e-ISSN 2828-0490
JURNAL MEDIA COMPUTER SCIENCE

Implementasi Automated Conflict Testing Untuk Mencegah Double Booking Pada Sistem Penjadwalan Klinik Gigi

Sulthan Zahran Sunata [1]; Wahyu Catur Wibowo [2]
[1),2)] ***Magister Teknologi Informasi,Universitas Indonesia***
Email: [1)] sulthan.zahran31@ui.ac.id ;[2)] wibowo@cs.ui.ac.id

How to Cite :
Sunata, S, Z., Wibowo, W, C. (2026). Implementasi Automated Conflict Testing Untuk Mencegah Double Booking Pada Sistem Penjadwalan Klinik Gigi. Jurnal Media Computer Science, 5(3).

ARTICLE HISTORY
Received [15 Juni 2026]
Revised [20 Juli 2026]
Accepted [23 Juli 2026]

KEYWORDS
Wireshark, Windump, Network Security Monitoring.

ABSTRAK

Double booking merupakan risiko utama pada sistem penjadwalan klinik gigi karena mengganggu alur pelayanan, meningkatkan waktu tunggu pasien, dan menurunkan efisiensi operasional dokter. Meskipun literatur penjadwalan layanan kesehatan banyak membahas optimasi slot, resource allocation, dan no-show, masih terbatas penelitian yang menunjukkan bagaimana aturan pencegahan konflik divalidasi otomatis pada sistem klinik yang diimplementasikan. Di sisi lain, literatur software testing lebih sering membahas otomatisasi secara umum daripada pengujian executable terhadap rule conflict checking. Penelitian ini bertujuan mengimplementasikan automated conflict testing untuk memverifikasi mekanisme pencegahan double booking pada sistem penjadwalan klinik gigi berbasis Django REST API. Metode yang diterapkan mencakup pengujian terotomasi pada level model dan API dengan skenario yang diturunkan dari aturan bisnis conflict checking. Hasil pengujian menunjukkan seluruh skenario berada pada status lulus dan rule conflict checking diterapkan secara seragam pada proses booking maupun reschedule. Kontribusi utama penelitian ini berupa pendekatan automated conflict testing yang executable, terstruktur, dan menunjukkan bahwa slot back-to-back tetap diterima, overlap aktif ditolak, dan approval reschedule tetap melewati validasi konflik.

ABSTRACT

Double booking poses a significant risk to dental clinic scheduling systems, as it disrupts service workflows, increases patient wait times, and reduces practitioner operational efficiency. While healthcare scheduling literature extensively covers slot optimization, resource allocation, and no-show issues, there is limited research demonstrating how conflict prevention rules are automatically validated within implemented clinic systems. Conversely, software testing literature tends to focus on general automation rather than executable testing for rule-based conflict checking. This study aims to implement automated conflict testing to verify double-booking prevention mechanisms in a Django REST API-based dental clinic scheduling system. The methodology involves automated testing at both the model and API levels, utilizing scenarios derived from conflict-checking business rules. Test results indicate that all scenarios passed and that conflict-checking rules were consistently applied to both initial booking and rescheduling processes. The study's primary contribution is an executable, structured automated conflict-testing approach that confirms the acceptance of back-to-back slots, the rejection of active overlaps, and the subjection of rescheduling requests to conflict validation.

PENDAHULUAN

Penjadwalan layanan merupakan komponen operasional penting dalam sistem informasi klinik karena berkaitan langsung dengan pengaturan waktu pelayanan pasien dan ketersediaan dokter. Pada klinik gigi, kualitas penjadwalan memengaruhi alur kerja administrasi secara menyeluruh. Ketika proses penjadwalan tidak dikelola dengan baik, klinik dapat mengalami bentrok jadwal, keterlambatan layanan, dan penurunan kualitas pelayanan (Ahmadi-Javid et al., 2017; Kuiper et al., 2021). Salah satu masalah paling krusial adalah double booking, yaitu kondisi ketika dua atau lebih janji temu tercatat pada slot waktu yang saling bertabrakan untuk sumber daya yang sama, yang berdampak pada ketidakpastian pelayanan dan ketidakpuasan pasien (Dantas et al., 2018; Kaandorp & Koole, 2007).

Aturan konflik pada modul penjadwalan rentan terhadap regresi ketika sistem terus dikembangkan, khususnya saat fitur reschedule, pembatalan appointment, atau penyesuaian jam kerja dokter ditambahkan. Pengujian manual sulit menjaga konsistensi ketika jumlah skenario konflik bertambah. Karena itu, diperlukan pendekatan automated testing yang secara khusus memverifikasi aturan conflict checking agar perubahan sistem tidak menyebabkan munculnya kembali risiko double booking (Barr et al., 2015; Godefroid et al., 2008; Pezze & Zhang, 2014).

Penelitian ini mengisi celah antara literatur penjadwalan layanan kesehatan yang kuat pada optimasi tetapi lemah pada verifikasi rule, dan literatur software testing yang membahas otomasi secara umum tanpa mengikatnya pada business rule konflik yang spesifik. Novelty penelitian ini adalah menghadirkan automated conflict testing yang menautkan rule penjadwalan, executable test case, dan bukti hasil uji dalam satu artefak yang dapat direplikasi. Tujuan penelitian meliputi: (1) mengimplementasikan aturan conflict checking pada sistem penjadwalan klinik gigi, (2) menyusun automated conflict testing untuk memverifikasi perilaku sistem pada skenario konflik yang relevan, dan (3) menyediakan rancangan pengujian dengan ketertelusuran antara aturan penjadwalan, test case, dan hasil pengujian.

LANDASAN TEORI

Penjadwalan Layanan Kesehatan dan Pencegahan Konflik

Riset appointment scheduling pada layanan kesehatan telah berkembang dari studi konseptual menuju model optimasi yang semakin rinci. (Ahmadi-Javid et al., 2017) menunjukkan bahwa literatur outpatient scheduling berfokus pada keputusan strategis, taktis, dan operasional untuk meningkatkan akses layanan dan utilisasi sumber daya. (Borgman et al., 2018; Kaandorp & Koole, 2007; Tsai & Teng, 2014) memperlihatkan bahwa sistem penjadwalan harus berhadapan dengan variasi durasi layanan, ketidakpastian permintaan, dan keterbatasan multi-resource. (Dantas et al., 2018; Luo et al., 2019) menegaskan bahwa no-show memengaruhi kapasitas efektif dan efisiensi layanan. (Kuiper et al., 2021) menekankan bahwa praktik penjadwalan nyata jauh lebih ambigu dan kaya constraint dibanding asumsi model teoritis, sehingga kebutuhan utamanya bukan hanya rule konflik yang baik di atas kertas, tetapi juga mekanisme automated testing yang dapat membuktikan rule tersebut tetap bekerja konsisten. (Yan et al., 2022) menguraikan bahwa konflik muncul dari perebutan sumber daya yang sama dan batas kapasitas operasional, sedangkan conflict dalam konteks penelitian ini dimaknai secara operasional sebagai kondisi yang menyebabkan appointment harus ditolak berdasarkan active workday, overlap interval, atau permintaan reschedule yang tidak valid (Borgman et al., 2018; Luo et al., 2019; Tsai & Teng, 2014).

Automated Testing dan Posisi Penelitian

Automated testing merupakan mekanisme penting untuk meningkatkan kualitas perangkat lunak dan mengurangi biaya regresi. (Godefroid et al., 2008) menunjukkan peluang besar otomasi pembangkitan test case melalui program analysis. (Anand et al., 2013) meninjau berbagai pendekatan automated test generation, termasuk symbolic execution, model-based testing, dan

e-ISSN 2828-0490
JURNAL MEDIA COMPUTER SCIENCE

combinatorial testing. (Barr et al., 2015; Pezze & Zhang, 2014) menekankan bahwa test oracle menjadi bottleneck utama dalam software testing automation, sehingga keputusan menerima atau menolak slot harus dapat dibandingkan secara eksplisit dengan expected result. (Taromirad & Runeson, 2025) memperkuat pentingnya assertion yang eksplisit untuk membedakan perilaku benar dan salah. (Ali et al., 2019; Lou et al., 2019; Rosero et al., 2016; Yoo & Harman, 2012) menunjukkan bahwa regression testing semakin penting ketika perangkat lunak terus berevolusi. (Butt et al., 2023; Garousi & Mantyla, 2016) menegaskan bahwa keberhasilan test automation sangat dipengaruhi konteks adopsi dan keputusan tentang apa yang perlu diotomatisasi.Tabel 1 merangkum perbedaan posisi penelitian ini terhadap studi-studi terpilih dari literatur yang dibahas.

Tabel 1. Perbandingan posisi penelitian ini dengan studi terkait

Aspek	Penelitian ini	Kaandorp dan Koole [2]	Kuiper dkk. [8]	Barr dkk. [11]	Ali dkk. [17]
Domain	Klinik gigi	Klinik rawat jalan	Klinik rawat jalan	Software testing umum	*Regression testing* industri
Fokus utama	*Conflict prevention* + *automated testing*	Optimasi slot jadwal	Praktik penjadwalan nyata	*Oracle problem*	Relevansi industri *regression testing*
Artefak executable	Ya (12 ACT, 36 *subcase*)	Tidak	Tidak	Tidak	Tidak
Traceability rule-testcase	Ya	Tidak	Tidak	Tidak	Tidak

METODE PENELITIAN

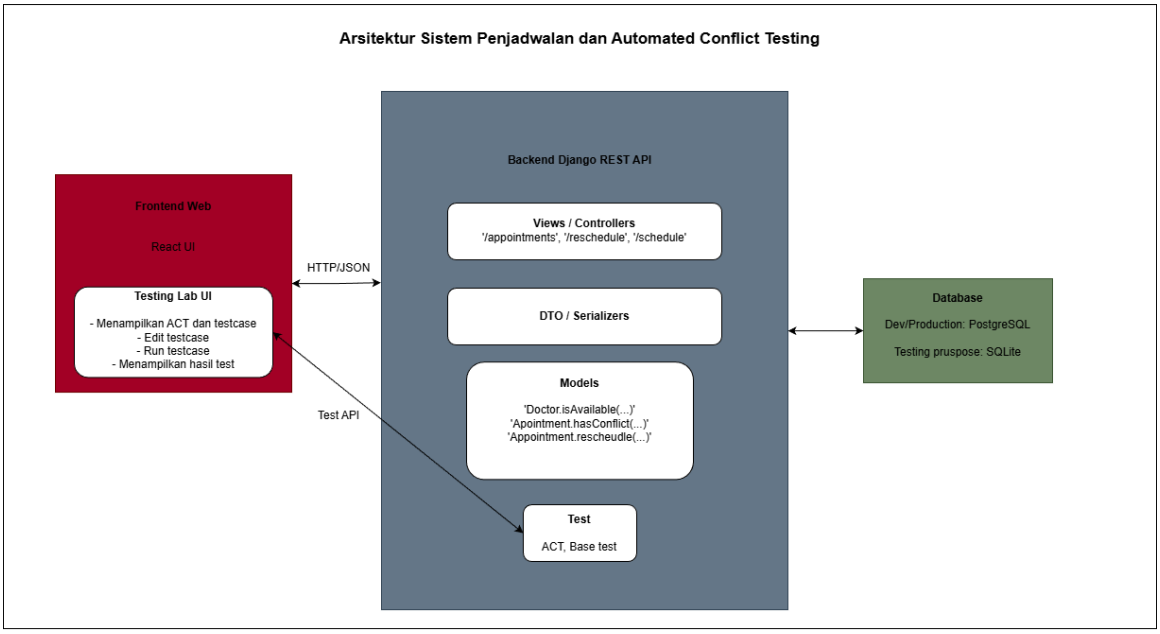
Tahapan Penelitian

Penelitian ini disusun dalam enam tahapan berurutan: (1) analisis kebutuhan dan identifikasi kondisi konflik pada modul penjadwalan; (2) perancangan *rule conflict checking* pada arsitektur Django REST API; (3) implementasi sistem beserta *endpoint booking* dan *reschedule*; (4) desain dua belas skenario *automated conflict testing* (ACT-01–ACT-12) dengan 36 *subcase*; (5) eksekusi *suite* pengujian secara otomatis pada *environment* terisolasi; dan (6) analisis hasil serta penyusunan klaim kontribusi. Tahapan tersebut bersifat iteratif terbatas: tahap eksekusi dapat memicu kembali tahap implementasi apabila ditemukan kegagalan testcase. Keseluruhan alur tahapan ditunjukkan pada Gambar 1.

Gambar 1. Tahapan Penelitian Implementasi Dan Validasi Automated Conflict Testing

Gambaran Sistem

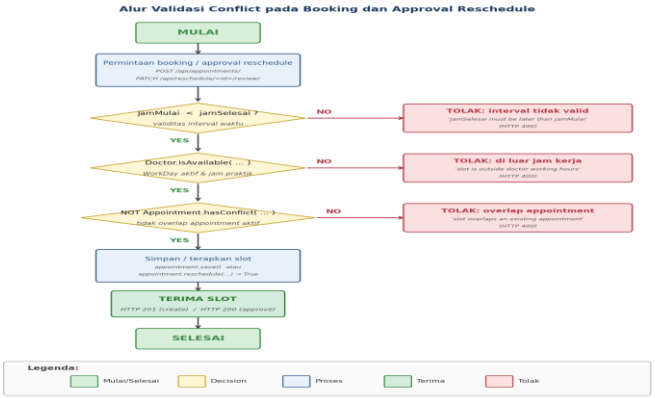
Sistem penjadwalan klinik gigi menggunakan arsitektur *frontend-backend*. *Backend* dibangun dengan Django REST API dan menangani validasi *conflict checking*. Entitas utama yang relevan adalah Doctor, WorkDay, Patient, Appointment, dan RescheduleRequest. Model Doctor berelasi *many-to-many* dengan WorkDay melalui jadwalMingguan untuk menentukan hari dan jam kerja aktif. Model Appointment menjadi pusat deteksi overlap melalui *method* hasConflict(...). *Automated conflict testing* ditempatkan pada *layer backend* karena keputusan menerima atau menolak slot sepenuhnya ditentukan oleh logika validasi di sisi server, sebagaimana ditunjukkan pada Gambar 2.



Gambar 2. Arsitektur Sistem Penjadwalan Klinik Gigi Dan Posisi Automated Conflict Testing Pada Layer Backend

Alur Validasi *Conflict*

Validasi *conflict* dilakukan melalui urutan pemeriksaan yang konsisten untuk pembuatan *appointment* baru maupun *approval reschedule*: (1) validitas interval (jamMulai < jamSelesai); (2) ketersediaan dokter via Doctor.isAvailable(...); dan (3) pengecekan *overlap* via Appointment.hasConflict(...). Hanya slot yang lolos seluruh tahapan yang dapat diterima sistem, seperti yang ditunjukkan pada Gambar 3. Alur ini menjadi landasan langsung bagi desain skenario ACT, di mana setiap kelompok skenario memetakan satu tahap validasi spesifik.



Gambar 3. Alur Validasi Conflict Yang Menjadi Dasar Penyusunan Skenario Automated Conflict Testing

Skenario *Automated Conflict Testing*

Suite automated conflict testing terdiri atas dua belas skenario utama dengan 36 *subcase* yang mencakup kondisi *nominal*, *boundary*, dan *negative case* pada dua level verifikasi. Level model memeriksa *rule* inti pada *method* Doctor.isAvailable(...), Appointment.hasConflict(...), dan Appointment.reschedule(...). Level API memeriksa apakah *rule* yang sama tetap ditegakkan pada *endpoint* POST /api/appointments/ dan PATCH /api/reschedule/<id>/review/. Setiap skenario diberi kode ACT dan diimplementasikan dalam *file test* terpisah pada folder backend/core/tests/scheduling/automated_conflict/. *Suite* dijalankan via python manage.py test core.tests.scheduling.automated_conflict --settings=clinic_scheduler.settings_test.

e-ISSN 2828-0490
JURNAL MEDIA COMPUTER SCIENCE

Tabel 2. Pemetaan Skenario ACT Terhadap Aturan Dan *Expected Behavior*

Kode ACT	Level	Aturan / Skenario	*Expected Behavior*
ACT-01	Model	Slot valid pada *active workday* dokter	*Availability check* menerima slot valid termasuk *boundary* buka dan tutup
ACT-02	Model	*Inactive workday* atau hari tidak cocok	*Availability check* menolak slot ketika *workday* tidak aktif
ACT-03	Model	*Reschedule* ke slot baru yang valid	Appointment.reschedule(...) berhasil dan memperbarui jam *appointment*
ACT-04	Model	*Reschedule* ke slot di luar jam kerja	Appointment.reschedule(...) ditolak, data *appointment* lama tidak berubah
ACT-05	Model	Deteksi *overlap* parsial dan interval penuh	*Conflict detector* mengenali bentrok waktu sebagai konflik
ACT-06	Model	*Reschedule* ke slot yang *overlap appointment* lain	*Reschedule* ditolak saat target berbenturan dengan slot aktif lain
ACT-07	API	*Create appointment* di luar jam kerja dokter	*Endpoint* mengembalikan HTTP 400 dan tidak menambah data
ACT-08	API	*Create appointment* dengan overlap	*Endpoint* mengembalikan HTTP 400 untuk seluruh variasi *overlap*
ACT-09	API	*Create appointment back-to-back*	Slot berdempetan tanpa irisan tetap diterima dengan HTTP 201
ACT-10	API	Pengaruh status *cancelled* pada *overlap checking*	*Appointment cancelled* diabaikan; konflik aktif tetap ditolak
ACT-11	API	*Approval reschedule* ke slot konflik	HTTP 400, status *request* tetap pending, *appointment* tidak berubah
ACT-12	API	*Approval reschedule* ke slot kosong dan valid	HTTP 200, status *approved*, *appointment* diperbarui

HASIL DAN PEMBAHASAN

Hasil

Pengujian dilakukan dengan mengeksekusi suite ACT pada *environment* clinic_scheduler.settings_test. Seluruh skenario ACT-01 sampai ACT-12 beserta 36 *subcase* berada pada status lulus. Tingkat keberhasilan *suite* mencapai 36/36 *subcase* (100%) untuk skenario yang didefinisikan. Pada level model, seluruh *rule* inti ketersediaan dokter, deteksi *overlap*, dan *reschedule* menunjukkan perilaku yang sesuai. Pada level API, *rule* yang sama tetap diterapkan ketika sistem menerima *request* pembuatan *appointment* baru maupun *approval reschedule* (Barr et al., 2015; Pezze & Zhang, 2014).

Tabel 3. Ringkasan Kuantitatif Hasil Eksekusi Suite Automated Conflict Testing.

Indikator Hasil Pengujian	Nilai
Total skenario utama (ACT)	12
Total *subcase executable*	36
Cakupan level model	6 ACT / 18 *subcase*
Cakupan level API	6 ACT / 18 *subcase*
Subcase lulus	36
Subcase gagal	0
Status akhir *suite*	*Pass*

Tabel 4. Perbandingan Expected Result Dan Actual Result Pada Tingkat ACT.

Kode ACT	Expected Result	Actual Result	Status
ACT-01	Slot valid pada *active workday* diterima, termasuk *boundary* buka dan tutup	Slot tengah hari, batas buka, dan batas tutup seluruhnya diterima	*Pass*
ACT-02	Slot ditolak ketika *WorkDay* tidak aktif atau hari tidak cocok	Slot tidak tersedia pada hari nonaktif	*Pass*
ACT-03	*Reschedule* ke slot valid berhasil dan *appointment* diperbarui	*Appointment* berhasil dipindahkan ke slot kosong	*Pass*
ACT-04	*Reschedule* ke slot di luar jam kerja ditolak	Permintaan sebelum jam buka dan melewati jam tutup seluruhnya ditolak	*Pass*
ACT-05	*Overlap* parsial maupun penuh terdeteksi sebagai konflik	*Overlap* sisi kiri, sisi kanan, dan *enclosing* interval terdeteksi	*Pass*
ACT-06	*Reschedule* ke slot berbenturan dengan *appointment* lain ditolak	*Appointment* tidak berubah ketika target *overlap* dengan slot aktif lain	*Pass*
ACT-07	Pembuatan *appointment* di luar jam kerja ditolak oleh API	*Endpoint* mengembalikan HTTP 400 dan tidak menambah data	*Pass*

e-ISSN 2828-0490
JURNAL MEDIA COMPUTER SCIENCE

ACT-08	Pembuatan *appointment* dengan *overlap* ditolak oleh API	HTTP 400 untuk seluruh variasi *overlap*	*Pass*
ACT-09	Pembuatan *appointment back-to-back* diterima	Slot berdempetan tanpa irisan berhasil dibuat dengan HTTP 201	*Pass*
ACT-10	*Appointment cancelled* diabaikan; konflik aktif tetap ditolak	*Booking* diterima jika hanya *cancelled*; ditolak jika masih ada konflik aktif	*Pass*
ACT-11	*Approval reschedule* ke slot konflik ditolak	HTTP 400, status *request* tetap pending, *appointment* tidak berubah	*Pass*
ACT-12	*Approval reschedule* ke slot valid berhasil	HTTP 200, *request approved*, *appointment* diperbarui	*Pass*

Pembahasan

Pada aspek ketersediaan dokter, pencegahan konflik berbasis WorkDay berjalan tepat pada batas operasional. ACT-01 membuktikan bahwa slot di dalam jam kerja, termasuk tepat pada batas buka dan tutup, tetap diterima. ACT-02, ACT-04, dan ACT-07 menunjukkan bahwa sistem menolak slot yang tidak memiliki WorkDay aktif atau berada di luar rentang operasional dokter, sejalan dengan pentingnya pengelolaan batas kapasitas layanan (Ahmadi-Javid et al., 2017).

Aturan *overlap* pada *appointment* aktif terbukti diterapkan dengan akurat di kedua level verifikasi. ACT-05 memperlihatkan bahwa detektor konflik mengenali *overlap* parsial di sisi kiri, sisi kanan, dan interval yang menutup penuh. Diperkuat oleh ACT-08 pada level API. ACT-09 menunjukkan sistem tidak terlalu restriktif: slot *back-to-back* yang tidak beririsan tetap diterima. Sistem mampu membedakan konflik yang harus ditolak dari *adjacency* yang masih sah, sesuai prinsip *oracle* yang sensitif terhadap pelanggaran berisiko tanpa menghasilkan *false rejection*(Barr et al., 2015; Pezze & Zhang, 2014).

Aspek penanganan status *appointment* dan *reschedule* menghasilkan temuan yang secara operasional paling kritis. ACT-10 menunjukkan *appointment* cancelled tidak dihitung sebagai konflik aktif, tetapi *subcase* lain dalam ACT-10 memperlihatkan bahwa sistem tidak melonggarkan aturan terhadap konflik aktif lain meskipun terdapat *appointment* cancelled di sekitarnya. ACT-11 dan ACT-12 membuktikan bahwa *approval reschedule* memvalidasi ulang konflik sebelum perubahan diterapkan, sehingga jalur *review* tidak menjadi celah yang melewati *conflict checking* (Rosero et al., 2016; Yoo & Harman, 2012).

Implementasi Conflict Checking

Listing 1 menampilkan fungsi validasi slot *appointment* pada *backend* yang menjadi unit utama yang diverifikasi oleh ACT level API.

Listing 1. Cuplikan fungsi validasi slot appointment pada backend

```
def validate_appointment_slot(
    doctor, tanggal, jam_mulai, jam_selesai,
    exclude_appointment_id=None,
):
```

```
if jam_mulai >= jam_selesai:
    raise serializers.ValidationError(
        {"jamSelesai": "End time must be later than start time."}
    )
is_available = doctor.isAvailable(
    tanggal,
    datetime.combine(tanggal, jam_mulai),
    datetime.combine(tanggal, jam_selesai),
)
if not is_available:
    raise serializers.ValidationError(
        {"non_field_errors": ["Requested slot is outside the doctor working hours."]}
    )
has_conflict = Appointment.hasConflict(
    doctor, tanggal, jam_mulai, jam_selesai,
    exclude_appointment_id=exclude_appointment_id,
)
if has_conflict:
    raise serializers.ValidationError(
        {"non_field_errors": ["Requested slot overlaps an existing appointment."]}
    )
```

Tingkat kelulusan 100% tidak otomatis berarti *suite* dirancang terlalu mudah; validitasnya justru ditopang oleh komposisi *suite* yang tidak hanya berisi kasus nominal, tetapi juga *negative case*, *boundary case*, *acceptance case*, dan *rejection case* pada dua level verifikasi. Jika implementasi terlalu permisif, ACT-08 dan ACT-11 seharusnya lolos secara salah; jika terlalu restriktif, ACT-01, ACT-09, dan ACT-12 seharusnya gagal. *Pass rate* 100% di sini merupakan indikasi konsistensi *oracle* terhadap *rule* yang diuji, bukan tanda bahwa *suite* dirancang terlalu mudah (Barr et al., 2015; Pezze & Zhang, 2014; Taromirad & Runeson, 2025). Dibandingkan *manual testing*, pendekatan ini menawarkan *repeatability* yang lebih tinggi untuk *rule-level regression*, khususnya pada kombinasi *boundary* dan *overlap* yang mudah terlewat saat verifikasi dilakukan secara visual (Alegroth et al., 2016; Bauer et al., 2024; Garousi & Mantyla, 2016).

Keterbatasan penelitian ini meliputi: pengujian belum mencakup *constraint* tambahan seperti ruang tindakan, cuti dokter, atau *blocked schedule* non-rutin; serta belum mengevaluasi kondisi konkurensi seperti dua *request booking* yang datang hampir bersamaan. Pembatasan tersebut bersifat sengaja karena penelitian diarahkan untuk memvalidasi *rule* inti pencegahan *double booking* pada satu dokter sebagai fondasi *suite regresi* pertama yang jelas hubungannya antara *business rule*, target teknis, *oracle*, dan hasil uji.

KESIMPULAN DAN SARAN

Kesimpulan

Penelitian ini menunjukkan bahwa mekanisme conflict checking pada sistem penjadwalan klinik gigi berhasil diimplementasikan untuk mencegah terjadinya double booking. Aturan inti yang berkaitan dengan active workday, batas jam pelayanan, overlap appointment, status appointment, serta validasi reschedule dapat ditegakkan secara stabil pada level model maupun level API. Automated conflict testing terbukti efektif memverifikasi aturan penjadwalan secara berulang, terstruktur, dan dapat ditelusuri melalui dua belas skenario ACT-01 sampai ACT-12 dengan 36 subcase yang seluruhnya lulus. Nilai utama pendekatan ini terletak pada kemampuannya

menghubungkan business rule, testcase, dan hasil pengujian ke dalam satu alur validasi yang terpadu dan dapat direplikasi pada sistem penjadwalan layanan kesehatan serupa.

Saran

Sebagai pengembangan lebih lanjut, penelitian berikutnya dapat memperluas cakupan conflict testing ke skenario yang lebih kompleks, seperti concurrent booking, blocked schedule non-rutin, cuti dokter, resource multi-constraint, dan pengujian di environment yang lebih mendekati production untuk memperkuat validitas eksternal.

DAFTAR PUSTAKA

Ahmadi-Javid, A., Jalali, Z., & Klassen, K. J. (2017). Outpatient appointment systems in healthcare: A review of optimization studies. *European Journal of Operational Research*, *258*(1), 3–34. https://doi.org/10.1016/j.ejor.2016.06.064

Alegroth, E., Feldt, R., & Kolstrom, P. (2016). Maintenance of automated test suites in industry: An empirical study on Visual GUI Testing. *Information and Software Technology*, *73*, 66–80. https://doi.org/10.1016/j.infsof.2016.01.012

Ali, N. B., Engstrom, E., Taromirad, M., Mousavi, M. R., Minhas, N. M., Helgesson, D., Kunze, S., & Varshosaz, M. (2019). On the search for industry-relevant regression testing research. *Empirical Software Engineering*, *24*, 2020–2055. https://doi.org/10.1007/s10664-018-9670-1

Anand, S., Burke, E. K., Chen, T. Y., Clark, J. A., Cohen, M. B., Grieskamp, W., Harman, M., Harrold, M. J., & McMinn, P. (2013). An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software*, *86*(8), 1978–2001. https://doi.org/10.1016/j.jss.2013.02.061

Barr, E. T., Harman, M., McMinn, P., Shahbaz, M., & Yoo, S. (2015). The oracle problem in software testing: A survey. *IEEE Transactions on Software Engineering*, *41*(5), 507–525. https://doi.org/10.1109/TSE.2014.2372785

Bauer, A., Frattini, J., & Alegroth, E. (2024). Augmented testing to support manual GUI-based regression testing: An empirical study. *Empirical Software Engineering*, *29*, 140. https://doi.org/10.1007/s10664-024-10522-z

Borgman, N. J., Vliegen, I. M. H., Boucherie, R. J., & Hans, E. W. (2018). Appointment scheduling with unscheduled arrivals and reprioritization. *Flexible Services and Manufacturing Journal*, *30*, 30–53. https://doi.org/10.1007/s10696-016-9268-0

Butt, S., Khan, S. U. R., Hussain, S., & Wang, W. L. (2023). A conceptual model supporting decision-making for test automation in Agile-based Software Development. *Data and Knowledge Engineering*, *144*, 102111. https://doi.org/10.1016/j.datak.2022.102111

Dantas, L. F., Fleck, J. L., Oliveira, F. L. C., & Hamacher, S. (2018). No-shows in appointment scheduling: A systematic literature review. *Health Policy*, *122*(4), 412–421. https://doi.org/10.1016/j.healthpol.2018.02.002

Garousi, V., & Mantyla, M. V. (2016). When and what to automate in software testing? A multi-vocal literature review. *Information and Software Technology*, *76*, 92–117. https://doi.org/10.1016/j.infsof.2016.04.015

Godefroid, P., de Halleux, J., Nori, A. V, Rajamani, S. K., Schulte, W., Tillmann, N., & Levin, M. Y. (2008). Automating software testing using program analysis. *IEEE Software*, *25*(5), 30–37. https://doi.org/10.1109/MS.2008.109

Kaandorp, G. C., & Koole, G. (2007). Optimal outpatient appointment scheduling. *Health Care Management Science*, *10*, 217–229. https://doi.org/10.1007/s10729-007-9015-x

Kuiper, A., de Mast, J., & Mandjes, M. (2021). The problem of appointment scheduling in outpatient clinics: A multiple case study of clinical practice. *Omega*, *98*, 102122. https://doi.org/10.1016/j.omega.2019.102122

Lou, Y., Chen, J., Zhang, L., & Hao, D. (2019). A survey on regression test-case prioritization. *Advances in Computers*, *113*, 1–46. https://doi.org/10.1016/bs.adcom.2018.10.001

Luo, L., Zhou, Y., & Han, B. T. (2019). An optimization model to determine appointment scheduling window for an outpatient clinic with patient no-shows. *Health Care Management Science*, *22*, 68–84. https://doi.org/10.1007/s10729-017-9421-7

Pezze, M., & Zhang, C. (2014). Automated test oracles: A survey. *Advances in Computers*, *95*, 1–48. https://doi.org/10.1016/B978-0-12-800160-8.00001-2

Rosero, R. H., Gomez, O. S., & Rodriguez, G. (2016). 15 years of software regression testing techniques: A survey. *International Journal of Software Engineering and Knowledge Engineering*, *26*(5), 675–689. https://doi.org/10.1142/S0218194016300013

Taromirad, M., & Runeson, P. (2025). Assertions in software testing: Survey, landscape, and trends. *International Journal on Software Tools for Technology Transfer*, *27*, 117–135. https://doi.org/10.1007/s10009-025-00794-1

Tsai, P.-F. J., & Teng, G.-Y. (2014). A stochastic appointment scheduling system on multiple resources with dynamic call-in sequence and patient no-shows for an outpatient clinic. *European Journal of Operational Research*, *239*(2), 427–436. https://doi.org/10.1016/j.ejor.2014.04.032

Yan, C., Huang, G. Q., Kuo, Y.-H., & Tang, J. (2022). Dynamic appointment scheduling for outpatient clinics with multiple physicians and patient choice. *Journal of Management Science and Engineering*, *7*(1), 19–35. https://doi.org/10.1016/j.jmse.2021.02.002

Yoo, S., & Harman, M. (2012). Regression testing minimization, selection and prioritization: A survey. *Software Testing, Verification and Reliability*, *22*(2), 67–120. https://doi.org/10.1002/stvr.430