iSENSE2.0: Meningkatkan Manajemen Crowdtesting yang Sadar Penyelesaian dengan Tagger Duplikat dan Pemeriksa Sanitas

JUNJIE WANG, Laboratorium Teknologi Perangkat Lunak Internet, Laboratorium Kunci Negara Ilmu Komputer, Institut Perangkat Lunak Akademi Ilmu Pengetahuan China; Akademi Ilmu Pengetahuan Universitas Tiongkok, Beijing, Tiongkok

YE YANG, Sekolah Sistem dan Perusahaan, Institut Teknologi Stevens, Hoboken, NJ, USA

TIM MENZIES, Departemen Ilmu Komputer, Universitas Negeri Carolina Utara, Raleigh, NC, AS QING WANG, Laboratorium Teknologi Perangkat Lunak Internet, Laboratorium Kunci Negara Ilmu Komputer, Institut Perangkat Lunak Akademi Ilmu Pengetahuan China; Akademi Ilmu Pengetahuan Universitas Tiongkok, Beijing, Tiongkok

ABSTRAK

Insinyur perangkat lunak mendapatkan pertanyaan tentang "berapa banyak pengujian yang cukup" secara teratur. Pendekatan yang ada dalam manajemen pengujian perangkat lunak menggunakan analisis berbasis pengalaman, risiko, atau nilai untuk memprioritaskan dan mengelola proses pengujian. Namun, sangat sedikit yang dapat diterapkan pada paradigma crowdtesting yang muncul untuk mengatasi informasi dan kontrol yang sangat terbatas atas crowdworker online yang tidak dikenal. Dalam praktiknya, memutuskan kapan harus menutup tugas crowdtesting sebagian besar dilakukan dengan menebak berdasarkan pengalaman dan sering kali menghasilkan crowdtesting yang tidak efektif. Lebih khusus lagi, ditemukan bahwa rata-rata 32% biaya pengujian adalah pengeluaran yang boros dalam praktik crowdtesting saat ini. Artikel ini bermaksud untuk mengatasi tantangan ini dengan memperkenalkan dukungan keputusan otomatis untuk memantau dan menentukan waktu yang tepat untuk menutup tugas crowdtesting.

Untuk itu, pertama-tama menyelidiki kebutuhan dan kelayakan prediksi dekat tugas crowdtesting berdasarkan dataset industri. Selanjutnya, ia mengusulkan pendekatan prediksi dekat bernama iSENSE2.0, yang menerapkan teknik sampling tambahan untuk memproses laporan crowdtesting yang tiba dalam urutan kronologis dan mengaturnya ke dalam kelompok berukuran tetap sebagai input dinamis. Kemudian, penanda duplikat menganalisis status duplikat dari laporan kerumunan yang diterima, dan penaksir penutupan berbasis CRC (Capture-ReCapture) menghasilkan keputusan penutupan berdasarkan status kedatangan bug dinamis. Selain itu, pemeriksa kewarasan berbasis cakupan dirancang untuk memperkuat stabilitas dan kinerja prediksi jarak dekat. Terakhir, evaluasi iSENSE2.0 dilakukan pada 56.920 laporan dari 306 tugas crowdtesting dari salah satu platform crowdtesting terbesar. Hasilnya menunjukkan bahwa median 100% bug dapat dideteksi dengan penghematan biaya 30%. Performa iSENSE2.0 tidak menunjukkan perbedaan yang signifikan dengan iSENSE pendekatan mutakhir, sedangkan yang terakhir bergantung pada tag duplikat, yang umumnya dianggap memakan waktu dan membosankan untuk diperoleh.

1. Pendahuluan

Crowdtesting adalah paradigma baru yang dapat meningkatkan efektivitas biaya pengujian perangkat lunak dan mempercepat prosesnya, terutama untuk aplikasi seluler [3–5, 45, 83]. Ini mempercayakan tugas pengujian kepada pekerja online yang beragam perangkat/konteks pengujian, pengalaman, dan keahliannya dapat secara signifikan berkontribusi pada hasil pengujian yang lebih andal, hemat biaya, dan efisien [4, 5]. Crowdtesting telah diadopsi oleh banyak organisasi perangkat lunak, termasuk namun tidak terbatas pada Google, Facebook, Amazon, dan Microsoft [6, 7]. Secara khusus, Google secara teratur menyebarkan crowdtesting untuk 14 lini produk utama mereka [6]. Menurut statistik terbaru dari Applause (juga dikenal sebagai uTest),1 manfaat memanfaatkan crowdtesting termasuk peningkatan rata-rata pada kapasitas pengujian sebesar 200%, jumlah rilis per tahun sebesar 150%, dan pengurangan rata-rata perbaikan kritis sebesar 50%.

Insinyur perangkat lunak mendapatkan pertanyaan tentang "berapa banyak pengujian yang cukup" secara teratur [26, 35, 41, 52]. Pengujian yang tidak memadai dapat menyebabkan kualitas perangkat lunak yang tidak memuaskan, sementara pengujian yang berlebihan dapat mengakibatkan penundaan jadwal yang potensial dan efektivitas biaya yang rendah. Hal ini terutama berlaku untuk crowdtesting, mengingat kompleksitas aplikasi seluler dan proses crowdtesting terdistribusi yang tidak dapat diprediksi. Pengujian aplikasi seluler sulit sebagian besar karena masalah kompatibilitas, karena aplikasi seluler dapat digunakan di seluruh perangkat yang memiliki sistem operasi berbeda (mis., iOS, Android), pabrikan (mis., Huawei, Samsung) dan jenis keypad (mis., virtual papan tombol, papan tombol keras) [8, 77]. Seseorang tidak dapat 100% yakin jika aplikasi yang diuji berfungsi dengan baik dengan perangkat tertentu, itu akan berjalan dengan lancar di perangkat lain.

Pendekatan yang ada dalam manajemen pengujian perangkat lunak mengadopsi analisis berbasis pengalaman, risiko, atau nilai [21, 32, 56, 60, 62, 76] untuk merencanakan dan mengelola proses pengujian secara efektif. Namun, sangat sedikit dari metode tersebut yang dapat diterapkan pada paradigma crowdtesting yang muncul di mana ada kebutuhan khusus untuk mengatasi informasi yang sangat terbatas dan kontrol atas crowdworker online yang tidak dikenal.

Dalam praktiknya, manajer proyek sangat bergantung pada pengalaman untuk memutuskan kapan harus menutup tugas crowdtesting. Strategi yang sering melibatkan penerapan kriteria seperti durasi tetap (misalnya, 5 hari) atau jumlah peserta terbatas (misalnya, menerima 400 laporan pengujian pertama). Penyelidikan kami pada data crowdtesting dunia nyata (Bagian 2.3) menunjukkan bahwa ada variasi besar dalam (a) tingkat kedatangan bug dalam tugas crowdtesting, (b) durasi tugas, dan (c) biaya yang dikonsumsi untuk mencapai tingkat kualitas yang setara . Untuk mengelola variasi besar seperti itu, manajer cenderung menggunakan ambang batas yang relatif besar untuk periode pengujian atau jumlah peserta. Ambang besar seperti itu dapat mengakibatkan proses crowdtesting yang tidak efektif, misalnya, rata-rata 32% pengeluaran yang sia-sia2 di platform crowdtesting eksperimental kami. Oleh karena itu, manajer memiliki tantangan yang signifikan dalam memutuskan kapan harus campur tangan dan menutup tugas, sehingga dapat meningkatkan efektivitas biaya crowdtesting.

Pekerjaan kami sebelumnya mengarah pada pengembangan pendekatan manajemen crowdtesting yang sadar penyelesaian bernama iSENSE [73], yang memantau kemajuan pengujian menuju penyelesaian dan memprediksi waktu penutupan tugas yang optimal, sehingga memfasilitasi praktik manajemen crowdtesting yang lebih efektif. Secara khusus, iSENSE memanfaatkan data kedatangan bug dinamis yang terkait dengan laporan crowdtesting untuk mendukung otomatisasi pengambilan keputusan. Ini dapat memberi manajer kepercayaan diri yang lebih besar untuk mencapai keuntungan efektivitas biaya dari crowdtesting, yaitu, dengan median pengurangan biaya 30%.

Namun, iSENSE memiliki satu batasan utama yang secara serius memengaruhi kepraktisan penerapan metode ini secara luas: iSENSE sangat bergantung pada pelabelan manual bug duplikat untuk memantau kemajuan pengujian berdasarkan laporan crowdtesting yang diterima secara dinamis. Proses pelabelan memakan waktu dan rawan kesalahan [69, 72, 81], terutama ketika berhadapan dengan sejumlah besar laporan di bawah konteks crowdtesting.

Untuk mengatasi keterbatasan ini, artikel ini mengusulkan dan menguji ekstensi iSENSE yang disebut iSENSE2.0. Ekstensi ini meningkatkan pekerjaan kami sebelumnya dalam dua cara penting:

- **Duplicate Tagger**, menggunakan teknik analisis semantik untuk secara otomatis menentukan dan memberi label status duplikat dari laporan crowdtesting yang masuk;
- **Sanity Checker**, mengintegrasikan mekanisme berbasis cakupan untuk mengurangi alarm palsu dalam prediksi dekat, yang berkontribusi pada penyebab utama kemacetan kinerja iSENSE.

Karena komponen baru ini memperluas dan memperkuat stabilitas dan kinerja iSENSE, versi iSENSE2.0 yang lebih baru ini menawarkan dukungan keputusan penutupan yang lebih baik untuk tugas-tugas crowdtesting.

Sisa artikel ini disusun sebagai berikut: Pertama, kami menyajikan latar belakang dan motivasi penelitian ini. Materi ini didorong oleh analisis empiris awal dan pengamatan pada kumpulan data crowdtesting industri. Ini mengarah pada kategorisasi dari tiga kurva kedatangan bug yang khas, yaitu, Naik-Tetap, Naik-Tetap-Sedikit Naik, Naik-Tetap-Naik. Kategori terakhir (yaitu, Rise-Stay-Rise) dapat menyebabkan kesalahan "alarm palsu" dengan iSENSE yang diusulkan sebelumnya.
Selanjutnya, kami menyajikan iSENSE2.0, yang menerapkan teknik sampling inkremental untuk memproses laporan pengujian massal yang tiba dalam urutan kronologis dan mengaturnya ke dalam kelompok berukuran tetap sebagai input dinamis. Penanda duplikat berdasarkan analisis semantik diintegrasikan untuk secara otomatis memberi label status duplikat laporan, dan penaksir penutupan berbasis CRC (Capture-ReCapture) kemudian digunakan untuk menghasilkan keputusan penutupan berbasis CRC berdasarkan status kedatangan bug dinamis. Sementara itu, iSENSE2.0 merancang pemeriksa kewarasan berbasis cakupan dengan mengukur cakupan istilah, yaitu, sejauh mana persyaratan tugas dicakup oleh laporan yang diterima, untuk mengatasi kemacetan kinerja di Kategori Naik-Tetap-Naik.

Evaluasi iSENSE2.0 dilakukan pada 56.920 laporan dari 306 tugas crowdtesting dari salah satu platform crowdtesting terbesar. Hasilnya menunjukkan bahwa median 100%

bug dapat dideteksi dengan penghematan 30% biaya.3 Apa yang kami tunjukkan adalah bahwa iSENSE2.0 berfungsi sebaik iSENSE, dan dengan cara yang secara signifikan meningkatkan kepraktisan pendekatan kami. Secara khusus, ini mengotomatiskan proses pelabelan duplikat yang kemudian diandalkan. Otomatisasi ini mengurangi sejumlah besar tenaga kerja yang diperlukan untuk memberi label data secara manual.
Kontribusi dari artikel ini adalah sebagai berikut:

- Selesaikan masalah besar dengan metode canggih sebelumnya dalam mengelola crowdtesting.
- Turunkan, dengan jumlah yang signifikan, biaya yang diperlukan untuk melakukan crowdtesting.
- Tunjukkan bahwa, meskipun kita secara otomatis melabeli data, kita masih dapat mempertahankan kinerja yang sama seperti metode canggih sebelumnya (yang membutuhkan lebih banyak upaya untuk komisi).

Artikel tersebut memperluas publikasi sebelumnya (disajikan pada ICSE 2019 [73]) sebagai berikut:

- Penyelidikan empiris tiga kategori kurva kedatangan bug untuk menggambarkan keterbatasan pekerjaan sebelumnya [73] dan memotivasi pendekatan ini (Bagian 2.3).

Gambar 1. Prosedur dari crowdtesting

- Pengembangan komponen Duplicate Tagger, yang memanfaatkan teknik analisis semantik untuk mengotomatiskan pelabelan duplikat laporan crowdtesting, yang dapat mengurangi upaya pelabelan manusia dan meningkatkan kepraktisannya (Bagian 3.2).
- Desain Sanity Checker berbasis cakupan untuk mengatasi kemacetan kinerja untuk tugas-tugas crowdtesting di Kategori Rise-Stay-Rise (Bagian 3.4).
- Evaluasi eksperimental pada kumpulan data yang lebih besar untuk membuktikan lebih lanjut efektivitas dan stabilitasnya (Bagian 2.2 dan Bagian 5).

2. LATAR BELAKANG DAN MOTIVASI

2.1 Latar belakang

Prosedur umum crowdtesting ditunjukkan pada Gambar 1. Secara umum, manajer menyiapkan tugas crowdtesting (termasuk perangkat lunak yang sedang diuji dan persyaratan pengujian) dan mendistribusikannya pada platform crowdtesting online tertentu. Persyaratan tugas, biasanya ditulis dalam bahasa alami, menetapkan tujuan pengujian penerbit tugas yang harus dipenuhi oleh pekerja massal.

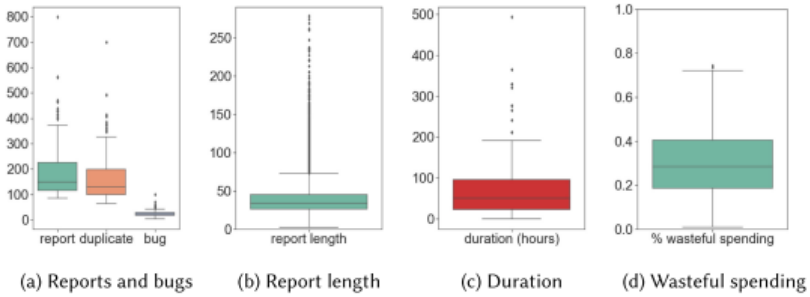
Crowdworkers dapat masuk ke tugas yang mereka minati dan mengirimkan laporan crowdtesting, biasanya meringkas input pengujian, langkah pengujian, hasil pengujian, dan sebagainya. Platform crowdtesting menerima dan mengelola laporan crowdtesting yang disampaikan oleh crowdworkers. Manajer proyek kemudian memeriksa dan memverifikasi setiap laporan untuk tugas mereka. Umumnya, setiap laporan akan dicirikan menggunakan dua atribut:

(1) apakah berisi bug4 yang valid; (2) jika ya, apakah itu bug duplikat yang sudah ada sebelumnya dilaporkan dalam laporan lain. Dalam artikel berikut, jika tidak ditentukan, ketika kami mengatakan "bug" atau "bug unik", yang kami maksud adalah laporan terkait berisi bug dan bug tersebut bukan duplikat dari yang dilaporkan sebelumnya. Dalam praktik saat ini, atribut ini ditentukan secara manual oleh manajer pengujian khusus, yang memakan waktu, membosankan, dan rawan kesalahan.

Manajer biasanya menetapkan periode tetap (misalnya, lima hari) atau jumlah peserta yang tetap (misalnya, merekrut 400 pekerja massal) untuk kriteria tugas crowdtesting yang mendekati. Jika salah satu kriteria terpenuhi terlebih dahulu, maka tugas akan ditutup secara otomatis. Ada skema pembayaran yang berbeda dalam crowdtesting, misalnya, bayar berdasarkan laporan (lihat Bagian 6.4 untuk detailnya). Umumnya, biaya tugas berkorelasi positif dengan jumlah laporan yang diterima, sehingga dengan waktu penutupan.

2.2 Kumpulan Data Baidu

Dataset eksperimental kami dikumpulkan dari platform crowdtesting Baidu,5 yang merupakan salah satu platform terbesar di China. Dataset berisi dua subset. Salah satunya mencakup semua tugas yang diselesaikan antara 1 Mei 2017 dan 1 Juli 2017, dengan semua tugas sebagai pengujian aplikasi seluler. Ini adalah dataset



Gambar2. Detail dari dari Baidu dataset

yang sama yang telah digunakan dalam penelitian kami sebelumnya pada prediksi tutup otomatis [73]. Subset lainnya mencakup tugas yang diselesaikan antara 1 Maret 2018, dan 1 April 2018, juga untuk pengujian aplikasi seluler. Secara total, ada 306 tugas crowdtesting dari berbagai domain6 dengan 56.920 laporan yang dikirimkan dan 7.613 bug yang terdeteksi. Jumlah rata-rata laporan per tugas adalah 149, dan median 86% dari semua laporan dalam tugas adalah untuk bug duplikat. Setiap laporan dijelaskan dengan median 34 istilah, dan 75% laporan dijelaskan dengan kurang dari 45 istilah. Gambar 2 memberikan distribusi jumlah laporan, jumlah bug unik, panjang deskripsi, durasi tugas, dan persentase pemborosan pengeluaran untuk membayar laporan yang tidak perlu, yang sebenarnya dapat disimpan dan memotivasi penelitian yang dilaporkan dalam artikel ini. dalam mencari dukungan penutupan tugas yang lebih efektif (lihat Bagian 2.3 untuk rinciannya).

2.3 Pengamatan dari Studi Percontohan

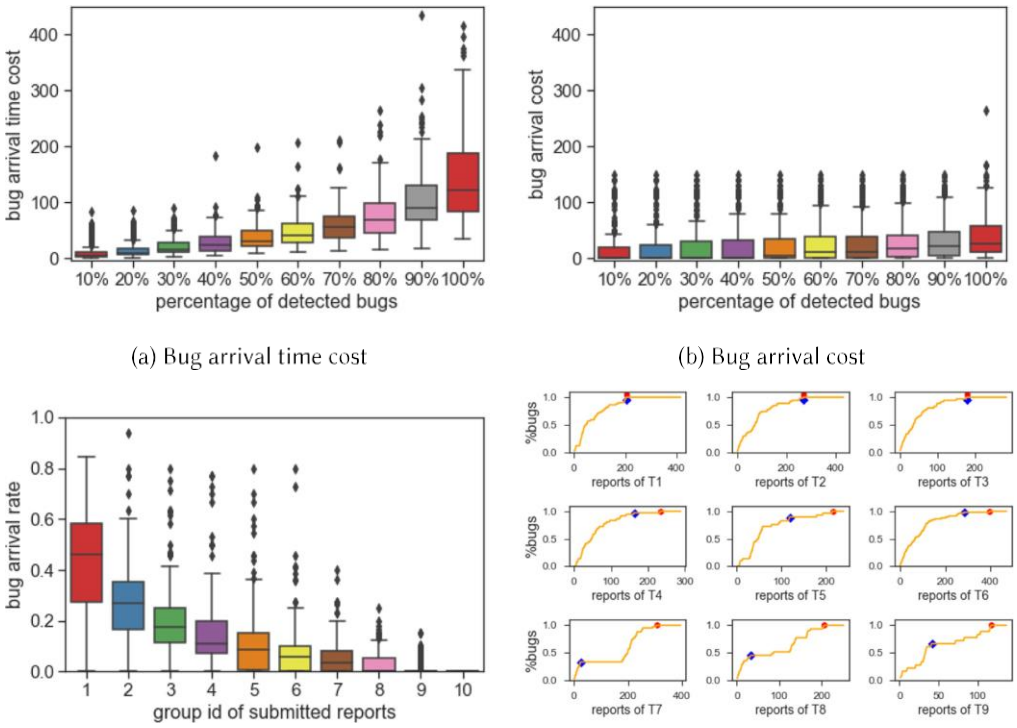
Untuk mengeksplorasi pola kedatangan bug dari crowdtesting, kami melakukan studi percontohan untuk menganalisis tiga metrik, yaitu, biaya waktu kedatangan bug, biaya kedatangan bug, dan tingkat kedatangan bug.

Untuk setiap tugas, pertama-tama kami mengidentifikasi waktu ketika bug K% telah terdeteksi, di mana K berkisar antara 10 hingga 100. Kemudian, biaya waktu kedatangan bug untuk suatu tugas dapat diturunkan menggunakan durasi (diukur dalam jam) antara pembukaannya waktu dan waktu menerima bug K%. Biaya kedatangan bug untuk suatu tugas dapat diturunkan menggunakan jumlah laporan yang dikirimkan dengan mencapai K% bug. Untuk memeriksa tingkat kedatangan bug, kami memecah laporan crowdtesting untuk setiap tugas menjadi 10 kelompok berukuran sama dalam urutan kronologis. Tarif untuk setiap grup diperoleh dengan menggunakan rasio antara jumlah bug unik yang baru dilaporkan dalam grup ini dan jumlah total laporan dalam grup. Selain itu, untuk setiap tugas crowdtesting, kami juga memeriksa persentase akumulasi bug (dilambangkan sebagai kurva kedatangan bug) untuk laporan X sebelumnya, di mana X berkisar dari 1 hingga jumlah total laporan tugas.

2.3.1 Mengkarakterisasi Pola Kedatangan Bug. Subbagian ini menyajikan pola kedatangan bug yang dicirikan dari hasil analisis.

(1) Variasi Besar dalam Biaya dan Waktu Kedatangan Bug. Gambar 3(a) dan 3(b) menunjukkan distribusi biaya waktu kedatangan bug dan biaya untuk semua tugas. Secara umum, ada variasi besar dalam biaya dan waktu kedatangan bug. Secara khusus, untuk mencapai bug K% yang sama, ada variasi besar di kedua metrik. Hal ini terutama berlaku untuk K% yang lebih besar. Misalnya, saat mendeteksi 90% bug, biaya kedatangan bug berkisar antara 3 jam hingga 149 jam dan dari 27 hingga 435 laporan.

(2) Penurunan Tingkat Kedatangan Bug dari Waktu ke Waktu. Gambar 3(c) menunjukkan tingkat kedatangan bug dari 10 kelompok yang rusak di semua tugas. Kita dapat melihat bahwa tingkat kedatangan bug menurun tajam selama



Gambar 3. Pengamatan pada data crowdtesting dunia nyata.

proses crowdtesting. Ini menandakan bahwa efektivitas biaya crowdtesting secara dramatis menurun untuk bagian akhir dari proses.

(3) Pengaruh Dataran Tinggi dari Kurva Kedatangan Bug. Gambar 3(d) menunjukkan kurva kedatangan bug tipikal untuk tugas-tugas crowdtesting. Sementara kurva kedatangan bug agak berbeda, perhatikan bahwa mereka semua menunjukkan "efek dataran tinggi" yang sama, setelah itu (yaitu, titik merah pada Gambar 3(d)) laporan baru tidak menemukan bug baru. Ini karena untuk tahap selanjutnya dari tugas crowdtesting, laporan yang dikirimkan sebagian besar berkontribusi pada duplikasi bug.

Kami mendefinisikan waktu tutup optimal sebagai waktu saat bug terakhir diterima, yaitu setelah tidak ada bug baru yang dikirimkan. Kami menganggap biaya yang dikeluarkan untuk laporan ini setelah waktu tutup yang optimal adalah pengeluaran yang sia-sia. Dalam 306 tugas eksperimental, ada rata-rata 32% pengeluaran boros. Efek dataran tinggi bersama dengan sejumlah besar pengeluaran yang sia-sia lebih lanjut menunjukkan peluang potensial dan kebutuhan praktis untuk memperkenalkan mekanisme penutupan awal (berdasarkan pengakuan dataran tinggi itu) untuk meningkatkan efektivitas biaya crowdtesting.

2.3.2 Mengkategorikan Kurva Kedatangan Bug. Kami menggunakan metode berbasis aturan dua langkah untuk mengelompokkan kurva kedatangan bug empiris ke dalam tiga kategori untuk lebih memotivasi penelitian ini.

Pada langkah pertama, setiap kurva kedatangan bug diperiksa dari awal hingga akhir untuk keberadaan interval datar dan titik lutut menurut aturan yang diturunkan secara empiris (yaitu, Aturan 1), yang menunjukkan bahwa selama interval lusinan laporan , jumlah bug unik yang terdeteksi tetap tidak berubah. Dengan kata lain, semua laporan yang diterima selama interval datar tidak melibatkan bug baru. Di detik langkah, kami

menggunakan tiga aturan lagi (yaitu, Aturan 2–4) untuk membandingkan tingkat kedatangan bug aktual dengan titik lutut dengan ambang batas yang diturunkan secara empiris untuk mengkategorikan kurva kedatangan bug ke dalam grup yang sesuai. Secara lebih spesifik, aturan tersebut dirangkum sebagai berikut:

Aturan (1) Jika kurva kedatangan bug berisi interval datar melebihi nilai yang ditentukan sebelumnya, misalnya 20, laporan bug yang sesuai dengan akhir interval diidentifikasi sebagai titik lutut kurva kedatangan bug; **Aturan (2)** Jika semua bug telah terungkap pada titik lutut yang diidentifikasi oleh Aturan 1, tugas termasuk dalam Kategori I Naik-Tetap; **Aturan (3)** Jika lebih dari 80% bug telah terungkap oleh titik lutut, tugas termasuk dalam Kategori II Naik-Tetap-Naik Sedikit; **Aturan (4)** Jika tidak, tugas tersebut termasuk dalam Kategori III Naik-Tetap-Naik.

Perhatikan bahwa angka 20 dan 80% adalah ambang batas yang diturunkan secara empiris untuk menganalisis dan menunjukkan tren, bukan untuk tujuan evaluasi studi. Hasil kategorisasi berbasis aturan menunjukkan bahwa di semua 306 tugas, proporsi (kasus) adalah 36% (110/306) milik Kategori I, 48% (148/306) milik Kategori II, dan 16% (48/306).) milik Kategori III, masing-masing. Gambar 3(d) mengilustrasikan 9 contoh tugas, 3 dari setiap kategori, dilambangkan dalam T1–T9 (Ti mewakili tugas crowdtesting i). T1, T2, dan T3 berasal dari kategori pertama Naik-Tetap. Kita dapat melihat bahwa untuk tugas-tugas ini, dengan peningkatan laporan yang dikirimkan, persentase bug yang terdeteksi pertama-tama akan meningkat tajam dan tetap tidak berubah selama bagian tugas selanjutnya. T4, T5, dan T6 berasal dari kategori kedua Naik-Tetap-Sedikit Naik. Untuk tugas kategori ini, dengan peningkatan laporan yang dikirimkan, persentase bug yang terdeteksi pertama-tama akan meningkat, tetap tidak berubah di bagian belakang tugas, dan sedikit meningkat. Kategori ketiga Rise-Stay-Rise lebih rumit, seperti yang ditunjukkan pada kurva T7, T8, dan T9. Berbeda dengan dua kategori pertama, kurva kedatangan bug di Kategori III memiliki beberapa peningkatan tajam, dihubungkan dengan interval datar di antaranya.

2.3.3 Tantangan dalam Prediksi Penutupan Tugas yang Akurat. Dalam Bagian 2.3.1, kami menyebutkan bahwa titik dataran tinggi menunjukkan waktu dekat, dan iSENSE yang kami usulkan sebelumnya [73] mencoba memprediksi waktu dekat berdasarkan titik dataran tinggi yang dikenali secara otomatis. Titik lutut dapat diperlakukan sebagai titik dataran tinggi yang dikenali secara otomatis. Dengan membandingkan antara titik lutut (yaitu, berlian biru pada Gambar 3(d)) dan titik dataran tinggi yang sebenarnya (yaitu, titik merah pada Gambar 3(d)), kami membahas kelemahan iSENSE yang diusulkan sebelumnya [73].

Untuk Kategori I, titik dataran tinggi biasanya dapat dikenali secara akurat berdasarkan tren kedatangan serangga, yaitu titik merah bertepatan dengan berlian biru. Jika platform crowdtesting dapat menutup tugas pada saat ini, sebagian besar biaya dapat dihemat tanpa mengorbankan kualitas pengujian (yaitu, jumlah bug yang terdeteksi).

Dibandingkan dengan Kategori I, tugas Kategori II menunjukkan sedikit peningkatan dalam jumlah bug setelah titik lutut yang dikenali secara otomatis (yaitu, berlian biru), yaitu, rata-rata 8,6% bug yang ditemukan setelah titik lutut dalam tugas pengujian massal eksperimental kami. Jika platform crowdtesting menutup tugas di titik lutut, tugas akan lebih hemat biaya, meskipun sebagian kecil bug tidak akan terdeteksi.

Untuk Kategori III, terdapat peningkatan jumlah bug yang signifikan setelah titik lutut yang dikenali secara otomatis, yaitu, rata-rata 35% bug ditemukan setelah titik lutut dalam tugas pengujian massal eksperimental kami. Keberadaan Kategori III dapat membawa kesalahan "alarm palsu" dengan pendekatan yang diusulkan sebelumnya iSENSE [73], karena titik lutut (yaitu, berlian biru) umumnya jauh lebih awal dari titik penutupan yang ideal (yaitu, titik merah). Selanjutnya, analisis semantik yang diadopsi dalam artikel ini berpotensi meningkatkan rasio kategori ini (lihat detail di Bagian 5.2); dengan demikian, bisa sangat menurunkan kinerja.

Singkatnya, titik puncak tugas crowdtesting di kategori pertama dan kategori kedua dapat dengan mudah didekati dan dikenali berdasarkan dinamika kedatangan bug. Sebaliknya, identifikasi titik dataran tinggi dari kategori ketiga bermasalah dan rentan terhadap alarm palsu yang

Tabel 1. Model Tangkap Ulang

		Crowdworker's detection capability	
		Identical	Different
Bug detection probability	Identical	M0 (*M0*)	Mt (*MtCH*)
	Different	Mh (*MhJK*, *MhCH*)	Mth (*Mth*)

berpengaruh negatif terhadap akurasi prediksi iSENSE. Ini memotivasi kami untuk mengembangkan strategi baru untuk mencegah alarm palsu seperti itu, untuk meningkatkan kinerja prediksi jarak dekat.

Untuk meringkas, karena ada variasi besar dalam biaya dan waktu kedatangan bug, pengambilan keputusan saat ini sebagian besar dilakukan dengan menebak-nebak. Hal ini menghasilkan efektivitas biaya yang rendah dari crowdtesting. Alternatif yang lebih efektif untuk mengelola crowdtesting adalah memantau dan memperingatkan manajer secara otomatis untuk menutup tugas w.r.t. tujuan deteksi bug yang ditentukan sebelumnya, mis., persentase tertentu dari total bug telah terdeteksi. Ini akan membantu untuk menghemat pemborosan biaya yang tidak perlu pada laporan bug yang datang nanti dan duplikat. Lebih lanjut, kurva kedatangan bug Kategori III menunjukkan bahwa kurva tersebut tidak cukup dan berisiko untuk hanya didasarkan pada dinamika kedatangan bug. Ada kebutuhan untuk menerapkan strategi tambahan untuk mengurangi alarm palsu dan memberikan keputusan penutupan yang lebih sensitif. Artikel ini bermaksud untuk mengatasi tantangan praktis ini dengan mengembangkan pendekatan baru untuk dukungan keputusan penutupan otomatis dalam manajemen crowdtesting untuk meningkatkan efektivitas biaya crowdtesting.

2.4 Model Tangkap-Tangkap Ulang

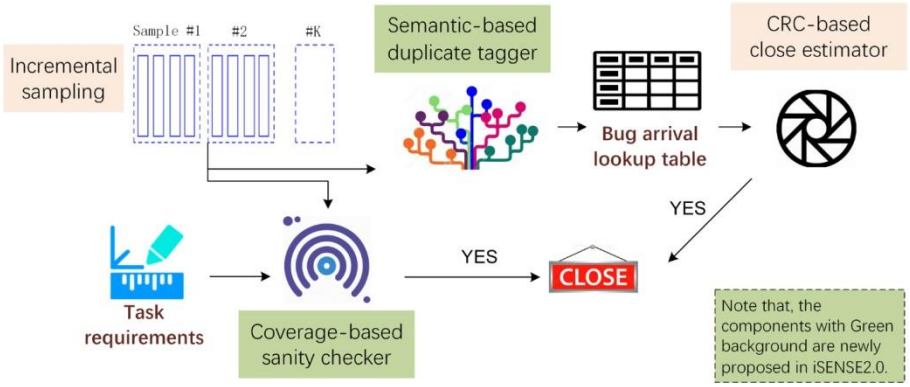
Subbagian ini memberikan latar belakang tentang model Capture-ReCapture (CRC), yang digunakan dalam Bagian 3.3. Model CRC pertama kali digunakan untuk memperkirakan ukuran populasi hewan dalam biologi [13-15, 39]. Dengan demikian, hewan ditangkap, ditandai, dan dilepaskan pada beberapa kesempatan perangkap. Jumlah hewan yang ditandai yang ditangkap kembali memungkinkan seseorang untuk memperkirakan ukuran populasi total berdasarkan tumpang tindih sampel. Ini juga telah diterapkan dalam inspeksi perangkat lunak untuk memperkirakan jumlah total bug [31, 43, 44, 59]. Model CRC yang ada dapat dikategorikan ke dalam empat jenis menurut pendeteksian bug (yaitu, identik vs berbeda) dan kemampuan crowdworkers dalam mendeteksi bug (yaitu, identik vs berbeda), seperti yang ditunjukkan pada Tabel1.

Model M0 mengandaikan semua bug dan crowdworker yang berbeda memiliki probabilitas deteksi yang sama. Model Mh mengandaikan bahwa bug memiliki probabilitas yang berbeda untuk dideteksi. Model Mt mengandaikan bahwa crowdworkers memiliki kemampuan deteksi yang berbeda. Model Mth mengandaikan probabilitas deteksi yang berbeda untuk bug dan crowdworker yang berbeda.

Berdasarkan empat model dasar CRC, berbagai estimator dikembangkan. Menurut tinjauan sistematis baru-baru ini [43], MhJK [13], MhCH [15], dan MtCH [14] adalah tiga estimator yang paling sering diselidiki dan paling efektif dalam rekayasa perangkat lunak. Selain itu, kami menyelidiki dua penduga lainnya (yaitu, M0 [39] dan Mth [40]) untuk memastikan keempat model dasar diselidiki.

3. PENDEKATAN

Gambar 4 menyajikan gambaran umum tentang iSENSE2.0. Ini berisi empat komponen utama, yaitu, sampling inkremental, tagger duplikat, estimator dekat berbasis CRC, dan pemeriksa kewarasan berbasis cakupan. Lebih khusus lagi, iSENSE2.0 terlebih dahulu memroses laporan crowdtesting menggunakan teknik sampling tambahan. Kedua, Duplikat Tagger berbasis semantik dirancang untuk mengotomatiskan pelabelan laporan pengujian duplikat; output dari komponen ini adalah tabel pencarian kedatangan bug yang merekam semua bug unik dan kemunculan duplikatnya. Ketiga, keputusan close berbasis CRC diturunkan berdasarkan tabel pencarian kedatangan bug dan estimator close berbasis CRC (Capture-ReCapture model). Keempat, Pemeriksa Sanitasi berbasis cakupan memverifikasi cakupan istilah di prediksi



Gambar 4. Ikhtisar iSENSE2.0.

waktu penutupan untuk mencegah alarm palsu yang terkait dengan fase Tetap, seperti yang dibahas dalam studi percontohan dari Bagian 2.3. Keputusan penutupan akhir dibatasi pada dua kondisi penjagaan: satu lulus keputusan penutupan berbasis CRC, dan yang lainnya lulus pemeriksaan kewarasan berbasis cakupan. Kami akan menyajikan setiap komponen secara lebih rinci di bawah ini.

3.1 Pengambilan Sampel Tambahan

Incremental sampling [50] adalah sampling komposit dan teknik pengolahan. Tujuannya adalah untuk mendapatkan sampel tunggal untuk analisis yang memiliki konsentrasi analitik yang mewakili unit keputusan. Ini meningkatkan keandalan dan pertahanan data sampling dengan mengurangi variabilitas bila dibandingkan dengan strategi pengambilan sampel diskrit konvensional.

Mempertimbangkan laporan crowdtesting yang diajukan dalam urutan kronologis (Bagian 2.1), ketika laporan smp- Size (smpSize adalah parameter input) diterima, iSENSE2.0 memperlakukannya sebagai grup perwakilan untuk mencerminkan beberapa sesi crowdtesting paralel.

3.2 Penanda Duplikat

Untuk mengatasi kelemahan bahwa iSENSE bergantung pada label duplikat manual, dalam penelitian ini, komponen Duplicate Tagger diusulkan, yang menerapkan teknik analisis semantik untuk secara otomatis mengidentifikasi status duplikat laporan crowdtesting.

3.2.1 Analisis Semantik. iSENSE2.0 menggunakan teknik penyisipan kata untuk menganalisis makna semantik dari laporan crowdtesting. Penyematan kata adalah teknik pembelajaran fitur dalam pemrosesan bahasa alami di mana kata-kata individu tidak lagi diperlakukan sebagai simbol unik, tetapi direpresentasikan sebagai vektor d-dimensi bilangan real yang menangkap makna semantik kontekstualnya [11, 48]. Perangkat lunak yang tersedia untuk umum7 digunakan untuk mendapatkan kata penyisipan laporan.

iSENSE2.0 pertama-tama melakukan pemrosesan bahasa alami dari deskripsi laporan. Setiap laporan melewati segmentasi kata standar8 dan penghapusan stopword untuk mengurangi noise, kemudian direpresentasikan sebagai sekumpulan istilah.

Dengan model penyisipan kata yang terlatih, setiap suku dapat diubah menjadi vektor d-dimensi di mana d diatur ke 100 seperti yang disarankan dalam penelitian sebelumnya [11, 48]. Sementara itu, setiap laporan ditransformasikan ke dalam matriks di mana setiap baris mewakili istilah dalam laporan dan setiap kolom mewakili dimensi penyisipan kata. Kemudian mengubah matriks laporan menjadi vektor dengan

Tabel 2. Contoh Tabel Pencarian Kedatangan Bug

	#1	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**	**#8**	**#9**	**#10**	**#11**	**#12**	…
Sample #1	1	1	1	0	0	0	0	0	0	0	0	0	
Sample #2	0	0	1	1	0	0	0	0	0	0	0	0	
Sample #3	0	0	1	0	1	0	0	0	0	0	0	0	
Sample #4	0	0	0	1	1	1	1	1	0	0	0	0	
Sample #5	0	0	1	1	0	0	0	1	1	1	1	0	
Sample #6	1	0	1	0	1	0	0	0	0	0	0	1	
Sample #7	…												

rata-rata semua vektor istilah yang dimuat dalam laporan, mengikuti pekerjaan sebelumnya [79, 80]. Secara khusus, mengingat matriks pelaporan yang memiliki total n baris, kami menyatakan baris ke-i dari matriks sebagai RI dan vektor laporan yang diubah vd dihasilkan sebagai berikut:

$$v_d = \frac{\sum_i r_i}{n}. \tag{1}$$

Dengan rumus di atas, setiap laporan pengujian kerumunan dapat direpresentasikan sebagai vektor penyisipan kata.

Untuk melatih model penyisipan kata, ini juga merayapi deskripsi tekstual dan ulasan aplikasi terkait dari toko aplikasi.9 Kemudian menggabungkan teks-teks ini dengan deskripsi semua laporan pengujian kerumunan dan menggunakannya untuk pelatihan model. Alasan mengapa kami menggunakan data ini adalah karena penelitian sebelumnya telah mengungkapkan bahwa untuk melatih model penyisipan kata yang efektif, dataset khusus domain dengan ukuran besar lebih disukai [79, 80]. Ukuran set data pelatihan kami adalah 1.160 megabita.

3.2.2 Penandaan Duplikat. Untuk laporan pengujian kerumunan baru yang akan datang dalam sampel yang diambil, tagger duplikat menghitung kesamaan kosinus antara vektor penyisipan kata dan setiap vektor penyisipan kata dari laporan sebelumnya dalam tugas ini dan memperoleh laporan sebelumnya spesifik k dengan kesamaan maksimum maxSim adalah tercapai. Jika maxSim kesamaan maksimum lebih kecil dari ambang kesamaan yang telah ditentukan (sim tiga, misalnya, 0,80), iSENSE2.0 melabelinya menggunakan tag baru (yaitu, nilai ID numerik) yang menunjukkan status duplikatnya; jika tidak, iSENSE2.0 melabeli laporan menggunakan tag laporan k.

3.2.3 Tabel Pencarian Kedatangan Bug. Selama proses pengujian massal tugas tertentu, iSENSE2.0 akan membuat dan secara dinamis memelihara tabel pencarian kedatangan bug dua dimensi untuk mencatat status duplikat dari laporan yang diterima untuk tugas tersebut.

Tabel 2 memberikan contoh ilustrasi. Setelah setiap sampel diterima, iSENSE2.0 terlebih dahulu menambahkan baris baru (misalkan baris i) pada tabel pencarian. Kemudian melewati setiap laporan yang terkandung dalam sampel ini. Jika laporan ditandai dengan tag yang sama dengan bug unik yang ada (misalkan kolom k), catat 1 pada baris i, kolom k. Jika laporan ditandai dengan tag baru, tambahkan kolom baru di tabel pencarian (misalkan kolom w), dan catat 1 di baris i, kolom w. Untuk sel kosong di baris i, isi dengan 0.

3.3 Pengukur Dekat berbasis CRC

iSENSE2.0 memperlakukan setiap sampel sebagai tangkapan (atau penangkapan kembali). Di akhir setiap pengambilan, setelah memperbarui tabel pencarian kedatangan bug, iSENSE2.0 memprediksi jumlah total bug dalam perangkat lunak[10].

Tabel 3. Arti dan Perhitungan Variabel

Var.	Meaning	Computation based on bug arrival lookup table	Example value
		Variables for computing M0	
n_1	Number of bugs in the first round	Count the number of columns with *1* in the first-half samples (i.e., samples 1–3)	5
n_2	Number of bugs in the second round	Count the number of columns with *1* in the second-half samples (i.e., samples 4–6)	11
m	Number of bugs detected in both rounds	Count the number of columns both with *1* in the first-half samples (i.e., samples 1–3) and with *1* in the second-half samples (i.e., samples 4–6)	4
		Variables for computing Mth, MhJK, MhCH, MtCH	
D	Actual number of bugs captured so far	Number of columns	12
t	Number of captures	Number of rows	6
n_j	Number of bugs detected in each capture	Number of cells with *1* in row j	3, 2, 2, 5, 6, 4
f_k	Number of bugs captured exactly k times in all captures, i.e., $\sum f_k = D$	Count the number of cells with *1* in each column and denote as r_i; f_k is the number of r_i with value k	1 = 7, 2 = 2, 3 = 2, 5 = 1
Z_p	Number of bugs captured only in the pth capture, i.e., $\sum Z_p = f_1$	For each row, examine the cell with *1*, if there are no other cells with *1* in the specific column, denote the cell as *true*, otherwise denote as *false*; Z_p is the number of cells with *true* in row p	1 = 1, 4 = 2, 5 = 3, 6 = 1
		Estimated total number of bugs	
	Estimation by M0	with n_1, n_2, m	14
	Estimation by Mth	with D, t, n_j, f_k	24
	Estimation by MhJK	with D, t, f_1	18
	Estimation by MhCH	with D, t, f_1, f_2, f_3	34
	Estimation by MtCH	with D, t, f_2, Z_p	18

berdasarkan tabel pencarian saat ini. Subbagian berikut memberikan rincian tentang bagaimana memanfaatkan lima estimator CRC yang disebutkan di atas (Bagian 2.4) untuk mendapatkan perkiraan bug.

3.3.1 Menerapkan Penaksir M0. Penduga M0 memprediksi jumlah total bug berdasarkan Persamaan (2) [39]. Tabel 3 menunjukkan arti dari setiap variabel dan bagaimana menghitung nilainya berdasarkan tabel pencarian kedatangan bug pada Tabel 2. Perhatikan bahwa kita hanya memperlakukan n1 × n2 sebagai jumlah total ketika m adalah 0

$$N_0 = \frac{n_1 \times n_2}{m} \tag{2}$$

3.3.2 Menerapkan Penaksir Mth. Penduga ke-M memprediksi jumlah total bug berdasarkan Persamaan (3) dan (4) [40]. Tabel 3 menunjukkan arti dari setiap variabel dan cara menghitung nilainya berdasarkan tabel pencarian kedatangan bug pada Tabel 2.

$$N_{th} = \frac{D}{C} + \frac{f_1}{C}\gamma^2, C = 1 - \frac{f_1}{\sum_{k=1}^{t} k f_k} \tag{3}$$

$$\gamma^2 = max\left\{\frac{\frac{D}{C}\sum_k k(k-1)f_k}{2\sum\sum_{j<k} n_j n_k} - 1, 0\right\} \tag{4}$$

3.3.3 Menerapkan Penaksir MhJK. Penduga MhJK mirip dengan metode Mth, kecuali persamaannya untuk menaksir jumlah total bug pada Persamaan (5) [13]. Tabel 3 menunjukkan arti dari masing-masing variabel, cara menghitung nilainya berdasarkan tabel pencarian kedatangan bug pada Tabel 2.

$$N = D + \frac{t-1}{t} f_1 \tag{5}$$

Perhatikan bahwa estimasi MhJK memiliki tiga ekspresi lainnya. Kami menggunakan keempat ekspresi dan memilih estimator yang tepat melalui pengujian hipotesis seperti yang disarankan dalam Referensi [13].

3.3.4 Silakan merujuk ke Referensi [13] untuk lebih jelasnya. 3.3.4 Menerapkan Penaksir MhCH. Penduga MhCH mirip dengan metode Mth, kecuali persamaannya untuk mengestimasi jumlah bug pada Persamaan (6) dan (7) [15]. Tabel 3 menunjukkan arti dari setiap variabel dan cara menghitung nilainya berdasarkan tabel pencarian kedatangan bug pada Tabel 2.

$$N = D + \frac{f_1^2}{2f_2} \tag{6}$$

Atau

$$N = D + \frac{[\frac{f_1^2}{2f_2}][1 - \frac{2f_2}{tf_1}]}{1 - \frac{3f_3}{tf_2}}, \text{ if } tf_1 > 2f_2, tf_2 > 3f_3, 3f_1f_2 > 2f_2^2 \tag{7}$$

3.3.5 Menerapkan Penaksir MtCH. Penduga MtCH juga mirip dengan metode Mth, kecuali persamaannya untuk mengestimasi jumlah total bug pada Persamaan (8) [14]. Tabel 3 menunjukkan arti dari setiap variabel dan cara menghitung nilainya berdasarkan tabel pencarian kedatangan bug pada Tabel 2.

$$N = D + \frac{\sum_{i=1}^{t} \sum_{j=i+1}^{t} Z_i Z_j}{f_2 + 1}$$

Segera setelah tugas pengujian massal dimulai, iSENSE2.0 dapat diterapkan untuk memantau kedatangan bug yang sebenarnya, terus memperbarui tabel pencarian kedatangan bug, serta terus memperkirakan jumlah total bug yang potensial. Keputusan dekat dibuat ketika jumlah total bug yang diprediksi sama dengan jumlah sebenarnya dari bug yang terdeteksi sejauh ini, dan prediksi tetap tidak berubah untuk dua penangkapan berturut-turut. Perhatikan bahwa pembatasan dua tangkapan berturut-turut adalah untuk memastikan stabilitas prediksi. Kami juga bereksperimen dengan batasan lain (yaitu, 1 hingga 5). Hasil ternyata pembatasan dengan 2 dapat memperoleh kinerja yang relatif baik dan stabil; karenanya, kami hanya menyajikan hasil ini karena keterbatasan ruang.

3.4 Pemeriksa Kewarasan Berbasis Cakupan

Ingat dari Bagian 1 bahwa salah satu batasan keputusan penutupan berbasis CRC adalah kemacetan kinerjanya terkait dengan kurva kedatangan bug Rise-Stay-Rise. Untuk mengatasinya, pemeriksa kewarasan berbasis cakupan dirancang di iSENSE2.0 untuk mengurangi kemungkinan alarm palsu dan memberikan dukungan keputusan yang lebih baik.

Cakupan kode pengujian telah lama dikenal sebagai indikator kualitas perangkat lunak. Berbagai kategori metrik cakupan kode diusulkan, misalnya, cakupan pernyataan, cakupan keputusan, cakupan kondisi, dan cakupan jalur [33, 46, 55]. Dalam konteks pengujian kerumunan, tidak mungkin untuk mendapatkan kode sumber aplikasi yang diuji karena pertimbangan rahasia. Oleh karena itu, kami mengusulkan metrik baru cakupan istilah untuk mengukur sejauh mana persyaratan tugas dieksplorasi di bawah lingkungan pengujian kerumunan. Dalam studi ini, cakupan istilah didefinisikan sebagai rasio istilah yang terdiri dari persyaratan tugas yang telah tercakup dalam laporan pengujian yang diterima. Cakupan jangka berlaku baik dalam konteks pengujian kerumunan karena sebagian besar tugas pengujian kerumunan akan memiliki persyaratan yang menentukan tujuan pengujian dan itu harus dipenuhi oleh pekerja kerumunan.

Penyaringan istilah deskriptif. Setelah pra-pemrosesan, laporan pengujian kerumunan diubah menjadi vektor istilah, seperti yang ditunjukkan pada Bagian 3.2. Seperti kebanyakan dokumen tekstual, tidak semua istilah sama pentingnya dalam laporan pengujian massal. Beberapa istilah mungkin muncul dalam sejumlah besar laporan tetapi sesuai dengan daya prediksi minimum, sementara istilah lain mungkin muncul dalam dokumen yang relatif sedikit tetapi memiliki peran penting dalam membedakan dokumen. Dalam studi ini, kami membuat daftar istilah deskriptif untuk memfasilitasi proses penyaringan yang efektif. Semua laporan pengujian kerumunan dan persyaratan tugas dari proyek historis di set pelatihan (lihat Bagian 4.3) diproses. Untuk mempermudah, kami menyebut keduanya sebagai "dokumen" dalam teks berikut. Setiap dokumen melewati prosedur pra-pemrosesan yang sama seperti yang disajikan dalam Bagian 3.2, dan setiap dokumen direpresentasikan menggunakan vektor istilah. Kami memberi peringkat istilah menurut jumlah dokumen di mana istilah muncul (yaitu, frekuensi dokumen, juga dikenal sebagai df) dan menyaring 5% istilah dengan frekuensi dokumen tertinggi dan 5% istilah dengan frekuensi dokumen terendah (yaitu, kurang istilah prediktif) mengikuti pekerjaan sebelumnya [18]. Perhatikan bahwa, karena dokumen dalam pengujian kerumunan sering pendek, istilah frekuensi (juga dikenal sebagai tf), yang merupakan metrik lain yang umum digunakan dalam pencarian informasi [61], tidak diskriminatif, jadi kami hanya menggunakan frekuensi dokumen untuk menentukan peringkat istilah . Dengan cara ini, daftar istilah deskriptif akhir terbentuk.

Cakupan istilah didefinisikan sebagai sejauh mana istilah dalam persyaratan tugas dicakup oleh laporan pengujian yang diajukan dalam tugas pengujian massal. Kami pertama-tama mewakili persyaratan tugas serta setiap laporan yang dikirimkan dalam ruang vektor dari daftar istilah deskriptif. Misalkan persyaratan tugas dilambangkan sebagai kumpulan istilah Lreq, dan misalkan kita telah menerima K laporan dan laporan ke-j dilambangkan sebagai kumpulan istilah Lptj . Jangka waktu cakupan diukur dengan menggunakan persamaan berikut:

$$term\ coverage = 1.0 - \frac{|L_{req} - \cup_j L_{pt_j}|}{|L_{req}|}, where\ j \in [1, K]. \quad (9)$$

Selama proses pengujian massal, dengan datangnya laporan pengujian kerumunan, iSENSE2.0 terus-menerus memperbarui cakupan istilah, dan keputusan yang ketat dari pemeriksa kewarasan dibuat ketika cakupan istilah melebihi covThres, yang merupakan parameter input.

4 DESAIN EKSPERIMEN

4.1 Pertanyaan Penelitian

Tiga pertanyaan penelitian dirumuskan untuk menyelidiki kinerja iSENSE2.0 yang diusulkan.

- RQ1: Seberapa efektif iSENSE2.0 dalam prediksi tutup otomatis?

Tujuan iSENSE2.0 adalah untuk memfasilitasi praktik pengujian kerumunan saat ini melalui dukungan keputusan otomatis. RQ1 adalah untuk menyelidiki efektivitas iSENSE2.0 dalam mendukung keputusan penutupan otomatis. Ingat bahwa iSENSE2.0 mengintegrasikan lima estimator CRC untuk keputusan dekat berbasis CRC; RQ1 juga menyajikan kinerja prediksi dekat dalam hal lima estimator CRC.

- RQ2: Sampai sejauh mana Pemberi Tag Duplikat dapat secara akurat melabeli laporan duplikat dalam laporan baru?

Dengan analisis semantik, iSENSE2.0 dapat secara otomatis melabeli status duplikat laporan pengujian kerumunan, sehingga mengkonsolidasikan data kedatangan bug sebagai masukan untuk penduga jarak dekat berbasis CRC. RQ2 adalah untuk menyelidiki keakuratan penanda duplikat dalam prediksi laporan duplikat. Kami juga menyajikan keakuratan kategori yang diidentifikasi dari kurva kedatangan bug, yang berasal dari status duplikat.

- RQ3: Apa pengaruh Sanity Checker dalam mengurangi alarm palsu?

Pemeriksa kewarasan berbasis cakupan diusulkan untuk mengurangi alarm palsu untuk mengatasi kemacetan kinerja dalam kategori Naik-Tetap-Naik. RQ3 adalah untuk menyelidiki pengaruh pemeriksa kewarasan dalam mengurangi alarm palsu. Kami menyelidiki lebih lanjut kontribusi pemeriksa kewarasan dalam meningkatkan kinerja prediksi dekat secara keseluruhan. Kami juga memeriksa kinerja prediksi dalam hal tiga kategori kurva kedatangan bug untuk lebih memahami kontribusinya.

4.2 Metrik Evaluasi

Kami mengukur efektivitas prediksi dekat berdasarkan dua metrik, yaitu, tingkat deteksi bug (yaitu, %bug) dan pengurangan biaya (yaitu, %reducedCost).

%bug adalah persentase bug yang terdeteksi oleh perkiraan waktu tutup. Kami memperlakukan jumlah bug yang terdeteksi secara historis sebagai jumlah total. Semakin besar % bug, semakin baik.

%reducedCost adalah persentase penghematan biaya berdasarkan perkiraan waktu tutup. Untuk mendapatkan metrik ini, pertama-tama kami memperoleh persentase laporan yang dikirimkan pada waktu dekat, di mana kami memperlakukan jumlah laporan yang dikirimkan secara historis sebagai jumlah total. Kami menganggap ini adalah persentase dari biaya yang dikonsumsi, dan %reducedCost diturunkan menggunakan 1 dikurangi persentase dari biaya yang dikonsumsi. Semakin besar %reducedCost, semakin baik.

Untuk lebih menunjukkan keunggulan pendekatan yang kami usulkan, kami melakukan uji Mann Whitney U satu sisi antara iSENSE2.0 yang kami usulkan dan alternatif lain. Kami menyertakan koreksi Bonferroni untuk melawan dampak dari beberapa tes hipotesis. Selain nilai p untuk menandakan signifikansi pengujian, kami juga menyajikan delta Cliff untuk menunjukkan ukuran efek pengujian. Kami menggunakan kriteria yang umum digunakan untuk menginterpretasikan tingkat efektivitas, yaitu Besar (0,474–

1,0), Median (0,33–0,474), Kecil (0,147–0,33), Dapat Diabaikan (−1, 0,147) (lihat detail di Referensi [17]).

Untuk mengevaluasi akurasi prediksi duplikat (RQ2), kami menggunakan recall@k mengikuti pekerjaan sebelumnya [54, 58, 70, 80]. Selain itu, dengan mempertimbangkan fakta bahwa setiap laporan memiliki beberapa duplikat, kami juga menyertakan metrik presisi@k untuk mendapatkan tampilan kinerja deteksi duplikat yang lebih komprehensif. Diberikan laporan kueri q, laporan duplikat kebenaran dasarnya menetapkan G(q) dan daftar laporan duplikat yang direkomendasikan atas-k yang dihasilkan oleh pendekatan kami R(q).

Recall@k memeriksa apakah rekomendasi top-k berguna. Definisi recall@k untuk laporan kueri q adalah sebagai berikut:

$$recall@k = \begin{cases} 1, & if\, G(q) \cap R(q) \neq \varnothing \\ 0, & Otherwise \end{cases}$$

Precision@k memeriksa proporsi laporan yang direkomendasikan dengan benar di antara rekomendasi top-k. Definisi presisi@k untuk laporan kueri q adalah sebagai berikut:

$$precision@k = \frac{|G(q) \cap R(q)|}{|R(q)|}.$$

Diberikan tugas, dengan setiap laporan bertindak sebagai laporan kueri, kami menghitung rata-rata recall@k dan presisi@k di antara semua laporan kueri untuk mendapatkan kinerja keseluruhan. Seperti pada pendekatan sebelumnya, kami menetapkan k sebagai 1, 3, 5, dan 10.

Untuk mengevaluasi peran pemeriksa kewarasan dalam mengurangi alarm palsu (RQ3), kami menyatakan falseAlarm@k (k adalah 0,60, 0,65, 0,70, 0,75, 0,80) sebagai jumlah tugas pengujian kerumunan yang %bugnya lebih kecil dari k pada waktu dekat.

4.3 Pengaturan Eksperimental

Kami menggunakan pengaturan data longitudinal yang umum digunakan [54, 68, 73]. Secara rinci, semua tugas diurutkan dalam urutan kronologis, dan kami menggunakan tugas N-1 sebelumnya sebagai set pelatihan untuk menyesuaikan parameter (lihat detail di Bagian 4.5) dan menggunakan tugas ke-N sebagai set pengujian untuk mengevaluasi kinerja iSENSE2 .0. Kami bereksperimen N dari 19 hingga 306 untuk memastikan kinerja yang relatif stabil, karena set pelatihan yang terlalu kecil akan membawa bias. Dengan cara ini, kami memperoleh kinerja dari 288 tugas pengujian (yaitu, tugas#19 hingga tugas #306).

Untuk RQ1, kami membandingkan kinerja pendekatan kami dengan dua baseline yang canggih dan umum digunakan (lihat Bagian 4.4) untuk menunjukkan keefektifannya. Untuk RQ2, kami memperoleh hasil prediksi duplikat dengan analisis semantik dan membandingkan dengan label duplikat kebenaran dasar. Kami juga memperoleh kategori kurva kedatangan bug dari setiap tugas yang diturunkan dari label duplikat yang diprediksi dan membandingkannya dengan kategori sebenarnya. Untuk RQ3, kami

membandingkan kinerja iSENSE2.0 dan variasinya tanpa pemeriksa kewarasan dan memeriksa kemunculan alarm palsu di kedua kasus, masing-masing. Perubahan dalam statistik kinerja dan alarm palsu akan digunakan untuk memahami efek dari pemeriksa kewarasan.

4.4 Garis Dasar

Untuk mengevaluasi lebih lanjut keuntungan dari iSENSE2.0 yang kami usulkan, kami membandingkannya dengan dua garis dasar.

iSENSE: Garis dasar ini adalah pendekatan mutakhir untuk prediksi dekat tugas pengujian kerumunan [73]. iSENSE mengekstrak dua atribut dari setiap laporan pengujian kerumunan yang diterima: (1) apakah laporan tersebut berisi bug yang valid; (2) apakah itu bug duplikat yang telah dilaporkan sebelumnya di laporan lain. Kemudian menerapkan model capture-recapture untuk memperkirakan jumlah total bug dari tugas pengujian kerumunan berdasarkan kedatangan dinamis bug unik bersama dengan laporan pengujian kerumunan. Ketika jumlah bug yang benar-benar terdeteksi sama dengan perkiraan jumlah total, itu menentukan waktu sebagai waktu dekat.

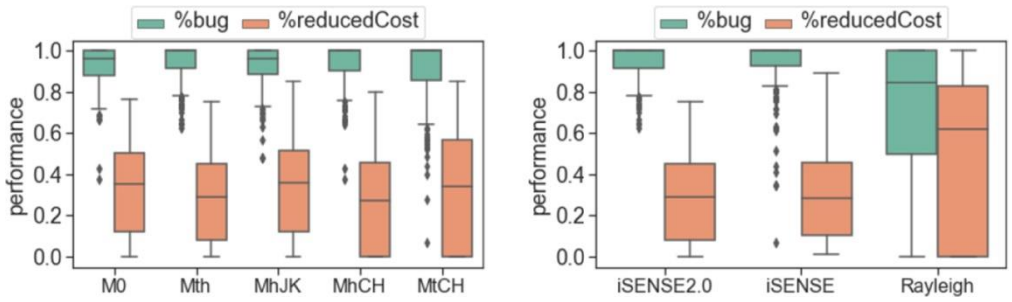
Rayleigh: Garis dasar ini diadopsi dari salah satu model paling klasik untuk memprediksi kedatangan cacat dinamis dalam pengukuran perangkat lunak. Umumnya, ini mengandaikan data kedatangan cacat mengikuti distribusi probabilitas Rayleigh [37, 57]. Dalam eksperimen ini, kami menerapkan kode agar sesuai dengan kurva Rayleigh tertentu (yaitu, model Rayleigh turunan) berdasarkan perkiraan data kedatangan bug setiap tugas, disusun menggunakan sampling tambahan, dan kemudian memprediksi total bug menggunakan model Rayleigh turunan. Ketika jumlah bug yang benar-benar terdeteksi sama dengan perkiraan jumlah total, ini menentukan titik waktu sebagai waktu tutup.

4.5 Parameter Tuning

Ada tiga parameter yang perlu disetel, yaitu size, sim three, dan covThres. Parameter pertama mewakili ukuran sampel untuk pengambilan sampel tambahan dari laporan pengujian kerumunan yang diterima secara dinamis, yaitu, berapa banyak laporan yang dipertimbangkan dalam setiap sampel (yaitu, setiap tangkapan di CRC). Parameter kedua diterapkan sebagai skor kesamaan minimal untuk dua laporan pengujian yang dianggap duplikat. Parameter terakhir mengontrol kriteria penghentian untuk pemeriksa kewarasan berbasis cakupan. Dalam studi ini, kami menyetel nilai parameter optimal berdasarkan set pelatihan (lihat Bagian 4.3) dan menerapkannya dalam set pengujian untuk mengevaluasi kinerja iSENSE2.0.

Lebih khusus lagi, kami bereksperimen dengan nilai smpSize mulai dari 6 hingga 16 dengan interval tambahan 2 di antaranya; nilai simThres dan covThres keduanya berkisar dari 0,6 hingga 1,0 dengan interval tambahan 0,1 di antaranya. Ini mengarah ke jumlah total 150 kombinasi pengaturan parameter. Untuk setiap kandidat kombinasi pengaturan parameter ini, kami memperoleh %bug dan %reducedCost berdasarkan set pelatihan (lihat Bagian 4.3). Kami kemudian memilih kombinasi nilai yang menghasilkan (1) median %bug terbesar dan median %reducedCost terbesar. Dalam kasus beberapa kandidat yang setara, kami memilih satu dengan %bug terbesar. Kami telah mencatat waktu untuk penyetelan parameter pada komputer dengan CPU Intel(R) Core(TM) i7 2,5

GHz PC dengan RAM 8 GB yang menjalankan OS Windows7 (64-bit), dalam hal set pelatihan yang berbeda, yaitu, jumlah tugas pelatihan mulai dari 18 hingga 305 (lihat Bagian 4.3). Biaya waktu untuk penyetelan parameter berkisar antara 9,4 menit hingga 53,2 menit. Namun demikian, perlu kami sampaikan bahwa pelatihan model dapat dilakukan secara offline;



Gambar 5. Efektifitas dari iSENSE2.0(RQ1)

karenanya, waktu pelatihan model hanya memberikan pengaruh kecil pada penerapan pendekatan di dunia nyata.

5 HASIL DAN ANALISIS

5.1 Jawaban untuk RQ1: Efektivitas iSENSE2.0

5.1.1 Perbandingan di Lima Penaksir CRC. Seperti yang disajikan dalam Bagian 3.3, iSENSE2.0 secara internal mengintegrasikan lima estimator berbasis CRC. Gambar 5(a) mengilustrasikan perbandingan kinerja prediksi iSENSE2.0 yang sesuai dengan masing-masing estimator CRC. Secara keseluruhan, kelima estimator dapat mencapai kinerja yang relatif tinggi dan stabil dalam prediksi dekat. Median %bug yang diperoleh kelima penduga di atas 96%, dan varians %bug dari kelima penduga ini lebih kecil dari 0,14. Selain itu, median %reducedCost yang dicapai oleh kelima estimator ini lebih besar dari 27%. Tidak ada perbedaan yang signifikan antara kinerja prediksi dekat dari estimator ini. Namun demikian, %bug median yang dicapai oleh Mth dan MhCH dapat mencapai 100%, dan varians %bug dari kedua penduga ini lebih kecil dari yang lain.

Dalam kegiatan inspeksi atau pengujian perangkat lunak tradisional, MhJK, MhCH, dan MtCH telah diakui sebagai penduga paling efektif untuk total bug [12, 29, 31, 44, 59, 66, 67]. Namun, dalam pengujian kerumunan, estimator Mth yang paling komprehensif (lihat Bagian 2.4) mengungguli estimator CRC lainnya.

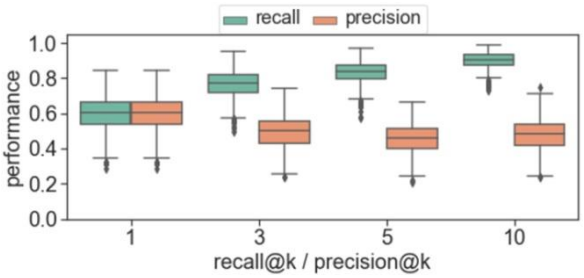
Hal ini masuk akal karena crowd testing dilakukan oleh sekelompok crowd worker yang beragam dengan berbagai tingkat kemampuan, pengalaman, perangkat pengujian, dan sifat bug dalam perangkat lunak yang diuji juga sangat bervariasi dalam hal jenis, penyebab, kesulitan deteksi, dan segera. Dalam kasus seperti itu, Mth, yang mengasumsikan probabilitas deteksi yang berbeda untuk bug dan pekerja kerumunan, dianggap sebagai estimator yang paling cocok untuk pengujian kerumunan.

Dalam eksperimen lebih lanjut, jika tidak disebutkan secara eksplisit, Mth dipilih sebagai estimator default di iSENSE2.0 karena performanya yang sedikit di atas yang lain. Akibatnya, hasil iSENSE2.0 mengacu pada yang dihasilkan dengan estimator ke-M.

Performa dengan estimator CRC terbaik. Median 100% bug dapat dideteksi dengan pengurangan biaya median 30% dengan iSENSE2.0. Hal ini menunjukkan efektivitas biaya tambahan 30% lebih bagi manajer jika dilengkapi dengan alat otomatisasi keputusan seperti iSENSE2.0 untuk memantau dan menutup tugas secara otomatis saat run-time. Pengurangan biaya adalah angka yang luar biasa ketika mempertimbangkan sejumlah besar tugas yang disampaikan dalam platform pengujian kerumunan. Selain itu, standar deviasi (0,7 pada %bug) relatif rendah, yang selanjutnya menandakan stabilitas iSENSE2.0.

Tabel 4. Hasil Uji Mann-Whitney U untuk Efektivitas (RQ1)

	%bug	%reducedCost
iSENSE2.0 vs. iSENSE	0.25 (0.05N)	0.67 (0.02N)
iSENSE2.0 vs. Rayleigh	0.00 (0.33M)	0.00 (0.23S)



Gambar 6. Ketepatan dari penanda duplikat

5.1.2 Perbandingan dengan Baseline. Gambar 5(b) menunjukkan kinerja iSENSE2.0 dibandingkan dengan dua baseline. Tabel 4 menyajikan hasil Uji Mann-Whitney antara iSENSE2.0 dan setiap baseline. Hasil menunjukkan bahwa pendekatan kami sebelumnya iSENSE juga dapat mendeteksi median 100% bug dengan 29% upaya yang disimpan, yang menunjukkan tidak ada perbedaan yang signifikan dengan iSENSE2.0 yang diusulkan dalam artikel ini. Namun, perlu dicatat bahwa iSENSE didasarkan pada label duplikat manual, sementara iSENSE2.0 sepenuhnya mengotomatiskan pelabelan duplikat, yang meringankan beban pelabelan manual, dan lebih praktis dan fleksibel. Sangat menjanjikan bahwa iSENSE2.0 dapat mencapai kinerja yang sama baiknya dengan intervensi manusia yang jauh lebih sedikit dibandingkan dengan pekerjaan yang ada.

Selain itu, varians %bug iSENSE2.0 lebih kecil daripada iSENSE (0,7 vs. 1.1). Secara khusus, %bug iSENSE2.0 minimum adalah 62,5%, sedangkan %bug minimum iSENSE hanya 6,7% dan ada 11 tugas crowdtesting yang %bug dengan iSENSE kurang dari 62,5%. Pencilan ini berpotensi disebabkan oleh kurva kedatangan bug "Naik-Tetap-Naik". Karena kami telah merancang pemeriksaan kewarasan berbasis cakupan di iSENSE2.0 untuk membantu mengatasi kemacetan kinerja yang disebabkan oleh kategori kurva kedatangan bug ini, ada lebih sedikit tugas crowdtesting dengan %bug yang cukup rendah dibandingkan dengan iSENSE.

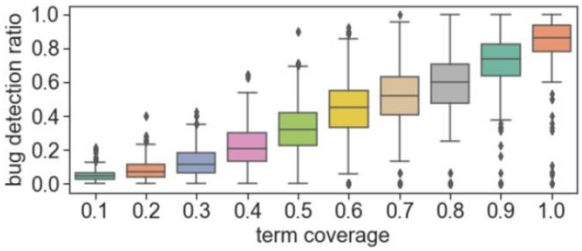
Selain itu, iSENSE2.0 secara signifikan mengungguli Rayleigh, yang didasarkan pada model kedatangan cacat tradisional. Distribusi probabilitas Rayleigh telah terbukti efektif dalam memodelkan kedatangan bug dalam pengujian perangkat lunak tradisional. Hasil di atas menyiratkan bahwa itu tidak cocok untuk crowdtesting. Ini mungkin karena proses deteksi bug dari tugas crowdtesting lebih terbuka dan rumit daripada proses pengujian perangkat lunak tradisional.

5.2 Jawaban untuk RQ2: Akurasi Tagger Duplikat

Gambar 6 menunjukkan hasil recall@k dan precision@k (k = 1, 3, 5, dan 10, masing-masing) dari tagger duplikat dalam prediksi laporan duplikat. Recall@1 adalah 0,29–0,84 di semua tugas eksperimental, dan median recall@1 adalah 0,61. Presisi median@k berkisar dari 0,47 hingga 0,51 ketika k adalah 3, 5, dan 10, menunjukkan presisi duplikat yang diprediksi hampir tidak berubah mengikuti keseluruhan peringkat.

Tabel 5. Hasil Identifikasi Kategori Bug Arrival Curve (RQ2)

		Ground Truth			
		Category I ***Rise-Stay***	***Category II*** ***Rise-Stay-Slight Rise***	***Category III*** ***Rise-Stay-Rise***	***Total***
Predicted	***Category I***	54 (19%)	23 (8%)	4 (1%)	81 (28%)
	Category II	22 (7%)	85 (29%)	15 (6%)	122 (42%)
	Category III	29 (10%)	36 (13%)	20 (7%)	85 (30%)
	Total	105 (36%)	144 (50%)	39 (14%)	



Gambar 7. Hubungan antara cakupan istilah dan deteksi bug (RQ3).

Dalam studi identifikasi duplikat yang ada, recall@1 dilaporkan sebagai 0,16–0,19 dalam Referensi [80], 0,36 dalam Referensi [58], 0,37–0,42 dalam Referensi [64], 0,42–0,57 dalam Referensi [54], 0,44– 0,79 dalam Referensi [70], dan seterusnya. Mempertimbangkan hasil yang ada di atas, hasil kami mendukung bahwa tagger duplikat yang diusulkan, yaitu, dengan median 61% untuk recall@1, umumnya mengungguli penelitian yang ada. Meskipun hasil yang diperoleh berdasarkan kumpulan data yang berbeda, namun dianggap memberikan kinerja yang memuaskan dalam mengotomatisasi bug duplikat dalam konteks penelitian ini.

Untuk memahami lebih lanjut kinerja pembuat tag duplikat, satu pertanyaan lagi adalah apakah/bagaimana Tagger Duplikat yang diperkenalkan berdampak pada bentuk kurva kedatangan bug. Untuk menjawab pertanyaan ini, kami mereproduksi hasil kategorisasi berbasis aturan (detail di Bagian 2.3.2) menggunakan label berpredikat dari Duplikat Tagger, alih-alih menggunakan label sebenarnya. Tabel 5 merangkum perbandingan antara kategorisasi kebenaran dasar dan yang menggunakan prediksi penanda duplikat. Perhatikan bahwa angka pada Tabel 5 diperoleh berdasarkan 288 tugas pengujian di bawah pengaturan eksperimental (lihat Bagian 4.3), tidak semua 306 tugas. Oleh karena itu, ada sedikit perbedaan antara distribusi kategoris seperti yang ditunjukkan pada baris terakhir dan distribusi kebenaran dasar yang diperkenalkan di Bagian 2.3.2, yang dihasilkan pada semua 306 tugas.

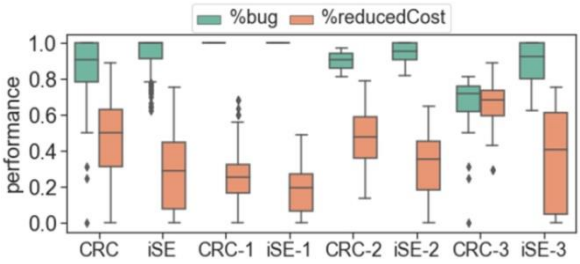
Secara keseluruhan, dengan menjumlahkan angka dalam sel diagonal, kita dapat menemukan bahwa 55% (19%+29%+7%) tugas eksperimental dapat dikategorikan dengan benar menggunakan label dari Duplicate Tagger. Kami mengakui bahwa ada sebagian besar (45%) tugas eksperimental yang dikategorikan secara tidak akurat, dan Duplicate Tagger cenderung menghasilkan lebih banyak tugas di Kategori III Naik-Tetap-Naik (30% vs. 14%) dibandingkan dengan kebenaran dasar. Hal ini memotivasi kami untuk merancang pemeriksa kesehatan berbasis cakupan untuk mengatasi kemacetan kinerja yang terkait dengan Kategori III.

5.3 Jawaban untuk RQ3: Pengaruh Pemeriksa Kewarasan

Karena dasar dari sanity checker adalah evaluasi kecukupan pengujian berdasarkan cakupan istilah, pertama-tama kami menyajikan hubungan antara cakupan istilah dan rasio deteksi bug (yaitu, persentase bug yang terdeteksi di antara semua bug) pada Gambar 7. Kita dapat melihat bahwa bug rasio deteksi berkorelasi positif dengan cakupan istilah, yang selanjutnya membuktikan rasionalitas istilah yang kami rancang

Tabel 6. Pengaruh Sanity Checker dalam Mengurangi False Alarm (RQ3)

	falseAlarm@0.60	falseAlarm@0.65	falseAlarm@0.70	falseAlarm@0.75	falseAlarm@0.80
CRC	26 (9%)	39 (14%)	52 (18%)	70 (24%)	88 (31%)
iSENSE2.0	0 (0%)	3 (1%)	4 (1%)	11 (4%)	21 (7%)



Gambar 8. Kontribusi pemeriksa kesehatan berbasis cakupan (RQ3)

	%bug	%reducedCost
CRC vs. iSENSE2.0	0.000 (0.36M)	0.000 (0.44M)
CRC vs. iSENSE2.0 in *Category I Rise-Stay*	1.000 (0.00N)	0.000 (0.26S)
CRC vs. iSENSE2.0 in *Category II Rise-Stay-Slight Rise*	0.000 (0.48L)	0.000 (0.43M)
CRC vs. iSENSE2.0 in *Category III Rise-Stay-Rise*	0.000 (0.81L)	0.000 (0.67L)

Tabel 7. Hasil Uji Mann-Whitney U untuk Kontribusi

Pemeriksa Sanity Berbasis Cakupan (RQ3)

pemeriksa kewarasan berbasis cakupan. Ketika semua persyaratan pengujian tercakup (yaitu, cakupan istilah adalah 100%), rata-rata 85% bug terdeteksi. Ini menunjukkan bahwa hanya menggunakan istilah cakupan tidak akan efektif untuk prediksi dekat, dan apa yang kami lakukan adalah mengintegrasikannya sebagai pemeriksa kewarasan di iSENSE2.0 kami.

Tabel 6 menyajikan jumlah dan persentase falseAlarm@k (k adalah 0.60, 0.65, 0.70, 0.75, 0.80), masing-masing, untuk iSENSE2.0 dan variasinya tanpa menerapkan pemeriksa kesehatan berbasis cakupan (yaitu, hanya dengan CRC berbasis close estimator), dilambangkan sebagai CRC pada Tabel 6. Kita dapat dengan mudah melihat bahwa sanity checker berbasis cakupan dapat mengurangi alarm palsu dalam prediksi dekat. Khususnya, tanpa sanity checker, ada 26 (9%) tugas eksperimental yang %bug-nya kurang dari 60%, dan angka ini nol jika dilengkapi dengan sanity checker. Tanpa sanity checker, ada 52 (18%) tugas eksperimental yang %bug-nya kurang dari 70%, dan angka ini 4 (1%) jika dilengkapi dengan sanity checker.

Selanjutnya, Gambar 8 menggambarkan perbandingan antara kinerja CRC dan iSENSE2.0, serta tiga variasinya. CRC-x dan iSE-x, masing-masing, menunjukkan kinerja CRC atau iSENSE2.0 untuk tugas eksperimental dari kategori prediksi x kurva kedatangan bug. Tabel 7 menyajikan hasil Uji Mann-Whitney antara keduanya.

Mari kita lihat dua kelompok kotak pertama, yaitu, CRC dan iSE. Jelas bahwa pemeriksa kewarasan berbasis cakupan dapat meningkatkan (atau setidaknya mempertahankan) persentase bug yang terdeteksi pada waktu tutup yang diprediksi. Meskipun berpotensi menunda waktu penutupan, yaitu dengan pengurangan biaya yang lebih sedikit, bahkan untuk tugas yang telah mendeteksi 100% bug. Ini dianggap dapat diterima karena dua alasan berikut: (1) jumlah bug yang terdeteksi lebih disukai daripada pengurangan

biaya; (2) bahkan dengan pemeriksa kewarasan, median pengurangan biaya 30% masih dapat dicapai.

Kami kemudian mengalihkan fokus kami pada kinerja di tiga kategori, yaitu, enam seri terakhir dari tiga kotak pada Gambar 8. Secara khusus, karena pemeriksa kewarasan dirancang khusus untuk meningkatkan masalah kinerja yang terkait dengan Kategori III Rise-Stay-Rise , kita dapat mengamati peningkatan besar dalam %bug median dari 0,71 (yaitu, CRC-3) menjadi 0,92 (yaitu, iSE-3). Hal ini menunjukkan, tanpa sanity checker, hanya median dari 71% bug yang dapat dideteksi hanya dengan model CRC; dan dengan pemeriksaan kewarasan, rata-rata 92% bug dapat dideteksi dengan iSENSE2.0 untuk tugas-tugas di Kategori III. Ini adalah bukti nyata tentang peningkatan yang dikaitkan dengan pemeriksa kewarasan.

Seperti yang diharapkan, efek sanity checker tidak begitu signifikan pada dua kategori lainnya seperti pada Kategori III. Untuk Kategori I Rise-Stay, kita dapat mengamati bahwa baik iSENSE2.0 dan CRC dapat mencapai %bug sebesar 1.00; dan CRC dapat menghemat lebih banyak upaya daripada iSENSE2.0. Seperti yang telah dibahas sebelumnya, waktu tutup yang optimal dapat ditentukan hanya berdasarkan kedatangan bug yang dinamis untuk kategori ini (lihat Bagian 2.3.2 untuk detailnya). Oleh karena itu, iSENSE2.0, yang mencakup pemeriksaan kewarasan, memberlakukan pembatasan tambahan dan berpotensi menunda waktu penutupan, yang menyebabkan lebih banyak pengeluaran biaya untuk membayar laporan duplikat. Untuk Kategori II Naik-Tetap-Sedikit Naik, pemeriksa kewarasan dapat sedikit meningkatkan tingkat deteksi bug, yaitu, dari 0,90 menjadi 0,95, dikompromikan dengan pengurangan biaya yang lebih rendah. Peningkatan % bug dapat mengurangi risiko pengujian yang tidak memadai. Berdasarkan umpan balik dari manajer pengujian, mendeteksi lebih banyak bug lebih menguntungkan daripada lebih banyak pengurangan biaya. Untuk praktisi crowdtesting, %bug dianggap sebagai tujuan yang sulit, sedangkan %reducedCost lebih seperti bonus tambahan.

6 PEMBAHASAN

6.1 Pengaruh Analisis Semantik Terhadap Kinerja

Pendekatan yang kami usulkan didasarkan pada status duplikat yang berasal dari analisis semantik laporan crowdtesting. Teknik deteksi duplikat yang canggih tidak dapat mencapai akurasi 100%, begitu pula dengan pendekatan kami. Meskipun demikian, prediksi dekat kami masih dapat mencapai median 100% bug dan 30% pengurangan biaya, yang tidak memberikan perbedaan signifikan dengan pekerjaan sebelumnya yang didasarkan pada tag duplikat berlabel manusia. Ini lebih lanjut menyiratkan pendekatan yang kami usulkan tahan terhadap kebisingan.

Kami mengakui bahwa berdasarkan tag duplikat berlabel otomatis, iSENSE2.0 tidak dapat memberikan prediksi akurat tentang biaya yang diperlukan untuk analisis tradeoff-close seperti iSENSE [73], karena total bug yang diprediksi memiliki kesalahan besar. Ketika dipersenjatai dengan teknik deteksi duplikat yang lebih akurat, akurasi prediksi ini dapat ditingkatkan dan kegunaan iSENSE2.0 dapat lebih ditingkatkan.

Saat ini, peneliti menggabungkan screenshot dan deskripsi tekstual untuk mendeteksi duplikat laporan crowdtesting [70] yang dapat mencapai kinerja yang lebih baik daripada hanya berdasarkan deskripsi tekstual. Teknik-teknik baru ini akan diselidiki dan diintegrasikan ke dalam iSENSE2.0 untuk lebih meningkatkan efektivitasnya.

6.2 Eksplorasi Lebih Lanjut Pemeriksa Kewarasan

Karena tahap "Tetap" dalam kurva kedatangan bug akan menyesatkan prediksi berbasis CRC untuk menentukannya sebagai waktu dekat, kami mencoba menggunakan asumsi ortogonal (yaitu, pemeriksaan kewarasan berbasis cakupan) selain tren kedatangan bug untuk membantu mengatasi kemacetan kinerja. Hasil di Bagian 5.3 menunjukkan bahwa bermanfaat untuk menggunakan pemeriksa kewarasan berdasarkan cakupan istilah, yang berpotensi dapat mematahkan fase datar dari kurva kedatangan bug dan secara signifikan meningkatkan tingkat deteksi bug pada waktu dekat yang disarankan. Namun, kinerja dalam kategori Rise-Stay menyiratkan bahwa hal itu juga berpotensi menurunkan efektivitas biaya crowdtesting, yaitu menunda waktu tutup. Oleh karena itu, eksplorasi lebih lanjut diperlukan untuk merancang pemeriksa kewarasan dengan hati-hati untuk lebih menurunkan biaya yang tidak perlu. Kemungkinan perbaikan adalah bereksperimen pada pengukuran baru untuk istilah cakupan atau teknik penyaringan istilah lainnya saat mengukur cakupan istilah. Alternatif lain adalah menggunakan informasi dari paket instalasi, selain persyaratan tugas, sebagai indikator dalam pemeriksa kesehatan.

Kemungkinan lain termasuk mengembangkan prediktor atau pengklasifikasi berbasis pembelajaran mesin untuk mengkategorikan tren kedatangan bug terlebih dahulu dan secara otomatis menerapkan metode yang paling sesuai dalam menanggapi tren kedatangan tertentu. Misalnya, jika bug muncul setelah Kategori I dan Kategori II, kerangka kerja dapat secara otomatis menerapkan metode berbasis CRC untuk memprediksi waktu tutup. Namun, jika bug muncul setelah Kategori III Rise-Stay-Rise, kerangka kerja dapat secara otomatis menerapkan iSENSE2.0 yang kami usulkan atau mengubah manajer proyek untuk berhati-hati dengan prediksi.

6.3 Penerapan Potensial untuk Konteks Lain

Kami berpendapat bahwa meskipun iSENSE2.0 termotivasi dan dirancang untuk mengatasi pengorbanan pengujian dalam paradigma crowdsourcing, sebenarnya memiliki potensi besar untuk diterapkan pada perangkat lunak lain dan konteks pengujian sistem. Salah satu contoh konteks adalah pengujian sistem perangkat lunak-intensif dalam domain pertahanan, mengingat kompleksitas sistem yang terus meningkat, kurangnya proses pengujian dan sumber daya, serta tekanan jadwal yang intens.

Menurut lokakarya SERC baru-baru ini tentang "Jaminan Sistem Berbasis Model (MBSA)" [2], yang berkaitan dengan praktik pengujian berbasis model di seluruh pemerintah AS, itu menyatakan bahwa "tidak ada proses pengujian formal di berbagai sistem di seluruh dunia. pemerintah. Ad hoc, manual, aktivitas pengujian terdesentralisasi membuat pengujian jaminan sistem menjadi sangat menantang." Akibatnya, ketika semua penguji harus menjawab pertanyaan umum seperti "kapan harus menghentikan pengujian", ada kekurangan metode analitis atau ilmiah untuk mendukung pengambilan keputusan tersebut. Keterbatasannya terletak pada kurangnya

proses, dan yang lebih mendasar, kurangnya metrik dan metode untuk mendukung pengukuran dinamis dan visibilitas proses di antara tim pengujian terdistribusi untuk mendukung perencanaan dan pengelolaan tugas pengujian yang lebih baik. Misalnya, kurva Rayleigh adalah model klasik untuk memprediksi kedatangan cacat dinamis dalam pengujian perangkat lunak dan rekayasa keandalan dalam domain pertahanan [37]. Namun, itu membutuhkan model pengukuran mendasar untuk diterapkan secara efektif. Selanjutnya, penelitian ini menunjukkan bahwa iSENSE2.0 yang diusulkan secara signifikan mengungguli model Rayleigh.

Salah satu kemungkinan penerapan iSENSE2.0 dalam konteks tersebut adalah untuk menyelidiki dan mengembangkan serangkaian model konteks pengujian dan kemudian mengadaptasi iSENSE2.0 ke konteks tersebut untuk mendukung pemantauan, agregat, dan prediksi kemajuan pengujian secara dinamis menuju penyelesaian di antara tim pengujian terdistribusi. Ini direncanakan sebagai pekerjaan masa depan.

6.4 Ancaman terhadap Validitas

Pertama, artikel ini memperlakukan jumlah laporan crowdtesting sebagai jumlah biaya saat mengukur pengurangan biaya (Bagian 4.2). Ini berlaku untuk skema pembayaran berdasarkan laporan (yaitu, pekerja paksa yang mengirimkan laporan bisa mendapatkan bayaran) [83], yang merupakan skema pembayaran yang umum digunakan. Untuk skema populer kedua yang dibayar oleh bug (mis., crowdworkers yang melaporkan bug bisa dibayar) [3, 25], iSENSE2.0 juga dapat mengurangi biaya dengan menutup tugas dengan benar (mis., lebih sedikit laporan bug berarti lebih sedikit biaya). Dan kami yakin iSENSE2.0 dapat memperoleh kinerja yang sebanding, karena proporsi bug dalam laporan hampir tidak berubah selama proses crowdtesting. Untuk skema populer ketiga yang dibayar oleh bug pertama (yaitu, crowdworkers yang melaporkan bug pertama bisa dibayar) [4]. Skema ini kurang sensitif terhadap keputusan penutupan tugas otomatis, karena di bawah skema ini, pembayaran kepada crowdworkers adalah konstan (yaitu, jumlah bug unik yang dikirimkan). Namun, dengan menutup tugas pada waktu yang tepat, platform berpotensi mengurangi biaya untuk mengelola laporan duplikat serta mempersingkat durasi tugas crowdtesting.

Kedua, pendekatan yang kami usulkan didasarkan pada status duplikat yang disimpulkan, menerapkan teknik analisis semantik pada deskripsi tekstual dari laporan crowdtesting. Pertunjukan tersebut mungkin sensitif terhadap bahasa atau kebiasaan menulis dari beragam pekerja kerumunan. Namun, kami telah bereksperimen pada lebih dari 50 ribu laporan crowdtesting yang, sampai batas tertentu, dapat mengatasi ancaman ini. Selain itu, kami memperlakukan status duplikat laporan crowdtesting sebagai status bug duplikat saat membuat tabel pencarian kedatangan bug, sedangkan situasi sebenarnya adalah beberapa laporan crowdtesting tidak mengandung bug. Perlakuan ini dapat diterima, karena close estimator berbasis CRC sebenarnya berlaku berdasarkan status duplikat dari laporan yang masuk, tidak peduli apakah itu bug atau bukan.

Ketiga, eksperimen kami didasarkan pada 306 tugas crowdtesting yang dikumpulkan dari salah satu platform pengujian crowdsourced China terbesar. Meskipun keragaman tugas, ukuran kumpulan data, dan kinerja yang konsisten secara relatif mengurangi risiko ancaman generalisasi, kami tidak dapat memastikan bahwa hasil penelitian kami

dapat digeneralisasikan di luar lingkungan tempat penelitian ini dilakukan. Eksplorasi lebih lanjut akan dilakukan terutama di platform crowdtesting lainnya.

7 PEKERJAAN TERKAIT

7.1 Crowdtesting

Crowdtesting telah diterapkan untuk memfasilitasi banyak tugas pengujian, misalnya, pembuatan kasus uji [16], pengujian kegunaan [28], analisis kinerja perangkat lunak [51], deteksi bug perangkat lunak dan reproduksi [27]. Meskipun aplikasi ini berhasil, ada beberapa tantangan dalam crowdtesting, yang ditunjukkan sebagai berikut:

Manajemen laporan crowdtesting: Sejumlah besar laporan crowdtesting akan tiba saat aplikasi dikirim untuk crowdtesting. Sangat penting untuk mengelola dan melakukan triase laporan ini secara efektif untuk memfasilitasi proses perbaikan bug berikut [1, 25, 83]. Untuk mengatasi tantangan ini, Feng et al. [22, 23] mengusulkan pendekatan untuk memprioritaskan laporan pengujian dalam crowdtesting. Mereka merancang strategi untuk secara dinamis memilih laporan pengujian yang paling berisiko dan beragam untuk pemeriksaan di setiap iterasi. Jiang dkk. [36] mengusulkan laporan pengujian kerangka pengelompokan fuzzy dengan menggabungkan laporan crowdtesting yang berlebihan dan multi-bug ke dalam cluster untuk mengurangi jumlah laporan pengujian yang diperiksa. Wang dkk. [68, 69, 72] mengusulkan pendekatan untuk secara otomatis mengklasifikasikan laporan crowdtesting. Pendekatan mereka dapat mengatasi distribusi data yang berbeda di antara domain perangkat lunak yang berbeda dan mencapai hasil klasifikasi yang baik. Liu dkk. [42] mengusulkan pendekatan otomatis untuk menghasilkan kata-kata deskriptif dari laporan crowdtesting untuk tangkapan layar berdasarkan model bahasa dan teknik Pencocokan Piramida Spasial. Hao dkk. [30] mengusulkan CTRAS untuk secara otomatis menggabungkan duplikat berdasarkan informasi tekstual dan tangkapan layar dan meringkas laporan pengujian duplikat menjadi laporan yang komprehensif dan dapat dipahami.

Manajemen crowdworker: Memilih crowdworker yang tepat diperingkatkan sebagai salah satu dari 10 faktor penting untuk berhasil melakukan crowdtesting [1]. Ini karena crowdworkers memiliki latar belakang yang berbeda dan tingkat pengalaman yang berbeda, dan apa yang dibutuhkan dari mereka juga berbeda dari satu proyek ke proyek lainnya [1, 84]. Untuk mengatasi tantangan ini, Wang et al. [71], Cui dkk. [18, 19], dan Xie et al. [78] mengusulkan pendekatan seleksi crowdworker untuk merekomendasikan crowdworker yang tepat untuk tugas-tugas crowdtesting tertentu. Pendekatan ini mempertimbangkan pengalaman crowdworker, relevansi dengan tugas, keragaman konteks pengujian, dan sebagainya, dan merekomendasikan sekumpulan pekerja yang dapat mendeteksi lebih banyak bug.

Manajemen dan perencanaan proses crowdtesting: Dalam artikel “peningkatan proses untuk pengujian perangkat lunak crowdsourced,” Alyahya et al. [10] menyebutkan bahwa tidak ada mekanisme untuk memantau kemajuan crowdworkers. Zogaj dkk. [84] menyarankan untuk menggabungkan mekanisme otomatis untuk memastikan kualitas dan kelayakan crowdtesting dengan proses crowdtesting. Selain itu, salah satu dari 10 faktor penting untuk berhasil melakukan pengujian crowdsourced adalah "perencanaan realistis" [1], yang memerlukan penentuan waktu dengan cermat

untuk memulai pengujian dan menghentikan pengujian untuk mengoptimalkan waktu pengiriman aplikasi ke pasar. Pekerjaan kami sebelumnya [73] mencoba untuk mengatasi tantangan ini dengan mengusulkan pendekatan manajemen crowdtesting iSENSE, yang dapat memprediksi waktu penutupan tugas yang optimal dan memfasilitasi praktik manajemen crowdtesting yang lebih efektif. Pekerjaan ini memperluas iSENSE untuk membuatnya lebih dapat diterapkan dalam praktik crowdtesting di dunia nyata dan lebih meningkatkan efektivitas biaya crowdtesting.

Ada tantangan lain yang disebutkan dalam pekerjaan yang ada yang memerlukan penelitian di masa depan, seperti bagaimana merancang mekanisme insentif yang tepat untuk memotivasi pekerja kerumunan [1, 83, 84], bagaimana mengoptimalkan dekomposisi dan desain tugas [34, 83], dan segera.

7.2 Manajemen Kualitas Perangkat Lunak

Banyak pendekatan yang ada mengusulkan analisis berbasis nilai atau risiko untuk memprioritaskan atau memilih kasus uji [21, 32, 49, 56, 60, 62, 74, 76, 82] untuk meningkatkan efektivitas biaya pengujian. Namun, tidak satu pun dari mereka yang berlaku untuk paradigma crowdtesting yang muncul di mana manajer biasanya tidak memiliki kendali atas perilaku dinamis crowdworker online dan kinerja yang tidak pasti.

Ada juga penelitian yang berfokus pada prediksi cacat dan estimasi usaha [9, 38, 47, 53, 63]. Bagian inti dari pendekatan ini adalah ekstraksi fitur dari kode sumber, atau repositori perangkat lunak. Namun, dalam crowdtesting, platform tidak dapat memperoleh kode sumber aplikasi ini, atau terlibat dalam proses pengembangan aplikasi ini.

Banyak pendekatan yang ada berfokus pada penerapan teknik over-sampling dan under-sampling untuk mengurangi masalah ketidakseimbangan data dalam prediksi [24, 65, 75]. Namun, yang kami hadapi adalah data kedatangan bug yang dinamis dan tidak pasti. Inilah sebabnya mengapa kami menggunakan sampling tambahan dalam penelitian ini.

Badan lain dari penelitian sebelumnya bertujuan untuk mengoptimalkan pemeriksaan perangkat lunak dengan memprediksi jumlah total dan sisa bug menggunakan model Capture-ReCapture (CRC). Eik dkk. [20] melaporkan pekerjaan pertama dalam menggunakan model capture-recapture dalam inspeksi perangkat lunak untuk memperkirakan jumlah bug yang tersisa dalam persyaratan dan artefak desain. Penelitian yang ada mengevaluasi pengaruh jumlah inspektur, jumlah bug aktual, ketergantungan dalam inspektur, dan gaya belajar inspektur individu pada akurasi penduga CRC [12, 20, 29, 31, 43, 44, 59, 66, 67]. Mereka didasarkan pada berbagai jenis model CRC, dan hasilnya ternyata MhJK, MhCH, dan MtCH adalah penduga yang paling efektif. Kami telah menggunakan kembali semua estimator ini dan mengevaluasinya secara eksperimental dalam crowdtesting.

8 KESIMPULAN

Manfaat crowdtesting sebagian besar dikaitkan dengan potensinya untuk menyelesaikan tes lebih cepat dan lebih murahz Termotivasi oleh pengamatan empiris dari platform crowdtesting industri, penelitian ini bertujuan untuk mengembangkan dukungan keputusan tertutup otomatis untuk mencapai efektivitas biaya tambahan.

Pendekatan prediksi dekat yang diusulkan iSENSE2.0 menerapkan teknik pengambilan sampel inkremental untuk mengatur laporan yang tiba secara dinamis dan penanda duplikat berbasis semantik untuk mengekstrak status duplikat laporan. iSENSE2.0 mendesain estimator close berbasis CRC (Capture-ReCapture) untuk menghasilkan keputusan close berdasarkan status kedatangan bug dinamis dan pemeriksa kewarasan berbasis cakupan untuk mengatasi kemacetan kinerja yang disebabkan oleh kurva kedatangan bug Rise-Stay-Rise.

Evaluasi dilakukan terhadap 56.920 laporan crowdtesting dari salah satu platform crowdtesting terbesar. iSENSE2.0 dapat memberikan manajer kemungkinan yang lebih besar untuk mencapai keuntungan efektivitas biaya dari crowdtesting. Dengan iSENSE2.0, median 100% bug dapat dideteksi dengan penghematan biaya 30% berdasarkan prediksi penutupan otomatis. Pekerjaan ini meningkatkan pendekatan mutakhir dengan penerapan yang lebih baik, karena yang terakhir bergantung pada tag duplikat yang umumnya dianggap memakan waktu dan membosankan untuk diperoleh.

Referensi :

[1] Sahil Deva. 2018. Retrieved from https://dzone.com/articles/10-critical-factors-to-successfully-perform-crowds.

[2] SERC. 2019. SERC Workshop on Model-Based System Assurance. Retrieved from https://sercuarc.org/event/sercworkshop-model-based-system-assurance/.

[3] Software Testing Help. 2020. Retrieved from http://www.softwaretestinghelp.com/crowdsourced-testingcompanies/.

[4] DZone. 2020. Retrieved from https://dzone.com/articles/top-10-benefits-of-crowd-testing-1.

[5] Software Testing Help. 2020. Retrieved from https://www.softwaretestinghelp.com/guide-to-crowdsourced-testing/.

[6] DZone. 2020. Retrieved from https://dzone.com/articles/why-google-uses-crowd-testing.

[7] Applause. 2020. Retrieved from https://www.applause.com/customers/.

[8] 360 logica. 2020. Retrieved from https://www.360logica.com/blog/challenges-faced-in-mobile-app-testing/.

[9] Amritanshu Agrawal and Tim Menzies. 2018. “Better data” is better than “better data miners” (benefits of tuning SMOTE for defect prediction). In Proceedings of the 40th International Conference on Software engineering.

[10] Sultan Alyahya and Dalal Alrugebh. 2017. Process improvements for crowdsourced software testing. Int. J. Adv. Comput. Sci. Applic. 8, 6 (2017), 32–40.

[11] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. J. Mach. Learn. Res. 3 (2003), 1137–1155.

[12] L. C. Briand, K. El Emam, B. G. Freimut, and O. Laitenberger. 2000. A comprehensive evaluation of capture-recapture models for estimating software defect content. IEEE Trans. Softw. Eng. 26, 6 (2000), 518–540.

[13] Kenneth P. Burnham and Walter Scott Overton. 1978. Estimation of the size of a closed population when capture probabilities vary among animals. Biometrika 65, 3 (1978), 625–633.

[14] Anne Chao. 1987. Estimating the population size for capture-recapture data with unequal catchability. Biometrics 43, 4 (1987), 783–791.

[15] Anne Chao. 1988. Estimating animal abundance with capture frequency data. J. Wildlife Manag. (1988), 295–300.

[16] Ning Chen and Sunghun Kim. 2012. Puzzle-based automatic testing: Bringing humans into the loop by solving puzzles. In Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. ACM, 140–149.

[17] Norman Cliff. 2014. Ordinal Methods for Behavioral Data Analysis. Psychology Press.

[18] Qiang Cui, Junjie Wang, Guowei Yang, Miao Xie, Qing Wang, and Mingshu Li. 2017. Who should be selected to perform a task in crowdsourced testing? In Proceedings of the IEEE 41st Annual Computer Software and Applications Conference, Vol. 1. IEEE, 75–84.

[19] Qiang Cui, Song Wang, Junjie Wang, Yuanzhe Hu, Qing Wang, and Mingshu Li. 2017. Multi-objective crowd worker selection in crowdsourced testing. In Proceedings of the 29th International Conference on Software Engineering and Knowledge Engineering. 218–223.

[20] Stephen G. Eick, Clive R. Loader, M. David Long, Lawrence G. Votta, and Scott Vander Wiel. 1992. Estimating software fault content before coding. In Proceedings of the 14th International Conference on Software Engineering. 59–65.

[21] Michael G. Epitropakis, Shin Yoo, Mark Harman, and Edmund K. Burke. 2015. Empirical evaluation of Pareto efficient multi-objective regression test case prioritisation. In Proceedings of the International Symposium on Software Testing and Analysis. ACM, 234–245.

[22] Yang Feng, Zhenyu Chen, James A. Jones, Chunrong Fang, and Baowen Xu. 2015. Test report prioritization to assist crowdsourced testing. In Proceedings of the 10th Joint Meeting on Foundations of Software Engineering. 225–236.

[23] Yang Feng, James A. Jones, Zhenyu Chen, and Chunrong Fang. 2016. Multi-objective test report prioritization using image understanding. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. IEEE, 202–213.

[24] Kehan Gao, Taghi M. Khoshgoftaar, and Randall Wald. 2014. The use of under- and oversampling within ensemble feature selection and classification for software quality prediction. Int. J. Reliab. Qual. Safe. Eng. 21, 01 (2014), 1450004.

[25] Ruizhi Gao, Yabin Wang, Yang Feng, Zhenyu Chen, and W. Eric Wong. 2018. Successes, challenges, and rethinking— An industrial investigation on crowdsourced mobile application testing. Empir. Softw. Eng. 24, 2 (2018), 1–25.

[26] Mohit Garg, Richard Lai, and S. Jen Huang. 2011. When to stop testing: A study from the perspective of software reliability models. IET Softw. 5, 3 (2011), 263–273.

[27] María Gómez, Romain Rouvoy, Bram Adams, and Lionel Seinturier. 2016. Reproducing context-sensitive crashes of mobile apps using crowdsourced monitoring. In Proceedings of the IEEE/ACM International Conference on Mobile Software Engineering and Systems. IEEE, 88–99.

[28] V. H. M. Gomide, P. A. Valle, J. O. Ferreira, J. R. G. Barbosa, A. F. da Rocha, and T. M. G. d. A. Barbosa. 2014. Affective crowdsourcing applied to usability testing. Int. J. Comput. Sci. Inf. Technol. 5, 1 (2014), 575–579.

[29] Anurag Goswami, Gursimran Walia, and Abhinav Singh. 2015. Using learning styles of software professionals to improve their inspection team performance. Int. J. Softw. Eng. Knowl. Eng. 25, 09–10 (2015), 1721–1726.

[30] Rui Hao, Yang Feng, James A. Jones, Yuying Li, and Zhenyu Chen. 2019. CTRAS: crowdsourced test report aggregation and summarization. In Proceedings of the 41st International Conference on Software Engineering (ICSE'19). 900–910.

[31] Young H. Chun. 2006. Estimating the number of undetected software errors via the correlated capture-recapture model. Euro. J. Oper. Res. 175, 2 (2006), 1180–1192.

[32] Christopher Henard, Mike Papadakis, Mark Harman, Yue Jia, and Yves Le Traon. 2016. Comparing white-box and black-box test prioritization. In Proceedings of the IEEE/ACM 38th International Conference on Software Engineering. IEEE, 523–534.

[33] Michael Hilton, Jonathan Bell, and Darko Marinov. 2018. A large-scale study of test coverage evolution. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE'18). 53–63.

[34] Mahmood Hosseini, Keith Phalp, Jacqui Taylor, and Raian Ali. 2014. The four pillars of crowdsourcing: A reference

model. In Proceedings of the IEEE 8th International Conference on Research Challenges in Information Science (RCIS'14).IEEE, 1–12.

[35] Javaid Iqbal, N. Ahmad, and S. M. K. Quadri. 2013. A software reliability growth model with two types of learning.

In Proceedings of the International Conference on Machine Intelligence and Research Advancement. IEEE, 498–503.

[36] He Jiang, Xin Chen, Tieke He, Zhenyu Chen, and Xiaochen Li. 2018. Fuzzy clustering of crowdsourced test reports

for apps. ACM Trans. Internet Technol. 18, 2 (2018), 18.

[37] Stephen H. Kan. 2002. Metrics and Models in Software Quality Engineering. Addison-Wesley Longman Publishing Co.,Inc.

[38] Ekrem Kocaguneli, Tim Menzies, and Jacky W. Keung. 2012. On the value of ensemble effort estimation. IEEE Trans.Softw. Eng. 38, 6 (2012), 1403–1416.

[39] Pierre S. Laplace. 1783. Sur les naissances, les mariages et les morts. Hist. I'Acad. Roy. Sci. (1783), 693.

[40] Shen-Ming Lee. 1996. Estimating population size for capture-recapture data when capture probabilities vary by time, behavior and individual animal. Commun. Stat.-Simul. Comput. 25, 2 (1996), 431–457.

[41] William E. Lewis. 2016. Software Testing and Continuous Quality Improvement. CRC Press.

[42] Di Liu, Xiaofang Zhang, Yang Feng, and James A. Jones. 2018. Generating descriptions for screenshots to assist crowdsourced testing. In Proceedings of the IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER'18). IEEE, 492–496.

[43] Gaoxuan Liu, Guoping Rong, He Zhang, and Qi Shan. 2015. The adoption of capture-recapture in software engineering: a systematic literature review. In Proceedings of the 19th International Conference on Evaluation and Assessmentin Software Engineering. ACM, 15.

[44] Narendar R. Mandala, Gursimran S. Walia, Jeffrey C. Carver, and Nachiappan Nagappan. 2012. Application of Kusumoto cost-metric to evaluate the cost effectiveness of software inspections. In Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. 221–230.

[45] Ke Mao, Licia Capra, Mark Harman, and Yue Jia. 2017. A survey of the use of crowdsourcing in software engineering. J. Syst. Softw. 126 (2017), 57–84.

[46] Alessandro Marchetto, Giuseppe Scanniello, and Angelo Susi. 2019. Combining code and requirements coverage with execution cost for test suite reduction. IEEE Trans. Softw. Eng. 45, 4 (2019), 363–390.

[47] Tim Menzies, Jeremy Greenwald, and Art Frank. 2007. Data mining static code attributes to learn defect predictors. IEEE Trans. Softw. Eng. 33, 1 (2007), 2–13.

[48] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. 2013. Distributed representations of words and phrases and their compositionality. In Proceedings of the Conference on Neural Information Processing Systems (NIPS'13). 3111–3119.

[49] Breno Miranda, Emilio Cruciani, Roberto Verdecchia, and Antonia Bertolino. 2018. FAST approaches to scalable similarity-based test case prioritization. In Proceedings of the 40th International Conference on Software Engineering (ICSE'18).

[50] Ligia Mora-Applegate and Mark Malinowski. 2012. Incremental Sampling Methodology. Technical Report. Interstate Technology and Regulatory Council (ITRC).

[51] R. Musson, J. Richards, D. Fisher, C. Bird, B. Bussone, and S. Ganguly. 2013. Leveraging the crowd: How 48,000 users helped improve Lync performance. IEEE Softw. 30, 4 (2013), 38–45.

[52] Glenford J. Myers, Corey Sandler, and Tom Badgett. 2011. The Art of Software Testing. John Wiley & Sons.

[53] Jaechang Nam, Wei Fu, Sunghun Kim, Tim Menzies, and Lin Tan. 2018. Heterogeneous defect prediction. IEEE Trans. Softw. Eng. 44, 9 (2018), 574–896.

[54] Anh Tuan Nguyen, Tung Thanh Nguyen, Tien N. Nguyen, David Lo, and Chengnian Sun. 2012. Duplicate bug report detection with a combination of information retrieval and topic modeling. In Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE'12). 70–79.

[55] Rohan Padhye, Caroline Lemieux, and Koushik Sen. 2019. JQF: Coverage-guided property-based testing in Java. In Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'19). 398–401.

[56] Annibale Panichella, Rocco Oliveto, Massimiliano Di Penta, and Andrea De Lucia. 2015. Improving multi-objective test case selection by injecting diversity in genetic algorithms. IEEE Trans. Softw. Eng. 41, 4 (2015), 358–383.

[57] Karl Pearson. 1905. The problem of the random walk. Nature 72, 1867 (1905), 342.

[58] H. Rocha, M. T. Valente, H. Marques-Neto, and G. C. Murphy. 2016. An empirical study on recommendations of similar bugs. In Proceedings of the IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER'16). 46–56.

[59] Guoping Rong, Bohan Liu, He Zhang, Qiuping Zhang, and Dong Shao. 2017. Towards confidence with capturerecapture estimation: An exploratory study of dependence within inspections. In Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering. 242–251.

[60] Ripon K. Saha, Lingming Zhang, Sarfraz Khurshid, and Dewayne E. Perry. 2015. An information retrieval approach for regression test prioritization based on program changes. In Proceedings of the IEEE/ACM 37th IEEE International Conference on Software Engineering, Vol. 1. IEEE, 268–279.

[61] Gerard Salton and Michael McGill. 1984. Introduction to Modern Information Retrieval. McGraw-Hill Book Company.

[62] August Shi, Tifany Yung, Alex Gyori, and Darko Marinov. 2015. Comparing and combining test-suite reduction and regression test selection. In Proceedings of the 10th Joint Meeting on Foundations of Software Engineering. ACM, 237– 247.

[63] Pradeep Kumar Singh, Dishti Agarwal, and Aakriti Gupta. 2015. A systematic review on software defect prediction. In Proceedings of the 2nd International Conference on Computing for Sustainable Global Development. IEEE, 1793–1797.

[64] C. Sun, D. Lo, S. Khoo, and J. Jiang. 2011. Towards more accurate retrieval of duplicate bug reports. In Proceedings of the International Conference on Automated Software Engineering (ASE'11). 253–262.

[65] Ming Tan, Lin Tan, Sashank Dara, and Caleb Mayeux. 2015. Online defect prediction for imbalanced data. In Proceedings of the 37th International Conference on Software Engineering, Vol. 2. IEEE Press, 99–108.

[66] Padmal Vitharana. 2017. Defect propagation at the project-level: Results and a post-hoc analysis on inspection efficiency. Empir. Softw. Eng. 22, 1 (2017), 57–79.

[67] G. S. Walia and J. C. Carver. 2009. Evaluating the effect of the number of naturally occurring faults on the estimates produced by capture-recapture models. In Proceedings of the International Conference on Software Testing Verification and Validation. 210–219.

[68] Junjie Wang, Qiang Cui, Qing Wang, and Song Wang. 2016. Towards effectively test report classification to assist crowdsourced testing. In Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ACM, 6.

[69] Junjie Wang, Qiang Cui, Song Wang, and Qing Wang. 2017. Domain adaptation for test report classification in crowdsourced testing. In Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track. IEEE Press, 83–92.

[70] Junjie Wang, Mingyang Li, Song Wang, Tim Menzies, and Qing Wang. 2019. Images don't lie: Duplicate crowdtesting reports detection with screenshot information. Inf. Softw. Technol. 110 (2019), 139–155.

[71] J. Wang, S. Wang, J. Chen, T. Menzies, Q. Cui, M. Xie, and Q. Wang. 2019. Characterizing crowds to better optimize worker recommendation in crowdsourced testing. IEEE Trans. Softw. Eng. (2019). DOI:https://doi.org/10.1109/TSE. 2019.2918520.

[72] Junjie Wang, Song Wang, Qiang Cui, and Qing Wang. 2016. Local-based active classification of test report to assist crowdsourced testing. In Proceedings of the 31st International Conference on Automated Software Engineering. IEEE, 190–201.

[73] Junjie Wang, Ye Yang, Rahul Krishna, Tim Menzies, and Qing Wang. 2019. iSENSE: Completion-aware crowdtestingm management. In Proceedings of the International Conference on Software Engineering (ICSE'19). 912–923.

[74] Kaiyuan Wang, Chenguang Zhu, Ahmet Celik, Jongwook Kim, Don Batory, and Milos Gligoric. 2018. Towards refactoring-aware regression test selection. In Proceedings of the 40th International Conference on Software Engineering (ICSE'18). 233–244.

[75] Song Wang, Taiyue Liu, and Lin Tan. 2016. Automatically learning semantic features for defect prediction. In Proceedings of the 38th International Conference on Software Engineering. ACM, 297–308.

[76] Song Wang, Jaechang Nam, and Lin Tan. 2017. QTEP: Quality-aware test case prioritization. In Proceedings of the 11th Joint Meeting on Foundations of Software Engineering. ACM, 523–534.

[77] L. Wei, Y. Liu, S. C. Cheung, H. Huang, X. Lu, and X. Liu. 2018. Understanding and detecting fragmentation-induced compatibility issues for Android apps. IEEE Trans. Softw. Eng. Early access. https://ieeexplore.ieee.org/document/8493348.

[78] Miao Xie, Qing Wang, Guowei Yang, and Mingshu Li. 2017. COCOON: Crowdsourced testing quality maximization under context coverage constraint. In Proceedings of the IEEE 28th International Symposium on Software Reliability Engineering. IEEE, 316–327.

[79] B. Xu, D. Ye, Z. Xing, X. Xia, G. Chen, and S. Li. 2016. Predicting semantically linkable knowledge in developer online forums via convolutional neural network. In Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE'16). 51–62.

[80] X. Yang, D. Lo, X. Xia, L. Bao, and J. Sun. 2016. Combining word embedding with information retrieval to recommend similar bug reports. In Proceedings of the International Symposium on Software Reliability Engineering (ISSRE'16). 127– 137.

[81] Zhe Yu, Nicholas A. Kraft, and Tim Menzies. 2018. Finding better active learners for faster literature reviews. Empir.Softw. Eng. 23, 6 (2018), 3161–3186.

[82] Lingming Zhang. 2018. Hybrid regression test selection. In Proceedings of the 40th International Conference on SoftwareEngineering. 199–209.

[83] Xiaofang Zhang, Yang Feng, Di Liu, Zhenyu Chen, and Baowen Xu. 2018. Research progress of crowdsourced softwaretesting. Journal of Software 29(1) (2018), 69–88.

[84] Shkodran Zogaj, Ulrich Bretschneider, and Jan Marco Leimeister. 2014. Managing crowdsourced software testing:A case study based insight on the challenges of a crowdsourcing intermediary. Journal of Business Economics 84, 3(2014), 375–405.