

Agile and InnerSource: A Match made in Heaven and in Hell!

Clare Dillon
Lero, University of Galway
clare.dillon@live.ie

Brian Fitzgerald
Lero, University of Limerick
brian.fitzgerald@ul.ie

J. Eduardo Ferreira Ribeiro
FEUP, University of Porto
jose.eduardo.ribeiro@fe.up.pt

Daniel Izquierdo-Cortázar
Bitergia
dizquierdo@bitergia.com

Abstract

InnerSource, the use of open source methods inside organizations, is gaining momentum. However, there is limited research available on how InnerSource may be adopted in tandem with Agile. The main objective of this paper is to examine areas of congruence and friction between Agile and InnerSource, so that practitioners may chart an optimal path to adopting InnerSource. Our analysis draws on a framework from previous studies which compared agile and open source. We find that InnerSource complements and diverges from Agile in many of the same ways open source does. Based on InnerSource practitioner feedback, we also identify an additional seven areas of potential friction that were not examined in previous Agile open source comparison studies. Our findings will help Agile practitioners assess when and how to adopt InnerSource for maximum positive impact in their organizations. Furthermore, we have identified a number of areas for follow-on research, investigating the adoption of both methods in diverse organizational contexts.

Keywords: Agile, InnerSource, Inner Source, Open Source

1. Introduction

Agile has become the *de facto* way of developing software at present, with 95% of respondents to a large-scale survey reporting the use of Agile software development practices (Digital.ai, 2022). In parallel with the rise of Agile methods, the prevalence of Open Source Software (OSS) has also increased dramatically, to the extent that it represents a significant proportion of all software available today – indeed, it has been estimated that OSS is present in between 96% and

99.9% of all large codebases, including those that are commercial in origin (Hoffmann et al., 2024). It is perhaps unsurprising that quite a lot of research has focused on the relationship between Agile methods and OSS (Chengalur-Smith et al., 2022; Gandomani et al., 2013; Goldman and Gabriel, 2005; Goth, 2007; Russo et al., 2009; Warsta and Abrahamsson, 2003).

*InnerSource*¹, the application of open source processes within an organization, has also risen significantly in recent years. Many organizations, including major technology companies, such as Microsoft, Huawei, Tencent, and PayPal, and even non-IT companies, such as Bloomberg, Bosch, and Nike, have embarked on significant InnerSource projects. InnerSource Commons, an online community of InnerSource practitioners, has over 3,000 participants across more than 750 organizations. InnerSource has also been identified in industry as a top strategic trend in software engineering (Gartner, 2023). Given the increased interest in InnerSource in companies globally, it is important to consider how Agile methods and InnerSource practices can be combined. This has not been the subject of much research to date, a notable exception being Morgan et al. (2022). In this paper, we examine the relationship between Agile methods and InnerSource, highlighting areas of congruence as well as potential friction between the two approaches. The paper primarily serves as a position paper on the combination of Agile and InnerSource, drawing upon the authors' extensive practical experience in the software industry and observations from the InnerSource Commons community. While all four authors are active researchers, they also bring a wealth of hands-on industry expertise, collectively amounting

¹We have chosen to use "InnerSource" rather than "inner source" to align with terminology used by the InnerSource Commons.

to 78 years—27 of which are specifically in Agile methodologies and 14 in InnerSource practices.

The remainder of this paper is structured as follows. In section 2, we consider research to date which has focused on the relationship between Agile and open source. In particular we draw on a framework used by Chengalur-Smith et al. (2022) to investigate areas of congruence and areas of friction between Agile and open source. Using this framework, we extend the analysis to Agile and InnerSource in section 3. In section 4, we consider areas and factors which arise uniquely in relation to Agile and InnerSource. Finally, in Section 5, we draw conclusions from the research and identify the next steps.

2. Agile and Open Source

As already mentioned, quite a lot of research has considered the relationship between Agile methods and OSS. In an early study, Warsta and Abrahamsson (2003) draw on Boehm's categorisation of Agile versus plan-driven methods (Boehm, 2002) by extending this to consider open source. Other researchers have drawn on the 12 principles of the Agile Manifesto to discuss the compatibility between Agile and open source (Goldman and Gabriel, 2005; Russo et al., 2009). Chengalur-Smith et al. (2022) review this research and identify two clusters of factors, namely areas in which Agile and open source are in harmony, and areas in which there is friction or disharmony. In summary the areas in which Agile and open source are in harmony include the following:

- Frequent releases of software
- Self-organizing teams
- Evolving nature of software requirements
- Avoid heavy up-front planning
- Refactoring
- Sub-division of work into smaller pieces
- Attitude to Change

The areas in which friction was identified between Agile and open source are the following:

- Location and mode of communication
- Size of development team
- Simplicity
- Role of customer

- Continuous Integration

Space limitations preclude a more comprehensive discussion of these findings here. A thorough discussion is provided in Chengalur-Smith et al. (2022).

3. Agile and InnerSource

In this paper, we consider the compatibility of Agile and InnerSource. As already mentioned, very little research has been conducted on this topic. Morgan et al. (2022) report on five case studies in companies on their implementation of Agile and InnerSource. They report that in two companies, the implementation of InnerSource preceded the implementation of Agile, in two companies, the implementation of Agile and InnerSource happened in parallel, and in one company, the implementation of InnerSource took place after the adoption of Agile. However, Morgan et al. (2022) do not provide a framework to compare Agile and InnerSource. We believe this would be useful for companies who seek to implement InnerSource in an Agile context. Hence, that is the focus of this research.

Agile teams typically consist of a dedicated set of individuals working on a specific project or set of features, with clearly defined roles and responsibilities. InnerSource involves contributors from across the entire organization, often working on various projects and contributing voluntarily or as part of their official role. InnerSource does not prescribe a specific process or framework but borrows principles from OSS development, such as pull requests, issue tracking, and community reviews. Agile often uses defined frameworks and methods like Scrum or XP, with specific ceremonies, roles, and artifacts. InnerSource and Agile share many principles and many of their practices complement each other. However, some practices differ to varying degrees, and a detailed understanding of similarities and differences would help organizations leverage the strengths of both approaches to improve software development practices.

It is important to note that InnerSource practices can differ depending on the organizational context. Where we have noted areas of friction or divergence between Agile and InnerSource below, the level of divergence may depend on where InnerSource is being adopted (e.g. as a grassroots effort vs more formally in product teams), which InnerSource practices are adopted, and how they are implemented. We have elaborated on this in Section 4. This research can also therefore be used to minimize areas of friction by helping organizations reflect on how they choose to implement InnerSource.

We draw on the comparison of agile and open source done by Chengalur-Smith et al. (2022) to explore the

areas of congruence and friction between Agile and InnerSource in Table 1 and 2. In most cases, the comparison with InnerSource practices resembles that with OSS.

One exception to this is the area of Software Requirements, where the need to synthesize requirements from across teams can cause additional friction for Agile teams. We have therefore moved this factor to Table 2 which lists the areas of potential friction between Agile and InnerSource practices. Other areas of potential friction arise from the fact that InnerSource assumes an unlimited number of remote teams may be collaborating asynchronously. Those teams may have different customers with conflicting needs and the teams themselves may be working in inconsistent development environments, using a diverse set of tools for communication.

4. Further Comparison of Agile and InnerSource

In the process of analysing the factors examined in Chengalur-Smith et al. (2022) in the context of InnerSource, and discussing them with InnerSource and Agile practitioners, we have identified some additional areas of alignment and divergence in Agile and InnerSource practices. These are explored below.

4.1. Team Roles

Common roles in Agile teams are often present in InnerSource teams. For many practicing Agile, adopting InnerSource may just involve implementing complementary practices alongside their traditional responsibilities. However, we have seen few examples of an equivalent to the formal role of Agile Coach or Scrum Master in InnerSource projects. Some of the InnerSource Commons community have suggested that formalizing InnerSource coaching in this way would not be a welcome development. We can hypothesize that this point of view may be an echo of sentiments in OSS ecosystems which often view formal corporate constructs as stifling autonomy. This may be a point of friction as InnerSource is adopted by Agile teams. However, we are just beginning to see the introduction of more formal organizational roles such as InnerSource Program Officers that are responsible for supporting InnerSource teams in their efforts, often providing direct guidance and advice. These roles may have more in common with Agile Coaches in the future as they evolve.

Some additional roles that are critical to InnerSource practices may be unfamiliar to Agile teams, including Trusted Committer and Contributor. At a high

level, Trusted Committers represent the interests of both their InnerSource community and the products the community is building. In addition to technical responsibilities, Trusted Committers have community-oriented responsibilities and commitments to servicing contributors outside their team.

In InnerSource, the Host Team is responsible for the stewardship, development, and maintenance of a particular project or codebase within the organization. A Contributor — as the name implies — makes contributions to the Host Team. These contributions could be code or non-code artifacts, such as bug reports, feature requests, or documentation. Contributors have commitments to fit in with the Host team way of working that may be different to their own team.

Product Owner (PO) is a role that is referenced in both Agile and InnerSource literature. In the context of InnerSource, the role may involve additional responsibilities above and beyond that which is traditionally expected in Agile. These responsibilities are documented in the InnerSource Commons Learning Path on Product Owners. As with Agile processes, in InnerSource the PO is responsible for defining and prioritizing requirements and stories for the community to implement. In addition, the PO interacts often with the Trusted Committer, (e.g., in making sure that a requested or contributed feature actually belongs to the product). In smaller, grassroots InnerSource communities, the Trusted Committer usually also acts as a PO, and the role may resemble the benevolent dictator common in OSS projects (InnerSource Commons Foundation, 2024).

4.2. Documentation

Agile prioritizes “working software over comprehensive documentation”, meaning the primary focus is on delivering functional software that meets user needs. In Agile, effort is allocated to only the necessary documentation that provides value for the team and stakeholders, with a preference for direct communication and collaboration. Agile also emphasizes direct communication and collaboration over extensive documentation. Face-to-face interactions, stand-up meetings, and other forms of direct communication may be preferred to ensure clarity and shared understanding (Fowler, Highsmith, et al., 2001).

InnerSource places a stronger emphasis on comprehensive and open documentation to facilitate broad, cross-organizational collaboration and ensure that contributors from different teams can understand and work on the project effectively. Where InnerSource

Table 1. Congruence between Agile and InnerSource Practices

Factor	Agile	InnerSource
Frequent Releases of Software	Strive toward working software at end of every sprint (Cockburn, 2004; Stapleton, 1997; Sutherland and Schwaber, 2011)	Encourages frequent contributions from internal teams. "Release early. Release often." adopted by InnerSource teams (Stol et al., 2014)
Self-organizing Teams	Reflect the principle of empowered developer teams acting creatively to fulfill necessary roles and adapt to change (Dyba, 2000; Nerur et al., 2005; Stapleton, 1997)	Contributors self-select tasks based on expertise and interest. Reflects organization practices (Stol et al., 2014). Organic collaborations happen across organization
Avoids heavy up-front planning	Strive to eliminate "heaviness" of plan-driven methods (Erickson et al., 2005) and "travel light" (Ambler, 2002; Beck, 2000)	Contribution intent shared early. Time explicitly allocated to support external contributions (Spier et al., 2023). Teams can dictate and guide planning requirements of contributing teams (InnerSource Commons Foundation, 2024)
Refactoring	Continuous evolution of initial functionality and improvement through refactoring (Beck, 2000)	InnerSource may require refactoring to modularize code to attract contributors and users (Stol et al., 2014)
Sub-division of work into smaller pieces	Development of small increments with rapid cycles (Abrahamsson et al., 2003; Nerur et al., 2005)	Tasks are broken down into smaller, manageable units for continuous integration (Sadler et al., 2021)
Attitude to Change	Embrace change even late in development (Beck, 2004; Williams and Cockburn, 2003)	Welcomes changes and new ideas even during the later stages of development

is being adopted widely across an organization, there is often an explicit effort to standardize documentation for a consistent experience as a contributor across projects. In industries with high levels of regulation due to legal requirements and standards compliance, documentation is crucial (Ferreira Ribeiro et al., 2023, 2024; Silva Cardoso Rodrigues et al., 2022), and the documentation that enhances InnerSource can aid as evidence for certification.

4.3. Tooling and Infrastructure

One challenge in InnerSource is potential differences in contributing teams' development environments (Stol et al., 2014). Contributors may not have experience of the Host team's tooling or prefer different coding languages. In some cases, they may not have the necessary licenses to access the Host team's coding environment. This is not an issue with an individual Agile team. If teams do not have common environments, both knowledge of the tools, and ability to access the tools (e.g. having a commercial license to use them) may add additional friction.

It should also be noted that even if the development environment is consistent across host and guest teams, challenges can arise to provide consistent access to a

host team's communication tools, either due to access (e.g. access to same Slack instances) or even language diversity (e.g. when host team may communicate in native language).

Because a primary motivation for adopting InnerSource is often code reuse, discoverability and findability of InnerSource projects is often an issue. Such challenges may be addressed with InnerSource portals or project directories (Cooper and Stol, 2018; Izquierdo-Cortázar et al., 2022; Oram, 2015). Similar tooling is typically not required for Agile teams .

Even when code is made visible across the organization, there can be challenges around accessing the host team's communication channels, for example, if informal conversations happen in Slack channels closed for certain teams. In InnerSource like OSS, individuals often view others' code to learn (not necessarily reuse) and they may "lurk" on communication channels to understand project norms before starting new contributions. If those channels are not easily findable and accessible, it can hamper the transfer of knowledge (Stol et al., 2014).

Table 2. Friction between Agile and InnerSource Practices

Factor	Agile	InnerSource
Location and mode of communication	Preference for co-located development teams, facilitating face-to-face “osmotic communication” (Cockburn, 2004) in daily stand-up meetings (Beck, 2004; Schwaber and Beedle, 2001)	Remote teams may be globally distributed. Teams rely on asynchronous communication methods, creating potential delays (Spier et al., 2023). In InnerSource, informal face-to-face decision making discouraged, and “If it’s not written down, it never happened” is often quoted (Bonewald, 2017)
Size of development team	Prefer small empowered teams (Dybå and Dingsøy, 2008; Stapleton, 1997)	No limit on team size. Enables teams of teams from across organization. As with OSS, this can enable individual contributions
Simplicity	Ensure simplest design possible – maximize the amount of work not done – “good enough” solution (Beck, 2000)	Additional docs may be required to be “good enough”. InnerSource can be used to resolve concurrent implementations into common platform. “Seed product” may be required to be good enough for reuse (Stol et al., 2014)
Evolving Nature of Software Requirements	Requirements emergent and adapted based on customer interaction and feedback (Boehm, 2002; Highsmith, 2000)	Requirement gathering in project teams may follow the normal of that team or may be very different. Need for coordination to incorporate requirements from different teams / customers into prioritization process (Stol et al., 2014)
Role of Customer	Active involvement of customer sought with Scrum role of Product Owner to act as proxy (Schwaber and Beedle, 2001)	Internal stakeholders act as customers, sometimes leading to conflicts of interest. Sometimes the customer is the developer themselves as, like in OSS, they “scratch their own itch” (Koch and Schneider, 2002; Raymond, 2001)
Continuous Integration	Core practice in Extreme Programming (Beck, 2000)	Depends on conformity of practices across the org. Challenges can arise where tooling differs from team to team in terms of skill gaps and access

4.4. Organization and Policy

Often, barriers to InnerSource are not of a technical nature, but rather organizational or sometimes political (Stol et al., 2014). Collaboration via InnerSource may also be constrained by an organization's context. For example, there may be transfer pricing issues related to sharing code across national borders (Buchner and Riehle, 2022). These constraints can be easily avoided when Agile teams are working independently. Alternatively, there may be policies in place constraining the sharing of code due to regulatory or policy compliance (Kalra and Afzal, 2023).

4.5. Motivations and Incentives

Individuals in Agile teams are often motivated by team-based incentives, project goals, and alignment with sprint objectives. Incentives can include financial rewards, career advancement, and team recognition.

Motivation to participate in InnerSource projects can be driven by a desire to share knowledge and make connections, as well as job satisfaction, and the desire to improve software quality (Dillon, 2024). Formal incentive programs for InnerSource can be difficult where individuals may be distributed across an organization. However, in some organizations, InnerSource contributions are linked to monetary rewards and the organization's promotion paths (Dey et al., 2022).

In addition, formal incentive programs may also be challenging when metrics and measurement of InnerSource success may not be available. Although many organizations report perceiving "measurable progress" against InnerSource goals (Dillon, 2024), it is also clear that measurement strategies are still immature and often rely on surveys and "gut feel".

4.6. Knowledge Sharing and Management

Within Agile teams, there is tacit knowledge and established methods of learning. For successful InnerSource adoption, there is often a requirement for more explicit knowledge-sharing practices including training and awareness about InnerSource practices, and additional documentation about host projects.

InnerSource practitioners also report learning through "lurking", by passively viewing code and observing communications within the Host team, and between the Host team and other external contributors.

In Agile, mentorship is often team-centric, with specific roles (Scrum Master, Agile Coach) and practices (pair programming) facilitating knowledge and skills transfer. Mentorship is also a key principle

of InnerSource (InnerSource Commons Foundation, 2016). In InnerSource, mentorship may be more organic, occurring through open contributions, code reviews, and community interactions. Mentor-mentee relationships can also span departments, creating additional social capital for the organization (Stol et al., 2014).

4.7. Culture and Mindset

An open culture is required for InnerSource to be successful. Agile teams value transparency through regular updates, daily stand-ups, and visible progress tracking (e.g., kanban boards). However, some of these practices require co-location to be an effective means of communication, and attitudes to visibility of code can vary. This can be challenging for InnerSource contributors who need to get familiar with project norms before contributing.

Organizational policies can mandate openness, but this can be a point of friction if developers feel that they are forced to make their code visible. Anecdotal feedback from practitioners suggests this may be due to a fear that their code may not be "up to scratch", or even a fear that efficient collaboration may make them redundant. Lack of access to tools can also hamper transparency and code visibility. Indeed, code may even be made visible, but transparency of processes and communications may still be constrained. It is acknowledged by InnerSource practitioners that explicit effort and additional resources (e.g. to provide awareness and education, build communities of practice) are often required to shift to a culture of openness and transparency at scale.

5. Conclusions

5.1. Limitations

This work is clearly limited to the extent that the empirical results are based on the experience of the authors, although our level of practitioner experience is significant in the substantive topics of Agile and InnerSource. Clearly however, our inherent biases will be present in the material presented. With that in mind, we have outline below significant plans to address this in future research on this topic.

5.2. Future Work

The key contribution of this work is the articulation of how Agile and InnerSource practices align and differ in significant and subtle ways. Although companies are increasingly adopting Agile and InnerSource practices

in tandem, little research is available on how to best approach implementing both methods in a harmonious way. Research on InnerSource overall is scarce and a detailed taxonomy of different InnerSource mechanisms and models for different organizational contexts is still lacking. There are therefore numerous opportunities for further research in this area. We believe there is an opportunity to further explore this topic in a Delphi Study including both Agile and InnerSource practitioners to give additional context to each of the areas we cover above. We would also like to conduct more detailed case studies to document successful implementations of InnerSource and Agile. Identifying best practices and common pitfalls could provide valuable guidance for organizations embarking on these methods. There is also a need to examine InnerSource and Agile in different organizational contexts (e.g. large versus small to medium enterprises, software practices spanning national borders).

We also see an opportunity to further explore how a combined Agile and InnerSource approach can enhance software development practices in particular industry contexts. Domain-specific standards and documents heavily regulate safety-critical systems, with one prominent example being the *DO-178C* standard for aerospace. Within such regulated environments, traditional Waterfall development processes are predominantly employed, diverging from the widespread adoption of Agile and InnerSource methods observed in the software industry. Leveraging Agile and InnerSource methods offer significant advantages alongside their promise to increase efficiency and knowledge sharing. Despite the potential benefits, the adoption of Agile and InnerSource practices within safety-critical domains like aerospace remains limited. Notably, standards like the *DO-178C* standard do not mandate or exclude specific software development methods, allowing for the integration of these types of methods and practices. Nonetheless, such methods and practices are underutilized within these contexts creating space for future research.

We conclude Agile and InnerSource may be a match made in heaven - though like many relationships, it will take work to get to that blissful state. There are many similarities between Agile and InnerSource principles and practices. There are also areas of potential friction, where InnerSource and Agile practices diverge. However, we believe these areas of potential friction can be addressed through thoughtful adoption strategies, and with further study we can build a stairway to heavenly Agile and InnerSource implementations.

References

- Abrahamsson, P., Warsta, J., Siponen, M., & Ronkainen, J. (2003). New directions on agile methods: A comparative analysis. *25th International Conference on Software Engineering*, 244–254.
- Ambler, S. (2002). *Agile modeling: Effective practices for extreme programming and the unified process*. John Wiley & Sons.
- Beck, K. (2000). *Extreme programming explained: Embrace change*. Addison-Wesley.
- Beck, K. (2004). *Extreme programming explained: Embrace change (2nd edition)*. Addison-Wesley.
- Boehm, B. (2002). Get ready for agile methods, with care. *IEEE Computer*, 35(1), 64–69.
- Bonewald, S. (2017). *Understanding the innersource checklist*. O'Reilly.
- Buchner, S., & Riehle, D. (2022). Calculating the costs of inner source collaboration by computing the time worked. <https://doi.org/10.24251/HICSS.2022.896>
- Chengalur-Smith, I., Nevo, S., & Fitzgerald, B. (2022). Enhancing hybrid oss development through agile methods and high media synchronicity. *SIGMIS Database*, 52(4), 92–118. <https://doi.org/10.1145/3508484.3508490>
- Cockburn, A. (2004). *Crystal clear: A human-powered methodology for small teams*. Addison-Wesley.
- Cooper, D., & Stol, K.-J. (2018). *Adopting innersource: Principles and case studies*. O'Reilly.
- Dey, T., Jiang, W., & Fitzgerald, B. (2022). Knights and gold stars a tale of InnerSource incentivization. *IEEE Software*, 39(6), 88–98. <https://doi.org/10.1109/MS.2022.3192647>
- Digital.ai. (2022). *15th annual state of agile report*. Retrieved June 1, 2022, from <https://digital.ai/resource-center/analyst-reports/state-of-agile-report>
- Dillon, C. (2024). State of innersource 2024. Retrieved June 12, 2024, from <https://youtu.be/8AnaQ7p6iLg>
- Dyba, T. (2000). Improvisation in small software organizations. *IEEE Software*, 17(5), 82–87.
- Dybå, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Inf. Softw. Technol.*, 50(9–10), 833–859. <https://doi.org/10.1016/j.infsof.2008.01.006>
- Erickson, J., Lyytinen, K., & Siau, K. (2005). Agile modeling, agile software development, and extreme programming: The state of research.

- Journal of Database Management*, 16(4), 88–100.
- Ferreira Ribeiro, J. E., Silva, J. G., & Aguiar, A. (2023). Beyond tradition: Evaluating agile feasibility in do-178c for aerospace software development.
- Ferreira Ribeiro, J. E., Silva, J. G., & Aguiar, A. (2024). Weaving agility in safety-critical software development for aerospace: From concerns to opportunities. *IEEE Access*, 12, 52778–52802. <https://doi.org/10.1109/ACCESS.2024.3387730>
- Fowler, M., Highsmith, J., et al. (2001). The agile manifesto. *Software development*, 9(8), 28–35.
- Gandomani, T. J., Zulzalil, H., Ghani, A. A. A., & Sultan, A. B. M. (2013). A systematic literature review on relationship between agile methods and open source software development methodology. *CoRR*, abs/1302.2748. <http://arxiv.org/abs/1302.2748>
- Gartner. (2023). *The gartner top strategic technology trends for software engineering*. Retrieved June 12, 2024, from <https://www.gartner.com/en/newsroom/press-releases/gartner-identifies-the-top-strategic-technology-trends-in-software-engineering-trends-for-2023>
- Goldman, R., & Gabriel, R. P. (2005). *Innovation happens elsewhere - open source as business strategy*. Elsevier. http://www.elsevier.com/wps/find/bookdescription.cws%5C_home/702604/description
- Goth, G. (2007). Sprinting toward open source development. *IEEE Software*, 24(1), 88–91. <https://doi.org/10.1109/MS.2007.28>
- Highsmith, J. (2000). *Adaptive software development: A collaborative approach to managing complex systems*. Dorset House, New York.
- Hoffmann, M., Nagle, F., & Zhou, Y. (2024). The value of open source software. *Harvard Business School Strategy Unit Working Paper*, (24-038).
- InnerSource Commons Foundation. (2016). Innersource commons foundation learning path - upleveling community members. Retrieved June 12, 2024, from <https://innersourcecommons.org/learn/learning-path/trusted-committer/04/>
- InnerSource Commons Foundation. (2024). *Innersource commons patterns*. Retrieved June 14, 2024, from <https://github.com/InnerSourceCommons/InnerSourcePatterns>
- Izquierdo-Cortázar, D., Alonso-Gutiérrez, J., Pérez García-Plaza, A., Robles, G., & González-Barahona, J. M. (2022). Starting the innersource journey: Key goals and metrics to measure collaboration. *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, 105–106. <https://doi.org/10.1145/3524842.3528044>
- Kalra, A., & Afzal, M. N. I. (2023). Transfer pricing practices in multinational corporations and their effects on developing countries' tax revenue: A systematic literature review. *International Trade, Politics and Development*, 7(3), 172–190.
- Koch, S., & Schneider, G. (2002). Effort, cooperation and coordination in an open source software project: Gnome. *Infor. Syst. J.*, 12(1), 27–42.
- Morgan, L., Gleasure, R., & Baiyere, A. (2022). Is inner source the next stage in the agile revolution? In A. Elbanna, S. McLoughlin, Y. K. Dwivedi, B. Donnellan, & D. Wastell (Eds.), *Co-creating for context in the transfer and diffusion of it* (pp. 130–136). Springer International Publishing.
- Nerur, S., Mahapatra, R., & Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. *Communications of the ACM*, 72–78.
- Oram, A. (2015). *Getting started with innersource. keys to collaboration and productivity inside your company*. O'Reilly.
- Raymond, E. (2001). *The cathedral & the bazaar: Musings on linux and open source by an accidental revolutionary*. O'Reilly Media.
- Russo, B., Scotto, M., Sillitti, A., & Succi, G. (2009). *Agile technologies in open source development*. Information Science Reference - Imprint of: IGI Publishing.
- Sadler, T., Ludmila, Tigges, J., & Rutledge, R. (2021). Innersource commons foundation learning path - contributor - mechanics of contributing. Retrieved June 12, 2024, from <https://innersourcecommons.net/resources/learningpath/contributor/04/>
- Schwaber, K., & Beedle, M. (2001). *Agile software development with scrum*. Prentice Hall.
- Silva Cardoso Rodrigues, J. M., Ferreira Ribeiro, J. E., & Aguiar, A. (2022). Improving documentation agility in safety-critical software systems development for aerospace. *2022 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 222–229. <https://doi.org/10.1109/ISSREW55968.2022.00071>
- Spier, S., Rutledge, R., Laura, & Drost-Fromm, I. (2023). Innersource commons foundation learning path - innersource and agile.

Retrieved June 12, 2024, from <https://innersourcecommons.org/learn/learning-path/project-leader/02/>

- Stapleton, J. (1997). *Dynamic systems development method: The method in practice*. Addison-Wesley.
- Stol, K.-J., Avgeriou, P., Babar, M. A., Lucas, Y., & Fitzgerald, B. (2014). Key factors for adopting inner source. *ACM Transactions on Software Engineering and Methodology*, 23(2), 18:1–18:35. <https://doi.org/10.1145/2533685>
- Sutherland, J., & Schwaber, K. (2011). The scrum papers: Nut, bolts, and origins of an agile framework. *Scrum Training Institute*.
- Warsta, J., & Abrahamsson, P. (2003). Is open source software development essentially an agile method. *Proceedings of the 3rd Workshop on Open Source Software Engineering*, 143–147.
- Williams, L., & Cockburn, A. (2003). Agile software development: It's about feedback and change. *IEEE Computer*, 36(6), 39–43.