Dampak Pengujian Shift-Left terhadap Perangkat Lunak Kualitas dalam Metodologi Agile: Sebuah Studi Kasus

Kus Andriadi
Jurusan Ilmu Komputer, Program Pascasarjana BINUS - Doktor Ilmu Komputer Universitas Bina Nusantara
Jakarta, Indonesia 11480
kus.andriadi@binus.ac.id

Haryono Soeparno
Jurusan Ilmu Komputer, Program Pascasarjana BINUS - Doktor Ilmu Komputer Universitas Bina Nusantara
Jakarta, Indonesia 11480
haryono@binus.edu

Ford Lumban Gaol
Departemen Ilmu Komputer, Program Pascasarjana BINUS - Doktor Ilmu Komputer Universitas Bina Nusantara
Jakarta, Indonesia 11480
fgaol@binus.edu

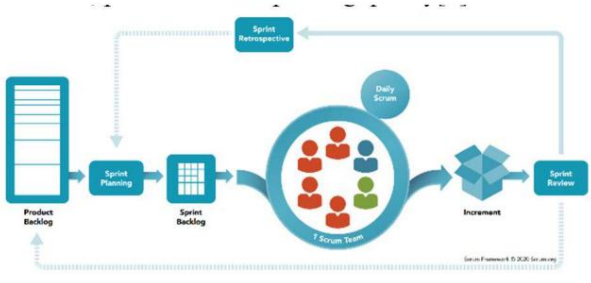
Yulyani Arifin
Jurusan Ilmu Komputer, Program Pascasarjana BINUS - Doktor Ilmu Komputer Universitas Bina Nusantara Jakarta, Indonesia 11480 yarifin@binus.edu

***Abstrak*— Inti dari Agile Methodology adalah perbaikan berkelanjutan, menghadirkan fitur kecil dengan cepat tanpa mengorbankan kualitas fitur; setiap sprint harus lebih baik dari sprint sebelumnya, dan yang lebih baik dapat berupa lebih sedikit bug, pengembangan yang lebih cepat, dan pengujian. Kami akan menyajikan cara mengurangi bug produksi dengan menyesuaikan iterasi sprint kami. Seperti yang kita ketahui, bug tidak dapat dihindari, tidak ada insinyur perangkat lunak yang dapat membuat perangkat lunak tanpa bug; namun, kita dapat mengurangi bug dalam produksi jika kita dapat menemukan bug di lingkungan yang lebih rendah sedini mungkin. Studi kasus dalam makalah ini diambil dari salah satu perusahaan teknologi di Indonesia, aktivitasnya dilakukan oleh Tim Quality Engineer (QA). Kami akan menunjukkan bahwa pengujian shift-left dapat membantu kita mengurangi bug dalam produksi. Pengujian adalah bagian dari metodologi agile, dan ide utama pengujian shift-left adalah memindahkan pengujian lebih awal dan dapat dilakukan oleh anggota tim mana pun, tidak hanya QA. Kami memasukkan pengujian shift-left dalam metodologi agile kami selama satu tahun pada tahun 2022 dan membandingkan hasilnya pada tahun sebelumnya.**

Kata kunci—kualitas perangkat lunak, pengujian perangkat lunak, agile, pengujian shift kiri.

I. PENDAHULUAN

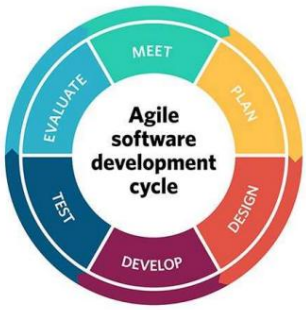
Inti dari metodologi agile adalah perbaikan berkelanjutan. Kita tidak memerlukan fitur yang signifikan dengan rilis dan penerapan yang besar; Dalam metodologi agile, kita perlu memberikan fitur kecil dengan cepat tanpa mengorbankan kualitas.
Metodologi agile diciptakan untuk mengatasi metodologi lama: Waterfall. Metodologi agile dapat meningkatkan kualitas perangkat lunak secara bertahap karena mengambil versi perangkat lunak sebelumnya sebagai pelajaran, dan membutuhkan waktu yang singkat dari pengembangan hingga rilis karena lebih seperti fitur-fitur kecil di setiap rilis. Pada akhirnya, tim akan lebih produktif, seiring dengan peningkatan kualitas[1].



Gambar 1. Alur Scrum [2]

Studi kasus dalam makalah ini diambil dari salah satu perusahaan teknologi di Indonesia. Tim QA mengumpulkan masalah dari aktivitas pengujian jangka waktu satu tahun dalam siklus pengembangan perangkat lunak perusahaan. Ini mencakup semua perangkat lunak di dalam perusahaan, jika perangkat lunak tersebut sudah digunakan untuk produksi, baik pengguna internal maupun eksternal (pelanggan).

Pengujian merupakan bagian dari metodologi agile, dan kita tahu bahwa perencanaan, perancangan, pembangunan, pengujian, dan peninjauan akan berulang di setiap sprint (lihat Gambar 2). Jadi, pengujian merupakan bagian yang tak terelakkan dari siklus pengembangan perangkat lunak [3]. Tim perlu beradaptasi dengan cepat untuk meningkatkan kepuasan p Kita akan menemukan banyak masalah selama proses berlangsung, tetapi itu tidak masalah. Kita perlu mencatatnya dan mendiskusikannya dalam sprint retrospectives, dan tim harus menemukan cara bersama untuk meningkatkannya dalam siklus pengembangan perangkat lunak berikutnya.
Bug yang berhubungan dengan logika adalah akar penyebab yang dominan. Seiring dengan berkembangnya perangkat lunak, bug semantik atau logika aka



Gambar 2. Iterasi metodologi Agile [5]

Pentingnya pengujian dalam siklus hidup pengembangan perangkat lunak adalah untuk meningkatkan keandalan, kinerja, dan faktor-faktor penting lainnya, yang dapat didefinisikan di bawah SRS (spesifikasi persyaratan perangkat lunak). Pelanggan dapat menunggu lebih lama untuk rilis perangkat lunak, tetapi mereka tidak suka bekerja dengan perangkat lunak yang cacat[6].

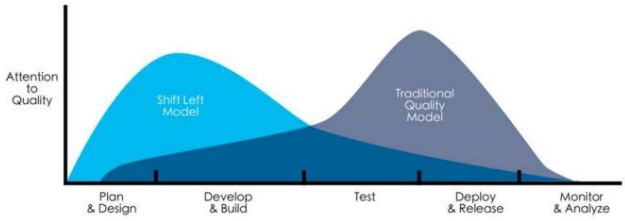
Pengujian dalam metodologi agile mencakup pengujian perangkat lunak, verifikasi, dan validasi perangkat lunak. Pengujian juga dapat diimplementasikan kapan saja dalam proses pengembangan [6]. Pada tahun 2002, bug perangkat lunak, atau kesalahan, sangat umum dan merugikan sehingga merugikan ekonomi AS sekitar $ 10 miliar.

$59,5 miliar per tahun, atau sekitar 0,6 persen dari produk domestik bruto, menurut sebuah studi yang baru dirilis yang ditugaskan oleh Institut Standar dan Teknologi Nasional (NIST) Departemen Perdagangan [7].

Dalam metodologi agile, kita memiliki sprint, sprint adalah periode waktu terbatas ketika sebuah tim bekerja untuk menyelesaikan sejumlah fitur [8]. Kita memiliki beberapa masalah dalam sprint kita, seperti kelebihan beban QA dengan pengujian di akhir sprint karena begitu banyak bug yang ditemukan setelah pengembang mengirimkan pekerjaan mereka. Tidak hanya itu, pengembang juga memiliki waktu terbatas untuk memperbaiki bug, karena waktu pengembang untuk mengembangkan fitur dalam satu sprint sudah menghabiskan lebih dari setengah timebox. Ini wajar karena kita menempatkan upaya pengujian di akhir sprint.

Kita perlu menemukan cara untuk mengurangi bug, karena jika bug terjadi, biaya yang dikeluarkan akan sangat mahal, baik biaya langsung untuk memperbaiki cacat maupun biaya tidak langsung karena berkurangnya kepuasan pelanggan, hilangnya bisnis, hilangnya peluang dan bertambahnya waktu pengembangan [9]

Untungnya, kami menemukan metode pengujian shift-left [10], pengujian shift-left adalah tentang mendorong pengujian ke tahap awal pengembangan perangkat lunak [11]. Kami menerapkan pengujian shift-left dalam satu tahun (2022) dan membandingkannya dengan tahun-tahun sebelumnya. Dalam makalah ini, kami membandingkan jumlah bug yang ditemukan di lingkungan produksi dan yang lebih rendah. Di akhir makalah ini, kami juga menyarankan strategi tambahan untuk meningkatkan pengujian shift-left.



Gbr. 3. Perbedaan bug yang ditemukan tanpa pengujian shift-kiri [12]

Pada gambar 3 di atas, kita dapat melihat bahwa dengan pengujian shift-left, kita dapat menemukan bug lebih awal. Jadi, kita dapat memperbaiki bug sesegera mungkin. Pengembang dapat membantu menguji dan menyelesaikan bug sebelum mengirimkannya ke QA.

II. PEKERJAAN TERKAIT

Penghindaran bug dapat dimaksimalkan jika pengembang menggunakan Integrated Development Editor (IDE) yang tepat saat mengembangkan perangkat lunak. IDE yang tepat berarti IDE yang dapat melengkapi, men-debug, dan menguji secara otomatis. Jika IDE dapat menyediakan pengujian template atau kelas, itu akan baik dan lebih baik untuk pengembangan perangkat lunak [13].

Tidak semua bug perlu segera diperbaiki, hal ini dapat diatasi dengan memberikan bobot pada bug [14]. Tujuannya adalah untuk mempercepat proses pengembangan dengan hanya berfokus pada bug utama. Pada akhirnya, cara ini hanya akan meninggalkan bug minor. Kita dapat memutuskan apakah akan memperbaikinya atau tidak.

Cara lain untuk mengurangi bug adalah dengan menggunakan static tools yang memindai proyek [15], tetapi ada kemungkinan tools tersebut memberikan hasil yang salah, seperti kode yang seharusnya bagus tetapi ditandai sebagai bug atau sebaliknya. Kelemahan lainnya adalah tools tersebut tidak dapat memeriksa alur bisnis, sehingga yang dapat ditemukan hanyalah bug seperti pointer null, variabel tidak diinisialisasi, kode yang tidak digunakan lagi, dan hal-hal lainnya.

Saat ini, banyak peneliti mencoba meningkatkan alat statis untuk mendapatkan hasil yang lebih baik. Seperti menggunakan pendekatan berbasis transformator untuk meningkatkan presisi analisis statis hingga 17,5% [16], mengevaluasi penemu bug [17], penilaian alat untuk keamanan kode sumber [18] atau menguji alat analisis statis untuk kerentanan kode C [19].

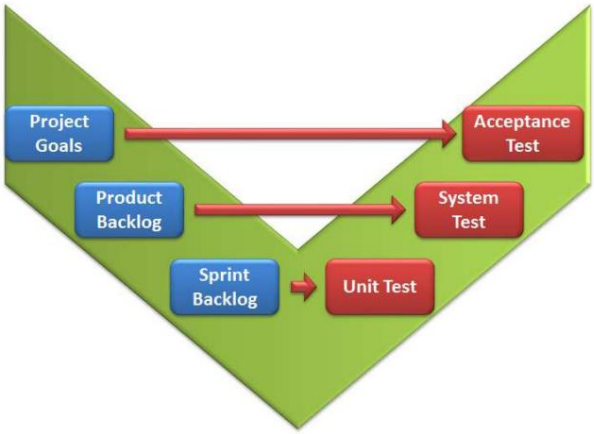
Pembelajaran mesin juga dapat memprediksi cacat perangkat lunak menggunakan model prediksi bug. Pembelajaran mesin menggunakan tiga algoritma ML yang diawasi untuk memprediksi kesalahan perangkat lunak di masa mendatang berdasarkan data historis [20]. Deteksi dan kategorisasi bug juga dapat dilakukan dengan menggunakan pendekatan penambangan data [21], dengan kategorisasi bug, tim dapat berfokus pada bug yang berdampak kritis terhadap kualitas.

Berdasarkan bagian-bagian karya terkait ini, kita mengetahui bahwa begitu banyak penelitian telah dilakukan dalam rekayasa perangkat lunak untuk mengurangi atau menghindari bug sebanyak mungkin. Bukan hanya tentang mengurangi atau menghindari, tetapi menemukan bug sedini mungkin dalam lingkungan yang lebih rendah.

Oleh karena itu, belum ada makalah yang membahas tentang efektivitas pengujian shift-kiri, dan belum ada penelitian yang dilakukan mengenai proses sprint scrum yang disesuaikan untuk menerapkan pengujian shift-kiri.

III. METODOLOGI

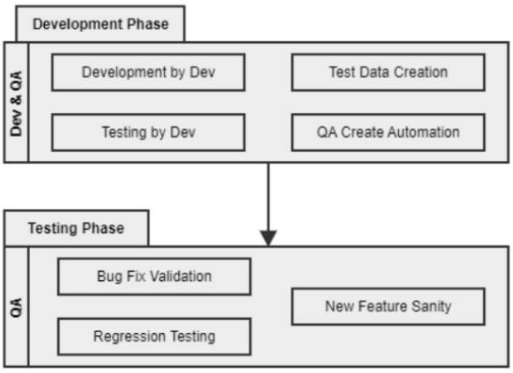
Ini adalah penelitian desain eksperimental, kami menambahkan pengujian shift-left dalam sprint scrum kami. Kami tidak mengubah alur pengujian scrum (gambar 4), jadi kami mempertahankan perencanaan sprint, rapat stand-up, dan retrospektif sprint. Yang kami ubah adalah saat Pengembang dan Penjaminan Kualitas (QA) bekerja sama dalam penyampaian fitur.



Gambar 4. Alur proses pengujian dalam Scrum [22]

Kami melakukannya selama setahun penuh pada tahun 2022, dan di akhir paper ini, kami akan membandingkan bug dengan tahun sebelumnya. Kami mengumpulkan daftar bug dari dasbor bug kami, dalam hal ini, kami menggunakan alat yang disebut Atlassian Jira. Kami juga memberikan batasan dan solusi yang memungkinkan untuk meningkatkan hasil metodologi kami. Batasannya bukan pada sisi negatif pengujian shift-left tetapi lebih pada kondisi sprint yang membuat pengiriman fitur berubah atau tertunda.

Implementasi pengujian shift-kiri kami dapat dijelaskan seperti diagram alur pada Gambar 5.



Gbr. 5. Implementasi pengujian shift-kiri kami

A. Tahap Pengembangan

Pengembang dan QA harus bekerja sama setiap hari dalam rapat tatap muka. Dalam tahap ini, komunikasi antara keduanya perlu dijaga. Aturan untuk pengembang adalah:

- **Menulis kode berdasarkan permintaan fitur. •**
  Membuat kasus pengujian untuk permintaan fitur. Pengembang dapat berdiskusi dengan QA tentang skenario pengujian.
- **Perbaikan Bug.**
- **Pengujian Fitur.**
- **Buat pengujian unit.**

Pengembang, dalam hal ini, perlu menguji fitur yang mereka kembangkan. Tidak perlu terlalu mendetail. Alur yang lancar seharusnya sudah cukup. Ini untuk memastikan bug yang muncul berkurang. Dalam fase ini, aturan untuk QA adalah:

- **Menulis kode berdasarkan permintaan fitur. •**
  Membuat pengujian otomatisasi untuk sprint sebelumnya.
- **Buat kasus uji untuk sprint saat ini. Pada tahap ini, QA perlu menjaga komunikasi dengan pengembang.**

- **Mendukung masalah produksi jika ada.**

Nomor tiga bersifat opsional; jika siklus pengembangan sudah baik dan bug telah dikurangi hingga ke tingkat minimum, masalah produksi dapat diminimalkan. Karena alur yang lancar telah dilalui oleh pengembang, QA dapat fokus pada pengujian otomasi. Ini akan menjadi hal yang baik karena, pada sprint berikutnya, mereka tidak perlu menguji lagi secara manual; mereka hanya perlu menjalankan pengujian otomasi.

B. Tahap Pengujian

Tahap ini bergantung pada tahap sebelumnya. Pada tahap sebelumnya, pengembang membantu QA memulai pengujian happy flow sehingga bug dapat ditemukan lebih awal dan diperbaiki sesegera mungkin.

QA akan menguji fitur berdasarkan persyaratan pengguna dan memastikan perilaku pengujian dari fase sebelumnya mengikuti kriteria. Dalam fase ini, QA akan menguji regresi, tidak hanya per API tetapi juga integrasi antar layanan.

Jika developer menemukan banyak bug selama pengujian fase pengembangan, hal itu akan memengaruhi fase pengujian. Jadi, itu berarti di sprint berikutnya, scrum master perlu meningkatkan komunikasi antara Developer dan QA dalam fase pengembangan. Begitu pula jika demikian, banyak bug ditemukan dalam fase pengujian, itu berarti developer tidak melakukan pengujian dengan baik dan scrum master harus menggali lebih dalam untuk menemukan masalah dalam sprint mereka. Biasanya, sisi perbaikan harus berada di developer, dan developer perlu lebih proaktif, dan inisiatif.

IV. HASIL DAN PEMBAHASAN

Kami memisahkan hasilnya menjadi tiga poin: Masalah yang ditemukan dalam produksi, masalah yang ditemukan di lingkungan yang lebih rendah, dan perbaikan terbaru Ketiga poin ini adalah parameter yang kami gunakan untuk melihat dampak pengujian shift-left pada kualitas perangkat lunak kami. Kami membandingkan masalah tahun 2022 dengan masalah tahun 2021.

Masalah yang dikumpulkan dari semua layanan yang kami miliki, jika layanan tersebut digunakan untuk produksi dan digunakan oleh pengguna, maka itu dihitung.

Kami menggunakan alat bernama Atlassian Jira dan membuat dasbor bug di sana. Saat pengembang atau QA menemukan bug, mereka harus membuat bug Jira dengan label tertentu seperti "BUGS_Q1_PROD" atau "BUGS_Q1_LOWENV".

Pengembang dan QA juga harus menentukan tingkat keparahan bug, apakah kritis, mayor, atau minor. Selama implementasi, kami tidak melihat tingkat keparahannya. Bug adalah bug, baik kritis maupun minor. Pimpinan hanya menggunakan tingkat keparahan untuk menentukan bug mana yang perlu diatasi terlebih dahulu.

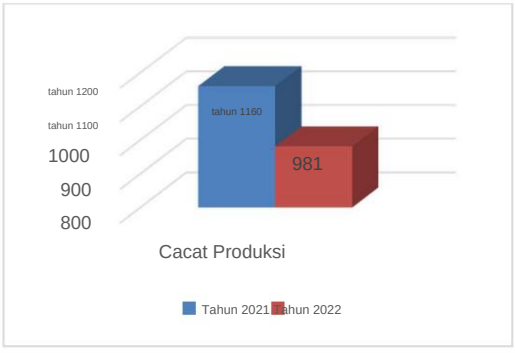
Dasbor Jira akan diperbarui secara otomatis, dan manajemen dapat melihat berapa banyak bug yang tersisa dalam produksi.
Kami bertemu dua minggu sekali untuk membahas bug yang tersisa di lingkungan produksi atau yang lebih rendah.

A. Masalah ditemukan dalam produksi.

Lingkungan produksi adalah tempat versi terbaru produk ditayangkan kepada pengguna yang dituju. [23]. Setiap cacat yang terjadi dalam produksi akan dirasakan oleh pengguna produk dan memengaruhi kepuasan pelanggan. Artinya, cacat pada fase ini harus ditangani dengan serius.

***1) Cacat produksi* Cacat**
produksi adalah cacat yang ditemukan dalam proses produksi. Cacat memiliki dampak yang besar terhadap siklus hidup pengembangan perangkat lunak, tindakan ini merupakan tindakan yang sangat mahal[24]. Perlu dicatat bahwa tidak semua cacat perlu segera diperbaiki, dan hal ini tergantung pada tingkat bug (kritis, mayor, atau minor) [25] [26].

Berdasarkan diagram pada Gambar 6, kita dapat melihat bahwa cacat produksi berkurang hingga 15%. Bug berkurang karena tim dapat menemukan bug lebih awal pada lingkungan yang lebih rendah.
Jadi, kami dapat memperbaikinya sesegera mungkin. Jadi, kami dapat meminimalkan bug yang muncul di lingkungan produksi.



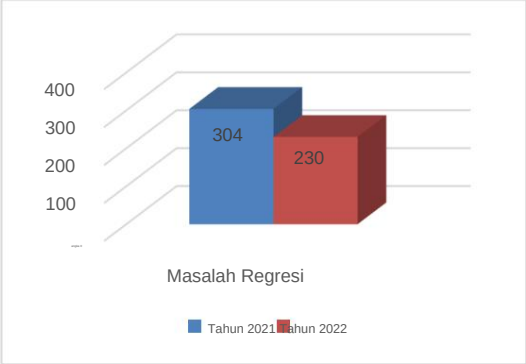
Gambar 6. Bagan cacat produksi

Cacat masih terlihat banyak, tetapi metodologi pencegahan cacat tidak selalu dapat mencegah semua cacat masuk ke aplikasi yang diuji. Pencegahan cacat mencerminkan tingkat kematangan disiplin pengujian yang tinggi, tetapi juga merupakan pengeluaran yang paling menguntungkan dari segi biaya yang terkait dengan seluruh upaya pengujian [24].

2) Masalah Regresi Masalah

regresi adalah masalah yang ditemukan setelah semua perubahan oleh pengembang dilakukan, biasanya disebut "Pengujian Regresi". Pengujian regresi adalah aktivitas pengujian yang dilakukan untuk memberikan keyakinan bahwa perubahan tidak membahayakan perilaku perangkat lunak yang ada. Rangkaian pengujian cenderung bertambah besar seiring dengan evolusi perangkat lunak[27]. QA akan memastikan bahwa perubahan awal dan perubahan terakhir tidak merusak alur bisnis [28].

Dalam pengujian regresi ini, berdasarkan gambar 7, kami dapat mengurangi masalah tersebut sebesar 24,3%



Gambar 7. Grafik masalah regresi

3) Masalah Integrasi

Pengujian integrasi dilakukan untuk memastikan semua modul perangkat lunak terintegrasi secara logis dan diuji sebagai satu kelompok [29]. Jadi, QA tidak hanya menguji perangkat lunak, tetapi juga API atau modul lain yang terhubung dengannya. Ini adalah bagian penting dari pengujian perangkat lunak, karena tidak ada gunanya jika Anda memiliki pengujian internal yang baik tetapi gagal begitu banyak saat diintegrasikan ke perangkat lunak atau modul lain.

Berdasarkan gambar 8 menunjukkan bahwa kita dapat mengurangi Masalah integrasi sebesar 57,14%

Gambar 8. Bagan masalah integrasi

B. Masalah ditemukan di lingkungan yang lebih rendah.

Sebelum melakukan produksi, kita perlu melakukan deploy di lingkungan yang lebih rendah. Pada dasarnya, lingkungan yang lebih rendah seperti lingkungan produksi tetapi terisolasi dari pengguna. [30]

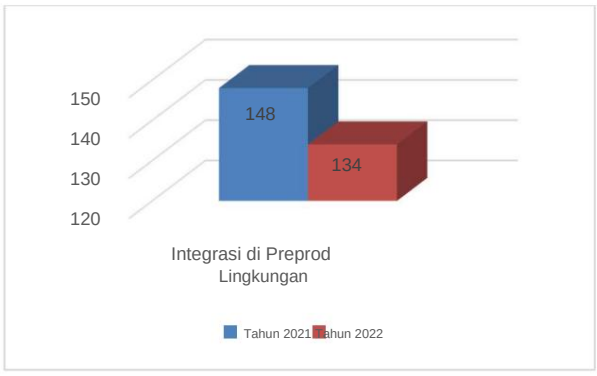
1) Lingkungan QA

Lingkungan ini dibuat untuk QA, jadi digunakan dalam fase pengujian, setelah pengembang selesai mengembangkan dan menguji fitur. Kita dapat melihat bahwa kita memiliki angka yang besar di sini, itu karena proyek besar kami untuk acara internal. Pada akhirnya, kita masih dapat mengurangi masalah sebesar 16,29%.

Gbr. 9. Masalah dalam lingkungan QA

2) Lingkungan Preprod Kita dapat mengatakan bahwa lingkungan ini meniru lingkungan produksi, kita dapat memiliki versi yang sama dan beberapa data nyata dalam preprod ini. Lingkungan ini digunakan untuk memastikan perubahan kita dalam pengembangan fitur sudah baik dan untuk mereproduksi bug dari produksi.

Berdasarkan gambar 10, kita dapat mengurangi jumlah masalah preprod sebesar 9,46%.

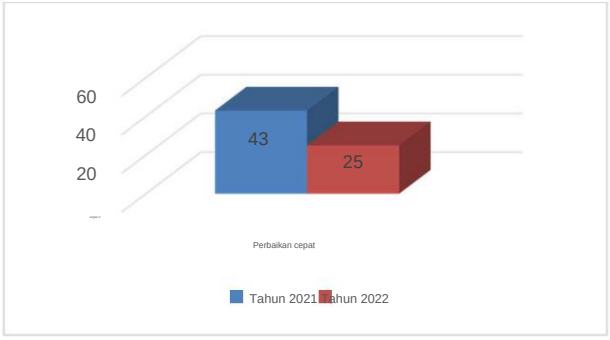


Gbr. 10. Masalah dalam lingkungan praproduksi

C. Perbaikan Cepat

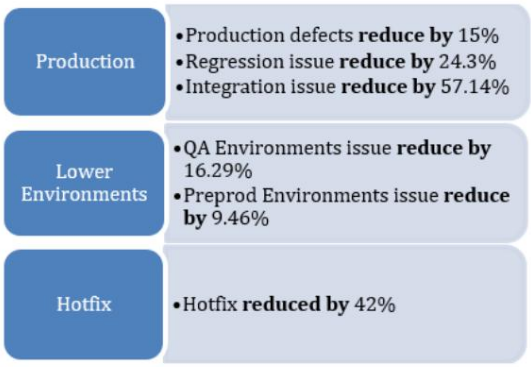
Hotfix adalah penerapan yang mendesak karena bug kritis atau besar. Hotfix dapat ditemukan oleh seseorang (bisa pengguna internal atau pengguna eksternal) yang menggunakan platform tersebut.

Seperti yang dijelaskan dalam bagan di bawah, hotfix berkurang 42% dibandingkan tahun sebelumnya. Artinya, kami dapat mengurangi bug kritis yang membuat pelanggan tidak senang saat menggunakan platform kami.



Gbr. 11. Masalah dalam lingkungan praproduksi

Ringkasan hasil pengujian shift kiri yang diterapkan dapat dilihat pada gambar 12 di bawah.



Gambar 12. Ringkasan hasil

V. KESIMPULAN DAN KARYA MASA DEPAN

Hasil studi kasus ini melengkapi kekurangan dari penelitian sebelumnya. Seperti yang kita ketahui, penelitian sebelumnya lebih banyak membahas teknis, deteksi bug, kategorisasi, tools, dan sebagainya. Hasilnya menunjukkan bahwa penerapan pengujian shift-kiri akan memberikan dampak positif pada kualitas perangkat lunak, jumlah total cacat akan berkurang secara signifikan.

Menggunakan pengujian shift-left dapat mengurangi jumlah bug yang muncul dalam produksi, tetapi kami tahu selalu ada ruang untuk perbaikan. Pengujian shift-left memerlukan kerja sama dari pengembang, QA, dan seluruh tim, termasuk scrum master dan manajer proyek. Dalam fase pengembangan, keduanya dapat membantu pengembang untuk menguji fitur. Ini akan menambah lebih banyak waktu bagi pengembang untuk membuat fitur dan memperbaiki bug.

Selain itu, QA perlu meningkatkan skrip otomatisasi mereka. Pengujian manual sangat memakan waktu, dengan pengujian otomatisasi, kita dapat menemukan bug dengan mudah. Kita dapat memeriksa apakah perubahan dalam sprint ini akan merusak perubahan dalam sprint sebelumnya atau tidak, jalankan saja otomatisasi dan kita dapat memeriksa bagian lainnya. Master Scrum perlu menyadari persyaratan ini, tim perlu memasukkan kerangka waktu bagi QA untuk membuat pengujian otomatisasi dalam perencanaan sprint.

Dalam manajemen sumber daya, pimpinan dapat menyarankan penambahan spesialisasi dalam QA, seperti QA untuk otomatisasi, QA untuk regresi, dan QA untuk mengelola masalah (atau orang).

REFERENSI

[1] A. Cockburn, "Pengembangan Perangkat Lunak Agile," Oktober 2001. Diakses: 03 Juli, [Online]. https://www.semanticscholar.org/paper/ Tersedia: Agile-Software-Development-Cockburn/ 2ca53807f49c49f82673273b8b10658c3cb032c6

[2] "Scrum", *Apa kerumunan modul?* https://www.scrum.org/resources/what-scrum-module [3] MA Jamil, M. Arif, NSA Abubakar, dan A. Ahmad, "Teknik Pengujian Perangkat Lunak: Tinjauan Literatur," dalam *Konferensi Internasional ke-6 tentang Teknologi Informasi dan Komunikasi untuk Dunia Muslim (ICT4M),* Jakarta, Indonesia: IEEE, Nov. 2016, hlm. 177–182. doi: 10.1109/ICT4M.2016.045.

[4] L. Tan, C. Liu, Z. Li, X. Wang, Y. Zhou, dan C. Zhai, "Karakteristik bug dalam perangkat lunak sumber terbuka," *Empir Software Eng,* vol. 19, no. 6, hlm. 1665–1705, Desember 2014, doi: 10.1007/s10664-013-9258-8.

[5] JR Johnivan, "Metodologi & Prinsip Pengembangan Agile Terbaik untuk 2023," *project-management.com,* 19 November 2022. https://project-management.com/agile-software-development-methodologies/ (diakses 24 Mei 2023).

[6] M. Parihar dan A. Bharti, "Pentingnya Pengujian Perangkat Lunak Dalam Siklus Hidup Pengembangan Perangkat Lunak," *Jurnal Penelitian Internasional,* Januari 2019, Diakses: 03 Juli 2023. [Online]. Tersedia: https://www.semanticscholar.org/paper/Importance-Of-Software-Pengujian-Dalam-Perangkat-Lunak-Kehidupan-Parihar-Bharti/56cc12fb0ed2d309178975e0f334cd79a9dc58b5

[7] M. Newman, "Kesalahan Perangkat Lunak Merugikan Perekonomian AS $59,5 Miliar Setiap Tahunnya," *Rilis Berita NIST,* https:// Juni. 28 tahun tahun 2002. www.abeacha.com/NIST_press_release_bugs_cost.html [8] "Scrum Sprints: Semua yang Perlu Anda Ketahui | Atlassian." https://www.atlassian.com/agile/scrum/sprints (diakses pada 24 Mei 2023).

[9] K. Beck, *Pemrograman ekstrem dijelaskan: merangkul perubahan.* Membaca, MA: Addison-Wesley, 2000.

[10] "Empat Jenis Pengujian Shift Left," 22 Maret 2015. https://insights.sei.cmu.edu/blog/four-types-of-shift-left-testing/ (diakses 24 Mei 2023).

[11] Testim, "Apa Itu Pengujian Shift Kiri? Panduan untuk Meningkatkan QA Anda," *Otomatisasi E2E berbasis AI dengan fleksibilitas seperti kode untuk pengujian Anda yang paling tangguh,* 11 Juni 2021. https://www.testim.io/blog/shift-left-testing-guide/ (diakses 17 Mei 2023).

[12] "Saat ini tahun 2017: Otomatisasi pengujian bukan lagi hal yang opsional saat membangun aplikasi seluler!," *Mobile First Cloud First,* 16 tempat 2017. https://mobilefirstcloudfirst.net/2017/01/2017-test-automation-not-optional-building-mobile-apps/ (diakses pada 23 Mei 2023).

[13] QD Soetens, P. Ebraert, dan S. Demeyer, "Menghindari bug secara proaktif dengan pemrograman berorientasi perubahan," dalam *Prosiding Lokakarya ke-1 tentang Pengujian Sistem Berorientasi Objek,* Maribor Slovenia: ACM, Juni 2010, hlm. 1–7. doi: 10.1145/1890692.1890699.

[14] J. Xuan *dkk.,* "Menuju Triase Bug yang Efektif dengan Teknik Pengurangan Data Perangkat Lunak," *IEEE Trans. Knowl. Data Eng.,* vol. 27, no. 1, hlm. 264–280, Januari 2015, doi: 10.1109/TKDE.2014.2324590.

[15] B. Johnson, Y. Song, E. Murphy-Hill, dan R. Bowdidge, "Mengapa pengembang perangkat lunak tidak menggunakan alat analisis statis untuk menemukan bug?," dalam *Konferensi Internasional ke-35 tentang Rekayasa Perangkat Lunak (ICSE) 2013,* San Francisco, CA, AS: IEEE, Mei 2013, hlm. 672–681. doi: 10.1109/ICSE.2013.6606613.

[16] A. Kharkar *et al.,* "Belajar mengurangi positif palsu dalam detektor bug analitik," dalam *Prosiding Konferensi Internasional ke-44 tentang Rekayasa Perangkat Lunak,* Pittsburgh Pennsylvania: ACM, Mei 2022, hlm. 1307–1316. doi: 10.1145/3510003.3510153.

[17] A. Delaitre, B. Stivalet, E. Fong, dan V. Okun, "Mengevaluasi Penemu Bug -- Pengujian dan Pengukuran Penganalisis Kode Statis," dalam *Lokakarya Internasional IEEE/ ACM ke-1 tahun 2015 tentang Kesalahan dan Kegagalan Kompleks dalam Sistem Perangkat Lunak Besar (COUFLESS),* Florence, Italia: IEEE, Mei 2015, hlm. 14–20. doi: 10.1109/COUFLESS.2015.10.

[18] G. Díaz dan JR Bermejo, "Analisis statis keamanan kode sumber: Penilaian alat terhadap tes SAMATE," *Teknologi Informasi dan Perangkat Lunak,* vol. 55, no. 8, hlm. 1462–1476, Agustus 2013, doi: 10.1016/j.infsof.2013.02.005.

[19] G. Chatzieleftheriou dan P. Katsaros, "Uji Coba Alat Analisis Statis untuk Mencari Kerentanan Kode C," dalam *Lokakarya Konferensi Perangkat Lunak dan Aplikasi Komputer Tahunan IEEE ke-35 tahun 2011,* Munich, Jerman: IEEE, 2011, 10.1109/COMPSACW.2011.26. Juli hal. 96–103. tempat:

[20] A. Hammouri, M. Hammad, M. Alnabhan, dan F. Alsarayrah, "Prediksi Bug Perangkat Lunak menggunakan Pendekatan Pembelajaran Mesin," *ijacsa,* vol. 9, no. 2, 2018, doi: 10.14569/IJACSA.2018.090212.

[21] D. ChandraYadav dan S. Pal, "Deteksi Bug Perangkat Lunak menggunakan Data Mining," *IJCA,* vol. 115, no. 15, hlm. 21–25, April 2015, doi: 10.5120/20228-2513.

[22] A. Abel, "Semangat untuk coding," *Uji dan Verifikasi dalam Scrum,* 12 April 2012. https://coding.abel.nu/2012/04/test-and-verification-in-scrum/

[23] "Apa itu Lingkungan Produksi?," *PagerDuty.* https://www.pagerduty.com/resources/learn/what-is-production-environment/ (diakses 24 Mei 2023).

[24] K. Ahmad dan N. Varshney, "Tentang Meminimalkan Cacat Perangkat Lunak selama Pengembangan Produk Baru Menggunakan Pendekatan Pencegahan yang Ditingkatkan," 2012. Diakses: 03 Juli 2023. [Online]. Tersedia: https://www.semanticscholar.org/paper/On-Minimizing-Software-Defects-during-New-Product-Ahmad-Varshney/22c2b51f7d839c317bfaac9b2a319cdfa924c03f

[25] N. Kanti-Singha Roy dan B. Rossi, "Menuju Peningkatan Klasifikasi Tingkat Keparahan Bug," dalam *Konferensi EUROMICRO ke-40 tahun 2014 tentang*

Rekayasa Perangkat Lunak dan Aplikasi Lanjutan, Verona: IEEE, Agustus 2014, hlm. 269–276. doi: 10.1109/SEAA.2014.51.

[26] AF Otoom, D. Al-Shdaifat, M. Hammad, dan EE Abdallah, "Prediksi tingkat keparahan bug perangkat lunak," dalam *Konferensi Internasional ke-7 tentang Sistem Informasi dan Komunikasi (ICICS) tahun 2016,* Irbid, Yordania: IEEE, April 2016, hlm. 92–95. doi: 10.1109/IACS.2016.7476092.

[27] S. Yoo dan M. Harman, "Minimalisasi, pemilihan dan penentuan prioritas pengujian regresi: sebuah survei," *Softw. Test. Verif. Reliab.,* hal. n/an/a, 2010, doi: 10.1002/stvr.430.

[28] WE Wong, JR Horgan, S. London, dan H. Agrawal, "Sebuah studi tentang pengujian regresi yang efektif dalam praktik," dalam *Prosiding The Eighth Simposium Internasional tentang Rekayasa Keandalan Perangkat Lunak,* Albuquerque, NM, AS: IEEE Comput. Soc, 1997, hlm. 264–274. doi: 10.1109/ISSRE.1997.630875.

[29] T. Hamilton, "Pengujian Integrasi: Apa itu, Jenis dengan Contoh," 11 Januari 2020. https://www.guru99.com/integration-testing.html (diakses 23 Mei 2023).

[30] "Lingkungan Pengujian: Panduan Pemula," *BrowserStack.* https://browserstack.com/guide/what-is-test-environment/ (diakses 24 Mei 2023).