Forschungsbeitrag zur Informatik

Stephan Krusche studierte von 2005 bis 2011 Informatik an der TU München und promovierte anschließend bis 2016 am Lehrstuhl für angewandte Softwaretechnik der TU München unter der Betreuung von Prof. Bernd Brügge. Als zertifizierter Scrum Master betreut er seit 2012 Projekte, in denen Software agil entwickelt wird. Sein Forschungsschwerpunkt liegt in der Kombination von agilen Methoden mit kontinuierlicher Software Entwicklung, sowohl in Industrieprojekten als auch in der Lehre. Im Rahmen seiner Promotion entwickelte er dazu das kontinuierliche Prozessmodell Rugby, das seit 2012 erfolgreich in Projektkursen mit 100 Studierenden eingesetzt wird.
Zusammen mit seinem Team konnte er Rugby 2014 in mehreren Projekten bei der Firma Capgemini einführen. Neben Zeiteinsparungen bei der Integration und Auslieferung von Software konnte Rugby die Bereiche Testen und Metriken verbessern. Die Einführung von Rugby in Unternehmen ist ein gutes Beispiel für Technologietransfer zwischen Wissenschaft und Industrie und verbessert die Wettbewerbsfähigkeit von Unternehmen. Stephan Krusche integrierte agile und kontinuierliche Methoden in das Übungskonzept einer Vorlesung über Projektmanagement mit 400 Studierenden im Jahr 2015. Ein Jahr später hielt er diese Vorlesung und verbesserte das Übungskonzept weiter. Er strebt eine Habilitation und Universitätskarrierie an.

Stephan Krusche

Kontinuierliche Software Entwicklung – Höhere Qualität durch die Einbeziehung von Nutzer Feedback

Die Softwaretechnik wurde 1968 auf der NATO Konferenz in Garmisch begründet [1]. Die Vision auf dieser Konferenz war es, von einer eher unstrukturierten Arbeitsweise zu einem definierten Prozessmodell zu wechseln. Prozessmodelle beschreiben wie Software entwickelt werden soll. Die ersten Software Prozessmodelle verwendeten Methoden aus anderen Ingenieurswissenschaften und aus Produktionsstätten. Dadurch entstanden lineare Ansätze, wie z.B. das Wasserfall Modell [2] oder das V-Modell, das Software Teams in deutschen Regierungsprojekten verwenden mussten [3]. Lineare Modelle nutzen eine definierte Vorgehensweise mit einem strikten Plan und festen Regeln. Jede Abweichung vom Plan wird als Fehler angesehen, der korrigiert werden muss. Der Hauptnachteil von linearen Ansätzen ist die Schwierigkeit auf Änderungen von Technologien oder Anforderungen nach dem Start des Projektes zu reagieren, da Phasen nacheinander und vollständig abgeschlossen werden. Anforderungen, die zur Analyse Phase am Projektanfang nicht berücksichtigt und erst später entdeckt werden, können nicht umgesetzt werden oder führen zu hohen Aufwänden.

Software Entwickler erkannten in den Folgejahren, dass lineare Modelle die Wirklichkeit nicht korrekt abbilden und entwickelten iterative und inkrementelle Vorgehensmodelle, wie z.B. das Spiralmodell [4] oder den Unified Process [5] (in deutsch: vereinheitlichter Prozess). In diesen Modellen durchlaufen Software Teams Projektphasen wie Analyse, Design und Implementierung mehrfach, wodurch Risiken früher erkannt und deren negative Auswirkungen vermieden werden können. Obwohl damit Änderungen nach dem Projektstart möglich waren, entsprachen diese Modelle weiterhin eher definierten Prozessen. Die Essenz der Software Entwicklung ist der Umgang mit Abweichungen und Änderungen. Software Entwicklung ist experimentelle Wissensarbeit, bei der Kreativität und die Reaktion auf Unerwartetes eine wichtige Rolle spielen [6]. Unsicherheit kann durch planmäßiges Vorgehen nicht beseitigt werden, sondern muss akzeptiert werden.

Kurzzusammenfassung:

Software wird unter immer dynamischeren Bedingungen entwickelt. Organisationen benötigen die Fähigkeit mit Ungewissheit umzugehen und auf unerwartete Änderungen zu reagieren. Rugby ist ein anpassbares und erweiterbares Prozessmodell zur kontinuierlichen Software Entwicklung und für die Lehre. Es integriert Arbeitsabläufe zur Verwaltung von Reviews, Releases und Feedback. Rugby wurde in drei Fallstudien angewandt: (1) in 62 Universitätsprojekten mit 500 Studierenden; (2) in einer Vorlesung mit 57 Team Projekten und 400 Studierenden; sowie (3) in 8 Industrieprojekten mit 31 Fachkräften im Unternehmen. Eine empirische Evaluation zeigt, dass kontinuierliche Lieferung, Einbeziehung von Nutzer Feedback und kontinuierliche Reviews die Qualität von Entwicklungs- und Lehrprozessen verbessern

In den 1990er Jahren entstanden agile Methoden mit der Philosophie, dass Software Entwicklung keinem definierten, sondern einem empirischen Vorgehen entspricht. Im Gegensatz zu definierten Modellen sehen empirische Modelle Abweichungen, Probleme und Fehler als Chancen an, die durch regelmäßige Inspektion erkannt und untersucht werden und die zu Anpassungen im Prozess führen. Scrum ist eine agile Vorgehensweise, bei der Software Teams in vierwöchigen Iterationen, sog. Sprints, entwickeln [6]. Innerhalb eines Sprints fokussiert sich das Team darauf, vorab definierte Anforderungen umzusetzen und so in die Software zu integrieren, dass die Software potentiell ausgeliefert werden kann. Innerhalb eines Sprints sind keine Änderungen möglich. Diese Regel soll das Team vor zu vielen Änderungen schützen, sie kann aber auch zu dem Problem führen, dass Entwickler falsch verstandene Anforderungen falsch umsetzen und erst am Sprint Ende davon erfahren.

„Organisationen benötigen die Fähigkeit mit Ungewissheit umzugehen und auf unerwartete Änderungen von Anforderungen und Technologien zu reagieren.“

Stephan Krusche

Software wird heutzutage in immer dynamischeren Zeiten entwickelt: Sich schnell wandelnde Märkte, komplexe und sich ändernde Kundenanforderungen, der Druck zu kürzeren Markteinführungszeiten und sich schnell weiterentwickelnde Technologien sind Rahmenbedingungen, die in Software Projekten eine Rolle spielen. Die digitale Transformation findet in allen wichtigen Industrien statt: ein Beispiel dafür sind mobile Anwendungen, deren Einsatz in kritischen Geschäftsbereichen in den letzten Jahren stark angestiegen ist. Kontinuierliche Software Entwicklung ist ein Ansatz um die Herausforderungen durch sich schnell ändernde Anforderungen und Technologien zu bewältigen. Kontinuierlich bezieht sich dabei auf die organisatorische Fähigkeit Software in schnellen Zyklen zu entwickeln, sie jederzeit ausliefern zu können, aus der Nutzung der Software zu lernen und das gewonnene Feedback in den Entwicklungsprozess einzubringen. [7]

Rugby ist ein Vorgehensmodell für kontinuierliche Software Entwicklung und wurde an der Technischen Universität München entwickelt [8]. Es basiert auf agilen Konzepten von Scrum [6] und parallelen Arbeitsabläufen vom Unified Process [5]. Rugby integriert drei Arbeitsabläufe für Review Management, Release Management und Feedback Management und ermöglicht Software Evolution, die kontinuierliche Anpassung von Software an die Änderungen der Umgebung und an die Bedürfnisse der Nutzer [9]. Arbeitsabläufe werden in Rugby am Anfang des Projektes auf die Gegebenheiten und die Besonderheiten eines Projekts angepasst. Notwendige Werkzeuge für die Arbeitsabläufe werden ausgewählt und aufgesetzt.

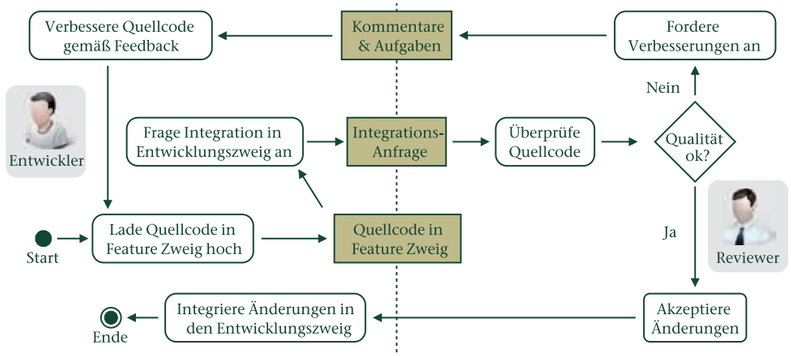

Abbildung 1 zeigt den Arbeitsablauf für Review Management als UML Aktivitätsdiagramm [11]. Dieser startet, wenn Entwickler eine neue Anforderung implementieren. Sie erzeugen dazu einen Feature Zweig, der einen separaten Bereich im Versionskontrollsystem darstellt und in dem sie alle Quellcode Änderungen für diese Anforderung einfügen. Entwickler können mehrere Feature Zweige anlegen um gleichzeitig mehrere Anforderungen parallel zu realisieren. Feature Zweige isolieren dabei alle Änderungen pro Anforderung und bieten den Vorteil, dass Entwickler ungestört von anderen Änderungen arbeiten können. Sobald eine Anforderung realisiert ist, fragen die Entwickler die Integration in den Hauptentwicklungszweig an. Sie erzeugen dafür eine Integrationsanfrage, die alle Änderungen am Quellcode für die Realisierung der Anforderung beinhaltet. Andere, erfahrene Entwickler im Team können in der Rolle des Reviewers diese Änderungen nun auf Konformität zur Softwarearchitektur und zu den Quellcode Richtlinien überprüfen. Wenn die Qualität nicht in Ordnung ist, fordern die Reviewer Verbesserungen an und hinterlassen Kommentare direkt im Quellcode.

Die Entwickler verbessern den Quellcode anhand dieser Kommentare und erzeugen weitere Änderungen im Feature Zweig, wodurch die Integrationsanfrage automatisch aktuali-

Abbildung 1: *Arbeitsablauf für Review Management als UML Aktivitätsdiagramm [KBB16]: Reviewer überprüfen den Quellcode von Entwicklern und fragen Verbesserungen an, bevor dieser in den Entwicklungszweig integriert wird.*

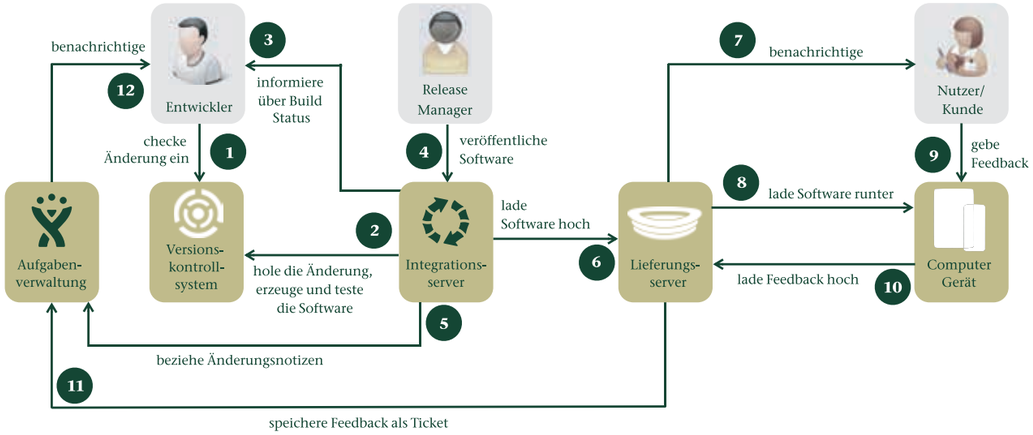

Abbildung 2:
Arbeitsablauf für Release Management [8]: Quellcode in der Versionskontrolle wird automatisch übersetzt in Software, die vom Release Manager veröffentlicht und dann vom Nutzer heruntergeladen werden kann. Feedback wird über die Aufgabenverwaltung den Entwicklern zur Verfügung gestellt.

siert wird. Bei der erneuten Überprüfung achten die Reviewer darauf, dass die Kommentare eingearbeitet und der Quellcode verbessert wurde. Sollte die Qualität noch nicht den definierten Standards entsprechen, fordern die Reviewer solange Verbesserungen, bis die Qualität stimmt. Erst dann akzeptieren sie die Änderungen, sodass die Entwickler sie vom Feature Zweig in den Hauptentwicklungszweig integrieren können.

Dieser Ablauf hat den Vorteil, dass nur geprüfter Quellcode in den Hauptentwicklungszweig gelangt, von dem aus alle weiteren Features implementiert werden. Mit diesem Ablauf wird verhindert, dass sich schlechter Quellcode und falsche Entwurfsentscheidungen in den Hauptentwicklungszweig ausbreiten [11]. Änderungen im Quellcode werden nur einmal und nicht mehrfach geprüft. Kollaboration und Wissensaustausch im Team verbessern sich. Sofern die Anforderungen nicht zu komplex sind und in wenigen Tagen entwickelt werden, sind die Reviews schnell und effizient und es entstehen keine Verzögerungen oder Integrationsprobleme durch die parallele Entwicklung.

Abbildung 2 zeigt den Arbeitsablauf für Release Management [8] mit dem Informationsfluss zwischen Akteuren und Werkzeugen: Er startet, wenn ein Entwickler Quellcode in das Versionskontrollsystem hoch lädt, unabhängig davon in welchem Zweig sich der Quellcode befindet. Der Integrationsserver holt sich die Änderungen, erzeugt die Software, führt Testfälle aus und informiert den Entwickler über das Ergebnis.

Der Release Manager im Team ist zuständig für die Auslieferung der Software und kann die Software jederzeit manuell veröffentlichen, d.h. den Kunden oder Testnutzern während der Entwicklung zur Verfügung stellen, sofern die Erzeugung der Software funktioniert hat. Die Software wird hochgeladen und dem Benutzer zur Verfügung gestellt. Der Nutzer kann die Software herunterladen und auf seinem Gerät direkt Feedback geben, das den Entwicklern in der Aufgabenverwaltung zur Verfügung gestellt wird. Die Entwickler haben nun die Möglichkeit das Feedback direkt zu adressieren oder es sich für einen späteren Zeitpunkt zu merken.

Diese Wahl hängt von der Art des Feedbacks ab. Abbildung 3 zeigt den Arbeitsablauf für Feedback Management mit Beispielen für Feedback [8]. Aktivitäten sind durch farbige Rechtecke gezeigt, deren Breite die Dauer der Aktivität darstellt. Übergänge zwischen Aktivitäten sind durch Pfeile dargestellt.

> *„Kontinuierliche Software Entwicklung ermöglicht es jederzeit Software auszuliefern, Nutzer Feedback zu sammeln und dieses in der Entwicklung zu berücksichtigen.“*
>
> *Stephan Krusche*

Das initiale Release R0 und das initiale Feedback F0 werden am Anfang des Projekts erzeugt, damit sichergestellt wird, dass die Arbeitsabläufe und Werkzeuge korrekt aufgesetzt sind. Beim darauffolgenden Release R1 stürzt die Anwendung ab und es wird automatisch ein Fehlerbericht F1 erzeugt, mit dem die Entwickler den Absturz nachvollziehen können. Entwickler können so direkt zur Implementierungsphase übergehen und den Fehler beheben. Handelt es sich wie bei F2 um eine Design Verbesserung, müssen sich die Entwickler erst Gedanken machen, wie sie das Design anpassen, bevor sie zur Implementierung übergehen. Wenn der Kunde im Feedback wie im Fall F3 eine neue Anforderung beschreibt, können die Entwickler entscheiden, diese Anforderung nicht direkt umzusetzen, sondern auf einen späteren Zeitpunkt zu verschieben, da sie sich im aktuellen Sprint auf die bestehenden Funktionalitäten konzentrieren möchten. Dieser Arbeitsablauf ermöglicht Entwicklern Feedback schnell umzusetzen und die geänderte Software direkt Nutzern oder Kunden zur Überprüfung zur Verfügung zu stellen, wodurch sich deren Motivation zur Feedback Bereitstellung erhöht.

Die drei Arbeitsabläufe für Review, Release und Feedback Management sind in Rugby in einem Prozess Model integriert und

„Es ist nicht nur wichtig, kontinuierliche Abläufe in der Industrie anzuwenden, sie müssen auch an der Universität unterrichtet werden, damit Software Entwickler der Zukunft diese Fähigkeiten verinnerlichen.“

Stephan Krusche

werden regelmäßig durchlaufen. Rugby wurde in drei Fallstudien in der Universität und in der Industrie verwendet und evaluiert. Seit 2011 wird Rugby erfolgreich in Projektkursen mit bis zu 100 Studierenden und 11 Teams an der Universität verwendet, bei der Studierende mobile iOS Anwendungen in möglichst realen Projekten für Industriepartner entwickeln [12]. Im Projektkurs 2014 konnten die Studierenden in drei Monaten durchschnittlich pro Team den Quellcode 96 mal überprüfen und die mobilen Anwendungen 64 mal ausliefern. Durchschnittlich wurden die Anwendungen 135 mal heruntergeladen und es wurden 27 Feedback Berichte erstellt.

Durch die Berücksichtigung der Verbesserungsvorschläge aus den Reviews konnte die Qualität des Quellcodes und der Architektur erhöht werden. Mittels Feedback konnte die Qualität der Anwendung für den Nutzer verbessert werden. Dadurch, dass alle Team Mitglieder jederzeit den aktuellen Software Stand ausprobieren und auf Smartphone bzw. Tablet anderen zeigen konnten, gab es weniger unnötige Diskussionen in Meetings, wodurch Zeit gespart werden konnte. Mehr als 80 % der Studierenden wollen die Arbeitsabläufe für Reviews, Releases und Feedback in künftigen Projekten verwenden.

In einer zweiten Fallstudie im Sommer 2015 wurden die Abläufe von Rugby in das Übungskonzept zur Vorlesung Software Entwicklung II – Projektorganisation und Management integriert, bei der über 400 Studierende registriert waren. In fünf Zentralübungen erlernten die Studierenden in individuellen, interaktiven und schrittweise aufgebauten Tutorials agile und kontinuierliche Konzepte, die sie anschließend in Team Projekten anwendeten. In einer Evaluation stimmten zwischen 77 % und 88 % der Übungsteilnehmer zu, dass sie ihre Fähigkeiten in den Übungen verbesserten und sich zuversichtlich zeigen, die gelernten Konzepte in künftigen Projekten anzuwenden.

Durch die Teilnahme an den Übungen konnten die Studierenden einen Notenbonus zwischen 0,3 und 1,3 erreichen. Eine Auswertung der Klausurergebnisse zeigte, dass Studierende, die erfolgreich an den Übungen teilnahmen, im Schnitt deutlich bessere Klausurnoten erreichten. Umso höher der Notenbonus war, umso besser war auch die durchschnittliche Klausurnote vor Abzug des Notenbonus. So erreichten Studierende ohne einen Notenbonus im Schnitt eine 3,9 und Studierenden mit dem höchsten Notenbonus im Schnitt eine 2,4. Dies zeigt, dass die Teilnahme an den Übungen den Studierenden auch die theoretischen Konzepte für die erfolgreiche Teilnahme an der Klausur vermittelte.

In einer dritten Fallstudie im Herbst 2014 wurde Rugby als Vorgehensmodell in 8 Industrieprojekten bei der Firma Capgemini verwendet [13], in denen mobile Anwendungen entwickelt wurden. Die Projektteams passten Rugby für ihre Bedürfnisse an und waren in der Lage den Aufwand für die Integration und Auslieferung der Anwendungen von mehreren Stunden auf wenige Minuten zu verringern, während sie die Anzahl der Auslieferungen der Software erhöhen konnten. Dadurch waren Kunden und Nutzer in der Lage häufiger die Anwendungen auszuprobieren und stärker die Entwicklung zu beeinflussen.

Abschließend lässt sich zusammenfassen, dass das Vorgehensmodell Rugby ein wichtiger Schritt auf dem Weg zur kontinuierlichen Software Entwicklung ist, mit der Software Teams den heutzutage immer dynamischeren Rahmenbedingungen begegnen können. Reviews erlauben es die Qualität des Quellcodes und der Software

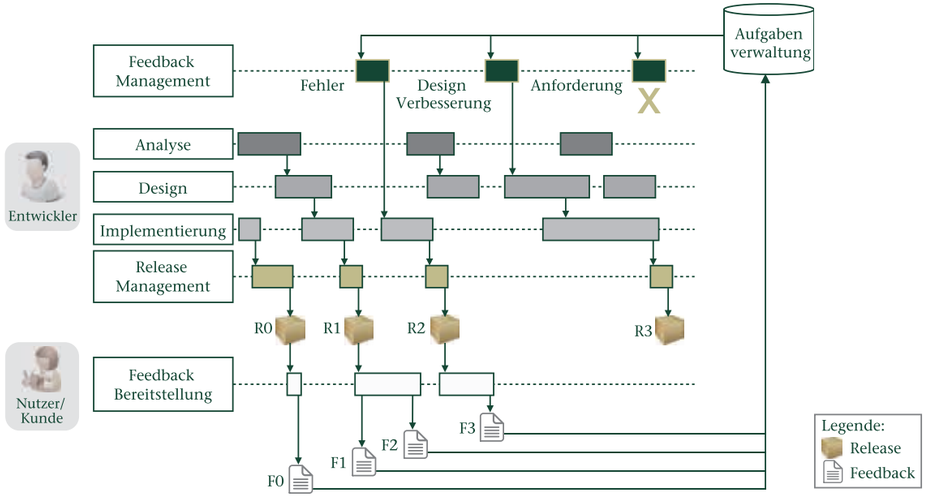

Abbildung 3:
Arbeitsablauf für Feedback Management [8]: Je nachdem um was es sich für ein Feedback handelt, werden unterschiedliche Arbeitsabläufe initiiert. Feedback kann direkt eingearbeitet werden, oder auf einen späteren Zeitpunkt verschoben werden, wenn es sich um größere Änderungen handelt.

Architektur zu verbessern und erhöhen die Zuverlässigkeit und die Wartbarkeit. Durch die Möglichkeit, die Software jederzeit auszuliefern, werden Änderungen früher erkannt. Das spart Zeit und verbessert die Produktqualität. Die Einbindung von Nutzer Feedback erlaubt es das richtige Produkt zu entwickeln und erhöht die Kundenzufriedenheit. Es ist nicht nur wichtig, die Konzepte und kontinuierlichen Arbeitsabläufe von Rugby in der Industrie anzuwenden, sie müssen auch an der Universität unterrichtet werden, damit die Software Entwickler und Projekt Manager der Zukunft diese Fähigkeiten verinnerlichen und beherrschen.

Literaturverzeichnis

[1] Peter Naur and Brian Randell. Software Engineering: Report of the 1968 conference in Garmisch, Germany. NATO Software Engineering Conference, 1969.

[2] Winston Royce. Managing the development of large software systems. In Proceedings of IEEE WES-CON, 1970.

[3] Bundesamt für Wehrtechnik und Beschaffung. Software Development Standard for the German Federal Armed Forces, V-Model, Software Lifecycle Process Model, BWB General Directive 250 edition, 1992.

[4] Barry Boehm. A spiral model of software development and enhancement. Computer, 21(5):61–72, 1988.

[5] Ivar Jacobson, Grady Booch, and James Rumbaugh. The Unified Software Development Process. Addison-Wesley, 1998.

[6] K. Schwaber and M. Beedle. Agile software development with Scrum. Prentice Hall, 2002.

[7] Jan Bosch. Continuous Software Engineering. Springer, 2014.

[8] Stephan Krusche, Lukas Alperowitz, Bernd Bruegge, and Martin Wagner. Rugby: An agile process model based on continuous delivery. In Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering, pages 42–50. ACM, 2014.

[9] Tom Mens. Introduction and roadmap: History and challenges of software evolution. Springer, 2008.

[10] Andrew Hunt and David Thomas. The pragmatic programmer: from journeyman to master. Addison-Wesley, 2000.

[11] Stephan Krusche, Mjellma Berisha, and Bernd Bruegge. Teaching Code Review Management using Branch Based Workflows. In Companion Proceedings of the 38th International Conference on Soft-ware Engineering, pages 384–393. ACM, 2016.

[12] Bernd Bruegge, Stephan Krusche, and Lukas Alperowitz. Software Engineering Project Courses with Industrial Clients. ACM Transactions on Computing Education, 15(4):17:1–17:31, 2015.

[13] Sebastian Klepper, Stephan Krusche, Sebastian Peters, Bernd Bruegge, and Lukas Alperowitz. Introducing continuous delivery of mobile apps in a corporate environment: A case study. In Proceedings of the 2nd International Workshop on Rapid Continuous Software Engineering, pages 5–11. IEEE/ACM, 2015.