilot with five QA professionals (3 practitioners via questionnaire and 2 managers via structured live feedback) in an anonymized organizational context. The goal was to assess perceived practical fit, not to establish causal effectiveness. Integrated feedback indicated positive utility and fairness signals for the non-competitive design, especially for private progression and contextual feedback. The main barrier reported by participants was contributor attribution in shared Test Plans, which can affect individual metric completeness and perceived fairness.

Scoring Engine: Rewarding Structural Quality. The scoring engine replaces simple artifact counts with evaluation of structural properties. Gherkin structure (given-when-then notation) receives the highest weight because this notation improves communication between developers, testers, and stakeholders and increases test reusability [4, 8]. Description completeness rewards documenting test intent and preconditions, information critical for long-term maintenance [17]. Traceability links connect tests to requirements, enabling coverage analysis [16]. Evidence attachment ensures that execution records include screenshots or logs that substantiate findings [15]. This addresses the core gap identified by Mäntylä et al. and Fulcini et al.: reward systems based only on counts distort behavior away from quality because they measure the wrong thing [8, 12]. By measuring what research identifies as indicators of portfolio value, QAQuest aligns the gamification incentive with the organizational goal. The current weight values are configuration defaults derived from literature-guided design and engineering judgment; they were not statistically optimized and should be interpreted as tunable parameters pending empirical calibration.

Badge Progression: Mastery Without Competition. QAQuest implements ten badges across two categories. Badges tied to artifact count (Test Architect, Defect Hunter, Retest Master, Plan Strategist) mark participation and experience depth. Badges tied to portfolio coverage percentages (Elite Documenter, Gherkin Guardian, Evidence Curator, Traceability Lord, Coverage Guardian, Sprint Finisher) recognize sustained investment in specific quality dimensions. Each badge has five levels calibrated to Flow Theory's prescription for graduated challenge: early levels are attainable with modest effort, providing psychological wins that prevent early abandonment [7]; later levels require consistent practice across extended sprints [9]. A central design choice is that badges are personal. Practitioners see only their own progression and never a ranked comparison with teammates. Most gamified testing tools employ badges anchored in competitive rank (e.g., "5th in team"), creating status threat that reduces motivation for lower-ranked individuals [14, 19]. By contrast, QAQuest replaces rank labels with competence-focused labels. Names like Architect, Guardian, and Master reinforce professional identity rather than social position, a mechanism that Self-Determination Theory predicts will produce more durable intrinsic motivation [5, 13, 18].

Portfolio Quality Bonus: Incentivizing Systemic Improvement. Beyond scoring individual artifacts, QAQuest computes a quality bonus from attribute coverage rates across the practitioner's entire portfolio. If 45 of 50 tests include description fields, the tester reaches 90% description coverage and earns a proportional bonus. The multipliers apply differential weights: Gherkin coverage (×1.5)

and within sprint resolution (×1.6) carry the highest values, reflecting their impact on structural quality [4] and delivery predictability [15]. This mechanic addresses an optimization gap created by artifact scoring. A practitioner could create a small number of structurally complete tests and receive significant rewards while leaving most of the portfolio undocumented [13, 16]. Portfolio bonuses make systemic coverage, not isolated wins, the target of the incentive.

Real Time Notifications: Closing the Feedback Loop. When a test issue is created or updated, QAQuest triggers a toast notification within seconds (e.g., "Excellent! You added Gherkin structure to TEST-457! +40 points"). The notification names the specific attribute rewarded, creating a direct stimulus-response loop that reinforces the behavior contributing to the score [10]. Contextual, action-specific feedback is significantly more effective at sustaining engagement than delayed aggregate reports [7, 9]. Most gamified testing tools report progress in daily or weekly summaries, which disconnects reward from action. Delivering feedback in the same interface where the tester is already working also eliminates context switching, supporting perceived ease of use [3].

3 User Interface and Dashboard Design

QAQuest organizes the interface into six informational blocks in a single dashboard overview. Figure 1 presents the full layout.

View 1: Operational Dashboard Overview. The dashboard consolidates personal points and level, badge progression, operational metrics, quality attributes, current sprint context, and sprint performance trend in a single screen. The design is intentionally personal and without comparison. Existing systems expose competitive rank or aggregate leaderboard position, creating status threat for lower-ranked performers [14, 19]. Showing only the practitioner's own metrics provides perceived usefulness [3] and supports competence perception without the motivation costs of comparative visibility [18].

To balance transparency and cognitive load, the interface currently shows metric values, per-dimension coverage, and contextual messages about rewarded actions, but it does not display the full weighting table inside the dashboard. The complete rubric remains documented in project artifacts and source code.

Views 2–4 (Badge Progress Strip, Quality Attribute Summary, and Sprint Context Panel) support personal self-regulation: they expose progression by badge, per-dimension quality coverage, and sprint constraints without peer comparison [5, 18, 23]. This design preserves competence feedback while avoiding status pressure.

View 5: Sprint Performance Trend. Positioned in the lower portion of the overview, this bar chart of points earned across recent sprints (6–10 sprints) provides a longitudinal perspective. Novelty fade, the gradual decline in engagement as game elements become familiar, is the primary sustainability risk in gamification [11], yet few gamified testing tools expose trend data to practitioners themselves. Making the trend visible enables the tester to notice declining engagement and self-correct, supporting reflective monitoring that Flow Theory associates with sustained intrinsic motivation [2, 9]. Figure 2 shows the Performance Trend.

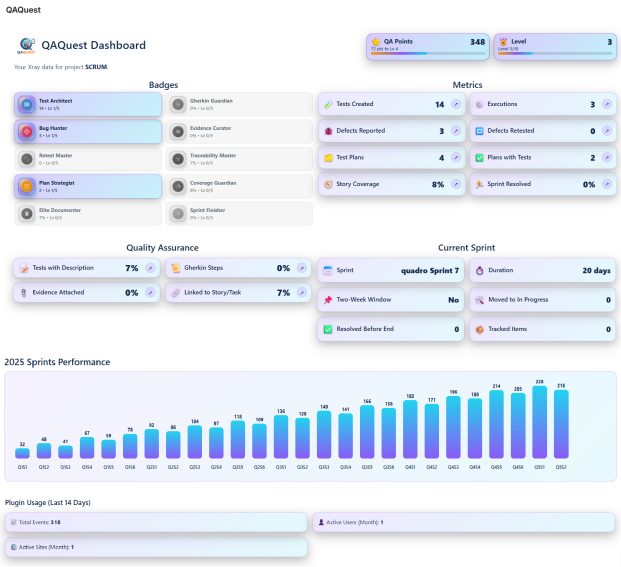

Figure 1: Operational Dashboard Overview: integrated layout with points/level, badges, metrics, quality, sprint context, and sprint trend.

Figure 2: Sprint Performance Trend: historical points across multiple sprints.

View 6: Plugin Usage Telemetry reports aggregated adoption metrics (total events, monthly active users, active projects) to administrators and QA leads, providing deployment evidence while preserving tester privacy [3, 8, 24].

4 Architecture

QAQuest follows a modular plugin architecture. In the reference implementation, it is built on Atlassian Forge, a serverless platform that runs logic within the Jira Cloud trust boundary. Issue creation and update events trigger webhook handlers; the backend computes scores, updates badge levels, enqueues reward notifications, and exposes data to the React frontend via Forge Resolver functions (`getMyGameReport`, `consumeRewardEvents`, `getUsageSummary`). All data is read-only from Jira via REST API v3 with JQL queries; no data is created or modified in Jira. Storage uses Forge Key-Value Store, requiring no external database. Processing occurs within the authenticated user's context so practitioners see only their own metrics.

Components: webhooks.js for event capture and scoring trigger; game-report.js for `scoreTest()`, `scoreExecution()`, portfolio bonus, level and badge computation; notifications.js for reward message formatting and routing; reward-events.js for event queue management; usage-analytics.js for telemetry tracking; React UI for dashboard polling and toast display.

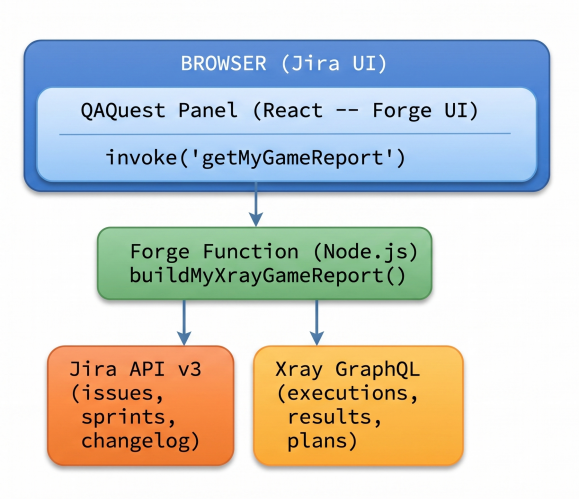

Figure 3: Execution flow: Jira event → handler → scoring → reward queue → UI polling.

4.1 Technical Implementation Details

QAQuest is implemented on Atlassian Forge using Node.js functions and React Custom UI. Backend integration with Jira and Xray is performed through `@forge/api` using REST API v3 endpoints and JQL search. The resolver layer in `src/index.js` exposes `getMyGameReport`, `consumeRewardEvents`, and `getUsageSummary`, which are polled by the frontend panel.

Event communication starts in `src/webhooks.js`. On issue create or update events, the handler loads full issue data and changelog, classifies the Xray entity type (Test, Test Execution, Test Plan), and calls scoring functions from `src/game-report.js`. Reward messages are sent through `src/notifications.js`, then queued in Forge KVS by `src/reward-events.js`. The frontend consumes these events through the resolver endpoint and renders toast feedback in context.

State and analytics are persisted in Forge KVS. Usage telemetry in `src/usage-analytics.js` tracks action type, source, daily volume, and monthly active users/sites. The badge rendering layer uses React and SVG assets defined in `static/badges/BadgeComponent.jsx` and `BadgeComponent.css`. Badge unlock levels are computed in backend logic (`BADGE_CATALOG` in `game-report.js`) and presented in frontend components through the same badge identifiers, ensuring consistency between scoring, notification, and visualization layers.

Scores are recomputed from current Jira/Xray artifact state whenever the report is queried. Therefore, improvements increase current points and percentage badges, while later removal of quality signals (e.g., deleting evidence or breaking traceability) can decrease current totals and corresponding badge levels for percentage-based dimensions.

4.2 Weighted Scoring Model

4.2.1 Test Case Scoring — `scoreTest`. The `scoreTest` function evaluates each Xray issue of type *Test* created by the authenticated user, assigning points per attribute as specified in Table 1.

Table 1: Attribute weights in `scoreTest`.

Attribute	Pts
Gherkin syntax in description	40
Description field populated	20
Traceability (link to Story/Task)	15
Evidence attached (attachment)	10
Descriptive title (summary)	10
Labels populated	5
Components populated	5
Priority defined	5
Maximum total	110

```
function hasGherkin(issue, xrayFieldId = null) {
  const descText = normalizeText(
    issue?.fields?.description
  );
  if (descText.includes('given')
      && descText.includes('when')
      && descText.includes('then')) {
    return true;
  }
  if (xrayFieldId) {
    const xrayText = normalizeText(
      issue?.fields?.[xrayFieldId]
    );
    return xrayText.includes('given')
        && xrayText.includes('when')
        && xrayText.includes('then');
  }
  return false;
}
```

Listing 1: Gherkin syntax detection in description or Xray Test Definition field.

Gherkin detection verifies the simultaneous presence of `given`, `when`, and `then` in the normalised text of the `description` field or Xray *Test Definition* custom field, after converting Atlassian Document Format (ADF) to plain text. The approach is deliberately permissive: it does not require strict indentation, step ordering, or isolated keywords, recognising that teams in early BDD adoption frequently produce informal scenarios. A consequence is possible false positives/negatives in edge cases; stricter parsing (e.g., ordered-step validation or grammar-based checks) is planned as a configurable mode.

4.2.2 Execution Scoring — `scoreExecution`. The `scoreExecution` function evaluates issues of type *Test Execution*, with emphasis on attributes indicating completeness of an auditable execution record (Table 2).

The higher weight for evidence in `scoreExecution` (15 vs. 10 in `scoreTest`) reflects a differentiated expectation: whereas a test case may exist before any execution, an execution without evidence is incomplete from an audit perspective [15].

4.2.3 Portfolio Quality Bonus. In addition to per-artifact scores, QAQuest computes a proportional quality bonus from attribute coverage rates across the entire portfolio. The multipliers apply differential weights: description (×1.2), Gherkin (×1.5), evidence

Table 2: Attribute weights in `scoreExecution`.

Attribute	Pts
Description populated	15
Execution environment defined	15
Evidence attached (attachment)	15
Descriptive title	10
Execution comment present	10
Assignee set	5
Recently updated	5
Maximum total	75

```
const qualityBonus =
  Math.round(toPercent(testsWithDescription, tests.length) *
       1.2) +
  Math.round(toPercent(testsWithGherkin, tests.length) *
      1.5) +
  Math.round(toPercent(testsWithEvidence, tests.length) *
      1.0) +
  Math.round(toPercent(testsWithTraceability, tests.length)
      * 1.3) +
  Math.round(toPercent(storiesWithCoverage, stories.length)
      * 1.4) +
  Math.round(toPercent(sprintResolvedBeforeEnd, sprintItems.
      length) * 1.6);
```

Listing 2: Portfolio quality bonus computation.

(×1.0), traceability (×1.3), story coverage (×1.4), and within-sprint resolution (×1.6). Gherkin and sprint resolution carry the highest multipliers, reflecting respectively their structural quality impact [4] and delivery predictability impact [15].

4.3 Level Progression Formula

A tester's level is computed by:

$$\text{level} = \min\Big(10,\ \lfloor \text{points}/140 \rfloor + 1\Big) \tag{1}$$

The threshold of 140 points per level was calibrated against two anchor cases from Table 1: a test case with strong structural properties (Gherkin + description + traceability + title = 85 pts) and one with minimal documentation (title + description only = 30 pts). In an illustrative scenario, a tester creating one of each during a sprint accumulates ≈115 pts in direct scoring; portfolio bonuses may push the total beyond 140 pts, yielding level 2 by sprint's end. This calibration follows the principle [2, 7] that reward cycles should be neither excessively short (creating dependency on immediate feedback) nor excessively long (causing demotivation through absence of visible progress). Level 10 requires 1,260 pts total; expected time-to-level is context-dependent (portfolio size, sprint cadence, and project activity) and should be validated empirically.

4.4 Badge Catalogue

QAQuest defines ten badges with five levels each, organized into two categories: badges based on count and badges based on coverage. Table 3 explains each badge individually in agile QA context and uses the corresponding SVG asset from the plugin package. Badge names are anchored in professional identity and competence (Architect, Guardian, Master) rather than in competitive rankings (champion, leader), because Self-Determination Theory predicts that competence-domain anchoring produces more durable intrinsic motivation than social-position rewards [5, 13, 18, 19].

The Gherkin Guardian first level threshold (40%) is set above typical baseline adoption rates, ensuring the badge becomes relevant only once a team has cleared the initial BDD adoption barrier [17]. The Evidence Curator starts at 20%, a conservative estimate of evidence attachment rates in agile teams, allowing early attainment, a scaffolding mechanism aligned with Flow Theory's prescription for graduated challenge onset [2, 7]. The Evidence Curator ceiling is 95%, not 100%, acknowledging that exploratory and ad hoc tests may legitimately produce no formalizable evidence; penalizing contextually justified exceptions would contradict the design principle of targets that support autonomy.

5 Limitations and Evaluation Plan

This camera-ready version clarifies that the present contribution is a deployable tool and design rationale, not a completed effectiveness study. We do not yet claim causal impact on motivation, artifact quality, or long-term engagement. A preliminary one-month pilot (N=5) provided early adoption and fairness signals, but with a small sample and one organizational context it remains exploratory. Main limitations are the permissive Gherkin heuristic, dependence on Jira/Xray (including sprint metadata), heuristic weight assignment (Tables 1 and 2), contributor attribution constraints in shared Test Plans, and partial in-dashboard rubric visibility. Next steps are a longitudinal mixed-method evaluation (telemetry, portfolio-quality indicators, and perception measures) plus sensitivity analyses for score weights and stricter parsing variants.

6 Conclusion

QAQuest addresses a persistent gap in test gamification research and practice: existing tools reward volume, introduce competitive risk, and fail to sustain engagement beyond initial novelty. Our system targets structural quality dimensions documented in testing literature as indicators of portfolio value, provides relative targets calibrated to each practitioner's own history, excludes competitive ranking to preserve team climate, and delivers immediate contextual feedback within the testing workflow. Every design choice (from the 40-point weight for Gherkin-related signals to the 140-point level threshold to the personal dashboard) is grounded in Self-Determination Theory and Flow Theory and formulated as an empirically testable proposition. Exploratory pilot signals in an anonymized industrial QA context support feasibility and perceived fairness of the non-competitive approach, while reinforcing the need to improve contributor attribution in shared Test Plans. The plugin is ready for deployment and provides a foundation for studying whether incentives focused on structural quality can sustain engagement without the behavioral distortions commonly seen in approaches focused only on volume. Immediate future work includes controlled and longitudinal evaluation, weight calibration, and optional strict Gherkin validation.

Table 3: QAQuest badge catalogue with individual rationale and visual asset.

Visual	Badge	Thresholds	Agile QA motivation
	Test Architect	[5, 15, 30, 50, 80]	Reinforces stable growth of reusable test assets across sprints instead of last-minute test creation.
	Defect Hunter	[2, 5, 10, 20, 40]	Rewards high-signal defect reporting that accelerates triage and shortens feedback loops in agile iterations.
	Retest Master	[1, 3, 6, 10, 20]	Encourages systematic revalidation after fixes, reducing reopen rate and increasing release confidence.
	Plan Strategist	[1, 3, 5, 8, 12]	Values explicit test planning links, improving predictability and planning quality in sprint ceremonies.
	Elite Documenter	[50, 70, 85, 95, 100]	Incentivizes complete test descriptions, improving maintainability and onboarding speed in rotating agile teams.
	Gherkin Guardian	[40, 60, 75, 90, 100]	Promotes behavior-focused scenario writing, strengthening shared understanding between QA, product, and development.
	Evidence Curator	[20, 40, 60, 80, 95]	Rewards attaching execution evidence, which improves auditability and defect reproducibility in distributed work.
	Traceability Lord	[40, 60, 80, 90, 100]	Encourages requirement traceability so scope changes can be assessed quickly without hidden quality gaps.
	Coverage Guardian	[30, 50, 70, 85, 95]	Reinforces story-level coverage visibility, supporting risk-based decisions during sprint review and release planning.
	Sprint Finisher	[30, 50, 70, 85, 95]	Encourages completion inside sprint boundaries, improving cadence stability and reducing carry-over volatility.

Artifact Availability

QAQuest is publicly available in the Atlassian Marketplace: QAQuest. The artifact package, including the source code and supporting materials for this submission, is available in a Figshare repository: QAQuest Source Code.

Acknowledgments

This research was partially supported by the Brazilian National Council for Scientific and Technological Development (CNPq) under grants 315765/2023-2 and 408651/2023-7.

References

[1] Paul P. Baard, Edward L. Deci, and Richard M. Ryan. 2004. Intrinsic Need Satisfaction: A Motivational Basis of Performance and Well-Being in Two Work Settings. *Journal of Applied Social Psychology* 34, 10 (2004), 2045–2068. doi:10.1111/j.1559-1816.2004.tb02690.x

[2] Mihaly Csikszentmihalyi. 1990. *Flow: The Psychology of Optimal Experience*. Harper and Row, New York.

[3] Fred D. Davis. 1989. Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly* 13, 3 (1989), 319–340. doi:10.2307/249008

[4] Gabriela Martins de Jesus, Fabiano Cutigi Ferrari, Daniel de Paula Porto, and Sandra Camargo Pinto Ferraz Fabbri. 2018. Gamification in Software Testing: A Characterization Study. In *Proceedings of the III Brazilian Symposium on Systematic and Automated Software Testing*. doi:10.1145/3266003.3266007

[5] Edward L. Deci and Richard M. Ryan. 2000. The "What" and "Why" of Goal Pursuits: Human Needs and the Self-Determination of Behavior. *Psychological Inquiry* 11, 4 (2000), 227–268. doi:10.1207/S15327965PLI1104_01

[6] Sebastian Deterding, Dan Dixon, Rilla Khaled, and Lennart Nacke. 2011. From Game Design Elements to Gamefulness: Defining Gamification. In *Proceedings of the 15th International Academic MindTrek Conference*. 9–15. doi:10.1145/2181037.2181040

[7] Stefan Engeser and Falko Rheinberg. 2008. Flow, Performance and Moderators of Challenge-Skill Balance. *Motivation and Emotion* 32, 3 (2008), 158–172. doi:10.1007/s11031-008-9102-4

[8] Tommaso Fulcini, Riccardo Coppola, Luca Ardito, and Marco Torchiano. 2023. A Review on Tools, Mechanics, Benefits, and Challenges of Gamified Software Testing. *Comput. Surveys* 55, 14s (2023), Article 310. doi:10.1145/3582273

[9] Clive J. Fullagar and E. Kevin Kelloway. 2009. 'Flow' at Work: An Experience Sampling Approach. *Journal of Occupational and Organizational Psychology* 82, 3 (2009), 595–615. doi:10.1348/096317908X357903

[10] Johannes Keller and Herbert Bless. 2008. Flow and Regulatory Compatibility: An Experimental Approach to the Flow Model of Intrinsic Motivation. *Personality and Social Psychology Bulletin* 34, 2 (2008), 196–209. doi:10.1177/0146167207310026

[11] Jonna Koivisto and Juho Hamari. 2019. The Rise of Motivational Information Systems: A Review of Gamification Research. *International Journal of Information Management* 45 (2019), 191–210. doi:10.1016/j.ijinfomgt.2018.10.013

[12] Mika V. Mäntylä and Kari Smolander. 2016. Gamification of Software Testing — An MLR. In *Proceedings of the 17th International Conference on Product-Focused Software Process Improvement (PROFES '16) (Lecture Notes in Computer Science, Vol. 10027)*. Springer, 611–614. doi:10.1007/978-3-319-49094-6_46

[13] Elisa D. Mekler, Florian Brühlmann, Alexandre N. Tuch, and Klaus Opwis. 2017. Towards Understanding the Effects of Individual Gamification Elements on Intrinsic Motivation and Performance. *Computers in Human Behavior* 71 (2017), 525–534. doi:10.1016/j.chb.2015.08.048

[14] Navid Memar, Aneesh Krishna, David A. McMeekin, and Tele Tan. 2020. Investigating Information System Testing Gamification with Time Restrictions on Testers' Performance. *Australasian Journal of Information Systems* 24 (2020). doi:10.3127/ajis.v24i0.2179

[15] Mariusz Nowostawski, Simon McCallum, and Deepti Mishra. 2018. Gamifying Research in Software Engineering. *Computer Applications in Engineering Education* 26, 5 (2018), 1641–1652. doi:10.1002/cae.21994

[16] Oscar Pedreira, Félix García, Nieves Brisaboa, and Mario Piattini. 2015. Gamification in Software Engineering — A Systematic Mapping. *Information and Software Technology* 57 (2015), 157–168. doi:10.1016/j.infsof.2014.08.007

[17] Wei Ren and Stephen Barrett. 2020. Toward Gamification to Software Engineering and Contribution of Software Engineer. In *ICMSS 2020: Proceedings of the 2020 4th International Conference on Management Engineering, Software Engineering and Service Sciences*. 1–5. doi:10.1145/3380625.3380628

[18] Richard M. Ryan and Edward L. Deci. 2017. *Self-Determination Theory: Basic Psychological Needs in Motivation, Development, and Wellness*. The Guilford Press, New York.

[19] Michael Sailer, Jan Ulrich Hense, Sarah K. Mayr, and Heinz Mandl. 2017. How Gamification Motivates: An Experimental Study of the Effects of Specific Game Design Elements on Psychological Need Satisfaction. *Computers in Human Behavior* 69 (2017), 371–380. doi:10.1016/j.chb.2016.12.033

[20] Katie Seaborn and Deborah I. Fels. 2015. Gamification in Theory and Action: A Survey. *International Journal of Human-Computer Studies* 74 (2015), 14–31. doi:10.1016/j.ijhcs.2014.09.006

[21] Helen Sharp, Nathan Baddoo, Sarah Beecham, Tracy Hall, and Hugh Robinson. 2009. Models of Motivation in Software Engineering. *Information and Software Technology* 51, 1 (2009), 219–233. doi:10.1016/j.infsof.2008.05.009

[22] Philipp Straubinger and Gordon Fraser. 2024. Improving Testing Behavior by Gamifying IntelliJ. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (ICSE '24)*. ACM, 1–13. doi:10.1145/3597503.3623339

[23] Maarten Vansteenkiste, Richard M. Ryan, and Bart Soenens. 2020. Basic Psychological Need Theory: Advancements, Critical Themes, and Future Directions. *Motivation and Emotion* 44, 1 (2020), 1–31. doi:10.1007/s11031-019-09818-1

[24] Viswanath Venkatesh, Michael G. Morris, Gordon B. Davis, and Fred D. Davis. 2003. User Acceptance of Information Technology: Toward a Unified View. *MIS Quarterly* 27, 3 (2003), 425–478. doi:10.2307/30036540

[25] Mark Zarb. 2011. Using Gamification to Motivate Software Engineering Students. In *Workshop on Emerging Trends in Software Engineering Education (WETSoE '11)*. 14–17.