IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 3, March 2025

DOI: 10.17148/IJARCCE.2025.14327

Total Team Quality: A Kickstart for Scrum Teams

Jay Yogesh Sampat

Technical Consultant, Tech Mahindra

Abstract: The concept of Total Team Quality represents a transformative approach to quality management within Scrum teams, moving away from traditional siloed testing and quality assurance methods toward a more integrated, collaborative model where quality becomes everyone's responsibility. This comprehensive analysis demonstrates that when properly implemented, Total Team Quality practices lead to significant improvements in both product quality and team velocity, with case studies showing up to 40% reduction in User Acceptance Testing (UAT) defects and duration alongside 20% increases in velocity. The approach centres on shift-left testing principles, shortened feedback cycles, reduced work-in-progress limits, and the cultivation of high-performing teams where quality is embedded into every phase of the development process rather than being an afterthought or separate activity.

Keywords: User Acceptance Testing (UAT), Scrum teams, SonarQube, work in progress (WIP), Test-Driven Development (TDD), Behaviour-Driven Development (BDD)

I. INTRODUCING TOTAL TEAM QUALITY

Total Team Quality is built upon a foundation of "shift-left" principles that aim to prevent defects from occurring rather than catching them after they have been created. This approach fundamentally changes how Scrum teams operate, moving quality activities earlier in the development lifecycle to eliminate the need for extensive defect remediation work later. The philosophy is supported by four core principles that guide implementation: the whole team takes responsibility for quality and testing; every opportunity is taken to shorten feedback cycles; teams build shared understanding of features and user stories; and development is guided by examples and tests rather than abstract requirements.

Case studies provide compelling evidence for the effectiveness of this approach. One team composed of six developers without dedicated testers built a new Java microservice with 175,000 lines of code across 75 features and 14 releases, yet encountered only four pre-production defects, four production defects, and zero regression defects. Their testing infrastructure included over 80% of unit test code coverage and 468 automated functional test cases that executed in just 35 minutes, with tests running daily and failures addressed immediately at daily standups.

A second case involving three Scrum teams working over a three-month Program Increment showed equally impressive results. While building three new microservices, the teams achieved approximately 20% improvement in velocity, 40% reduction in UAT defects, and 40% reduction in UAT duration. Test coverage reached 100% in one service and over 80% in others, with all services achieving A-grade maintainability, reliability, and security metrics in SonarQube.

The Total Team Quality approach introduces distinct differences in how Scrum teams operate, including a focus on work-in-progress limits, shortened feedback cycles, quality-centric continuous learning culture, behaviour-driven development, modern testing approaches, and test-driven development. The crucial takeaway is that teams can achieve both increased speed and superior quality simultaneously, but this requires commitment to working differently than traditional software development approaches.

II. CREATING HIGH PERFORMING TEAMS

Implementing Total Team Quality requires high-performing teams that function as cohesive units rather than collections of individuals. As Driskell and Salas observed in their research, "Teams working as a cohesive unit perform far better than collections of individuals for knowledge-rich, problem-solving tasks that require high amounts of information". This insight is particularly relevant for complex software development projects where multiple perspectives and skill sets must be integrated.

High-performing teams do not emerge spontaneously by implementing Scrum ceremonies or following Agile practices mechanically. They develop through distinct formation stages as described by Tuckman's model: forming, storming, norming, and performing.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 3, March 2025

DOI: 10.17148/IJARCCE.2025.14327

Organizations must recognize that reaching the performing stage typically takes months of deliberate effort, coaching, and relationship building among team members. During this development journey, teams cultivate essential attributes including psychological safety that encourages risk-taking without fear of punishment, diverse knowledge and skills that enable independent decision-making, trust that allows for healthy conflict, shared vision with clear goals, mutual accountability, reliable commitment fulfillment, understanding of organizational impact, and enjoyment of their work together.

The transition from traditional "relay race" approaches to product development toward a more holistic "rugby" approach is central to team ownership and quality responsibility. As Hirotaka Takeuchi and Ikujiro Nonaka noted in their influential Harvard Business Review article, "The traditional sequential or 'relay race' approach to product development... may conflict with the goals of maximum speed and flexibility. Instead, a holistic or 'rugby' approach—where a team tries to go the distance as a unit, passing the ball back and forth—may better serve today's competitive requirements". In this rugby model, quality becomes everyone's concern, impacting activities from requirements and planning through architecture, implementation, and deployment.

Cultivating an Agile mindset requires restructuring how team members view their roles. Instead of testers focusing solely on finding bugs, they should consider how to prevent defects from occurring, make testing more efficient, and build testing skills throughout the team. Similarly, programmers must move beyond merely completing programming tasks in isolation and focus on helping teammates complete user stories, making code testable, and writing appropriate tests themselves. This shift embodies true team-based development where members are self-organizing, decide collaboratively how to approach work, maintain high-bandwidth communication, and participate in all development aspects according to their competencies.

The team charter serves as a critical tool for establishing operational norms. Effective teams create charters collaboratively that clarify how members will hold each other accountable, share information, reach consensus, request help, and build psychological safety. Working agreements within these charters define how teams want to work together, creating transparency and accountability that new members can quickly understand and adopt. Similarly, a shared Definition of Done ensures each increment meets quality standards by establishing clear criteria that must be met before work is considered complete, such as passing acceptance tests, maintaining code standards, and addressing non-functional requirements.

III. WORK IN PROGRESS LIMITS

Controlling work in progress (WIP) represents one of the most powerful yet underutilized quality techniques in Scrum environments. The simple maxim "Stop starting and start finishing" encapsulates the core principle that teams should focus on completing existing work rather than beginning new tasks. When teams impose smaller WIP limits, they naturally experience faster flow of value and receive feedback more quickly, allowing issues to be identified and addressed earlier in the development cycle. This approach directly contradicts traditional management practices that emphasize keeping individuals fully utilized, instead prioritizing team completion over individual utilization metrics.

The implementation of WIP limits encourages "swarming" behaviour where multiple team members collaborate intensively on a small number of user stories to rapidly move them to completion. This collaborative approach represents a significant departure from the common antipattern of assigning stories to individuals, which creates silos and bottlenecks. Setting WIP limits lower than the number of developers creates beneficial "slack" that enables team members to help each other, share knowledge, and collectively improve quality. The practice of preventing new stories from starting until others are completed forces the team to address impediments immediately rather than working around them by starting new work.

Team-first thinking fundamentally reshapes how Scrum events function. The Daily Stand-Up transforms from a status meeting with the standard three questions ("What did you do yesterday? What will you do today? What is blocking you?") into a planning opportunity focusing on how the team will move each user story toward completion. This shift in perspective changes the nature of the meeting from reporting individual progress to collaborative problem-solving and resource allocation. Similarly, Sprint Goals provide a commitment mechanism that offers more flexibility than committing to specific user stories, allowing teams to adapt their approach while maintaining focus on delivering business value.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 3, March 2025
DOI: 10.17148/IJARCCE.2025.14327

Effective leadership plays a critical role in establishing WIP limits by coaching teams to build trust, communication, autonomy, and a continuous learning culture. Leaders should resist the temptation to focus on individual productivity metrics and instead measure teams based on their ability to deliver sprint goals and stakeholder satisfaction. Well-conducted Sprint Reviews and Retrospectives become essential learning opportunities rather than mere ceremonies, helping teams continuously refine their approach to work management. Individual team members must adopt the team-first mindset, actively helping the team succeed as a whole and contributing to areas outside their specialized roles or comfort zones.

IV. SHORTER FEEDBACK CYCLES

The traditional feature-first delivery approach often delays critical feedback until late in the development process. In this model, architects and systems engineers decompose business requirements (features) into technical components that Scrum teams build individually before integration, testing, and stakeholder demonstrations occur. This sequence creates a significant risk illustrated by the "cost of change curve" – the later defects or changes are identified, the exponentially more expensive they become to address. By the time stakeholders see working software and provide feedback, the cost of implementing changes has increased dramatically.

Story-first delivery presents a compelling alternative that identifies defects at the earliest possible opportunity. This approach emphasizes continuous testing where all tests run continuously during development: developers run unit tests before committing code, fast tests execute on every code commit, slower tests run at least once daily, and all tests include accumulated regression tests to prevent quality regression. The philosophy that "every newly passing test instantly becomes a regression test" ensures quality is maintained throughout development rather than degrading over time. This continuous testing strategy fundamentally improves productivity by allowing teams to spend less time fixing defects and more time enhancing functionality, while automated testing replaces resource-intensive manual testing efforts.

Comparing traditional and best practices reveals stark differences in testing approaches. Traditional practices focus most thorough testing at the feature level, treat user stories as technical requirements, require integration testing for user story acceptance, and involve stakeholders only when reviewing completed features. These practices increase costs and lower software quality by delaying feedback. In contrast, best practices focus user stories on business processes and rules (with technical details as subtasks), encourage teams to design, code, unit test, and functional test stories within a single iteration, centre testing on user stories, test all requirements with stories, and use integrated testing specifically for integration rather than requirements verification. These practices decrease costs substantially while increasing quality through earlier feedback.

The business impact of shorter feedback cycles extends beyond technical considerations. Organizations implementing shorter feedback loops report faster time-to-market, higher customer satisfaction, and reduced rework costs. When stakeholders can interact with working software earlier and more frequently, their input becomes more relevant and implementable. Teams also experience improved morale as they spend less time fixing bugs and more time creating value – the positive reinforcement of seeing features work correctly sooner creates a virtuous cycle of quality improvement and satisfaction.

V. FOCUS ON QUALITY AND CONTINUOUS LEARNING CULTURE

"Without Quality, there is No Speed," observed W. Edwards Deming, capturing the essential relationship between quality and productivity that underpins the Total Team Quality approach. Rather than viewing quality activities as overhead that slows delivery, high-performing Scrum teams recognize that quality investments accelerate development by reducing rework, support costs, and technical debt. This perspective requires balancing the execution of the current sprint with adequate preparation for future sprints, ensuring that the team maintains its delivery cadence without compromising quality standards.

Successful teams incorporate quality activities into their definition of done and refuse to compromise on completing these activities even under delivery pressure. This might include practices such as peer code reviews, automated testing, documentation updates, and refactoring. By treating these activities as non-negotiable parts of the development process rather than optional extras, teams establish quality as a fundamental value rather than an afterthought. This commitment extends to addressing technical debt proactively instead of allowing it to accumulate, which would eventually cripple development velocity.

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 3, March 2025

DOI: 10.17148/IJARCCE.2025.14327

A continuous learning culture forms the foundation for sustained quality improvement. Teams allocate time specifically for learning how to perform their jobs more effectively, whether through formal training, knowledge-sharing sessions, or experimental time. This investment may appear to reduce immediate productivity but yields substantial long-term benefits through improved skills, better solutions, and reduced defects. The learning culture extends to how teams manage quality issues – successful teams adopt a zero-tolerance approach to defects, failed tests, or broken builds, addressing these immediately rather than allowing quality problems to accumulate.

The role of specialists evolves significantly in quality-focused teams. Rather than creating bottlenecks through dedicated specialists who become overloaded with work, teams share knowledge and cross-train to distribute quality responsibilities. Testing specialists may initially lead quality practices but increasingly focus on building testing capabilities throughout the team rather than personally executing all tests. Similarly, architecture and security specialists collaborate with developers to embed quality thinking into daily work rather than performing separate specialized reviews. This collaborative approach removes bottlenecks while raising overall team capabilities.

VI. TEST FIRST AND BEHAVIOUR-DRIVEN DEVELOPMENT

The Test First paradigm encompasses several software development practices where tests are written before production code, representing a significant departure from traditional approaches where testing follows implementation. This category includes Test-Driven Development (TDD), where developers create and run unit tests for code they are about to write, demonstrate test failure, and then add just enough code to pass the test. It also includes Behaviour-Driven Development (BDD), where acceptance tests are defined for each agile requirement before development begins, providing clear guidance for implementation. Both practices share the fundamental principle that test creation should precede coding rather than following it.

Effective implementation of these practices requires genuine collaboration between the Scrum team and Product Owner. The "Three Amigos" approach – bringing together business representatives, developers, and testers – helps ensure that stories are valuable, feasible, small enough to fit within an iteration, and contain minimal gaps or ambiguities. This collaboration is particularly important when decomposing requirements into manageable pieces. Techniques that maintain business value down to the user story level, such as Story Mapping and Example Mapping, prove more effective than purely technical decomposition approaches that can disconnect stories from clear business outcomes.

Acceptance Criteria play a crucial role in Test First approaches, serving as both requirements clarification and the test plan for user stories. Effective criteria should be understandable and reviewable by non-technical stakeholders, favouring formats that accomplish this goal. The Gherkin format (using Given-When-Then structure) is particularly valuable as it provides a clear, consistent structure that business stakeholders can understand while simultaneously guiding test automation. When properly implemented, test automation frameworks can directly execute against the examples in the Acceptance Criteria, creating a seamless connection between requirements and verification.

The quality of test automation frameworks deserves the same attention as production applications. As the search results note, "Test automation frameworks are applications, and need as much care as the production application". Teams frequently underestimate the engineering effort required to create maintainable, reliable test automation, leading to brittle tests that generate false positives or become maintenance burdens. Successful teams apply the same engineering principles to test code as production code, including refactoring, code reviews, and architectural consideration. This investment pays dividends through reduced maintenance costs and higher confidence in test results.

VII. TEST STRATEGIES

Effective testing requires thoughtful strategic approaches rather than simply executing more tests. One critical strategy involves testing the UI separately from system functionality, writing separate BDD scenarios for UI elements and underlying business functionality. This separation allows for more focused testing of each component and reduces the brittleness often associated with UI-driven tests. Importantly, teams should avoid testing business rules via the UI whenever possible, as this creates slow, fragile tests that become maintenance burdens. Instead, business rules should be tested directly through APIs or service interfaces where possible, with UI testing focused specifically on interaction patterns and visual presentation.

Test doubles provide essential tools for effective testing strategies. Defined as "simplified versions of depended-upon components that are slow, expensive, not available, or random," test doubles enable isolated testing of components without requiring the entire system to be assembled.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 3, March 2025
DOI: 10.17148/IJARCCE.2025.14327

These include stubs (providing canned answers to calls), mocks (pre-programmed with expectations), fakes (working implementations with shortcuts), and spies (recording calls made). By strategically employing test doubles, teams can test components in isolation, control test environments, and dramatically increase test execution speed. This approach supports shift-left testing by allowing developers to test extensively before integration.

Continuous Integration and Continuous Testing form the backbone of modern test strategies. The practice of having "everyone push commits to main every day" encourages frequent integration that prevents large, complex merges. The workflow typically involves developers pulling from the main branch to merge changes locally, verifying code correctness including passing unit tests, and pushing to main only if builds pass. This discipline ensures that the main branch remains in a constantly deployable state, with quality built-in rather than assessed after the fact. When combined with automated test execution at multiple levels (unit, component, API, UI), this approach provides rapid feedback on potential issues.

The implementation of effective test strategies requires appropriate tooling and infrastructure. Test environments should be easily reproducible and consistent, ideally created programmatically through infrastructure-as-code practices. Data management presents challenges, with successful teams developing strategies for test data creation, isolation, and cleanup that allow tests to run independently and concurrently. Test result reporting deserves special attention, with effective strategies making test failures immediately visible and providing clear diagnostic information to accelerate resolution. These infrastructure investments pay dividends through faster feedback cycles and improved quality.

VIII. METRICS AND MEASUREMENT FOR QUALITY-FOCUSED TEAMS

Measuring quality effectively requires moving beyond traditional metrics that focus solely on defect counts or test coverage percentages. High-performing Scrum teams implement comprehensive measurement systems that capture both leading indicators (predictive of future quality) and lagging indicators (reflecting past performance). Leading indicators might include automated test coverage, code complexity trends, technical debt measurements, and peer review participation rates. Lagging indicators typically encompass production incidents, customer-reported defects, and system reliability statistics. By tracking both types of metrics, teams gain insights into both their current quality status and likely future outcomes.

Test automation metrics deserve particular attention within quality measurement systems. Beyond simple coverage percentages, teams should monitor metrics such as test reliability (the percentage of tests that consistently pass or fail without flakiness), test execution time (which impacts feedback cycle speed), and the distribution of tests across different testing pyramids levels. The automation pyramid concept suggests teams should have more unit tests than integration tests, and more integration tests than end-to-end tests, creating a pyramid shape. Monitoring this distribution helps teams identify imbalances that might indicate testing inefficiencies or gaps.

Lead time and cycle time metrics provide invaluable insights into process efficiency and quality. Lead time measures the period from when work is requested until it is delivered, while cycle time measures from when work begins until completion. When these metrics lengthen, they often indicate quality issues or process inefficiencies that require attention. Teams can use cumulative flow diagrams to visualize work progression and identify bottlenecks, while control charts help distinguish between common cause variation (normal process fluctuation) and special cause variation (significant anomalies requiring investigation). These visualization techniques make quality trends immediately apparent.

Metrics should drive continuous improvement rather than simply reporting status. High-performing teams establish regular metric review practices where they analyse trends, identify improvement opportunities, and implement changes. When metrics indicate quality issues, teams conduct root cause analysis to address underlying problems rather than symptoms. Importantly, metrics should never be used punitively against individuals, as this creates perverse incentives to game the system rather than improve actual quality. Instead, metrics should facilitate learning, experimentation, and genuine process improvement that benefits the entire team and organization.

IX. IMPLEMENTING TOTAL TEAM QUALITY: TRANSFORMATION ROADMAP

Transitioning to a Total Team Quality approach requires deliberate change management rather than simply announcing new practices. Organizations should begin with an honest assessment of current quality practices, team structures, and organizational barriers to quality. This assessment provides a baseline from which to measure improvement and helps identify the highest-priority change areas. Successful transformations typically start with a pilot team that can demonstrate the approach's effectiveness before broader implementation. This pilot creates internal case studies and champions who can support wider adoption.

 This work is licensed under a Creative Commons Attribution 4.0 International License

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 3, March 2025
DOI: 10.17148/IJARCCE.2025.14327

Leadership alignment represents a critical success factor for quality transformations. Executives and managers must understand that quality is not merely a technical concern but a strategic business investment. Leaders need to demonstrate this commitment through actions such as allocating time for quality activities, recognizing and rewarding quality-focused behaviours, and resisting pressure to compromise quality standards for short-term delivery goals. When leadership consistently prioritizes quality through both words and actions, teams feel empowered to make necessary changes to their working practices.

Skill development forms an essential component of the transformation roadmap. Teams typically need training and coaching in practices such as Test-Driven Development, Behaviour-Driven Development, automated testing, and collaborative specification techniques. This training should combine formal instruction with hands-on practice and ongoing coaching. Organizations should recognize that productivity may temporarily decrease as teams learn new skills, but this investment yields substantial returns through improved quality and reduced rework. The development of internal champions who can support their peers accelerates skill adoption across the organization.

Organizational policies and structures often require adjustment to support Total Team Quality implementation. Traditional separation between development and quality assurance functions may need reconsideration, potentially reorganizing into cross-functional product teams with embedded quality expertise. Performance evaluation systems should shift from individual productivity metrics toward team outcomes and quality contributions. Procurement and vendor management practices may need updating to ensure external partners align with internal quality practices. These structural changes create an environment where quality-focused behaviours become natural rather than exceptional.

Continuous improvement mechanisms ensure the transformation sustains over time rather than regressing to previous practices. Regular retrospectives at team, program, and organizational levels help identify improvement opportunities and track transformation progress. Communities of practice around quality topics facilitate knowledge sharing across teams and prevent silos. Ongoing measurement against established quality metrics provides visibility into improvement trends. Celebration of quality successes reinforces the importance of these practices and motivates continued commitment. Through these mechanisms, organizations evolve from implementing discrete quality practices toward developing a genuine quality culture that sustains excellence regardless of delivery pressures or organizational changes.

X. CONCLUSION

Total Team Quality represents a transformative approach to software development that fundamentally reconceptualizes how quality is achieved within Scrum teams. Rather than treating quality as a separate activity performed by specialized personnel after development, this approach embeds quality thinking and practices throughout the development lifecycle and across the entire team. The evidence from case studies demonstrates that this approach delivers substantial benefits including faster delivery, higher quality, reduced defects, and improved team satisfaction.

The successful implementation of Total Team Quality depends on several interconnected elements. High-performing teams with clear charters and working agreements provide the foundation for quality collaboration. Work-in-progress limits force focus on completion rather than starting new work. Shorter feedback cycles identify issues when they're least expensive to fix. A continuous learning culture encourages ongoing improvement. Test-first approaches like TDD and BDD provide guidance for implementation. Comprehensive test strategies ensure appropriate coverage across the system. Effective metrics drive improvement rather than merely reporting status. A deliberate transformation roadmap manages the organizational change required.

Organizations seeking to implement Total Team Quality should recognize that this represents a journey rather than a destination. The transformation requires patience, persistence, and leadership commitment to overcome inevitable challenges and resistance. However, the potential rewards – dramatically improved quality, accelerated delivery, reduced costs, and increased customer satisfaction – make this investment worthwhile. As development environments become increasingly complex and customer expectations for quality continue to rise, the Total Team Quality approach offers a proven path to sustainable excellence that benefits both organizations and the customers they serve.

REFERENCES

[1]. https://www.wrike.com/agile-guide/faq/how-to-measure-agile-team-performance/
[2]. https://www.simpliaxis.com/resources/who-is-responsible-for-quality-in-a-scrum-team
[3]. https://www.dynatrace.com/news/blog/what-is-shift-left-and-what-is-shift-right/
[4]. https://agiletestingdays.com/blog/continuous-testing-in-agile-and-continuous-delivery-environments/

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 3, March 2025

DOI: 10.17148/IJARCCE.2025.14327

[5]. https://resources.scrumalliance.org/Article/behavior-driven-development
[6]. https://agilepainrelief.com/blog/scrum-team-size.html
[7]. https://www.testrail.com/blog/whole-team-approach/
[8]. https://www.scrum.org/resources/blog/tips-improving-effectiveness-scrum-teams
[9]. https://www.linkedin.com/pulse/shift-left-testing-why-its-essential-agile-apx0c
[10]. https://agilevelocity.com/blog/5-good-habits-high-quality-scrum-teams/
[11]. https://www.atlassian.com/agile/scrum/scrum-metrics
[12]. https://www.linkedin.com/pulse/how-did-i-kickstart-new-scrum-team-shalini-thirunilathil
[13]. https://www.scrum.org/forum/scrum-forum/28335/kick-start-after-splitting-existing-team-two
[14]. https://www.scrum.org/forum/scrum-forum/73105/how-define-success-scrum-team
[15]. https://www.reddit.com/r/QualityAssurance/comments/15q0ukm/can_a_full_qa_team_be_a_full_operating_scrum/
[16]. https://www.splunk.com/en_us/blog/learn/continuous-testing.html
[17]. https://www.agilealliance.org/glossary/bdd/
[18]. https://www.indeed.com/career-advice/career-development/agile-metrics
[19]. https://www.bmc.com/blogs/what-is-shift-left-shift-left-testing-explained/
[20]. https://www.globalapptesting.com/the-ultimate-guide-to-agile-testing
[21]. https://www.scrum.org/forum/scrum-forum/27202/bdd-scrum
[22]. https://www.aha.io/roadmapping/guide/agile/agile-metrics
[23]. https://www.reddit.com/r/agile/comments/hgqmgn/seeking_inspiration_for_new_scrum_team_kickoff/
[24]. https://www.softwaretestingmagazine.com/knowledge/unpacking-shift-left-testing-benefits-key-to-reducing-osts-and-boosting-collaboration/
[25]. https://monday.com/blog/rnd/agile-testing/
[26]. https://en.wikipedia.org/wiki/Behavior-driven_development
[27]. https://www.mountaingoatsoftware.com/blog/advice-on-conducting-agile-project-kickoff-meetings