



1. Das Erfordernis verteilter Entwicklung

Softwareentwicklung – vor noch nicht langer Zeit noch die Aufgabe Einzelner oder Weniger – ist längst als Teamsportart etabliert. Doch wer glaubt, damit sei die Entwicklung beendet, irrt: In Zukunft werden nicht die Entwicklergruppen die Innovation prägen, die in Bezug auf die interne Zusammenarbeit ihre Hausaufgaben gemacht haben, sondern jene, die darüber hinaus auch noch das Zusammenspiel mit ihrem Umfeld gemeistert haben. Dem Umfeld gehören dabei nicht nur Auftraggeber und Benutzer an, sondern immer stärker auch andere Entwicklergruppen. Grund dafür sind Standardisierung und Spezialisierung. Wer für jedes Problem eine eigene Lösung bereitstellen möchte, schafft sich unnötigen Aufwand. Vielmehr gilt es, auf vorhandenen und erprobten Lösungen aufzubauen und die eigene Expertise gezielt auf neue Problemfelder zu richten. Bei kleineren Projekten genügt es dabei, auf fertige Komponenten, Bibliotheken, Frameworks und APIs zurück zu greifen. Bei größeren Projekten, etwa in Konzernen oder bei gemeinsamen Entwicklungen unabhängiger Unternehmen, erfordert es Zusammenarbeit und Austausch schon während der Entwicklung: Die Schlagworte lauten „verteilte Entwicklung“ oder „Global Software Engineering“.

2. Warum Entwicklungsprojekte scheitern und was von Open Source zu lernen ist

Es ist kein Geheimnis, dass diese schöne neue Welt auch eine beachtliche Schattenseite hat. Die bekannten Projektrisiken werden durch die Verteilung erst einmal verstärkt, neue Risiken kommen hinzu. Wer schon einmal mit den Themen Offshoring oder Outsourcing in Berührung gekommen ist, kennt die speziellen Hürden [Dam07]. Sprachliche und kulturelle Eigenheiten müssen überwunden werden, falls die Kooperation international ist. Kommunikationsprobleme treten bereits auf, wenn die Entwickler in unterschiedlichen Gebäuden in einer Stadt sitzen. Findet die Zusammenarbeit über Unternehmensgrenzen hinweg statt, ist ein stetiger Abfluss von Wissen und Kompetenzen zu befürchten [EMJ08]. Sobald andere Interessensgruppen nicht nur als Zulieferer, sondern – beispielsweise bei einem abteilungsübergreifenden Projekt – als gleichberechtigte Partner eigene Anforderungen, Vorstellungen und Wünsche mitbringen, wird zusätzlich eine Harmonisierung der Interessen erforderlich.

Bemerkenswert ist, dass es sehr ähnliche Voraussetzungen sind, in denen Free/Open Source Software (FOSS) Projekte scheinbar mühelos gedeihen [Fin03]:

- | Extrem verteilte, teils sehr große Teams, teils mit starker Fluktuation
- | Meist ähnliche, teils aber auch widersprüchliche Anforderungen und Ziele der Beteiligten
- | Volatile Umwelteinflüsse, stetiger Wandel von Anforderungen und Prioritäten
- | Generell knappe Ressourcen, die in ständiger Konkurrenz zu anderen Projekten oder anderen Interessen und Verpflichtungen der Projektteilnehmer stehen

Da Open Source-Projekte überwiegend offen und transparent sind, können Erfolge und Misserfolge von jedermann nachvollzogen und als Anregung für die eigene Arbeit genutzt werden.

Sind Methoden und Prozessmodelle Open Source-Entwicklung auch für kommerzielle Softwareentwicklung einsetzbar?



Prof. Dr. Volker Gruhn

Institut für Informatik/Professur für
Angewandte Telematik/e-Business

Universität Leipzig

3. Open Source Best Practices als Chancen für die kommerzielle Softwareentwicklung

Einige Technologien, die ihren Ursprung in der FOSS-Bewegung haben oder ihr zumindest ihre aktuelle Verbreitung verdanken, sind längst auch in der kommerziellen Softwareentwicklung etabliert [RJT09]. Dazu gehören interaktive und agile Prozesse, Versionierungswerkzeuge (SVN, CVS), Wikis und viele mehr. Andere Verfahren sind dabei sich einzureihen. Die Verbreitung von Software Forges beispielsweise nimmt stetig zu [RJT09]. Zu beachten ist, dass es hierbei nicht einfach nur um eine Werkzeuggattung geht, sondern dass mit derartigen Werkzeugen immer auch gewisse Paradigmen Einzug in die Entwicklung halten.

Organisationen, die Software nicht für externe Kunden, sondern für den eigenen Bedarf entwickeln (z. B. Finanzdienstleister, Energieunternehmen, aber auch der öffentliche Dienst) können von den FOSS Build- und Test-Praktiken einiges lernen – da die Endnutzer zugleich Mitarbeiter sind, könnten sie optimal als „Aktive Nutzer“ eingebunden werden.

In anderen Bereichen, etwa der Projektorganisation und der verteilten Softwareentwicklung, schlummern noch ungeahnte Potenziale. Solange nur die Methoden, nicht aber die Software von FOSS übernommen wird, kann die „Offenheit“ übrigens schon am Werkstor enden. Sofern es um Kernkompetenzen geht, sollte sie das auch [ZOM08]. Bei der Entwicklung von einfachen Werkzeugen, Commodities und dergleichen kann sich dagegen auch die Zusammenarbeit mit anderen Unternehmen, bis hin zum Engagement in einem echten FOSS-Projekt, lohnen.

Literatur

[Dam07] Damian, Daniela, Stakeholders in Global Requirements Engineering: Lessons Learned from Practice, IEEE Software, vol. 24, no. 2, pp. 21-27, 2007.

[EMJ08] Ebert, Christof, et al., Managing Risks in Global Software Engineering, International Conference on Global Software Engineering, pp. 131-140, 2008.

[Fin03] Fink, Martin, The business and economics of Linux and open Source, Prentice Hall, 2003.

[RJT09] Riehle, Dirk et al., Open Collaboration within Corporations Using Software Forges, IEEE Software, vol. 26, no. 2, pp. 52-58, 2009.

[ZOM08] Ziemer, Sven, Østerlie, Thomas, Pentti, Martti: Balancing Flexibility and Control in Open Source Requirements Engineering, COSI Deliverable, pp. 48-57, 2008