Problemas que afectan a la Calidad de Software en Entrega Continua y Pruebas Continuas

Maximiliano A. Mascheroni[1,2], Emanuel Irrazábal[1],

[1] Facultad de Ciencias Exactas y Naturales y Agrimensura.
Universidad Nacional del Nordeste. Corrientes, Argentina.
[2] Facultad de Informática. Universidad Nacional de La Plata.
La Plata, Buenos Aires, Argentina.

{mascheroni, eirrazabal}@exa.unne.edu.ar

Abstract. Muchas organizaciones que desarrollan software de manera ágil buscan adoptar el enfoque de entrega continua, el cual permite la liberación del software a producción en cualquier momento. Sin embargo, uno de los principales desafíos es mantener la calidad del mismo gestionando adecuadamente el tiempo que se invierte en las pruebas. En este sentido, se han reportado numerosos problemas relacionados con la calidad del software al adoptar este enfoque. Asimismo, algunos autores y empresas han propuesto soluciones para hacer frente a algunos de ellos, pero surge el interrogante de si las pruebas continuas son el elemento clave para estas prácticas. En este trabajo, se presenta un resumen de los principales problemas que afectan a la calidad de software en entrega continua y la relación que existe entre los mismos. Al final, se propone como resultado del análisis de estudios previos una definición al concepto de “pruebas continuas".

Keywords: entrega continua, calidad de software, pruebas continuas.

1 Introducción

El desarrollo ágil de software ha incorporado una gran variedad de buenas prácticas y procesos para producir software de manera más veloz y eficiente [1]. Una de las más conocidas es la Integración Continua (CI). CI es una práctica del desarrollo de software, en donde los desarrolladores integran el código fuente con frecuencia y cada integración es verificada por un sistema que construye el código y lo prueba automáticamente [2]. Esta práctica en sus inicios estaba soportada solamente por pruebas unitarias. Pero con la aparición de nuevas herramientas como Selenium WebDriver[1], niveles más altos de pruebas podían ser automatizados, y así introducidos en el proceso de CI. Sin embargo, para ejecutar pruebas de manera automática sobre una versión del código en ejecución, es necesario el despliegue del

[1] Selenium Webdriver es una herramienta que permite la automatización de la interacción con páginas web. https://www.seleniumhq.org/projects/webdriver/

mismo a un ambiente de desarrollo. Esto dio lugar a la aparición de nuevas herramientas para automatizar el despliegue de código, y así incorporar este proceso, junto con la ejecución de pruebas automatizadas de distintos niveles en CI.

Posteriormente, se añade la idea de realizar despliegues automatizados en producción. Por un lado, algunas compañías buscaban automatizar el proceso completo, desde la integración del código fuente, hasta el despliegue a producción. Por otro lado, otras empresas, deciden automatizar todo el flujo, pero el despliegue a producción se realizaría de manera automática al presionar un botón cuando el cliente lo indique. El primer enfoque se denomina Despliegue Continuo (DC) y el segundo Entrega Continua (CD, del inglés Continuous Delivery). En todas estas prácticas (CI, DC y CD), las pruebas automatizadas cumplen un rol fundamental, ya que deben brindar a los desarrolladores confiabilidad en cada versión del producto que se construye. Para ello, los autores de CD proponen un patrón llamado *conducto de despliegue*, más conocido en inglés como Deployment Pipeline (DP) [3]. Con DP, el flujo se divide en diferentes etapas, donde a medida que se avanza de una etapa a la otra, se aumenta la confiabilidad, usualmente al costo de más tiempo de ejecución [4].

De esta manera, encontrar un equilibrio entre un tiempo de ejecución pequeño y un lote de pruebas que aseguren la calidad de cada entregable es una tarea difícil de llevar a cabo. En la literatura, pueden encontrarse revisiones y casos de estudio que reportan problemas relacionados con CD, y la mayor parte de ellos afectan a la calidad.

Con el fin de contribuir a la apertura de nuevas líneas de investigación en el área de ingeniería de software, en este trabajo se realiza un resumen de los problemas más frecuentes reportados en CD, que afectan a la calidad del software. Asimismo, se realiza un análisis de definiciones de "pruebas continuas", proponiendo una más general.

Además de esta sección introductoria, en la sección 2 se resume brevemente en qué consiste CD y sus elementos principales. En la sección 3 se describen los problemas que afectan a la calidad. La sección 4 representa el apartado de pruebas continuas. Finalmente, en la sección 5 se perfilan conclusiones y trabajos futuros.

2 Entrega Continua

Según Humble y Farley [3], CD fue creado para resolver el siguiente interrogante: “Si alguien tiene una buena idea, ¿cómo se la hace llegar a los usuarios lo más rápido posible?”. CD se define formalmente como un enfoque en el cual los equipos mantienen la producción de software en ciclos cortos de tiempo, asegurando que el producto pueda ser lanzado de manera fiable en cualquier momento [3, 5]. El objetivo es lanzar a producción una versión del software libre de defectos “con solo apretar un botón” [3]. Los principales beneficios de CD son [3, 4]: riesgo más bajo al realizar el despliegue; progreso real; y feedback inmediato por parte del usuario final.

La clave principal de CD es permitir a los clientes o la gente involucrada en el negocio, requerir que el desarrollo actual de una versión del software pueda ser lanzado a producción en el mismo momento en que se lo solicita. Sin embargo, para

lograrlo se deben cumplir ciertos requisitos. Los autores recomiendan las siguientes prácticas para la implementación de CD [3, 4]:

- Integración continua.
- Gestión de la configuración, no solo del código fuente, sino de cualquier artefacto relacionado al desarrollo del software, como por ejemplo esquemas de bases de datos, archivos de configuración, etc.
- Automatización de pruebas de software, en la mayor cantidad de niveles posibles (unitarias, de integración, funcionales, etc.)
- Generación de una cultura centrada en la comunicación, colaboración e integración entre desarrolladores de software y profesionales de operaciones en las tecnologías de la información (DevOps).

Además, para que el producto se encuentre libre de defectos, es necesario que los lotes de pruebas automatizados brinden confiabilidad. Como se mencionó anteriormente, DP es un patrón que permite dividir el flujo en etapas. Aún no existe un modelo estándar para la aplicación de las etapas de DP [5], y cada empresa lo implementa según sus necesidades. Sin embargo, el DP presentado por Humble y Farley [3] tiene las siguientes etapas (ver Fig. 1):

1. Introducción de cambios: incluye la compilación del código, ejecución de pruebas unitarias, análisis estático del código fuente y creación de artefactos.
2. Ejecución de pruebas de aceptación automatizadas.
3. Ejecución de pruebas no funcionales automatizadas: se deben incluir pruebas de capacidad, carga y sobrecarga.
4. Ejecución de pruebas manuales.

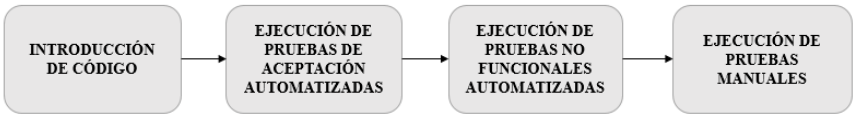

Fig. 1. Deployment Pipeline (conducto de despliegue).

En la actualidad, existen herramientas en el mercado que facilitan la implementación de las etapas de CD. Por otro lado, algunas empresas optan por integrar herramientas de CI con herramientas de DC. Finalmente, existen módulos y plugins de las propias herramientas de CI que permiten el despliegue de software. Algunos ejemplos de estas herramientas pueden verse en la Tabla 1, donde se especifica su aplicación (CI, DC, CD, plugin o módulo), el tipo de licencia (libre, comercial o ambos) y finalmente su sitio web. Sin embargo, si bien las mismas facilitan la implementación de las etapas de DP, todavía no existe un guía, modelo o lista de factores claves para el diseño, desarrollo y ejecución de pruebas en CD. En la literatura, pueden encontrarse revisiones sistemáticas que estudian los problemas de CD, y mencionan que aún se encuentran inconvenientes relacionados a las pruebas en este tipo de enfoque. Además, empresas y casos de estudios, reportan que mantener la calidad de software resulta un desafío al implementar CD.

Tabla 1. Herramientas más populares para implementar DP.

Herramienta	Aplicación	Licencia	URL
GoCD	Entrega Continua	Libre	https://www.gocd.org/
Jenkins	Integración Continua	Libre	https://jenkins.io/
Travis CI	Integración Continua	Ambos	https://travis-ci.org/
Bamboo	Integración Continua	Comercial	https://atlassian.com/software/bamboo
Strider CD	Despliegue Continuo	Libre	https://github.com/Strider-CD
Team City	Integración Continua	Ambos	https://jetbrains.com/teamcity/
Chef	Despliegue Continuo	Ambos	https://www.chef.io/chef
Ansible Tower	Despliegue Continuo	Comercial	https://www.ansible.com/products/tower
Buddy	Entrega Continua	Ambos	https://buddy.works/
Puppet	Despliegue Continuo	Ambos	https://puppet.com/
Wercker	Entrega Continua	Ambos	http://www.wercker.com
Jenkins Pipelines	Plugin	Libre	https://jenkins.io/solutions/pipeline/
Bamboo Deployments	Módulo	Comercial	https://atlassian.com/software/bamboo
Puppet Pipelines	Módulo	Comercial	https://puppet.com/products/puppet-pipelines

3 Problemas que afectan a la calidad de software

Actualmente, CD es un concepto que está siendo adoptado por empresas que desarrollan software de manera ágil. Como se mencionó anteriormente, esta disciplina soporta prácticas ágiles y reduce los tiempos de lanzamiento de nuevas versiones del producto al mercado, de semanas a solo horas. Sin embargo, de acuerdo a Prusak [6], "la industria aún no ha cerrado el círculo cuando se trata de implementar un proceso completo de entrega continua". Si bien la literatura contiene instrucciones y buenas prácticas de cómo adoptar CD, su implementación ha sido un desafío en la práctica [7]. Es por ello que para algunos autores [6, 8, 9], las pruebas continuas (Continuous Testing) son el elemento faltante en el proceso de desarrollo continuo, que incluye las prácticas mencionadas.

Algunas organizaciones aún no han podido adoptar CD completamente [10], y otras han encontrado muchos obstáculos [5, 10–15]. La mayor parte de los problemas reportados están relacionados principalmente con pruebas e integración [7].

A continuación, se presenta una lista de problemas relacionados con la calidad de software, que han sido reportados en revisiones de la literatura, casos de estudio y artículos empíricos, cuyo objetivo era el estudio y la implementación de CI, DC o CD.

- **Pruebas que consumen mucho tiempo de ejecución** [15–17]: existen muchos tipos de pruebas que pueden implementarse en CD. Sin embargo, ejecutar un gran lote de pruebas es una tarea que lleva mucho tiempo. Además, los cambios son introducidos con más frecuencia, y por eso, es necesario la ejecución de regresiones lo más rápido posible.
- **Pruebas no deterministas (Flaky Tests)** [15, 18–21]: Una prueba no determinista es aquella que podría generar un resultado positivo o negativo por la misma versión del software. Son pruebas que producen los llamados "falsos positivos" y por su inestabilidad son más conocidas como "flaky tests". Una de las características más importantes de CD es la confiabilidad, y pruebas que fallan aleatoriamente no son confiables.

- **Automatización de pruebas de interfaz gráfica de usuario** [22–25]: Las pruebas de interfaz de usuario eran ejecutadas de manera manual, pero con la aparición de herramientas como Selenium WebDriver, fue posible comenzar a automatizarlas. Sin embargo, la interfaz de usuario es la parte de la aplicación que cambia con más frecuencia y realizar pruebas sobre la misma, puede ocasionar "flaky tests".
- **Resultados de ejecución de pruebas ambiguos** [19, 26]: En entornos de desarrollo continuo, los desarrolladores deben ser notificados de cualquier defecto introducido, detectado a través de las pruebas. Cuando los resultados de la ejecución no son claros, es decir, no detallan el error, su causa, el lugar donde ocurrió, capturas de pantalla, etc., se tienen resultados ambiguos.
- **Automatización de pruebas web con contenido dinámico** [8, 25, 27]: Las aplicaciones web más modernas, utilizan tecnologías como Ajax, React o Angular, que generan contenido dinámico. Incorporar pruebas automatizadas de este tipo de tecnologías en un DP es complejo, ya que una determinada página web pudo haber alcanzado el estado esperado final, pero el contenido dinámico aún se encuentra cargando.
- **Pruebas en Big Data** [28]: Big data es el proceso de utilizar grandes conjuntos de datos que no se pueden procesar utilizando técnicas tradicionales. Realizar verificaciones sobre este conjunto de datos es un nuevo reto que involucra varias técnicas y herramientas que aún están madurando, y por lo tanto es difícil incorporarlas en CD.
- **Pruebas de datos** [29]: Los datos son muy importantes para diferentes tipos de sistemas y los errores en estos son costosos. Si bien las pruebas de software han recibido gran atención en diferentes niveles, las pruebas de datos han sido poco tenidas en cuenta. Es por ello, que no existen soluciones totales, para la implementación de pruebas de datos en un entorno de CD.
- **Pruebas en dispositivos móviles** [30]: Las pruebas móviles han traído consigo muchos desafíos. Entre ellos se encuentran: el proceso de pruebas en sí, los niveles y tipos de pruebas a considerar, los diferentes tipos de dispositivos y los costos de las pruebas automáticas. Todos estos factores dificultan la implementación de pruebas en dispositivos móviles en CD.
- **Pruebas automatizadas de requerimientos no funcionales** [16, 20]: Si bien las pruebas unitarias, de integración y funcionales, han sido estudiadas ampliamente en la literatura e implementadas por muchas organizaciones en la práctica, las pruebas no funcionales han sido poco consideradas.
- **Pruebas de aplicaciones compuestas por servicios en la nube** [31]: Actualmente, existe una gran cantidad de servicios en la nube, y el tamaño de los datos que deben manejar es muy grande. Probar flujos compuestos por estos servicios es muy complejo. Además, es necesario proporcionar garantías de calidad de servicio (QoS, Quality of Service) [32].
- **TaaS en entrega continua** [33]: Las pruebas como un servicio (TaaS), es un modelo en cual las pruebas son ejecutadas por un proveedor de servicios en lugar de un equipo de pruebas perteneciente a la misma organización que desarrolla el software. Realizar TaaS en CD, resulta una tarea muy compleja, puesto que los componentes y procesos del proveedor de servicios deben adaptarse a las características del DP de la organización contratante.

- **Pruebas de servicios web** [34–36]: Si bien existen muchas herramientas para probar este tipo de arquitecturas, es difícil integrarlas en CD.

Los problemas mencionados representan dificultades para la implementación del proceso de pruebas en CD correctamente. Además, los mismos se encuentran relacionados entre sí, de tal manera, que la ocurrencia de uno produce directamente otro, o lo puede producir. Estas relaciones se muestran en la Fig. 2.

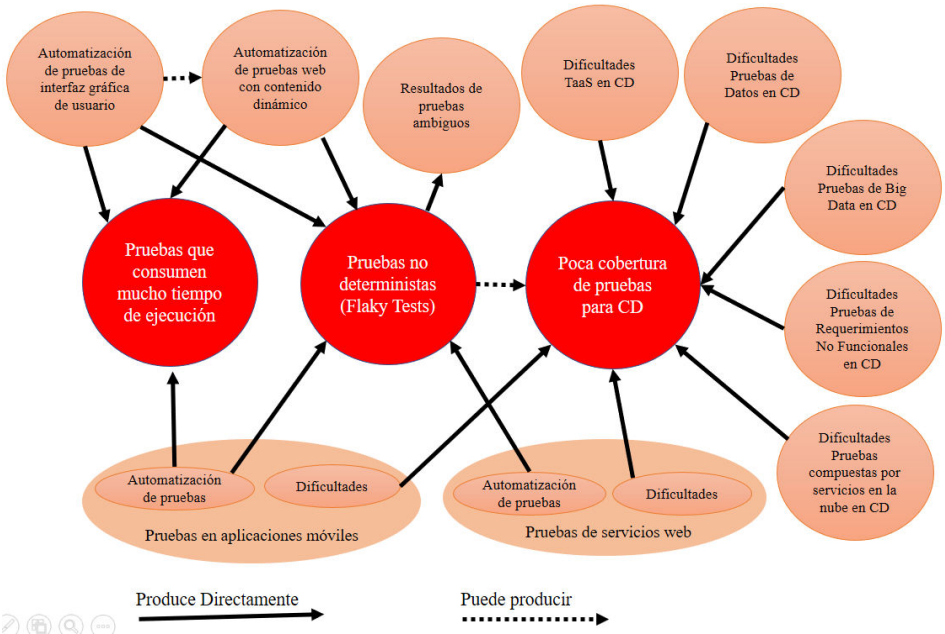

Fig. 2. Relación entre los problemas de pruebas en CD.

En la Fig. 2 puede apreciarse que los problemas más importantes son las ***pruebas que consumen mucho tiempo de ejecución*** y las ***pruebas automatizadas no deterministas***. Además, en base a los problemas relacionados con dificultades en la implementación de ciertos tipos de pruebas, se ha agregado un problema más: ***poca cobertura de pruebas en CD***. Estos factores han sido impedimentos para algunas organizaciones en la adopción de CD.

4 Pruebas Continuas

Las pruebas continuas o Continuous Testing (CT), es un término introducido por Edward Smith en el año 2000 [37], como un componente adicional de TDD en la ejecución de pruebas unitarias. Consistía en un proceso de ejecución de pruebas durante todo el día (continuamente), mientras se desarrollaba el código. Sin embargo, este concepto ha ido evolucionando con el correr del tiempo.

En al año 2003, Saff y Ernst [38], introdujeron el concepto de CT como una manera de reducir el tiempo que se pierde al ejecutar las pruebas unitarias. Para ello, entre los años 2004 y 2005 [39], proponen un plugin para el entorno de desarrollo integrado eclipse, el cual permite la ejecución de pruebas unitarias y de integración, mientras el desarrollador escribe código en su computadora. En el año 2010, este concepto se extiende a otros tipos de pruebas [40]: pruebas de especificación, pruebas de diseño, pruebas sobre el código fuente, pruebas funcionales, pruebas no funcionales, pruebas de instalación, pruebas de soporte y pruebas de mantenimiento.

Posteriormente, en el año 2011, en [31] se presenta CT para cloud computing. Los autores afirmaron que CT puede ser usado para realizar pruebas en aplicaciones SaaS. Además, como las aplicaciones están compuestas de servicios, CT puede ser llevado a cabo antes y después de la composición de estos servicios, incluso durante su ejecución como un sistema de monitoreo (Continuous Monitoring).

En el 2013, Google define CT como la "ejecución de cualquier tipo de prueba tan pronto como sea posible, para detectar cualquier tipo de defecto relacionado con un cambio realizado por el desarrollador" [41].

Entre los años 2015 y 2016, surgieron nuevos enfoques relacionados a CT. Erder y Pureur [42], consideran a CT como la utilización de técnicas de automatización para mejorar la velocidad de las pruebas mediante la integración de las etapas de desarrollo y pruebas. Para Moe et al. [43], CT consiste en probar el código fuente inmediatamente después de ser integrado en el repositorio, realizando regresiones. Virmani [44], expresa que CT consiste en automatizar cada uno de los casos de pruebas, ejecutándolos en cada integración sin intervención del usuario. Rossi et al. [45], presentan CT para desarrollo móvil, y se lo define como "el proceso de ejecutar todas las pruebas continuamente en un laboratorio de dispositivos móviles". Finalmente, Duvall et al. [46], afirman que "en un ambiente de CI, las pruebas deben ser ejecutadas continuamente".

El concepto de CT ha ido evolucionando con el correr de los años. En el comienzo, solo contemplaba la ejecución de pruebas unitarias de manera continua, especialmente en el entorno de trabajo del desarrollador. En la actualidad, no solo se aplica a las pruebas unitarias, sino a todo tipo de prueba que pueda ser automatizada (Ver Fig. 3).

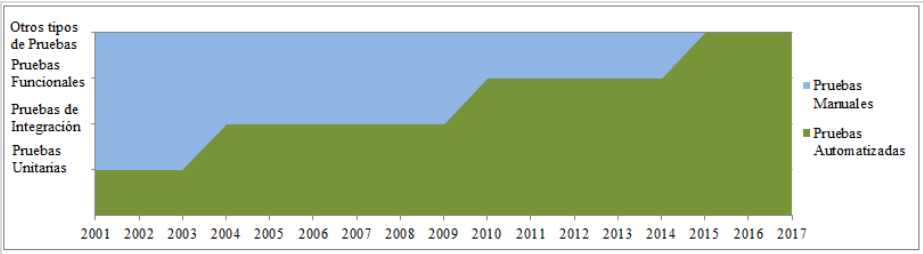

Fig. 3. Evolución de las pruebas automatizadas durante los años.

Esto podría indicar que la inclusión de las diferentes etapas de pruebas durante los años en las definiciones de CT, está relacionada con la aparición de herramientas y técnicas de automatización. Sin embargo, todas estas definiciones se basaron en el enfoque de Smith [37] y Saff y Ernst [38, 39], pero aún no existe una definición

formal para CT que englobe a todas las propuestas presentadas hasta la actualidad. De esta manera, tomando el análisis de las definiciones anteriores, podemos definir a CT como **el proceso de automatizar y ejecutar cualquier tipo de caso de prueba tan pronto como sea posible, para brindar a los desarrolladores retroalimentación rápida y así detectar defectos críticos antes de salir a producción**.

A partir de esta definición, y tomando soluciones existentes, es posible la construcción de un conjunto de procedimientos para atacar los problemas de la implementación del proceso de pruebas en entornos de CI, DC y CD.

5 Conclusiones y trabajo futuro

En este trabajo se presentó un relevamiento de una parte del estado del arte de la calidad del software en entornos ágiles, especialmente en aquellos que utilizan el enfoque de entrega continua. El mismo abarcó problemas comunes que afectan a la calidad, las relaciones que existen entre ellos, y las pruebas continuas.

Dentro de los problemas más comunes, se encuentran la ejecución de lotes de pruebas (regresiones) de larga duración, los resultados de pruebas no deterministas (flaky tests), y la baja cobertura de pruebas.

Por otro lado, diferentes autores afirman que las pruebas continuas son la clave para resolver los problemas de calidad en entrega continua. Así, se realizó un análisis de los diferentes enfoques que contemplan a las pruebas como continuas, y partiendo del mismo, se propuso una definición más concreta de pruebas continuas.

Como trabajo futuro se propone un análisis de las soluciones a estos problemas, para la elaboración de un modelo basado en las pruebas continuas. El mismo estará formado por un conjunto de procedimientos que una organización pueda seguir, para implementar el proceso de pruebas en entornos ágiles que buscan alcanzar exitosamente los enfoques de entrega continua.

Agradecimientos

Este trabajo se enmarca en dos proyectos de investigación: “Metodologías y herramientas emergentes para contribuir con la calidad del software” (PI 17F018 SCyT UNNE), y "Análisis e Implementación de tecnologías emergentes en sistemas computacionales de aplicación regional" (PI 17F017 SCyT UNNE).

Referencias

1. Cohn, M.: Succeeding with agile: Software development using scrum. Addison-Wesley Educational Publishers Inc, Upper Saddle River, NJ (2009).
2. Fowler, M.: Continuous Integration, https://martinfowler.com/articles/continuousIntegration.html.
3. Humble, J., Farley, D.: Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Pearson Professional, Upper Saddle River, NJ (2010).

4. Fowler, M.: Continuous Delivery, https://martinfowler.com/bliki/ContinuousDelivery.html.
5. Chen, L.: Continuous Delivery: Huge Benefits, but Challenges Too. IEEE Software. 32, 50–54 (2015).
6. Prusak, O.: Continuous testing: The missing link in the continuous delivery process, https://www.blazemeter.com/blog/continuous-testing-missing-link-continuous-delivery-process.
7. Laukkanen, E., Itkonen, J., Lassenius, C.: Problems, causes and solutions when adopting continuous delivery — A systematic literature review. Information and Software Technology. 82, 55–79 (2017).
8. Mascheroni, M.A., Irrazábal, E.: Continuous Testing and solutions for testing problems in Continuous Delivery: A Systematic Literature Review, (in press).
9. BlazeMeter: Continuous Testing in practice. Completing the Continuous Delivery Process, https://info.blazemeter.com/shift-left-testing, (2015).
10. Debbiche, A., Dienér, M., Svensson, R.B.: Challenges when adopting continuous integration: A case study. In: Product-Focused Software Process Improvement. pp. 17–32. Springer, Cham (2014).
11. Fitzgerald, B., Stol, K.-J.: Continuous software engineering and beyond: Trends and challenges. In: Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering. pp. 1–9. ACM, New York, NY, USA (2014).
12. Claps, G.G., Berntsson Svensson, R., Aurum, A.: On the journey to continuous deployment: Technical and social challenges along the way. Information and Software Technology. 57, 21–31 (2015).
13. Fitzgerald, B., Stol, K.-J.: Continuous software engineering: A roadmap and agenda. Journal of Systems and Software. 123, 176–189 (2017).
14. Mäntylä, M.V., Adams, B., Khomh, F., Engström, E., Petersen, K.: On rapid releases and software testing: a case study and a semi-systematic literature review. Empir Software Eng. 20, 1384–1425 (2015).
15. Neely, S., Stolt, S.: Continuous Delivery? Easy! Just change everything (well, maybe it is not that easy). In: 2013 Agile Conference. pp. 121–128 (2013).
16. Chen, L.: Continuous Delivery: Overcoming adoption challenges. Journal of Systems and Software. 128, 72–86 (2017).
17. Brooks, G.: Team pace keeping build times down. Agile Conference. pp. 294–297 (2008).
18. Laukkanen, E., Lehtinen, T.O.A., Itkonen, J., Paasivaara, M., Lassenius, C.: Bottom-up adoption of continuous delivery in a stage-gate managed software organization. In: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. pp. 45:1–45:10. ACM, New York, NY, USA (2016).
19. Cannizzo, F., Clutton, R., Ramesh, R.: Pushing the boundaries of testing and continuous integration. In: Agile 2008 Conference. pp. 501–505 (2008).
20. Leppänen, M., Mäkinen, S., Pagels, M., Eloranta, V.P., Itkonen, J., Mäntylä, M.V., Männistö, T.: The highways and country roads to continuous deployment. IEEE Software. 32, 64–72 (2015).
21. Debbiche, A., Dienér, M.: Assessing challenges of continuous integration in the context of software requirements breakdown: A case study, (2014).
22. Alégroth, E., Feldt, R., Ryrholm, L.: Visual GUI testing in practice: challenges, problems and limitations. Empir Software Eng. 20, 694–744 (2015).
23. Borjesson, E., Feldt, R.: Automated system testing using visual gui testing tools: A comparative study in industry. In: Verification and Validation 2012 IEEE Fifth International Conference on Software Testing. pp. 350–359 (2012).
24. Pradhan, L.: User interface test automation and its challenges in an industrial scenario, (2012).
25. Suwala, P.: Challenges with modern web testing, (2015).

26. Ståhl, D., Bosch, J.: Automated software integration flows in industry: A multiple-case study. In: Companion Proceedings of the 36th International Conference on Software Engineering. pp. 54–63. ACM, New York, NY, USA (2014).
27. Huizinga, D., Kolawa, A.: Automated defect prevention: Best practices in software management. John Wiley & Sons, Hoboken, N.J (2007).
28. Garg, N., Singla, S., Jangra, S.: Challenges and techniques for testing of big data. Procedia Computer Science. 85, 940–948 (2016).
29. Muşlu, K., Brun, Y., Meliou, A.: Data debugging with continuous testing. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. pp. 631–634. ACM, New York, NY, USA (2013).
30. Muccini, H., Francesco, A.D., Esposito, P.: Software testing of mobile applications: Challenges and future research directions. In: 2012 7th International Workshop on Automation of Software Test (AST). pp. 29–35 (2012).
31. Tsai, W.T., Zhong, P., Balasooriya, J., Chen, Y., Bai, X., Elston, J.: An approach for service composition and testing for cloud computing. In: 2011 Tenth International Symposium on Autonomous Decentralized Systems. pp. 631–636 (2011).
32. Barquet, L.A., Tchernykh, A., Yahyapour, R.: Performance Evaluation of Infrastructure as Service Clouds with SLA Constraints. Computación y Sistemas. 17, 401–411 (2013).
33. Tilley, S., Floss, B.: Hard problems in software testing: Solutions using testing as a service. Morgan & Claypool Publishers (2014).
34. Mascheroni, M.A., Irrazábal, E.: A Design Pattern Approach for RESTful tests: A case study. Obras colectivas en ciencias de la computación. Cali, Colombia (2018).
35. Savchenko, D.I., Radchenko, G.I., Taipale, O.: Microservices validation: Mjolnirr platform case study. 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). pp. 235–240 (2015).
36. Mascheroni, M.A., Irrazábal, E.: Framework para la creación y ejecución de pruebas automatizadas sobre servicios REST. Presented at the XXII Congreso Argentino de Ciencias de la Computación (CACIC). (2016).
37. Smith, E.: Continuous testing. Presented at the 17th International Conference on Testing Computer Software (2000).
38. Saff, D., Ernst, M.D.: Reducing wasted development time via continuous testing. 14th International Symposium on Software Reliability Engineering, pp. 281–292 (2003).
39. Saff, D., Ernst, M.D.: Continuous Testing in Eclipse. Electronic Notes in Theoretical Computer Science. 107, 103–117 (2004).
40. Burgin, M., Debnath, N.: Intelligent testing systems. In: 2010 World Automation Congress. pp. 1–6 (2010).
41. Penix, J.: Large-scale test automation in the cloud (Invited Industrial Talk). Presented at the IEEE 34th International Conference on Software Engineering (ICSE) (2012).
42. Erder, M., Pureur, P.: Continuous Architecture: Sustainable Architecture in an Agile and Cloud-Centric World. Morgan Kaufmann, Waltham, MA (2015).
43. Moe, N.B., Cruzes, D., Dybå, T., Mikkelsen, E.: Continuous software testing in a globally distributed project. In: 2015 IEEE 10th International Conference on Global Software Engineering. pp. 130–134 (2015).
44. Virmani, M.: Understanding DevOps bridging the gap from continuous integration to continuous delivery. In: Fifth International Conference on the Innovative Computing Technology (INTECH 2015). pp. 78–82 (2015).
45. Rossi, C., Shibley, E., Su, S., Beck, K., Savor, T., Stumm, M.: Continuous Deployment of Mobile Software at Facebook (Showcase). In: ACM SIGSOFT: International Symposium on the Foundations of Software Engineering (FSE 2016) (2016).
46. Duvall, P., Matyas, S., Glover, A.: Continuous integration: Improving software quality and reducing risk. Pearson Professional, Upper Saddle River, NJ (2007).